ITS
Institut
Teknologi
Sepuluh Nopember

TUGAS AKHIR - KI141502

DESAIN DAN ANALISIS PENERAPAN ALGORITMA MO DAN STRUKTUR DATA *SEGMENT TREE* PADA STUDI KASUS PERMASALAHAN SPOJ KLASIK DCEPCA09-MMM

MIFTAKHUL AKHYAR
NRP 05111440000143

Dosen Pembimbing I
Rully Soelaiman, S.Kom., M.Kom.

Dosen Pembimbing II
Wijayanti Nurul Khotimah S.Kom., M.Sc.

DEPARTEMEN INFORMATIKA
Fakultas Teknologi Informasi dan Komunikasi
Institut Teknologi Sepuluh Nopember
Surabaya, 2018

(Halaman ini sengaja dikosongkan)

TUGAS AKHIR - KI141502

DESAIN DAN ANALISIS PENERAPAN ALGORITMA MO DAN STRUKTUR DATA *SEGMENT TREE* PADA STUDI KASUS PERMASALAHAN SPOJ KLASIK DCEPCA09-MMM

MIFTAKHUL AKHYAR
NRP 05111440000143

Dosen Pembimbing I
Rully Soelaiman, S.Kom., M.Kom.

Dosen Pembimbing II
Wijayanti Nurul Khotimah S.Kom., M.Sc.

DEPARTEMEN INFORMATIKA
Fakultas Teknologi Informasi dan Komunikasi
Institut Teknologi Sepuluh Nopember
Surabaya, 2018

(Halaman ini sengaja dikosongkan)

UNDERGRADUATE THESIS - KI141502

DESIGN AND ANALYSIS OF MO'S ALGORITHM AND SEGMENT TREE DATA STRUCTURE IMPLEMENTATION ON CASE STUDY OF SPOJ CLASSICAL PROBLEM DCEPCA09-MMM

MIFTAKHUL AKHYAR
NRP 05111440000143

Supervisor I
Rully Soelaiman, S.Kom., M.Kom.

Supervisor II
Wijayanti Nurul Khotimah S.Kom., M.Sc.

DEPARTMENT OF INFORMATICS
Faculty of Information and Communication Technology
Institut Teknologi Sepuluh Nopember
Surabaya, 2018

(Halaman ini sengaja dikosongkan)

LEMBAR PENGESAHAN

DESAIN DAN ANALISIS PENERAPAN ALGORITMA MO DAN STRUKTUR DATA *SEGMENT TREE* PADA STUDI KASUS PERMASALAHAN SPOJ KLASIK DCEPCA09-MMM

TUGAS AKHIR

Diajukan Guna Memenuhi Salah Satu Syarat
Memperoleh Gelar Sarjana Komputer
pada
Bidang Studi Algoritma dan Pemrograman
Program Studi S1 Informatika
Fakultas Teknologi Informasi dan Komunikasi
Institut Teknologi Sepuluh Nopember

Oleh :

MIFTAKHUL AKHYAR
NRP: 05111440000143

Disetujui oleh Dosen Pembimbing Tugas Akhir :

Rully Soelaiman, S.Kom., M.Kom. .....
NIP: 197002131994021001 (Pembimbing 1)

Wijayanti Nurul Khotimah S.Kom., M.Sc. .....
NIP: 198603122012122004 (Pembimbing 2)

SURABAYA
Juli 2018

(Halaman ini sengaja dikosongkan)

DESAIN DAN ANALISIS PENERAPAN ALGORITMA MO DAN STRUKTUR DATA *SEGMENT TREE* PADA STUDI KASUS PERMASALAHAN SPOJ KLASIK DCEPCA09-MMM

Nama	:	**MIFTAKHUL AKHYAR**
NRP	:	**05111440000143**
Departemen	:	**Informatika FTIK**
Pembimbing I	:	**Rully Soelaiman, S.Kom., M.Kom.**
Pembimbing II	:	**Wijayanti Nurul Khotimah S.Kom., M.Sc.**

Abstrak

Perkembangan teknologi informasi dalam beberapa dekade terakhir sangat pesat, terutama dalam hal proses komputasi yang memungkinkan pengambilan keputusan dapat dilakukan dengan cepat dan akurat. Salah satu bidang permasalahan yang menarik untuk dieksplorasi dalam bidang komputasi adalah query processing.

Query processing adalah suatu permasalahan untuk memproses beberapa query dari suatu data dengan indeks yang dinamis sesuai dengan kriteria masing-masing query. Tujuan dari query processing adalah menyelesaikan semua query dengan waktu yang efisien dan biaya komputasi yang rendah oleh karena itu diperlukan algoritma yang optimal untuk mencapai hal tersebut. Terdapat dua pendekatan pada metode query processing yaitu online query dan offline query. Pada online query setiap masukan query akan diproses secara langsung. Sedangkan pada offline query, query akan terlebih dahulu disimpan dan diurutkan dengan fungsi tertentu kemudian diproses dan akan menampilkan hasil sebanyak jumlah query yang ada.

Topik Tugas Akhir ini akan mengangkat permasalahan yang terdapat pada Online Judge SPOJ dengan kode DCEPCA09. Pada permasalahan ini, diberikan sebuah array bilangan dengan panjang N dan dua set bilangan i dan j sebanyak Q. Bilangan i dan j merepresentasikan indeks yang ada pada array. Dari setiap interval array indeks i dan j diminta untuk mencari nilai mean, median dan mode. Pada permasalahan ini memiliki tantangan untuk merancang algoritma yang dapat melakukan pemrosesan tiap case secara optimal pada array yang dinamis karena array yang diproses untuk setiap case berbeda sesuai dengan indeks i dan j dengan batasan yang sudah ditentukan. Dengan batasan waktu selama 0.5 detik, nilai N antara 2 sampai 10^4 *dan jumlah query sampai* 10^4 *maka apabila dikerjakan dengan cara naif akan menghasilkan TLE.*

Pada Tugas Akhir ini akan diimplementasikan Algoritma Mo yang merupakan algoritma offline query dengan menggunakan struktur data segment tree. Segment tree digunakan dengan memanfaatkan nilai agregasi untuk menyelesaikan permasalahan pada Tugas Akhir ini yaitu untuk menemukan nilai mean, median dan mode. Hasil dari Tugas Akhir ini diharapkan dapat mendapatkan implementasi algoritma dan struktur data yang tepat untuk memecahkan permasalahan di atas secara optimal dan diharapkan dapat memberikan kontribusi pada perkembangan ilmu pengetahuan dan teknologi informasi.

***Kata-Kunci**: range query, offline query, algortima mo, segment tree*

DESIGN AND ANALYSIS OF MO'S ALGORITHM AND SEGMENT TREE DATA STRUCTURE IMPLEMENTATION ON CASE STUDY OF SPOJ CLASSICAL PROBLEM DCEPCA09-MMM

Name	:	**MIFTAKHUL AKHYAR**
NRP	:	**05111440000143**
Major	:	**Informatics FTIK**
Supervisor I	:	**Rully Soelaiman, S.Kom., M.Kom.**
Supervisor II	:	**Wijayanti Nurul Khotimah S.Kom., M.Sc.**

Abstract

The development of information technology in recent decades is very rapid, especially in terms of computing processes that enable decision making to be done quickly and accurately. One of the most interesting issues to explore in computing is query processing.

Query processing is a problem for processing multiple queries from a data with a dynamic index according to the criteria of each query. The purpose of query processing is to solve all queries with efficient time and low computational cost and therefore an optimal algorithm is needed to achieve this. There are two approaches to query processing methods: online query and offline query. In the online query each input query will be processed directly. While in the offline query, the query will be first stored and sorted with a certain function then processed and display the results as much as the number of existing queries.

This final project topic will use the problems contained in Online Judge SPOJ with DCEPCA09 code. In this problem, an array of numbers of length N and two sets of numbers i and j of Q are given. The numbers i and j represent the indices in the array.

From each array interval between indices i and j are required to find the mean, median and mode values. On this case it has a challenge to design algorithms that can optimally process each case on a dynamic array, because the array processed for each case is different according to index i and j with the limit set by the problem setter. With time limits for 0.5 seconds, the length of array(N) up to 10^4 *and the number of queries(Q) up to* 10^4 *if the problem processed naively will result in TLE.*

This Final Project will be implemented using Mo's algorithm which is an offline query algorithm by using data structure segment tree. Segment tree is used by utilizing the aggregation value to solve the problem in this Final Project to find mean, median and mode. The results of this Final Project is expected to determine the implementation of appropriate algorithms and data structure to solve the above problems optimally and can contribute to the development of science and information technology.

Keywords*:range query, offline query, mo's algorithm, segment tree*

KATA PENGANTAR

بِسْمِ ٱللَّهِ ٱلرَّحْمَٰنِ ٱلرَّحِيمِ

Puji syukur penulis panjatkan kepada Allah SWT. atas rezeki, berkah, kekuatan dan karunia-Nya sehingga penulis dapat menyelesaikan Tugas Akhir yang berjudul :

DESAIN DAN ANALISIS PENERAPAN ALGORITMA MO DAN STRUKTUR DATA *SEGMENT TREE* PADA STUDI KASUS PERMASALAHAN SPOJ KLASIK DCEPCA09-MMM.

Dengan selesainya Tugas Akhir ini diharapkan apa yang telah dikerjakan penulis dapat memberikan kontribusi bagi perkembangan ilmu pengetahuan terutama di bidang teknologi informasi serta bagi diri penulis sendiri selaku peneliti.

Penulis mengucapkan terima kasih kepada semua pihak yang telah memberikan dukungan baik secara langsung maupun tidak langsung selama penulis mengerjakan Tugas Akhir maupun selama menempuh masa studi antara lain:

- Ibu Ani Shofiyah selaku ibu penulis yang selalu memberikan perhatian serta kasih sayang dan tidak pernah lelah menemani penulis.
- Bapak Keman selaku bapak penulis yang selalu memberikan motivasi, mengajarkan tentang kehidupan, dan menjadi sosok teladan yang sangat dihormati oleh penulis.
- Bapak Rully Soelaiman, S.Kom., M.Kom. selaku Dosen Pembimbing yang telah banyak memberikan ilmu, pandangan, nasihat, motivasi, dan bimbingan selama penulis menempuh masa perkuliahan maupun selama pengerjaan Tugas Akhir ini.
- Ibu Wijayanti Nurul Khotimah S.Kom., M.Sc. selaku Dosen Pembimbing yang telah memberikan bimbingan

dan arahan dalam pengerjaan Tugas Akhir ini.

- Nur Fadlilah selaku adik penulis yang memotivasi penulis untuk bisa menjadi kakak yang baik dan dapat dijadikan panutan.
- Suriya yang selalu memberikan perhatian, motivasi, dan mendengarkan keluh kesah penulis.
- Keluarga Besar *Orang Sibuk* yang memberikan penulis tempat tinggal dan menjadi teman-teman terdekat selama perkuliahan.
- Rekan dan sahabat Laboratorium Rekayasa Perangkat Lunak yang memberikan tempat bernaung yang menyenangkan dengan guyonan-guyonan receh yang menyegarkan.
- Rekan dan sahabat developer *Lantai 4* yang selalu menjadi teman untuk berbagi ilmu, cerita dan pengalaman.
- Angkatan TC 2014 yang menjadi teman penulis selama masa perkuliahan.
- Seluruh dosen, karyawan, dan teknisi yang sudah memberikan ilmu dan mengisi hari penulis selama masa perkuliahan.

Penulis menyadari masih ada kekurangan pada Tugas Akhir ini sehingga Penulis mengharapkan kritik dan saran yang membangun untuk pembelajaran dan perbaikan supaya Tugas Akhir ini menjadi lebih baik. Semoga melalui Tugas Akhir ini penulis dapat memberikan manfaat untuk pembaca.

Surabaya, 8 Juni 2018

Miftakhul Akhyar

DAFTAR ISI

ABSTRAK **vii**

ABSTRACT **ix**

KATA PENGANTAR **xi**

DAFTAR ISI **xiii**

DAFTAR TABEL **xvii**

DAFTAR GAMBAR **xix**

DAFTAR KODE SUMBER **xxv**

BAB I PENDAHULUAN **1**
1.1 Latar Belakang 1
1.2 Rumusan Masalah 2
1.3 Batasan Masalah 2
1.4 Tujuan 3
1.5 Manfaat 4
1.6 Metodologi 4
1.7 Sistematika Penulisan 5

BAB II DASAR TEORI **7**
2.1 Deskripsi Umum 7
2.1.1 Ukuran Pemusatan Data 7
2.1.1.1 Nilai Rata-Rata (*mean*) 7
2.1.1.2 Nilai Tengah (*median*) 8
2.1.1.3 Modus (*mode*) 10
2.1.2 *Offline Query* 11
2.2 Algoritma Mo 11
2.2.1 Deskripsi Algoritma 11
2.2.2 Pembagian Blok 12
2.2.3 Pengurutan *Query* 14

2.2.4 Pemrosesan *Query* 15
2.2.5 Kompleksitas algoritma 19
2.3 Struktur Data *Segment Tree* 20
2.3.1 Deskripsi Struktur Data 20
2.3.2 Implementasi *Segment Tree* 22
2.3.2.1 Operasi *Update* 25
2.3.2.2 Operasi *Query* 27
2.3.3 Variasi Nilai Agregasi *Segment Tree* . . . 29
2.3.3.1 Nilai Agregasi: Banyak Data . . 29
2.3.3.2 Nilai Agregasi: Frekuensi Maksimal 31
2.4 Permasalahan DCEPCA09-MMM pada SPOJ . . 32
2.5 Penyelesaian Permasalahan DCEPCA09-MMM . 34

BAB III DESAIN DAN PERANCANGAN 37
3.1 Desain Penyelesaian Permasalahan *DCEPCA09-MMM* 37
3.1.1 Definisi Umum Program 37
3.1.2 Desain Algoritma 39
3.1.2.1 Membaca Masukan data 40
3.1.2.2 Memadatkan dan Menentukan Indeks Posisi Data 40
3.1.2.3 Menghitung Konstanta Algoritma Mo 44
3.1.2.4 Menentukan Urutan *Query* . . . 45
3.1.2.5 Desain Fungsi *solveMMM* . . . 46
3.1.2.6 Desain Fungsi *update* 48
3.1.2.7 Desain Fungsi *findMedian* . . . 51
3.1.2.8 Desain Fungsi *cari* 52
3.1.2.9 Menampilkan nilai *mean, median* dan *mode* 53
3.1.2.10 Desain Data *Generator* 54

BAB IV IMPLEMENTASI **57**
4.1 Lingkungan Implementasi 57
4.2 Implementasi Penyelesaian Permasalahan DCEPCA09-MMM 57
4.2.1 Penggunaan *Library*, Konstanta, Variabel Global, *Struct* dan *Template* 57
4.2.2 Implementasi Fungsi untuk Memadatkan *Array* dan Menentukan Indeks Posisi Data 62
4.2.3 Implementasi Fungsi untuk Menghitung Konstanta Algoritma Mo 65
4.2.4 Implementasi Fungsi untuk Mengurutkan *Query* 65
4.2.4.1 Implementasi Fungsi *compare* . 66
4.2.5 Implementasi Fungsi *solveMMM* 66
4.2.6 Implementasi Fungsi *update* 68
4.2.7 Implementasi Fungsi *findMedian* 69
4.2.8 Implementasi Fungsi *cari* 70
4.2.9 Implementasi Fungsi untuk Menampilkan Hasil 71
4.2.10 Implementasi Data *Generator* 72

BAB V UJI COBA DAN EVALUASI **75**
5.1 Lingkungan Uji Coba 75
5.2 Uji Coba Kebenaran 75
5.3 Uji Coba Kinerja 107
5.3.1 Pengaruh Nilai Jumlah Banyak Data terhadap Efisiensi Waktu Pemrosesan . . . 108
5.3.2 Pengaruh Nilai Jumlah *Query* terhadap Efisiensi Waktu Pemrosesan 109
5.3.3 Pengaruh Pengurutan *Query* pada Algortima Mo terhadap Efisiensi Waktu Pemrosesan 112

BAB VI KESIMPULAN DAN SARAN **117**
6.1 Kesimpulan 117
6.2 Saran 118

DAFTAR PUSTAKA **119**

Lampiran A Kode Sumber Penyelesaian DCEPCA09-MMM **121**

Lampiran B Data Uji Coba Kebenaran Pada Perbandingan Metode Naif **127**

Lampiran C Hasil Uji Coba Kinerja **131**

BIODATA PENULIS **135**

DAFTAR TABEL

4.1 Tabel Daftar Variabel Global Permasalahaan DCEPCA09-MMM Bagian 1 58
4.2 Tabel Daftar Variabel Global Permasalahaan DCEPCA09-MMM Bagian 2 59

5.1 Tabel Perhitungan Nilai Blok untuk Setiap $Query$ 79
5.2 Tabel Perhitungan Nilai Blok untuk Setiap $Query$ 79
5.3 Tabel Hasil Perhitungan $Mean$, $Median$, $Mode$ pada $Query$ dengan Batas 1 dan 2 90
5.4 Tabel Hasil Perhitungan $Mean$, $Median$, $Mode$ pada $Query$ dengan Batas 1 dan 2 97
5.5 Tabel Hasil Perhitungan $Mean$, $Median$, $Mode$ pada $Query$ dengan Batas 2 dan 4 103
5.6 Tabel Hasil Perhitungan $Mean$, $Median$, $Mode$ pada Semua $Query$ 103
5.7 Tabel Hasil Perhitungan $Mean$ Secara Manual . . 103
5.8 Tabel Hasil Perhitungan $Median$ Secara Manual 103
5.9 Tabel Hasil Perhitungan $Mode$ Secara Manual . . 104
5.10 Hasil Uji Coba Perbandingan Nilai *Mean* Hasil Metode Naif dengan Metode Algoritma Mo Menggunakan *Segment Tree* 106
5.11 Hasil Uji Coba Perbandingan Nilai *Median* Hasil Metode Naif dengan Metode Algoritma Mo Menggunakan *Segment Tree* 106
5.12 Hasil Uji Coba Perbandingan Nilai *Mode* Hasil Metode Naif dengan Metode Algoritma Mo Menggunakan *Segment Tree* 107
5.13 Tabel Pengaruh Jumlah Data Terhadap Kinerja Waktu Pemrosesan 108
5.14 Tabel Pengaruh Jumlah Query Terhadap Kinerja Waktu Pemrosesan 110

5.15 Tabel Perbandingan Kinerja Waktu Pemrosesan dengan dan Tanpa Pengurutan $Query$ pada Kasus Perubahan Jumlah Data 113
5.16 Tabel Perbandingan Kinerja Waktu Pemrosesan dengan dan Tanpa Pengurutan $Query$ pada Kasus Perubahan Jumlah $Query$ 114

C.1 Hasil Uji Coba Waktu Komputasi pada Kasus Perubahan Jumlah Data dengan Jumlah *Query* Tetap 131
C.2 Hasil Uji Coba Waktu Komputasi pada Kasus Perubahan Jumlah *Query* dengan Jumlah Data Tetap 132
C.3 Hasil Uji Coba Waktu Komputasi Tanpa Pengurutan Query pada Kasus Perubahan Jumlah Data dengan Jumlah *Query* Tetap 133
C.4 Hasil Uji Coba Waktu Komputasi Tanpa Pengurutan Query pada Kasus Perubahan Jumlah *Query* dengan Jumlah Data Tetap 134

DAFTAR GAMBAR

2.1 Ilustrasi Frekuensi Data untuk Mencari $Mean$ . . 8
2.2 Ilustrasi Data Terurut untuk Mencari $Median$ Data Ganjil 9
2.3 Ilustrasi Data Terurut untuk Mencari $Median$ Data Genap 9
2.4 Ilustrasi Frekuensi Data untuk Mencari $Mode$ . . 10
2.5 Ilustrasi Pembagian Blok pada Algoritma Mo . . . 13
2.6 Ilustrasi Inisialisasi Pemrosesan $Query$ 16
2.7 Ilustrasi Pemrosesan $Query$ dengan Batas {0,4} . 17
2.8 Ilustrasi Pemrosesan $Query$ dengan Batas{0,7} . 18
2.9 Komponen pada *Segment Tree* 20
2.10 Representasi $Array$ pada *Segment Tree* dengan Nilai Agregasi Penjumlahan Data 23
2.11 Jumlah $Node$ pada *Segment Tree* 24
2.12 Rekursi dan Pengisian Data pada *Leaf Node* . . . 24
2.13 Rekursi dan Pengisian Data Hasil Agregasi pada $Node$ 24
2.14 Rekursi pada $Root$ dan Pengisian Nilai Total Agregasi 25
2.15 Rekursi dan Perubahan Nilai pada $Node$ [4,4] . . 26
2.16 Rekursi dan Perubahan Nilai pada $Node$ [4,5] . . 26
2.17 Rekursi dan Perubahan Nilai pada $Node$ [4,7] . . 26
2.18 Rekursi dan Perubahan Nilai pada $Node$ [0,7] . . 27
2.19 $Node$ [0,3] yang Sepenuhnya Berada pada Rentang $Query$ 28
2.20 $Node$ [4,4] yang Sepenuhnya Berada pada Rentang $Query$ 28
2.21 $Node$ pada *Segment Tree* 30
2.22 Representasi *Segment Tree* dengan Agregasi Banyak Data 30
2.23 Representasi *Segment Tree* dengan Agregasi Frekuensi Maksimal 32

2.24 Contoh Masukan dan Luaran Permasalahan DCEPCA09-MMM 33

3.1 *Pseudocode* Fungsi *main* Bagian 1 38
3.2 *Pseudocode* Fungsi *main* Bagian 2 39
3.3 Ilustrasi Jumlah $node$ pada Segment Tree dengan Nilai Agregasi Banyak Data 41
3.4 Ilustrasi Pengurutan Data Hasil Masukan 42
3.5 Ilustrasi Iterasi ke 1 dan 4 dengan Data Unik . . . 42
3.6 Ilustrasi Iterasi ke 5 dan 6 dengan Data Tidak Unik 43
3.7 Ilustrasi Penentuan Posisi Indeks Data Terurut . . 43
3.8 $Pseudocode$ Menentukan Indeks Urutan Data . . 44
3.9 $Pseudocode$ Menentukan Konstanta $AlgoritmaMo$ 45
3.10 $Pseudocode$ Membaca Masukan *Query* 46
3.11 $Pseudocode$ Fungsi $compare$ 46
3.12 $Pseudocode$ Fungsi $solveMMM$ 48
3.13 $Pseudocode$ Fungsi $update$ 50
3.14 $Pseudocode$ Fungsi $findMedian$ Bagian 1 51
3.15 $Pseudocode$ Fungsi $findMedian$ Bagian 2 52
3.16 $Pseudocode$ Fungsi $cari$ 53
3.17 $Pseudocode$ Menampilkan Hasil Sesuai Urutan Bagian 1 53
3.18 $Pseudocode$ Menampilkan Hasil Sesuai Urutan Bagian 2 54
3.19 *Pseudocode* Fungsi *Generator* Bagian 1 54
3.20 *Pseudocode* Fungsi *Generator* Bagian 2 55

4.1 Ilustrasi Pengurutan Data Masukan kedalam Variabel $sorted$ 62
4.2 Ilustrasi Pengurutan Data Masukan kedalam Variabel $sorted$ 63
4.3 Ilustrasi Pengurutan Data Masukan kedalam Variabel $sorted$ 63

4.4 Ilustrasi Pengurutan Data Masukan kedalam Variabel $sorted$ 64

5.1 Contoh Kasus Uji Permasalahan DCEPCA09-MMM 76
5.2 Pengurutan Data Masukan ke $Array$ Baru 76
5.3 Pemadatan Data Hasil Pengurutan ke $Array$ Baru 77
5.4 Pemetaan Data dan Posisi Data dari Hasil Pemadatan Data 77
5.5 Pemetaan Ulang Data Masukan Awal Berdasarkan Map Posisi Data 78
5.6 Hasil Pembagian Data menjadi 3 Blok 79
5.7 Inisialisasi Nilai pada *Segment Tree* 80
5.8 Penamaan $Node$ pada *Segment Tree* untuk Memudahkan Penjelasan 80
5.9 Inisialisasi Nilai *currentL* dan *currentR* 81
5.10 Pergeseran Nilai *currentR* dari indeks -1 ke Indeks 0 81
5.11 Perubahan Nilai Agregasi Frekuensi Data dan Indeks Data Karena Penambahan Data 2 82
5.12 Perubahan Nilai Agregasi Jumlah Data Karena Penambahan Data 2 82
5.13 Perubahan Nilai Agregasi Banyak Data Karena Penambahan Data 2 83
5.14 Pergeseran Nilai *currentR* dari indeks 0 ke Indeks 1 83
5.15 Perubahan Nilai Agregasi Frekuensi Data dan Indeks Data Karena Penambahan Data 1 84
5.16 Perubahan Nilai Agregasi Jumlah Data Karena Penambahan Data 1 84
5.17 Perubahan Nilai Agregasi Banyak Data Karena Penambahan Data 1 84
5.18 Pergeseran Nilai *currentR* dari indeks 1 ke Indeks 2 85

5.19 Perubahan Nilai Agregasi Frekuensi Data dan Indeks Data Karena Penambahan Data 0 85
5.20 Perubahan Nilai Agregasi Jumlah Data Karena Penambahan Data 0 86
5.21 Perubahan Nilai Agregasi Banyak Data Karena Penambahan Data 0 86
5.22 Pergeseran Nilai *currentL* dari indeks 0 ke Indeks 1 86
5.23 Perubahan Nilai Agregasi Frekuensi Data dan Indeks Data Karena Pengurangan Data 2 87
5.24 Perubahan Nilai Agregasi Jumlah Data Karena Pengurangan Data 2 87
5.25 Perubahan Nilai Agregasi Banyak Data Karena Pengurangan Data 2 88
5.26 Pergantian Operasi Query Mengubah Batas Kanan dan Kiri menjadi 0 dan 4 90
5.27 Pergeseran Nilai *currentR* dari indeks 2 ke Indeks 3 91
5.28 Perubahan Nilai Agregasi Frekuensi Data dan Indeks Data Karena Penambahan Data 3 92
5.29 Perubahan Nilai Agregasi Jumlah Data Karena Penambahan Data 3 92
5.30 Perubahan Nilai Agregasi Banyak Data Karena Penambahan Data 3 92
5.31 Pergeseran Nilai *currentR* dari indeks 3 ke Indeks 4 92
5.32 Perubahan Nilai Agregasi Frekuensi Data dan Indeks Data Karena Penambahan Data 3 93
5.33 Perubahan Nilai Agregasi Jumlah Data Karena Penambahan Data 3 93
5.34 Perubahan Nilai Agregasi Banyak Data Karena Penambahan Data 3 94
5.35 Pergeseran Nilai *currentL* dari indeks 1 ke Indeks 0 94
5.36 Perubahan Nilai Agregasi Frekuensi Data dan Indeks Data Karena Penambahan Data 2 95

5.37 Perubahan Nilai Agregasi Jumlah Data Karena Penambahan Data 2 95
5.38 Perubahan Nilai Agregasi Banyak Data Karena Penambahan Data 2 95
5.39 Pergantian Operasi Query Mengubah Batas Kanan dan Kiri menjadi 2 dan 4 97
5.40 Pergeseran Nilai *currentL* dari indeks 0 ke Indeks 1 98
5.41 Perubahan Nilai Agregasi Frekuensi Data dan Indeks Data Karena Pengurangan Data 2 99
5.42 Perubahan Nilai Agregasi Jumlah Data Karena Pengurangan Data 2 99
5.43 Perubahan Nilai Agregasi Banyak Data Karena Pengurangan Data 2 99
5.44 Pergeseran Nilai *currentL* dari indeks 1 ke Indeks 2 100
5.45 Perubahan Nilai Agregasi Frekuensi Data dan Indeks Data Karena Pengurangan Data 1 100
5.46 Perubahan Nilai Agregasi Jumlah Data Karena Pengurangan Data 1 101
5.47 Perubahan Nilai Agregasi Banyak Data Karena Pengurangan Data 1 101
5.48 Hasil Luaran Program Menggunakan Data Uji Coba 104
5.49 Hasil Uji Coba pada Situs Penilaian SPOJ 105
5.50 Grafik Pengaruh Jumlah Data Terhadap Kinerja Waktu Pemrosesan 109
5.51 Grafik Pengaruh Jumlah $Query$ Terhadap Kinerja Waktu Pemrosesan 111
5.52 Grafik Perbandingan Hasil Waktu Menggunakan Pengurutan $Query$ dan Tanpa Pengurutan $Query$ pada Kasus Perubahan Jumlah Data 112
5.53 Grafik Perbandingan Hasil Waktu Menggunakan Pengurutan $Query$ dan Tanpa Pengurutan $Query$ pada Kasus Perubahan Jumlah $Query$ 113

5.54 Grafik Rasio Waktu Menggunakan Pengurutan $Query$ dan Tanpa Pengurutan $Query$ pada Kasus Perubahan Jumlah Data 115
5.55 Grafik Rasio Waktu Menggunakan Pengurutan $Query$ dan Tanpa Pengurutan $Query$ pada Kasus Perubahan Jumlah $Query$ 116

DAFTAR KODE SUMBER

4.1 Penggunaan *Library* dan Konstanta pada Penyelesaian Permasalahan DCEPCA09-MMM . 58
4.2 Penggunaan Variabel Global pada Penyelesaian Permasalahan DCEPCA09-MMM Bagian 1 60
4.3 Penggunaan Variabel Global pada Penyelesaian Permasalahan DCEPCA09-MMM Bagian 2 60
4.4 *Struct* $answer$ pada Penyelesaian Permasalahan DCEPCA09-MMM 60
4.5 *Struct* $valMO$ pada Penyelesaian Permasalahan DCEPCA09-MMM 61
4.6 *Struct* seg_tree pada Penyelesaian Permasalahan DCEPCA09-MMM 61
4.7 Penggunaan *Template Fast I/O* pada Penyelesaian Permasalahan DCEPCA09-MMM 61
4.8 Implementasi Fungsi Memadatkan *Array* dan Menentukan Indeks Posisi Data 64
4.9 Implementasi Fungsi Menghitung Konstanta Algoritma Mo 65
4.10 Implementasi Fungsi Mengurutkan *Query* 65
4.11 Implementasi Fungsi *compare* 66
4.12 Implementasi Fungsi *solveMMM* 67
4.13 Implementasi Fungsi *update* 69
4.14 Implementasi Fungsi *findMedian* 70
4.15 Implementasi Fungsi *cari* 71
4.16 Implementasi Fungsi untuk Menampilkan Hasil . 71
4.17 Implementasi Data Generator 72
A.1 Kode Sumber Lengap Penyelesaian DCEPCA09-MMM Bagian 1 121
A.2 Kode Sumber Lengap Penyelesaian DCEPCA09-MMM Bagian 2 122
A.3 Kode Sumber Lengap Penyelesaian DCEPCA09-MMM Bagian 3 123
A.4 Kode Sumber Lengap Penyelesaian DCEPCA09-MMM Bagian 4 124

A.5 Kode Sumber Lengap Penyelesaian DCEPCA09-MMM Bagian 5 125
A.6 Kode Sumber Lengap Penyelesaian DCEPCA09-MMM Bagian 6 126

BAB I

PENDAHULUAN

Pada bab ini akan dijelaskan latar belakang, rumusan masalah, batasan masalah, tujuan, metodologi dan sistematika penulisan Tugas Akhir.

1.1 Latar Belakang

Dalam *competitive programming* terdapat permasalahan yang didasarkan pada proses pencarian *query* nilai berdasarkan rentang data tertentu. Pemrosesan *range query* adalah suatu permasalahan untuk menyelesaikan beberapa *query* dari suatu data dengan rentang indeks yang dinamis sesuai dengan batasan rentang data masing-masing *query*. Tujuan dari pemrosesan *query* adalah menyelesaikan semua *query* dengan waktu yang efisien dan penggunaan memori yang minimal. Dengan demikian diperlukan algoritma yang optimal dan struktur data yang sesuai untuk mencapai hal tersebut. Terdapat dua pendekatan pada metode pemrosesan *query* yaitu *online query* dan *offline query*. Pada *online query* setiap *query* akan diproses secara langsung. Sedangkan pada *offline query*, *query* akan terlebih dahulu disimpan dan diurutkan untuk kemudian diselesaikan dan akan menampilkan hasil sebanyak jumlah *query* yang ada.

Topik tugas akhir ini akan mengangkat permasalahan yang terdapat pada *Online Judge SPOJ* dengan kode DCEPCA09. Pada permasalahan ini, diberikan sebuah *array* bilangan dengan panjang N dan dua set bilangan i dan j sebanyak Q. Bilangan i dan j merepresentasikan indeks pada *array*. Untuk setiap interval *array* indeks i dan j diminta untuk mencari nilai $mean$, $median$ dan $mode$.

Permasalahan ini memiliki tantangan untuk mendesain algoritma yang dapat melakukan pemrosesan *query* secara optimal pada *array* yang dinamis karena *array* yang diproses untuk setiap case berbeda sesuai dengan indeks i dan j. Dengan

batasan waktu selama 0.5 detik, nilai N antara 2 sampai 10^4 dan jumlah *query* sampai 10^4 maka apabila dikerjakan dengan cara *native* akan menghasilkan TLE.

Pada Tugas Akhir ini penulis akan menggunakan pendekatan *offline query* untuk menyelesaikan permasalahan SPOJ klasik DCEPCA09-MMM dengan menggunakan Algoritma Mo dan struktur data *segment tree*. Hasil dari Tugas Akhir ini diharapkan dapat memberikan gambaran mengenai performa Algoritma Mo dan penggunaan struktur data *segment tree* untuk menyelesaikan permasalahan *query* pada permasalahan SPOJ klasik DCEPCA09-MMM secara efektif dan efisien serta dapat memberikan kontribusi pada perkembangan ilmu pengetahuan dan teknologi informasi.

1.2 Rumusan Masalah

Rumusan masalah yang diangkat dalam Tugas Akhir ini adalah sebagai berikut:

1. Bagaimana mendesain dan mengimplementasikan Algoritma Mo menggunakan struktur data *segment tree* untuk menyelesaikan permasalahan SPOJ klasik DCEPCA09-MMM secara efisien?
2. Bagaimana menguji dan menganalisis performa algoritma dan struktur data yang sudah diimplementasikan untuk menyelesaikan permasalahan SPOJ klasik DCEPCA09-MMM?

1.3 Batasan Masalah

Permasalahan yang dibahas pada Tugas Akhir ini memiliki beberapa batasan, yaitu sebagai berikut:

1. Batas maksimum panjang *array* adalah N dengan rentang 2 sampai 10000.

2. Batas maksimum *test case* adalah Q dengan rentang 1 sampai 10000.
3. Batas maksimum bilangan dalam *array* adalah $A[i]$ dengan rentang 0 sampai 10^8.
4. Batas indeks untuk setiap *test case* adalah i dan j dengan syarat $0 <= i < N$ dan $i <= j < N$.
5. Apabila hasil perhitungan $mean$, $median$ dan $mode$ berupa bilangan pecahan maka yang diambil sebagai jawaban adalah bilangan desimalnya.
6. Apabila $mode$ yang didapatkan lebih dari satu maka diambil nilai yang paling besar.
7. *Dataset* yang digunakan adalah dataset pada permasalahan SPOJ klasik DCEPCA09-MMM.
8. Batas maksimum waktu untuk menyelesaikan pemrosesan *dataset* adalah 0.5 detik.
9. Batas maksimum ukuran file *source code* yang dihasilkan adalah $50000B$.
10. Batas maksimum memori RAM yang digunakan untuk pemrosesan *dataset* adalah $1536MB$.

1.4 Tujuan

Tujuan dari Tugas Akhir ini adalah sebagai berikut:

1. Mendesain dan mengimplementasikan Algoritma Mo menggunakan struktur data *segment tree* untuk menyelesaikan permasalahan SPOJ klasik DCEPCA09-MMM secara efisien.
2. Menguji dan menganalisis performa algoritma dan struktur data yang telah diimplementasikan untuk menyelesaikan permasalahan SPOJ klasik DCEPCA09-MMM.

1.5 Manfaat

Tugas Akhir ini diharapkan dapat membantu memahami:

1. Metode kompresi data untuk digunakan pada *segment tree* agar didapatkan jumlah *node* yang minimal.
2. Pemrosesan *range query* dengan mengimplementasikan Algoritma Mo dan menggunakan nilai agregasi pada struktur data *segment tree* untuk menyelesaikan permasalahan SPOJ klasik DCEPCA09-MMM.

1.6 Metodologi

Metodologi yang digunakan dalam pengerjaan Tugas Akhir ini adalah sebagai berikut:

1. Penyusunan proposal Tugas Akhir Tahap pertama dalam proses pengerjaan Tugas Akhir ini adalah menyusun proposal Tugas Akhir. Pada proposal Tugas Akhir ini diajukan sebuah optimisasi pada permasalahan SPOJ klasik DCEPCA09-MMM dengan penggunaan algoritma dan gagasan solusi yang berkaitan dengan permasalahan tersebut. Luaran dari tahap ini adalah proposal Tugas Akhir.
2. Studi literatur Tahap kedua dalam proses pengerjaan Tugas Akhir ini adalah studi literatur. Pada tahap ini dilakukan pencarian informasi dan studi metode penyelesaian masalah terkait penggunaan struktur data yang tepat, dan algoritma yang terkait dengan permasalahan SPOJ klasik DCEPCA09-MMM. Materi-materi tersebut didapatkan dari buku-buku, *paper*, *internet*, serta materi perkuliahan yang terkait. Luaran dari tahap ini adalah daftar pustaka dan referensi.
3. Desain Pada tahap ini dilakukan desain rancangan algoritma dan struktur data yang digunakan dalam solusi

untuk memecahkan permasalahan SPOJ klasik DCEPCA09-MMM. Luaran dari tahap ini adalah algoritma dan struktur data yang digunakan dalam implementasi.

4. Implementasi algoritma dan struktur data Implementasi algoritma adalah tahapan untuk membangun aplikasi yang digunakan dari desain algoritma dan struktur data yang sudah dilakukan pada tahap desain untuk menyelesaikan permasalahan permasalahan SPOJ klasik DCEPCA09-MMM. Luaran dari tahap ini adalah implementasi algoritma yang dibangun menggunakan bahasa pemrograman C++ dan menggunakan IDE Eclipse.
5. Uji coba dan evaluasi Tahap pengujian dan evaluasi adalah tahap untuk menguji *program* yang telah dibuat hasil implementasi algoritma pada permasalahan SPOJ klasik DCEPCA09-MMM. Pengujian dan evaluasi dilakukan hingga mendapatkan hasil *accepted* dari *Online Judge* $SPOJ$. Luaran dari tahapan ini adalah status *accepted* dari *Online Judge* $SPOJ$.
6. Penyusunan buku Tugas Akhir Pada tahap ini dilakukan penyusunan laporan dan buku Tugas Akhir yang berisi dokumentasi hasil pengerjaan Tugas Akhir.

1.7 Sistematika Penulisan

Berikut adalah sistematika penulisan buku Tugas Akhir ini:

1. BABI: PENDAHULUAN Bab ini berisi latar belakang, rumusan masalah, batasan masalah, tujuan, metodologi dan sistematika penulisan Tugas Akhir.
2. BAB II: DASAR TEORI Bab ini berisi dasar teori mengenai permasalahan dan algoritma penyelesaian yang digunakan dalam Tugas Akhir.
3. BAB III: DESAIN Bab ini berisi desain algoritma dan struktur data yang digunakan dalam penyelesaian

permasalahan.

4. BAB IV: IMPLEMENTASI Bab ini berisi implementasi berdasarkan desain algortima yang telah dilakukan pada tahap desain.
5. BAB V: UJI COBA DAN EVALUASI Bab ini berisi uji coba dan evaluasi dari hasil implementasi yang telah dilakukan pada tahap implementasi.
6. BAB VI: KESIMPULAN Bab ini berisi kesimpulan yang didapat dari hasil uji coba yang telah dilakukan.

BAB II

DASAR TEORI

Pada bab ini akan dijelaskan mengenai dasar teori yang menjadi dasar pengerjaan Tugas Akhir ini.

2.1 Deskripsi Umum

Pada subbab ini akan dibahas istilah, definisi, dan deskripsi umum yang digunakan di dalam buku ini. Pembahasan dalam subbab ini memiliki tujuan supaya pembaca dapat lebih mudah memahami dan mengerti isi dari buku ini.

2.1.1 Ukuran Pemusatan Data

Ukuran pemusatan data adalah ukuran yang dapat mewakili data secara keseluruhan apabila data sudah diurutkan [1]. Permasalahan pada Tugas Akhir ini adalah menemukan ukuran pemusatan data yang berupa *mean*, *median* dan *mode* dari sekumpulan data dengan rentang tertentu untuk setiap *query*.

2.1.1.1 Nilai Rata-Rata (*mean*)

Nilai rata-rata atau *mean* adalah nilai yang diperoleh dari jumlah sekelompok data dibagi dengan banyaknya data[1]. Menentukan nilai rata-rata secara umum dapat dirumuskan seperti pada persamaan 2.1.

$$mean = \frac{\sum_{i=1}^{n}(x_i)}{n} \tag{2.1}$$

Dimana x menyatakan data dan n menyatakan banyaknya data. Untuk data yang sudah dikelompokkan dan dihitung frekuensinya(f), nilai $mean$ dapat dihitung dengan rumus pada persamaan 2.2.

$$mean = \frac{\sum_{i=1}^{n}(x_i \times f_i)}{\sum_{i=1}^{n}(f_i)} \tag{2.2}$$

Sebagai contoh terdapat data {6,5,3,7,7,1,3} maka untuk memperoleh $mean$ dapat dihitung dengan persamaan 2.1:

$$mean = \frac{6+5+3+7+7+1+3}{6} = \frac{32}{6} = 5,33$$

Apabila data sudah diurutkan dan dikelompokkan berdasarkan frekuensinya maka dapat dihitung dengan menggunakan rumus pada Persamaan 2.2:

Data	1	3	5	6	7
Frekuensi	1	2	1	1	2

Gambar 2.1: Ilustrasi Frekuensi Data untuk Mencari $Mean$

$$mean = \frac{(1\times 1)+(3\times 2)+(5\times 1)+(6\times 1)+(7\times 2)}{6}$$

$$= \frac{32}{6} = 5,33$$

Didapatkan nilai $mean$ untuk data {6,5,3,7,7,1,3} adalah $5,33$. Pada permasalahan DCEPCA09-MMM terdapat batasan bahwa nilai $mean$ yang menjadi keluaran adalah nilai bilangan bulat dari hasil perhitungan. Sehingga pada tugas akhir ini keluaran untuk nilai $mean$ data {6,5,3,7,7,1,3} adalah 5.

2.1.1.2 Nilai Tengah (*median*)

Nilai tengah atau $median$ adalah nilai data yang terletak di tengah setelah data diurutkan. Dengan demikian, median membagi data menjadi dua bagian yang sama besar[1]. Jika jumlah data ganjil, mediannya adalah data yang berada tepat di tengah. Nilai $median$ dapat dirumuskan dengan persamaan 2.3.

$$median = x_{\frac{n+1}{2}} \tag{2.3}$$

Jika jumlah data genap, mediannya adalah hasil bagi jumlah dua data yang berada di tengah. Nilai $median$ dapat dirumuskan dengan persamaan 2.4.

$$median = \frac{x_{\frac{n}{2}} + x_{\frac{n}{2}+1}}{2} \tag{2.4}$$

Sebagai contoh terdapat data {6,5,3,7,7,1,3} maka untuk memperoleh nilai $median$ dari data tersebut data terlebih dahulu diurutkan. Sehingga didapatkan data terurut pada Gambar 2.2.

Data	1	3	3	5	6	7	7
Urutan	1	2	3	4	5	6	7

Gambar 2.2: Ilustrasi Data Terurut untuk Mencari $Median$ Data Ganjil

Karena jumlah data adalah ganjil maka nilai $median$ dapat ditentukan dengan menggunakan rumus 2.3

$$median = x_{\frac{n+1}{2}} = x_{\frac{7+1}{2}} = x_4$$

Dari hasil perhitungan didapatkan bahwa nilai $mean$ terdapat pada data urutan ke 4 yaitu 5.

Sebagai ilustrasi untuk data berjumlah genap, terdapat data {6,5,3,7,7,1,3,4}. Setelah diurutkan akan didapat data seperti pada Gambar 2.3.

Data	1	3	3	4	5	6	7	7
Urutan	1	2	3	4	5	6	7	8

Gambar 2.3: Ilustrasi Data Terurut untuk Mencari $Median$ Data Genap

Karena jumlah data adalah genap maka nilai $median$ dapat ditentukan dengan menggunakan rumus 2.4

$$median = \frac{x_{\frac{n}{2}} + x_{\frac{n}{2}+1}}{2} = \frac{x_{\frac{8}{2}} + x_{\frac{8}{2}+1}}{2}$$

$$= \frac{x_4 + x_5}{2} = \frac{4+5}{2} = 4,5$$

Didapatkan nilai $median$ untuk data {6,5,3,7,7,1,3,4} adalah $4,5$. Pada permasalahan DCEPCA09-MMM terdapat batasan bahwa nilai $median$ yang menjadi keluaran adalah nilai bilangan bulat dari hasil perhitungan. Sehingga pada tugas akhir ini keluaran untuk nilai $median$ data {6,5,3,7,7,1,3} adalah 5 dan untuk data {6,5,3,7,7,1,3,4} adalah 4.

2.1.1.3 Modus (*mode*)

Modus atau $mode$ adalah nilai yang paling sering muncul dalam data atau nilai yang memiliki frekuensi paling besar[1]. Pada permasalahan DCEPCA09-MMM terdapat batasan apabila dalam data memiliki nilai $mode$ lebih dari satu maka nilai $mode$ yang menjadi keluaran adalah nilai $mode$ yang paling besar.

Sebagai ilustrasi terdapat data {6,5,3,7,7,1,3,4}. Frekuensi untuk data tersebut adalah seperti yang ditunjukkan pada Gambar 2.4.

Data	1	3	4	5	6	7
Frekuensi	1	2	1	1	1	2

Gambar 2.4: Ilustrasi Frekuensi Data untuk Mencari $Mode$

Dari Gambar 2.4 nilai yang memiliki frekuensi tertinggi adalah 3 dan 7 yang memiliki nilai frekuensi 2. sehingga pada permasalahan DCEPCA09-MMM nilai keluaran dari data diatas adalah 7.

2.1.2 *Offline Query*

Offline query adalah penyelesaian permasalahan $query$ dengan cara mengurutkan $query$ yang menjadi masukan dengan urutan tertentu untuk kemudian dikerjakan. Perbedaan dengan *online query* adalah pada *online query* setiap $query$ akan langsung dikerjakan sesuai dengan urutan masukannya dan akan langsung ditampilkan hasilnya. Sedangkan pada *offline query*, $query$ diurutkan terlebih dahulu kemudian $query$ yang sudah terurut akan dikerjakan semuanya, setelah selesai hasil ditampilkan sesuai dengan urutan masukan.

Kelebihan dari *offline query* adalah dapat mengubah urutan pengerjaan $query$ sehingga didapatkan waktu pengerjaan dengan biaya dan waktu yang optimal. Optimalisasi tersebut didapatkan dengan memanipulasi data hasil dari operasi $query$ sebelumnya untuk digunakan pada $query$ yang akan diproses selanjutnya, sehingga mengurangi operasi-operasi berulang yang bisa didapatkan dari hasil $query$ sebelumnya. Oleh karena itu, urutan $query$ akan berpengaruh terhadap efisiensi penyelesaian permasalahan[2].

2.2 Algoritma Mo

Pada subbab ini akan dibahas mengenai Algoritma Mo yang merupakan implementasi dari *offline query* untuk menyelesaikan permasalahan pada Tugas Akhir ini.

2.2.1 Deskripsi Algoritma

Algoritma Mo adalah salah satu algoritma pengurutan $query$ pada *offline query*. Algoritma ini digunakan untuk tipe permasalahan sebagai berikut: diberikan sebuah $array$ A dengan data N dan $query$ sebanyak Q. Setiap $query$ memiliki nilai L dan R sebagai batasan indeks yang akan diproses pada $array$

untuk setiap $query$[3].

Terdapat beberapa syarat yang harus dipenuhi untuk menerapkan Algoritma Mo yaitu:

1. Nilai dari *array* tidak berubah karena proses $query$.
2. Semua $query$ sudah diketahui sebelumnya.
3. Jika $query$ dengan batasan $F([L, R])$ sudah diketahui, maka bisa didapatkan $F([L + 1, R])$, $F([L - 1, R])$, $F([L, R + 1])$ dan $F([L, R - 1])$ dengan kompleksitas waktu masing-masing $\mathcal{O}(F)$.[4]

Pada permasalahan DCEPCA09-MMM data array tidak mengalami perubahan nilai selama pemrosesan $query$. Semua $query$ sudah didapatkan dari masukan sebelum pemrosesan $query$. Fungsi $F()$ adalah fungsi yang digunakan untuk menemukan nilai $mean$, $median$ dan $mode$ dari data [5]. Sehingga dapat diimplementasikan algoritma Mo pada permasalahan DCEPCA09-MMM. Dalam mengimplementasikan algoritma Mo terdapat beberapa tahapan yang dilakukan. Dimulai dari pembagian blok, pengurutan $query$ dan pemrosesan $query$.

2.2.2 Pembagian Blok

Data $array\ A$ akan dikelompokkan pada suatu blok. Setiap $query$ menempati blok tertentu berdasarkan nilai dari indeks batas kiri L. Banyaknya blok didapatkan dari:

$$blok = \sqrt{N} \tag{2.5}$$

Nilai blok adalah bilangan bulat, apabila $\sqrt{N}$ menghasilkan bilangan desimal maka akan dibulatkan ke atas agar jumlah blok dapat menampung semua elemen data. Setiap blok akan berisi data sebanyak:

$$isi = \frac{N}{blok} \tag{2.6}$$

Jika hasil pembagian menghasilkan nilai desimal hasil akan dibulatkan ke atas[6].

Sebagai ilustrasi, terdapat suatu $array$ A dengan data {6,5,3,7,7,1,3,4,11,2,10,7,1}. Dari data tersebut akan dilakukan $query$ untuk mencari jumlah data pada rentang indeks L dan R. Dengan urutan masukan $query$ sebagai berikut: {0,7}, {4,9}, {2,12}, {3,7}, {0,4}. Dari data tersebut dengan jumlah data sebanyak 13 didapatkan jumlah $blok$ dan isi:

$$blok = \lceil \sqrt{13} \rceil = \lceil 3,6 \rceil = 4$$

$$isi = \lceil \frac{13}{4} \rceil = \lceil 3,25 \rceil = 4$$

Dari 13 data akan dibagi menjadi 4 blok dengan masing-masing blok berisi 4 data, kecuali untuk blok terakhir dapat memiliki data kurang dari 4.

	Blok 0				Blok 1				Blok 2				Blok 3
Index	0	1	2	3	4	5	6	7	8	9	10	11	12
Data	**6**	**5**	**3**	**7**	**7**	**1**	**3**	**4**	**11**	**2**	**10**	**7**	**1**

Gambar 2.5: Ilustrasi Pembagian Blok pada Algoritma Mo

Dari blok yang sudah didapatkan, selanjutnya ditentukan blok untuk masing-masing $query$. Berdasarkan nilai dari indeks batasan L. Sehingga didapatkan hasil:

$$\{0,7\} \Longrightarrow Blok\ 0$$

$$\{4,9\} \Longrightarrow Blok\ 1$$

$$\{2,12\} \Longrightarrow Blok\ 0$$

$$\{3,7\} \Longrightarrow Blok\ 0$$

$$\{0,4\} \Longrightarrow Blok\ 0$$

Setelah didapatkan blok dari setiap $query$, langkah selanjutnya adalah mengurutkan pengerjaan $query$.

2.2.3 Pengurutan *Query*

Terdapat beberapa aturan untuk melakukan pengurutan $query$ sesuai dengan ketentuan Algoritma Mo[3]

1. Prioritas pertama adalah operasi $query$ dengan nilai blok paling kecil.
2. Apabila $query$ memiliki nilai blok yang sama, prioritas kedua adalah operasi $query$ dengan batas kanan R paling kecil.
3. Apabila $query$ memiliki nilai blok dan batas kanan R yang sama. prioritas ketiga adalah operasi $query$ yang urutan masuknya paling awal.

Melanjutkan contoh pada pembagian blok subbab 2.2.2 dengan $query$:

$$\{0, 7\} \Longrightarrow Blok\ 0, R : 7$$

$$\{4, 9\} \Longrightarrow Blok\ 1, R : 9$$

$$\{2, 12\} \Longrightarrow Blok\ 0, R : 12$$

$$\{3, 7\} \Longrightarrow Blok\ 0, R : 7$$

$$\{0, 4\} \Longrightarrow Blok\ 0, R : 4$$

Pengurutan $query$ dengan aturan Algoritma Mo akan menghasilkan urutan:

$$\{0, 4\} \Longrightarrow Blok\ 0, R : 4$$

$$\{0, 7\} \Longrightarrow Blok\ 0, R : 7$$

$$\{3, 7\} \Longrightarrow Blok\ 0, R : 7$$

$$\{2, 12\} \Longrightarrow Blok\ 0, R : 12$$

$$\{4, 9\} \Longrightarrow Blok\ 1, R : 9$$

Setelah didapatkan urutan pengerjaan $query$, selanjutnya setiap query akan diproses sesuai dengan urutan tersebut.

2.2.4 Pemrosesan *Query*

Pemrosesan query dimulai dengan melakukan inisialisasi $currentL$ yang merepresentasikan posisi L pada saat iterasi dengan nilai 0, dan nilai $currentR$ yang merepresentasikan posisi R dengan nilai -1. Posisi $currentL$ dan $currentR$ akan diubah secara bertahap satu persatu, bisa penambahan atau pengurangan sesuai dengan batas kiri L dan batas kanan R dari $query$.

Terdapat empat kemungkinan saat mengubah $currentL$ dan $currentR$ dari satu operasi $query$ ke operasi $query$ lainnya, yaitu:

1. Nilai $currentL$ kurang dari batas kiri L. Maka yang dilakukan adalah menambah nilai dari $currentL$ sampai memiliki nilai yang sama dengan batas kiri L. Secara bersamaan mengurangi nilai-nilai elemen *array* yang memiliki indeks $currentL$ dari total yang tersimpan.
2. Nilai $currentL$ lebih dari batas kiri L. Maka yang dilakukan adalah mengurangi nilai dari $currentL$ sampai memiliki nilai yang sama dengan batas kiri L. Secara bersamaan menambah nilai-nilai elemen *array* yang memiliki indeks $currentL$ dari total yang tersimpan.
3. Nilai $CurrentR$ kurang dari batas kanan R. Maka yang dilakukan adalah menambah nilai dari $currentR$ sampai memiliki nilai yang sama dengan batas kanan R. Secara bersamaan menambah nilai-nilai elemen *array* yang memiliki indeks $currentR$ dari total yang tersimpan.
4. Nilai $CurrentR$ lebih dari batas kanan R. Maka yang dilakukan adalah mengurangi nilai dari $currentR$ sampai memiliki nilai yang sama dengan batas kanan R. Secara

bersamaan mengurangi nilai-nilai elemen *array* yang memiliki indeks $currentR$ dari total yang tersimpan.

Dalam proses perubahan posisi $currentL$ atau $currentR$ akan dilakukan proses perhitungan untuk mencari solusi permasalahan dalam rentang indeks $currentL$ dengan $currentR$. Sehingga ketika $currentL$ sama dengan L dan $currentR$ sama dengan R maka akan didapatkan hasil keluaran untuk $query$ dengan batasan tersebut [3].

Sebagai ilustrasi, melanjutkan simulasi pada pengurutan $query$ subbab 2.2.3 dengan urutan $query$ yang sudah didapatkan yaitu {0,4}, {0,7}, {3,7}, {2,12}, {4,9}. Selanjutnya akan dicari jumlah data dalam rentang batas L dan R masing-masing $query$.

Langkah pertama dilakukan inisialisasi nilai $currentL = 0$ dan $currentR = -1$, $jumlahdata = 0$ karena belum ada elemen yg ditambahkan. Sesuai dengan hasil pengurutan, $query$ pertama yang dikerjakan adalah {0,4}.

CurrentR = -1

CurrentL,CurrentR	CL												
Query[L,R]	L				R								
Index	0	1	2	3	4	5	6	7	8	9	10	11	12
Data	6	5	3	7	7	1	3	4	11	2	10	7	1

Hasil Jumlah Data	0

Gambar 2.6: Ilustrasi Inisialisasi Pemrosesan $Query$

Sesuai dengan ketentuan pemrosesan $query$ karena kondisi posisi $currentL$ sama dengan L maka posisi $currentL$ tidak perlu diubah, sedangkan posisi $currentR$ kurang dari R. Maka akan dilakukan penambahan nilai $currentR$ secara bertahap sampai memiliki nilai yang sama dengan R. Secara bersamaan menambah nilai dari $jumlahdata$ dengan data pada indeks $currentR$. Ketika nilai $currentL$ sama dengan L dan $currentR$ sama dengan R maka pemrosesan untuk $query$ tersebut telah selesai dan akan dilanjutkan ke $query$ selanjutnya. Untuk $query$

0

CurrentL,CurrentR	CL,CR												
Query[L,R]	L				R								
Index	0	1	2	3	4	5	6	7	8	9	10	11	12
Data	6	5	3	7	7	1	3	4	11	2	10	7	1
Hasil Jumlah Data	0 + (6) = 6												

CurrentL,CurrentR	CL	CR											
Query[L,R]	L				R								
Index	0	1	2	3	4	5	6	7	8	9	10	11	12
Data	6	5	3	7	7	1	3	4	11	2	10	7	1
Hasil Jumlah Data	6 + (5) = 11												

CurrentL,CurrentR	CL		CR										
Query[L,R]	L				R								
Index	0	1	2	3	4	5	6	7	8	9	10	11	12
Data	6	5	3	7	7	1	3	4	11	2	10	7	1
Hasil Jumlah Data	11 + (3) = 14												

CurrentL,CurrentR	CL			CR									
Query[L,R]	L				R								
Index	0	1	2	3	4	5	6	7	8	9	10	11	12
Data	6	5	3	7	7	1	3	4	11	2	10	7	1
Hasil Jumlah Data	14 + (7) = 21												

CurrentL,CurrentR	CL				CR								
Query[L,R]	L				R								
Index	0	1	2	3	4	5	6	7	8	9	10	11	12
Data	6	5	3	7	7	1	3	4	11	2	10	7	1
Hasil Jumlah Data	14 + (7) = 21												

Gambar 2.7: Ilustrasi Pemrosesan $Query$ dengan Batas {0,4}

CurrentL,CurrentR	CL				CR								
Query[L,R]	L							R					
Index	0	1	2	3	4	5	6	7	8	9	10	11	12
Data	6	5	3	7	7	1	3	4	11	2	10	7	1
Hasil Jumlah Data	21												

CurrentL,CurrentR	CL					CR							
Query[L,R]	L							R					
Index	0	1	2	3	4	5	6	7	8	9	10	11	12
Data	6	5	3	7	7	1	3	4	11	2	10	7	1
Hasil Jumlah Data	21 + (1) = 22												

CurrentL,CurrentR	CL						CR						
Query[L,R]	L							R					
Index	0	1	2	3	4	5	6	7	8	9	10	11	12
Data	6	5	3	7	7	1	3	4	11	2	10	7	1
Hasil Jumlah Data	22 + (3) = 25												

CurrentL,CurrentR	CL							CR					
Query[L,R]	L							R					
Index	0	1	2	3	4	5	6	7	8	9	10	11	12
Data	6	5	3	7	7	1	3	4	11	2	10	7	1
Hasil Jumlah Data	25 + (4) = 29												

Gambar 2.8: Ilustrasi Pemrosesan $Query$ dengan Batas $\{0,7\}$

$\{0,4\}$ didapatkan nilai hasil *jumlah data* adalah 21. Berikutnya akan diproses untuk $query$ urutan selanjutnya yaitu $\{0,7\}$ yang akan diilustrasikan pada Gambar 2.8. Pada operasi $query$ $\{0,7\}$ didapatkan hasil *jumlah data* = 29. Dari ilustrasi dapat dilihat bahwa untuk mengerjakan $query$ $\{0,7\}$ dengan Algoritma Mo tidak perlu menghitung lagi *jumlah data* dari indeks $\{0,4\}$ karena sudah dihitung pada $query$ sebelumnya. Sehingga dengan Algortima Mo dapat diminimalisasi perhitungan nilai yang mengalami $overlapping$. Dengan optimalisasi tersebut, akan mengurangi kompleksitas algoritma untuk menyelesaikan permasalahan yang dikerjakan.

Proses ini akan dilakukan untuk $query - query$ selanjutnya sampai semua $query$ diselesaikan. Setelah semua $query$

diselesaikan, hasil setiap $query$ akan ditampilkan sesuai dengan urutan masukan $query$.

2.2.5 Kompleksitas algoritma

Kompleksitas utama Algoritma Mo terletak pada proses perubahan indeks batas kiri dan batas kanan. Pada perubahan indeks batas kiri, untuk setiap dua operasi $query$ yang ada di dalam satu blok, batas kiri antara dua operasi $query$ tidak akan berubah lebih dari $\sqrt{N}$ langkah. Kemudian perubahan dari batas kiri $query$ blok pertama ke batas kiri $query$ blok selanjutnya tidak akan berubah lebih dari $2\sqrt{N}$. Sehingga untuk perubahan batas kiri antara $query$ satu dengan selanjutnya tidak akan berubah lebih dari $\mathcal{O}(\sqrt{N})$. Untuk operasi $query$ sejumlah Q, terdapat Q langkah saat menambah atau mengurangi elemen untuk setiap kemungkinan sebanyak $\mathcal{O}(Q\sqrt{N})$ dengan setiap langkah membutuhkan kompleksitas $\mathcal{O}(K)$. Sehingga dapat disimpulkan Kompleksitas untuk mengubah batas kiri dari semua operasi $query$ adalah $\mathcal{O}(Q\sqrt{N}K)$[2].

Urutan $query$ pada setiap blok yang sama diurutkan berdasarkan nilai batas kanan dari terkecil ke terbesar. Sehingga $currentR$ akan bergerak dari terkecil ke terbesar dalam satu blok untuk melakukan $query$. Pada kondisi terburuk, posisi batas kanan berada pada nilai yang paling besar dan $query$ pada blok selanjutnya batas kanannya adalah nilai paling kecil. Sehingga untuk setiap blok, batas kanan akan bertambah sampai dengan N. Dikarenakan jumlah blok yang dimiliki adalah $\sqrt{N}$, dan setiap langkah membutuhkan kompleksitas $\mathcal{O}(K)$, maka dapat disimpulkan Kompleksitas untuk mengubah batas kanan dari semua operasi $query$ adalah $\mathcal{O}(N\sqrt{N}K)$[2].

Kompleksitas dari Algoritma Mo terdiri dari perubahan indeks batas kiri dan batas kanan sehingga didapatkan kompleksitasnya adalah $\mathcal{O}((N+Q)\sqrt{N}K)$ [2]. N adalah jumlah banyaknya data, Q adalah jumlah dari operasi $query$, dan

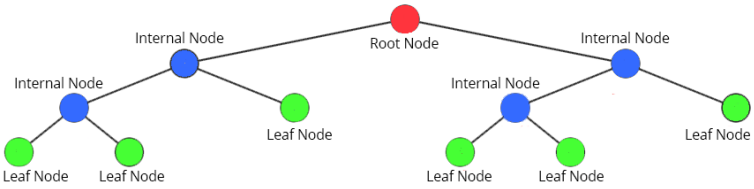
K adalah kompleksitas yang dibutuhkan dalam menyelesaikan permasalahan untuk setiap penambahan atau pengurangan data. Pada contoh ilustrasi 2.2.4 K adalah kompleksitas untuk menghitung *jumlah data* setiap penambahan atau pengurangan data. Sehingga kompleksitas K adalah $linear$ $\mathcal{O}(1)$. Keseluruhan kompleksitas Algoritma Mo untuk contoh ilustrasi pada subbab 2.2.4 untuk mencari *jumlah data* adalah $\mathcal{O}((N+Q)\sqrt{N})$.

2.3 Struktur Data *Segment Tree*

Struktur data *segment tree* biasa digunakan untuk kasus dimana terdapat beberapa $query$ untuk rentang tertentu pada suatu $array$ dan terdapat perubahan pada nilai $array$. Pada subbab ini akan dibahas mengenai struktur data *segment tree* yang akan digunakan untuk menyelesaikan permasalahan pada tugas akhir ini.

2.3.1 Deskripsi Struktur Data

Segment tree pada dasarnya adalah sebuah *binary tree* yang digunakan untuk menyimpan nilai interval atau segmen dari suatu rentang data. Setiap $node$ pada *segment tree* merepresentasikan nilai interval tersebut[7]. Gambar 2.9 merepresentasikan komponen dari suatu *segment tree*.



Gambar 2.9: Komponen pada *Segment Tree*

Tinjau sebuah $array$ A dengan panjang N dan representasinya dalam suatu *segment tree* T:

1. *Root node* pada *segment tree* T merepresentasikan nilai agregasi keseluruhan $array\ A[0 : N-1]$.
2. Setiap *leaf node* pada *segment tree* T merepresentasikan sebuah elemen $array$, sehingga akan tedapat N *leaf node* pada *segment tree*.
3. *Internal node* pada *segment tree* T merepresentasikan nilai agregasi dari interval $array\ A[i\ :\ j]$ atau *leaf node* yang menjadi $child$-nya.

Root node dari *segment tree* akan merepresentasikan keseluruhan $array\ A[0 : N-1]$. Kemudian dari *root node* akan dipecah menjadi setengah segmen atau interval, sehingga akan dihasilkan dua $child$ dari *root node* yang masing-masing merepresentasikan $array\ A[0\ :\ (N-1)/2]$ dan $A[(N-1)/2+1\ :\ (N-1)]$. Pembagian interval atau segmen menjadi setengah akan dilakukan hingga mencapai *leaf node*. Sehingga *segment tree* akan memiliki tinggi $log_2 N$. Terdapat *leaf node* sebanyak N yang merepresentasikan N data pada $array$. Jumlah *internal node* termasuk *root node* adalah $N-1$. Sehingga total keseluruhan $node$ pada *segment tree* untuk merepresentasikan *array* dengan data sebanyak N adalah $2 \times N - 1$.

Struktur data *segment tree* bersifat statis sehingga ketika *segment tree* sudah terbentuk, strukturnya sudah tidak bisa diubah, tapi nilai dari element $node$ masih dapat dilakukan perubahan[7]. Struktur data *segment tree* juga bersifat rekursif. Karena sifatnya yang rekursif *segment tree* mudah untuk diimplementasikan. Pada *Segment tree* menyediakan dua operasi untuk memproses data:

1. $Update$: Operasi ini digunakan untuk merubah nilai data dari $array$. Perubahan tersebut akan merubah nilai $node$ dari *segment tree* yang merepresentasikan interval pada data

yang dirubah.

2. $Query$: Operasi ini digunakan untuk menemaukan nilai dari interval atau segmen pada rentang tertentu dan mengembalikannya sebagai solusi dari permasalahan pada rentang tersebut.

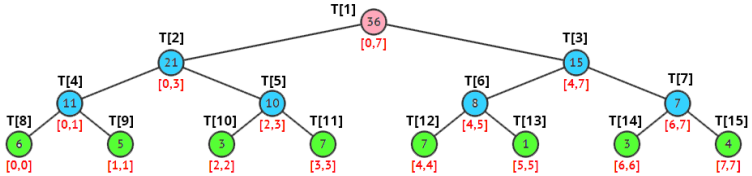
2.3.2 Implementasi *Segment Tree*

Karena *segment tree* adalah sebuah *binary tree*, maka $array$ dapat digunakan untuk merepresentasikan *segment tree*. Selanjutnya adalah menentukan nilai agregasi yang akan disimpan pada *internal node* dari *segment tree*. Sebagai contoh, untuk kasus menemukan jumlah nilai data pada $array$ dengan rentang L dan R seperti contoh pada subbab 2.2.2, maka setiap *node* kecuali *leaf node* akan menyimpan nilai jumlah dari *child node*-nya.

Setelah mengetahui nilai agregasi yang akan disimpan, *segment tree* dapat dibangun menggunakan metode rekursif *bottom-up*. Dimulai dari *leaf node* dan naik sampai ke *root node* sekaligus memperbarui nilai *internal node* yang berada pada jalur dari *leaf node* ke *root node* tersebut. *leaf node* mewakili satu elemen. Di setiap langkah akan digabungkan dua *child node* untuk membentuk *internal node*. Setiap *internal node* akan mewakili nilai agregasi interval dari *child node*-nya. Rekursi akan berakhir di *root node* yang mewakili nilai agregasi seluruh $array$.

Sebagai ilustrasi, representasi sebuah $array$ A{6,5,3,7,7,1, 3,4} pada *segment tree* dengan nilai agregasi jumlah nilai data ditunjukkan oleh Gambar 2.10. Dari Gambar 2.10 dapat dilihat setiap $node$ mewakili nilai agregasi interval dari *node-node* dibawahnya. *Segment tree* T menggunakan $array$ linear pada Gambar 2.10 merepresentasikan interval $array$ A pada rentang:

$$T[1] \Rightarrow A[0, 7]$$

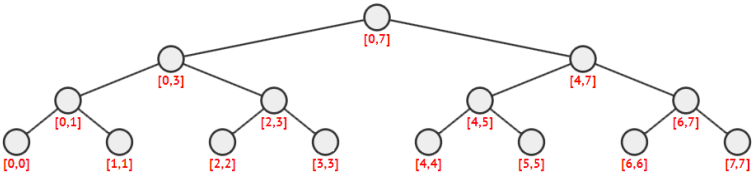


Gambar 2.10: Representasi *Array* pada *Segment Tree* dengan Nilai Agregasi Penjumlahan Data

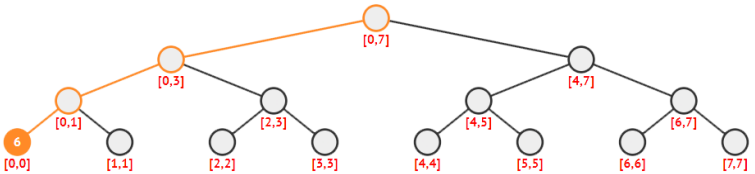
$$T[2] \Rightarrow A[0,3]$$
$$T[3] \Rightarrow A[4,7]$$
$$T[4] \Rightarrow A[0,1]$$
$$T[5] \Rightarrow A[2,3]$$
$$T[6] \Rightarrow A[4,5]$$
$$T[7] \Rightarrow A[6,7]$$
$$T[8] \Rightarrow A[0,0]$$
$$T[9] \Rightarrow A[1,1]$$
$$T[10] \Rightarrow A[2,2]$$
$$T[11] \Rightarrow A[3,3]$$
$$T[12] \Rightarrow A[4,4]$$
$$T[13] \Rightarrow A[5,5]$$
$$T[14] \Rightarrow A[6,6]$$
$$T[15] \Rightarrow A[7,7]$$

Untuk membangun *segment tree* pada Gambar 2.10 untuk $array$ A{6,5,3,7,7,1,3,4} dengan jumlah data sebanyak 8, maka diperlukan $node$ sebanyak $2 \times N - 1 = 15$.

Membangun *segment tree* dengan metode rekursif



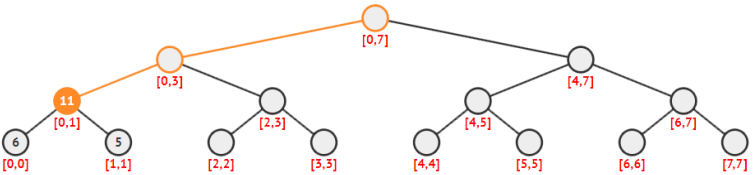
Gambar 2.11: Jumlah $Node$ pada *Segment Tree*



Gambar 2.12: Rekursi dan Pengisian Data pada *Leaf Node*

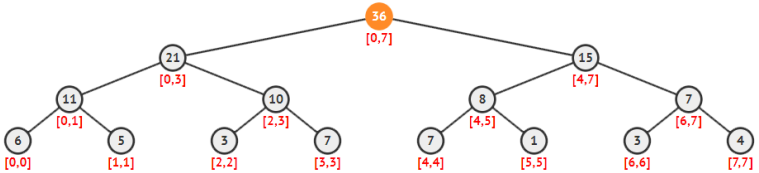
$bottom - up$ dimulai dari mengisi data pada *leaf node* melewati *root node*. Setelah sampai pada *leaf node*, $node$ akan diisi dengan data sesuai dengan elemen $array$ yang direpresentasikannya. Seperti ditunjukkan pada Gambar 2.12.

Setelah $child$ dari $node$ sudah diisi data oleh proses rekursi, selanjutnya $parent$ dari $child$ tersebut yang akan diisi dengan nilai dari agregasi dari kedua childnya. Sebagai contoh setelah $node$ ke-9 dan $node$ ke-10 sudah diisi dengan data 6 dan 5. Maka $node$ ke-4 sebagai $parent$ dari kedua $node$ tersebut akan disi dengan data 11 karena nilai agregasinya adalah jumlah data. Seperti diilustrasikan pada Gambar 2.13.



Gambar 2.13: Rekursi dan Pengisian Data Hasil Agregasi pada $Node$

Proses rekursi ini akan terus dilakukan hingga semua $node$ terisi sampai ke *root node*. Pada *root node* akan terisi nilai jumlah keseluruhan data pada $array$, seperti yang ditunjukkan oleh Gambar 2.14.



Gambar 2.14: Rekursi pada $Root$ dan Pengisian Nilai Total Agregasi

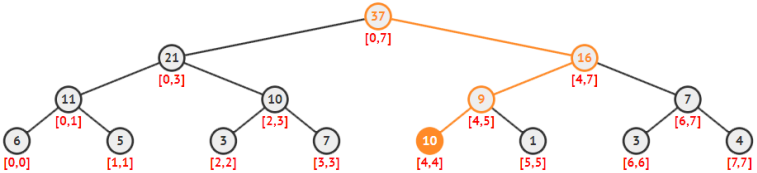
Setelah *segment tree* sudah terbentuk kemudian dapat dilakukan operasi $query$ atau $update$.

2.3.2.1 Operasi *Update*

Untuk melakukan operasi $update$ yang dilakukan adalah mencari *leaf node* yang berisi elemen untuk diperbarui. Pencarian dapat dilakukan dengan pergi ke *child node* kiri atau *child node* kanan tergantung pada interval yang berisi elemen yang akan diperbarui. Setelah menemukan *leaf node*-nya, selanjutnya nilai *leaf node* tersebut akan diperbarui dan menggunakan pendekatan $bottom - up$ untuk memperbarui $internalnode$ yang dilalui pada jalur dari *leaf node* sampai ke *root node*. Kompleksitas dari operasi $update$ adalah $\mathcal{O}(logN)$[8].

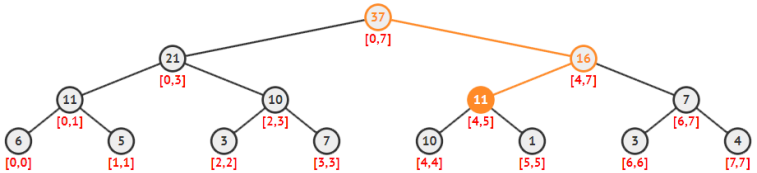
Sebagai ilustrasi operasi $update$, *segment tree* yang sudah dibangun pada subbab 2.3.2 akan dilakukan pembaruan nilai 7 pada $array$ $A[4]$ dengan nilai 10. Langkah yang dilakukan adalah melakukan rekursi mencari *leaf node* yang berisi elemen untuk diperbarui melalui *root node*. Dari *root node* menuju *child* kiri, karena interval pada *child* kiri tidak mencakup indeks 4 maka rekursi dilakukan menuju *child* kanan.

Dari *child* kanan akan dilakukan rekursi hingga menuju *leaf node* [4,4] dan merubah nilai $node$ menjadi 10 seperti ditunjukkan pada Gambar 2.15.

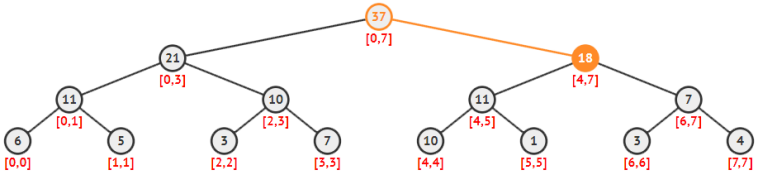


Gambar 2.15: Rekursi dan Perubahan Nilai pada $Node$ [4,4]

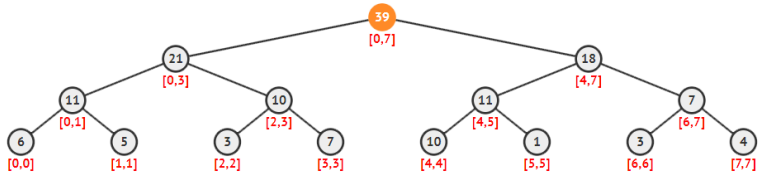
Rekursi $bottom - up$ akan merubah nilai $node - node$ yang dilewati untuk mencapai $node$ [4,4]. Dalam contoh kasus ini adalah $node$ [4,5], [4,7] dan [0,7]. Karena nilai pada $node$ tersebut dipengaruhi oleh nilai pada [4,4]. Perubahan nilai pada setiap $node$ ditunjukkan oleh Gambar 2.16 sampai 2.18.



Gambar 2.16: Rekursi dan Perubahan Nilai pada $Node$ [4,5]



Gambar 2.17: Rekursi dan Perubahan Nilai pada $Node$ [4,7]



Gambar 2.18: Rekursi dan Perubahan Nilai pada $Node$ [0,7]

2.3.2.2 Operasi *Query*

Untuk melakukan operasi $query$ pada *segment tree* dengan rentang dari indeks L ke R, akan dilakukan rekursi pada *segment tree* dimulai dari *root node* dan diperiksa apakah interval yang diwakili oleh $node$ berada dalam rentang dari L ke R. Terdapat tiga kemungkinan kondisi ketika melakukan $query$ rentang pada *node* di *segment tree*:

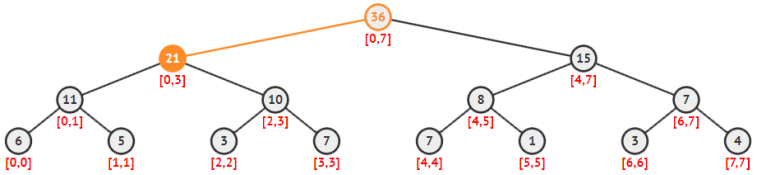
1. Rentang yang direpresentasikan oleh $node$ sepenuhnya didalam rentang yang dicari pada $query$.
2. Rentang yang direpresentasikan oleh $node$ sepenuhnya diluar rentang yang dicari pada $query$.
3. Rentang yang direpresentasikan oleh $node$ sebagian di dalam dan sebagian di luar rentang yang dicari pada $query$.

Jika terjadi kondisi pertama maka nilai yang dikembalikan adalah nilai node tersebut. jika kondisi kedua maka nilai yang dikembalikan adalah 0. Jika kondisi ketiga maka nilai yang dikembalikan adalah jumlah dari hasil operasi $query$ $child$ kiri dan $child$ kanan dari $node$ tersebut. Kompleksitas dari operasi $query$ adalah $\mathcal{O}(logN)$[8].

Sebagai contoh, menggunakan *segment tree* yang sudah dibangun pada subbab 2.3.2 untuk melakukan $query$ mencari jumlah nilai data dengan batasan [0,4]. Untuk operasi $query$ dengan batas $[0, 4]$, yang dilakukan adalah melakukan rekursi mulai dari *root node* ke *child node*-nya dan dilakukan pengecekan hingga rentang dari node yang dikunjungi sudah

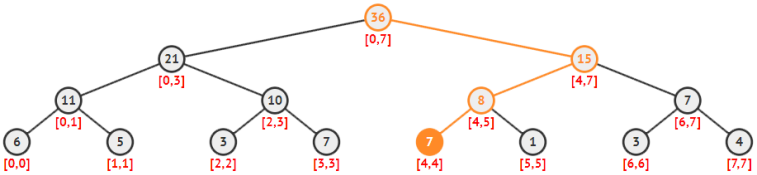
mewakili batasan pada $query$.

Langkah pertama dimulai dari *root node*. Karena *root node* merepresentasikan nilai $[0, 7]$ yang sebagian berada didalam dan sebagian diluar dari rentang query, maka hasil query adalah jumlah hasil $query$ pada $child$-nya. Dari $child$ kiri yang merepresentasikan rentang $[0, 3]$ yang sepenuhnya didalam rentang $query$ yang dicari, maka nilai 21 pada $node$ tersebut akan digunakan sebagai nilai kembalian dari $child$ kiri.



Gambar 2.19: $Node$ [0,3] yang Sepenuhnya Berada pada Rentang $Query$

Dari $child$ kanan yang merepresentasikan rentang $[4, 7]$ yang sebagian didalam dan sebagian diluar rentang $query$ yang dicari, maka akan dilakukan pencarian pada $child$-nya. Setelah melalui beberapa rekursi maka akan didapatkan $node$ yang merepresentasikan rentang $[4, 4]$ yang sepenuhnya berada pada rentang $query$ yang dicari. Sehingga nilai 9 pada $node$ tersebut yang akan dijadikan nilai kembalian untuk $child$ kanan.



Gambar 2.20: $Node$ [4,4] yang Sepenuhnya Berada pada Rentang $Query$

Sehingga solusi dari $query$ rentang $[0, 4]$ adalah nilai $node$ $[0, 3]$ ditambah dengan nilai $node$ $[4, 4]$ yaitu 29.

2.3.3 Variasi Nilai Agregasi *Segment Tree*

Nilai yang disimpan dalam suatu $node$ seperti jumlah nilai data pada contoh pada subbab 2.3.2 disebut dengan nilai agregasi. Nilai agregasi digabungkan dengan prinsip pembagian pertanggung jawaban melalui segmen menjadikan *segment tree* menjadi struktur data yang serba bisa.

Nilai agregasi bisa berupa apa saja, nilai maksimum, jumlah elemen, hasil perkalian, banyaknya pola tertentu yang muncul, sebuah himpunan, dan sebagainya. Selain itu nilai agregasi yang disimpan pada *segment tree* bisa lebih dari satu jenis nilai. Hal ini menjadi kelebihan *segment tree* bila dibandingkan dengan struktur data untuk permasalahan *dynamic range query* lainnya seperti *Binary Indexed Tree*, yang nilai agregasinya cukup terbatas[9].

Selain nilai agregasi jumlah nilai data yang sudah dicontohkan pada subbab 2.3.2 dan dapat digunakan untuk membantu menemukan nilai $mean$, *segment tree* juga dapat menyimpan nilai agregasi banyaknya data, yang dapat digunakan untuk menemukan $median$ dari suatu data. Selain itu nilai agregasi frekuensi maksimal juga dapat digunakan untuk menemukan $mode$ dari suatu data. Ketiga nilai agregasi tersebut akan digunakan untuk menjawab permasalahan pada tugas akhir ini.

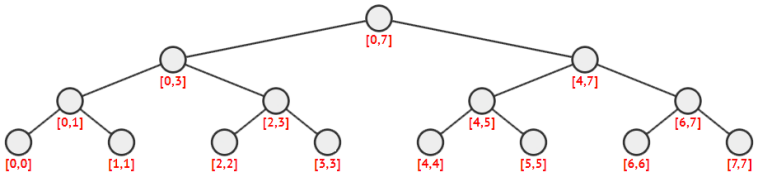
2.3.3.1 Nilai Agregasi: Banyak Data

Nilai agregasi banyak data dapat digunakan untuk menemukan nilai urutan data ke-i dalam kondisi data terurut pada suatu array dengan kondisi:

1. Data $array$ sudah dalam kondisi terurut.
2. Nilai indeks *leaf node* adalah representasi dari nilai data.
3. Nilai $parent\ node$ berisi jumlah dari $child\ node$-nya.

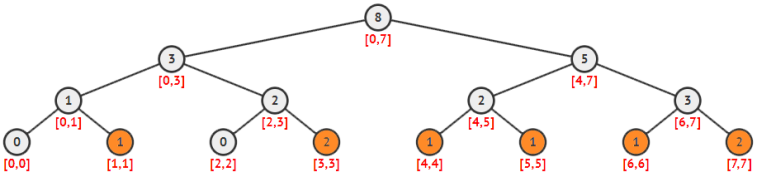
Sebagai ilustrasi, pada Gambar 2.21 terdapat suatu segment

tree, *leaf node* [0,0], [1,1], [2,2]...[7,7] adalah representasi dari nilai 0,1,2...7 pada $array$. Sehingga *leaf node* tersebut akan diisi dengan frekuensi dari nilai data yang direpresentasikannya. Nilai agregasi dari *node* adalah jumlah dari *child node*-nya. Sehingga $root$ pada segment tree ini akan menyimpan jumlah banyaknya data[10].



Gambar 2.21: $Node$ pada *Segment Tree*

Sebagai contoh untuk data $array$ {1,3,3,4,5,6,7,7} akan menghasilkan *segment tree* seperti pada Gambar 2.22.



Gambar 2.22: Representasi *Segment Tree* dengan Agregasi Banyak Data

Dapat dilihat bahwa pada $node$ [0,0] memiliki nilai 0 karena data $array$ tidak memiliki nilai 0. Sedangkan $node$ [1,1] memiliki nilai 1 karena pada data $array$ terdapat nilai 1 sejumlah 1. Untuk melakukan $query$ data urutan ke-i dapat dilakukan dengan rekursi dari *root node*. Selanjutnya dicek apakah nilai dari $child$ kiri kurang dari, sama dengan atau lebih dari i, jika kurang dari atau sama dengan i pencarian akan diteruskan ke $child$ kiri. Jika lebih dari i maka akan dicari data ke [i – *nilai node*] pada $child$ kanan. Pencarian akan berhenti ketika

sudah mencapai *leaf node* yang merupakan solusi dari data ke-i yang dicari.

Sebagai contoh untuk mencari data urutan ke-4, dimulai dari $root$ kemudian di cek data pada $child$ kirinya. Ternyata $child$ kiri memiliki nilai kurang dari 4 yaitu 3. Sehingga pencarian dilanjutkan di $child$ kanan pada $node[4, 7]$ untuk mencari data urutan ke-1, hal ini karena pada $child$ kiri sudah terdapat 3 data. Sehingga untuk mencari urutan ke-4 cukup mencari urutan pertama pada $child$ kanan. Dari $node[4, 7]$ dilakukan pengecekan apakah $child$ kiri memiliki nilai kurang dari atau sama dengan 1, karena memenuhi pencarian dilanjutkan ke $child$ kiri $node[4, 5]$. Dari $node[4, 5]$ dilakukan hal yang sama hingga sampai pada *leaf node*$[4, 4]$. Solusi dari $query$ adalah indeks pada $node$, sehingga nilai urutan data ke-4 adalah 4.

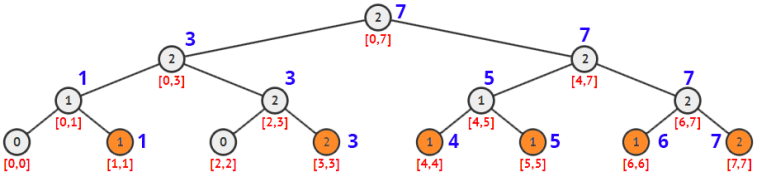
2.3.3.2 Nilai Agregasi: Frekuensi Maksimal

Nilai agregasi frekuensi maksimal dapat digunakan untuk menemukan nilai yang paling sering muncul atau $mode$ pada suatu data dengan kondisi:

1. Data $array$ sudah dalam kondisi terurut.
2. Nilai indeks *leaf node* adalah representasi dari nilai data.
3. Nilai $parent\ node$ berisi frekuensi maksimal $child\ node$-nya dan nilai data yang memiliki frekuensi tersebut.
4. Jika $child\ node$ memiliki frekuensi yang sama, maka diambil nilai data $array$ yang lebih besar.

Sama seperti pada subbab 2.3.3.1, *leaf node* [0,0], [1,1], [2,2]...[7,7] adalah representasi dari nilai 0,1,2...7 pada $array$. Sehingga *leaf node* tersebut akan diisi dengan frekuensi dari nilai data yang direpresentasikannya. Nilai agregasi dari *node* adalah frekuensi maksimal dan nilai datanya dari *child node*. Sehingga $root$ pada segment tree ini akan menyimpan frekuensi maksimal dan nilai $mode$ dari keseluruhan data.

Sebagai contoh untuk data $array$ {1,3,3,4,5,6,7,7} akan menghasilkan *segment tree* seperti pada Gambar 2.23.



Gambar 2.23: Representasi *Segment Tree* dengan Agregasi Frekuensi Maksimal

Dapat dilihat bahwa pada $node$ [0,0] memiliki nilai 0 karena data $array$ tidak memiliki nilai 0. Sedangkan $node$ [1,1] memiliki nilai 1 karena pada data $array$ terdapat nilai 1 sejumlah 1. Apabila $child$ memiliki nilai yang sama maka akan diambil data $array$ yang lebih besar. Seperti yang terjadi pada $node$[4,5] dan [0,7]. Operasi $query$ dilakukan sama dengan yang sudah dijelaskan pada subbab 2.3.2.2 hanya saja nilai yang dibandingkan dan dikembalikan adalah frekuensi maksimal dan nilai $array$-nya. Pada pengerjaan tugas ini, nilai agregasi dari $root$ sudah mengakomodir untuk mencari nilai $mode$ dari data.

2.4 Permasalahan DCEPCA09-MMM pada SPOJ

Permasalahan yang diangkat dalam Tugas Akhir ini bersumber dari *online judge* $SPOJ$ yaitu permasalahan *MMM* dengan kode soal $DCEPCA09$[5]. Pada permasalahan ini, diberikan sebuah *array* bilangan dengan panjang N dan dua set bilangan i dan j sebanyak Q. Bilangan i dan j merepresentasikan indeks pada *array*. Untuk setiap interval *array* indeks i dan j diminta untuk mencari nilai $mean$, $median$ dan $mode$.

Format masukan untuk permasalahan DCEPCA09-MMM adalah:

1. Baris pertama adalah *integer* N, yaitu jumlah elemen dari *array* A.
2. Baris kedua adalah *array* A dengan jumlah sebanyak N elemen yang berisi angka yang menjadi sebuah baris dipisahkan dengan spasi.
3. Baris pertama adalah *integer* Q, yaitu jumlah $query$ yang akan dilakukan.
4. Baris selanjutnya adalah dua pasang bilangan i dan j sebanyak Q yang merepresentasikan batas indeks $query$.

Format keluaran berisi tiga pasang nilai yang menunjukkan nilai $mean$,$median$ dan $mode$ sebanyak Q, hasil diurutkan sesuai dengan urutan masukan $query$. Jika hasil nilai $mean$,$median$ dan $mode$ berupa nilai desimal cukup ditampilkan desimalnya saja sebagai jawaban. Contoh dari format masukan dan keluaran dapat dilihat pada Gambar 2.24.

```
Input:
5
6 5 3 7 7
2
1 2
0 4

 Output:
4 4 5
5 6 7
```

Gambar 2.24: Contoh Masukan dan Luaran Permasalahan DCEPCA09-MMM

Batasan-batasan pada permasalahan DCEPCA09-MMM:

1. Implementasi algoritma menggunakan bahasa pemrograman C++.
2. Batas maksimum panjang *array* adalah N dengan rentang 2 sampai 10000.

3. Batas maksimum *test case* adalah Q dengan rentang 1 sampai 10000.
4. Batas maksimum bilangan dalam *array* adalah $A[i]$ dengan rentang 0 sampai 10^8.
5. Batas indeks untuk setiap *test case* adalah i dan j dengan syarat $0 <= i < N$ dan $i <= j < N$.
6. Apabila hasil perhitungan $mean$, $median$ dan $mode$ berupa bilangan pecahan maka yang diambil sebagai jawaban adalah bilangan desimalnya.
7. Apabila $mode$ yang didapatkan lebih dari satu maka diambil nilai yang paling besar.
8. *Dataset* yang digunakan adalah dataset pada permasalahan SPOJ klasik DCEPCA09-MMM.
9. Batas maksimum waktu untuk menyelesaikan pemrosesan *dataset* adalah 0.5 detik.
10. Batas maksimum ukuran file *source code* yang dihasilkan adalah $50000B$.
11. Batas maksimum memori RAM yang digunakan untuk pemrosesan *dataset* adalah $1536MB$.

2.5 Penyelesaian Permasalahan DCEPCA09-MMM

Pada subbab ini akan dijelaskan bagaimana menyelesaikan permasalahan DCEPCA09-MMM dengan penjelasan permasalahan seperti pada subbab 2.4.

Pada deskripsi soal tersebut diketahui bahwa terdapat operasi $query$ untuk menemukan nilai $mean$,$median$ dan $mode$ dengan rentang data pada indeks i sampai j. Operasi $query$ akan diurutkan menggunakan Algoritma Mo untuk membuat waktu berjalannya solusi menjadi optimal dan efektif.

Dalam penyelesaian permasalahan DCEPCA09-MMM akan digunakan struktur data *Segment Tree* yang menyimpan nilai agregasi nilai penjumlahan data yang digunakan untuk mencari

$mean$, agregasi nilai jumlah banyaknya data untuk $median$ dan agregasi nilai untuk mencari nilai $mode$. Karena banyaknya jenis nilai agregasi yang disimpan, maka *segment tree* akan menggunakan $struct\ array$.

Untuk melakukan optimalisasi, data masukan $array$ akan diringkas dan dikelompokkan berdasarkan urutan dari terkecil ke terbesar sehingga didapatkan data terurut unik. Banyak data terurut unik ini yang akan digunakan untuk membangun *segment tree*. Sehingga nilai yang direpresentasikan pada *leaf node* adalah urutan posisi data.

Desain penyelesaian pada tugas ini dilakukan dengan mengkombinasikan Algoritma Mo dan Segment Tree. Dimulai dari menginisialisasi sebuah tree dengan nilai 0 untuk semua node. Sehingga untuk setiap pergeseran nilai batas kanan atau batas kiri pada Algoritma Mo dilakukan operasi $update$ pada *segment tree*. Operasi $update$ dapat berupa memasukkan data atau mengeluarkan data dari *segment tree* yang akan memperbarui nilai agregasi. Sampai batas kiri dan kanan sesuai dengan batas $query$. Hasil akhir dari sebuah operasi $query$ akan disimpan di dalam sebuah *struct array* $answers[]$ dengan indeks sesuai urutan masuk operasi $query$.

Dengan kompleksitas Algoritma Mo sebesar $\mathcal{O}((M + N)\sqrt{N}K)$, K dalam kasus ini adalah operasi $update$ pada *segment tree*. Sehingga akan didapatkan kompleksitas keseluruhan sebesar $\mathcal{O}((M + N)\sqrt{N}logN)$

(Halaman ini sengaja dikosongkan)

BAB III

DESAIN DAN PERANCANGAN

Pada bab ini akan dijelaskan desain dan perancangan algoritma yang digunakan dalam pengerjaan Tugas Akhir ini.

3.1 Desain Penyelesaian Permasalahan *DCEPCA09-MMM*

Pada subbab ini akan dijelaskan desain penyelesaian permasalahan *DCEPCA09-MMM*

3.1.1 Definisi Umum Program

Program akan menerima masukan sebuah bilangan N yang merepresentasikan jumlah *array* pada permasalahan DCEPCA09-MMM diikuti dengan masukan sejumlah N data disimpan pada *array* A. Setelah mendapatkan data $array$ A, program akan mengurutkan *array* A dari terkecil ke terbesar untuk untuk disimpan pada variabel baru. Dari data masukan yang sudah terurut tersebut akan dipadatkan menjadi $array$ data unik terurut. Kemudian data masukan $array$ A dipetakan dengan array data unik terurut, sehingga didapatkan urutan posisi tiap data awal dalam kondisi data unik sudah terurut.

Program akan membagi N jumlah data menjadi blok-blok kecil sesuai dengan yang sudah dijelaskan pada subbab 2.2.2. Setelah menerima masukan data *array* A, masukan selanjutnya yang akan diterima adalah bilangan Q diikuti dengan sepasang bilangan i dan j sebanyak Q. Bilangan Q merepresentasikan jumlah *query*, bilangan i merepresentasikan indeks yang menjadi batas kiri sedangkan j merepresentasikan indeks batas kanan dari *query*. Kemudian program akan mengurutkan operasi *query* berdasarkan nilai i dan j sesuai dengan ketentuan pada subbab 2.2.3.

Operasi *query* yang sudah diurutkan akan dikerjakan sesuai dengan urutannya dan untuk setiap jawaban dari satu operasi *query* akan disimpan ke dalam sebuah *segment tree* dan

digunakan kembali untuk menjawab operasi *query* selanjutnya. Untuk mencari nilai *mean* digunakan agregasi nilai penjumlahan data dari *segment tree*. Untuk mencari *median* digunakan *segment tree* yang menyimpan banyaknya data pada *leaf* sampai dengan *root*. Untuk mencari *mode* digunakan *segment tree* yang menyimpan nilai agregasi frekuensi maksimum dan indeks data tersebut.

Hasil keluaran dari program adalah jawaban dari operasi *query* yang mencari nilai $mean$, $median$ dan $mode$ pada range i dan j sesuai dengan urutan operasi *query* saat dimasukkan ke dalam program. Format keluaran sesuai dengan subbab 2.4. *Pseudocode* dari fungsi *main* ditunjukkan oleh $pseudo\ code$ pada Gambar 3.1 dan Gambar 3.2.

```
1: Input Jumlah array N
2: for (i = 0 to N − 1)
3:         input A[i]
4: end for
5: x[i] ← sorted A[i]
6: uniqueVal[0] ← A[0]
7: Inisialisasi posisi[x[0]] ← 0
8: Inisialisasi jumlah ← 1
9: for(tmp = 1 to tmp < N)
10:        if(x[tmp] ! = x[tmp − 1])
11:                uniqueVal[tmp] ← A[tmp]
12:                posisi[x[tmp]] ← jumlah
13:                jumlah ← jumlah + 1
14:        end if
15: end for
```

Gambar 3.1: *Pseudocode* Fungsi *main* Bagian 1

```
16: for(i = 0 to N − 1)
17:         data[i] ← posisi[data[i]]
18: end for
19: blok ←√N
20: isi ← blok/N
21: Input Q
22: for(tmp = 0 to tmp < Q)
23:         input i
24:         input j
25:         valMOs[tmp].BlockNumber ← i/isi
26:         valMOs[tmp].RValue ← j
27:         valMOs[tmp].inputOrder ← tmp
28: end for
29: sort(valMOs, valMOs + m, compare)
30: solveMMM()
31: for(tmp = 0 to tmp < Q)
32:         print answers[tmp].mean
33:         print answers[tmp].median
34:         print answers[tmp].mode
35: end for
```

Gambar 3.2: *Pseudocode* Fungsi *main* Bagian 2

3.1.2 Desain Algoritma

Program terdiri dari satu fungsi utama yaitu $solveMMM$ dan beberapa fungsi lain yang mempunyai tugas-tugas yang lebih spesifik untuk menyelesaikan submasalah. Untuk menyelesaikan permasalahan DCEPCA09-MMM diperlukan $subprogram$ untuk membaca masukan data, menentukan indeks urutan tiap data, menentukan konstanta Algoritma Mo, menentukan urutan $query$ dengan fungsi $compare$, menghitung nilai $mean$, $median$, dan $mode$ menggunakan fungsi $solveMMM$ yang mengimplementasikan algoritma Mo dan struktur data *segment*

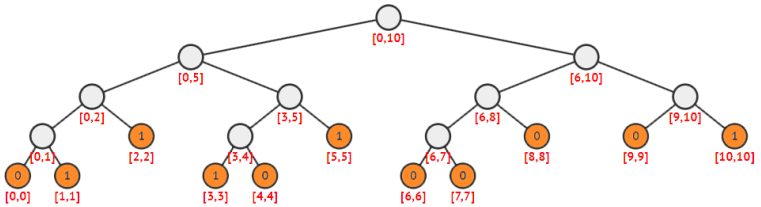
tree dengan bantuan fungsi $update$, $findMedian$ dan fungsi $cari$. Kemudian menampilkan hasil perhitungan sesuai dengan urutan masukan data. Pada subbab ini akan dijelaskan desain algoritma dari bagian-bagian fungsi tersebut dan algoritma yang digunakan.

3.1.2.1 Membaca Masukan data

Pada permasalahan DCEPCA09-MMM dengan batasan waktu hanya $0,5$ detik, diperlukan optimalisasi pada proses membaca dataset, untuk meningkatkan performa program. Pada Tugas AKhir ini akan digunakan *template Fast I/O* untuk membaca masukan *array N*, jumlah *query Q* beserta dengan batas kiri i dan batas kanan j.

3.1.2.2 Memadatkan dan Menentukan Indeks Posisi Data

Jumlah data yang diproses pada *segment tree* harus ditentukan dari awal sebelum membangun *segment tree* tersebut, karena sifat *segment tree* yang statis sehingga tidak memungkinkan perubahan struktur datanya. Oleh karena itu data harus diminimalisasi seminimal mungkin. Pada *segment tree* dengan nilai agregasi frekuensi data atau banyak data, jumlah $leaf$ bergantung pada nilai maksimal pada $array$. Sebagai contoh jika terdapat lima data dengan nilai {2,10,3,5,1} maka perlu dibuat 11 $leaf\ node$ Seperti yang ditunjukkan Gambar 3.3 karena $leaf$ dimulai dari data 0. Tentu hal tersebut tidak efisien, Oleh karena itu data perlu ditata ulang agar didapatkan komposisi data yang minimal dengan tetap merepresentasikan data masukan. Untuk itu akan digunakan indeks posisi data.

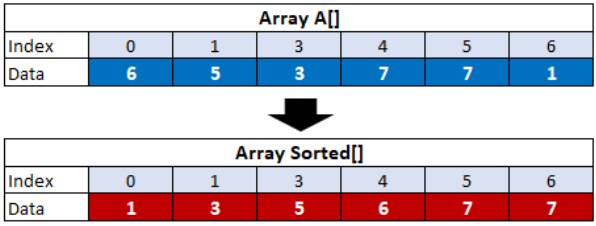


Gambar 3.3: Ilustrasi Jumlah $node$ pada Segment Tree dengan Nilai Agregasi Banyak Data

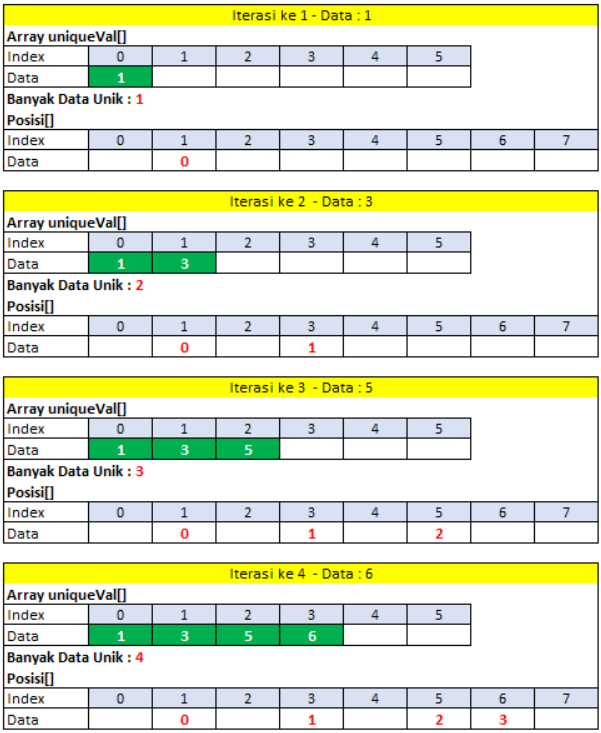
Indeks posisi data adalah indeks data pada $array$ ketika data sudah diurutkan dari nilai terkecil ke nilai terbesar dengan kondisi urutan hanya mempertimbangkan data unik. Sehingga apabila dalam $array$ terdapat data yang berulang maka data-data tersebut hanya menempati satu posisi. Dengan mengetahui indeks urutan data maka pengurutan $array$ hanya perlu dilakukan satu kali. Indeks urutan data akan digunakan sebagai masukan untuk perhitungan dan konstruksi *segment tree* menggunakan Algoritma Mo. Sehingga dengan hanya mempertimbangkan data unik dan urutannya banyak $node$ yang digunakan untuk membangun *segment tree* menjadi lebih minimal.

Langkah menentukan indeks urutan data dimulai dari mengurutkan data $array$ A kemudian menyimpan hasil pengurutan kedalam $array$ baru seperti ilustrasi pada gambar 3.4. Data yang sudah terurut akan diiterasi untuk membentuk $array$ baru berisikan data unik yang sudah terurut. Data pada iterasi indeks ke i akan dibandingkan dengan indeks ke $i - 1$ apakah memiliki nilai yang sama. Pada saat iterasi, program juga akan menyimpan posisi data pada kondisi terurut pada $array$ posisi seperti ditunjukkan pada Gambar 3.5 dan Gambar 3.6. Dari array posisi yang disimpan, akan didapatkan posisi masing-masing data $array$ A dengan cara melakukan iterasi

pada *array* A dan melihat nilai pada data posisi. Sehingga akan didapat indeks urutan data seperti pada ilustrasi Gambar 3.7.



Gambar 3.4: Ilustrasi Pengurutan Data Hasil Masukan



Gambar 3.5: Ilustrasi Iterasi ke 1 dan 4 dengan Data Unik

Iterasi ke 5 - Data : 7

Array uniqueVal[]

Index	0	1	2	3	4	5
Data	1	3	5	6	7	

Banyak Data Unik : 5

Posisi[]

Index	0	1	2	3	4	5	6	7
Data		0		1		2	3	4

Iterasi ke 6 - Data : 7

Array uniqueVal[]

Index	0	1	2	3	4	5
Data	1	3	5	6	7	

Banyak Data Unik : 5

Posisi[]

Index	0	1	2	3	4	5	6	7
Data		0		1		2	3	4

Gambar 3.6: Ilustrasi Iterasi ke 5 dan 6 dengan Data Tidak Unik

Array A[]

Index	0	1	2	3	4	5
Data	6	5	3	7	7	1

Array Data[]

Index	0	1	2	3	4	5
Data	posisi[6]	posisi[5]	posisi[3]	posisi[7]	posisi[7]	posisi[1]
	3	2	1	4	4	0

Gambar 3.7: Ilustrasi Penentuan Posisi Indeks Data Terurut

Dapat dilihat bahwa dengan mendapatkan hasil indeks urutan data, nilai maksimal pada data menjadi lebih kecil, dari 7 pada data asli menjadi 4 pada indeks urutan data. Hal tersebut akan meminimalisir *node* pada SEGMENT TREE terutama untuk nilai agregasi banyaknya data dan frekuensi maksimal, seperti yang sudah dijelaskan pada subbab 2.3.3.2 dan 2.3.3.1.

Potongan *pseudocode* program untuk menentukan indeks urutan ditunjukkan oleh Gambar 3.8.

```
1: Require Jumlah array N, data array A[]
2: x[i] ← sorted A[i]
3: uniqueVal[0] ← A[0]
4: Inisialisasi posisi[x[0]] ← 0
5: Inisialisasi jumlah ← 1
6: for(tmp = 1 to tmp < N)
7:       if(x[tmp] ! = x[tmp − 1])
8:             uniqueVal[tmp] ← A[tmp]
9:             posisi[x[tmp]] ← jumlah
10:            jumlah ← jumlah + 1
11:      end if
12: end for
13: for(i = 0 to N − 1)
14:      data[i] ← posisi[data[i]]
15: end for
```

Gambar 3.8: *Pseudocode* Menentukan Indeks Urutan Data

3.1.2.3 Menghitung Konstanta Algoritma Mo

Algoritma Mo berfungsi untuk mengurutkan operasi *query* menjadi urutan pengerjaan yang efisien dengan memanfaatkan hasil operasi sebelumnya dan membagi data menjadi beberapa blok, dimana setiap *query* menempati blok tersebut. Untuk melakukan pengurutan diperlukan konstanta yang akan menentukan urutan *query*. Konstanta Algoritma Mo terdiri dari jumlah blok dan banyak data setiap blok seperti yang sudah dijelaskan pada subbab 2.2.2. Jumlah blok dihitung dari akar pangkat dua dari banyaknya data sedangkan banyak data untuk setiap blok didapatkan dengan membagi banyak data dengan jumlah blok. Potongan *pseudocode* program untuk menentukan konstanta Algoritma Mo ditunjukkan oleh Gambar 3.9.

```
1: Require Jumlah array N
2: blok ←√N
3: if(blok x blok ! = N)
4:         blok ← blok + 1
5: end if
6: isi ← N/blok
7: if(N/blok ! = 0)
8:         isi ← isi + 1
9: end if
```

Gambar 3.9: *Pseudocode* Menentukan Konstanta *AlgoritmaMo*

Setelah menentukan konstanta Algoritma Mo selanjutnya adalah melakukan pengurutan urutan pengerjaan operasi *query* yang sudah ditentukan menggunakan *std sort* dan fungsi *boolean compare*.

3.1.2.4 Menentukan Urutan *Query*

Setiap operasi *query* memiliki batasan indeks kiri dan indeks kanan. Pada Algoritma Mo blok dari suatu *query* ditentukan oleh batas indeks kiri. Untuk menentukan blok dari batasan indeks didapatkan dengan membagi indeks kiri dengan konstanta Algoritma Mo yaitu banyak data setiap blok. Potongan *pseudocode* untuk menentukan blok setiap *query* terdapat pada Gambar 3.10.

Setelah mendapatkan blok dari setiap *query*. Operasi *query* akan diurutkan menggunakan *stdsort*. Fungsi *boolean compare* digunakan untuk menjadi argumen *std sort* dengan ketentuan:

1. Prioritas pertama adalah operasi *query* dengan nilai blok paling kecil.
2. Apabila *query* memiliki nilai blok yang sama, prioritas kedua adalah operasi *query* dengan batas kanan R paling kecil.

3. Apabila $query$ memiliki nilai blok dan batas kanan R yang sama. prioritas ketiga adalah operasi $query$ yang urutan masuknya paling awal.

Potongan desain fungsi *compare* terdapat pada Gambar 3.11.

```
1: Input Q
2: for(tmp = 0 to tmp < Q)
3:         input i
4:         input j
5:         valMOs[tmp].BlockNumber ← i/isi
6:         valMOs[tmp].RValue ← j
7:         valMOs[tmp].inputOrder ← tmp
8: end for
9: sort(valMOs, valMOs + m, compare)
```

Gambar 3.10: $Pseudocode$ Membaca Masukan *Query*

```
1: Require valMO x,y (untuk dibandingkan)
2: if(x.BlockNumber ! = y.BlockNumber)
3:         return x.BlockNumber < y.BlockNumber
4: end if
5: if(x.RValue ! = y.RValue)
6:         return x.RValue < y.RValue
7: end if
8: return x.inputOrder < y.inputOrder
```

Gambar 3.11: $Pseudocode$ Fungsi $compare$

3.1.2.5 Desain Fungsi *solveMMM*

Fungsi $solveMMM$ digunakan untuk menentukan perubahan yang terjadi pada *segment tree* dan nilai-nilai agregasinya melalui fungsi $update$. Dengan menerapkan Algoritma Mo, perubahan pada *segment tree* ditentukan oleh

pergeseran nilai indeks kiri ($currentL$) dam indeks kanan($currentR$) untuk menyesuaikan batasan dari $query$. Terdapat 4 kondisi yang mungkin terjadi ketika terjadi pergeseran indeks:

1. Nilai $currentL$ kurang dari batas kiri L. Maka yang dilakukan adalah menghilangkan data dengan indeks $currentL$ pada *segment tree*, kemudian menambah nilai dari $currentL$.
2. Nilai $currentL$ lebih dari batas kiri L. Maka yang dilakukan adalah mengurangi nilai dari $currentL$, kemudian menambahkan data dengan indeks $currentL$ pada *segment tree*.
3. Nilai $CurrentR$ kurang dari batas kanan R. Maka yang dilakukan adalah menambah nilai dari $currentR$, kemudian menambahkan data dengan indeks $currentR$ pada *segment tree*.
4. Nilai $CurrentR$ lebih dari batas kanan R. Maka yang dilakukan adalah menghilangkan data dengan indeks $currentR$ pada *segment tree*, kemudian mengurangi nilai dari $currentR$.

Hal tersebut dilakukan sampai memenuhi batasan $query$, ketika sudah sesuai dengan batasan maka akan dicari nilai $mean$,$median$ dan $mode$ menggunakan *segment tree* yang sudah diperbarui. Nilai mean didapat dari nilai agregasi nilai penjumlahan data pada $root$ dibagi dengan banyak data pada rentang $query$. $Median$ dicari dengan menggunkan fungsi $findMean$ yang memanfaatkan nilai agregasi jumlah data. $Mode$ diambil dari nilai agregasi frekuensi maksimal yang ada pada $root$.

Potongan $pseudocode$ fungsi $solveMMM$ ditunjukkan oleh Gambar 3.12.

```
1: for(tmp = 0 to tmp < Q)
2:     now ← valMOs[tmp].inputOrder
3:     right ← query[now].right
4:     left ← query[now].left
5:     while(currentR < right)
6:         currentR + +
7:         update(1,0,jumlahDataUnik-1,data[currrentR],1)
8:     end while
9:     while(currentR > right)
10:        update(1,0,jumlahDataUnik-1,data[currrentR],-1)
11:        currentR − −
12:    end while
13:    while(currentL < left)
14:        update(1,0,jumlahDataUnik-1,data[currrentL],-1)
15:        currentL + +
16:    end while
17:    while(currentL > left)
18:        currentL − −
19:        update(1,0,jumlahDataUnik-1,data[currrentL],1)
20:    end while
21:    answers[now].mean ←
               seg_tree[1].jumlahData/banyak_data
22:    answers[now].median ← findMedian()
23:    answers[now].mode ←
               val[seg_tree[1].indeksData]
24: end for
```

Gambar 3.12: *Pseudocode* Fungsi $solveMMM$

3.1.2.6 Desain Fungsi *update*

Fungsi update digunakan untuk memperbarui nilai pada *node segment tree*. Fungsi *update* dijalankan ketika terjadi perubahan batas kanan atau kiri pada Algoritma Mo yang berakibat pada

bertambahnya atau berkurangnya data pada rentang $query$. Sehingga diperlukan adanya perubahan nilai pada *segment tree*. Nilai-nilai agregasi yang dirubah pada $segmenttree$ adalah:

1. $frekuensiData$: frekuensi kemunculan data pada rentang $query$.
2. $indeksData$: merupakan variabel bantu $frekuensiData$ untuk menyimpan indeks dari data yang frekuensi datanya disimpan oleh $node$.
3. $banyakData$: banyaknya data pada rentang $query$.
4. $jumlahData$: nilai penjumlahan data pada rentang $query$.

Proses pembaruan nilai pada $segmenttree$ dilakukan secara rekursif. Dimulai dari pembaruan data pada $leaf$ melalui $root$. Untuk memproses pembaruan diperlukan informasi $data$ yang akan diproses dan status pemrosesan apakah menambah atau mengurangi yang dikirim melalui parameter $value$. Nilai $left$ dan $right$ yang merepresentasikan rentang yang direpresentasikan oleh node *segment tree*.

Dengan mekanisme rekursi, program ini memiliki konfisi berhenti ketika $data$ yang akan masuk berada diluar rentang indeks yang dicakupi *segment tree* jika diluar maka program akan memanggil fungsi $return$.Selain itu Jika nilai $left$ dan $right$ sama, maka rekursi sudah sampai pada $leaf$ dan akan dilakukan pembaruan nilai agregasi. Setelah selesai melakukan pembaruan program akan memanggil fungsi $return$.

Program melakukan rekursi dari $root$ kemudian ke $child$ kiri dan $child$ kanannya. setelah kembali dari $child$-nya program akan memperbarui nilai agregasi $parent$-nya, untuk nilai $frekuensiData$ dan $indeksData$ akan diambil dari nilai $child$ yang memiliki nilai lebih besar. Untuk nilai $banyakData$ dan $jumlahData$ akan diambil dari hasil permjumlahan dari $child$-nya. Potongan $pseudocode$ fungsi $update$ dapat dilihat pada Gambar 3.13.

```
1: Require index, left, right, data, value
2: if(data < left or data > right)
3:       return
4: end if
5: if(left = right)
6:       seg_tree[index].indeksData ← left
7:       seg_tree[index].frekuensiData += value
8:       seg_tree[index].banyakData += value
9:       seg_tree[index].jumlahData +=
                    uniqueVal[left] × value
10:      return
11: end if
12: update(2 * index, left, (left + right)/2, data, value)
13: update(2 * index + 1, (left + right)/2 + 1,
     right, data, value)
14: if (seg_tree[2 * index].frekuensiData >
            seg_tree[2 * idx + 1].frekuensiData)
15:      seg_tree[index] ← seg_tree[2 * index]
16: else
17:      seg_seg_tree[index] ← seg_tree[2 * index + 1]
18: end if
19: seg_trees[index].banyakData =
     seg_trees[2 * index].banyakData+
     seg_trees[2 * index + 1].banyakData
20: segtrees[index].jumlahData =
     segtrees[2 * index].jumlahData+
     segtrees[2 * index + 1].jumlahData
21: return
```

Gambar 3.13: *Pseudocode* Fungsi *update*

3.1.2.7 Desain Fungsi *findMedian*

Fungsi $findMedian$ digunakan untuk menemukan nilai $median$ dari suatu $query$. Fungsi ini membutuhkan masukan berupa $index$ yang menunjukkan urutan $query$ yang akan dicari nilai $median$-nya. Untuk mencari nilai median perlu dihitung banyaknya bilanganan pada rentang $query$ tersebut. Banyak bilangan bisa didapatkan dari nilai agregasi banyak data pada $root$. Setelah mengetahui banyak bilangannya ada dua kemungkinan yaitu berjumlah ganjil atau genap.

Apabila banyak bilangan berjumlah ganjil maka nilai $median$ berada tepat di tengah urutan data. Sehingga akan dicari data ke $(banyak\_bilangan + 1/2)$ pada *segment tree*. Apabila banyak bilngan berjumlah genap maka $median$ adalah setengah dari total dua bilangan tengah. Sehingga perlu dicari data ke $banyak\_bilangan/2$ dan data ke $(banyak\_bilangan + 1/2)$ untuk kemudian dijumlahkan dan dibagi 2.

$Pseudocode$ untuk fungsi $findMedian$ dapat dilihat pada Gambar 3.14 dan Gambar 3.15. Untuk mencari urutan data ke-i digunakan fungsi $cari$ yang memanfaatkan nilai agregasi banyaknya data.

```
1: banyakbilangan ← seg_tree[1].banyak_data
2: if (banyakbilangan%2 = 1)
3:       return uniqueVal[cari(1, 0, jumlahDataUnik−
           1, banyakbilangan/2) + 1]
4: else
5:       kiri ← cari(1, 0, jumlahDataUnik − 1,
           banyakbilangan/2)
```

Gambar 3.14: $Pseudocode$ Fungsi $findMedian$ Bagian 1

```
6:      kanan ← cari(1, 0, jumlahDataUnik − 1,
          banyakbilangan/2) + 1
7:      return (uniqueVal[kiri] + uniqueVal[kanan])/2
8: end if
```

Gambar 3.15: $Pseudocode$ Fungsi $findMedian$ Bagian 2

3.1.2.8 Desain Fungsi *cari*

Fungsi cari digunakan untuk menemukan data pada urutan tertentu dengan memanfaatkan nilai agregasi $banyakData$ pada *segment tree*. Dengan parameter ke yang merepresentasikan urutan data yang dicari variabel $index$,$left$ dan $right$ untuk melakukan rekursi. Kondisi rekursi akan berhenti ketika $left$ dan $right$ bernilai sama, yang berarti rekursi sudah sampai pada $leaf$.

Pencarian dilakukan mulai dari $root$ dan dilakukan pengecekan ke $child$ kirinya. Jika nilai $banyakData$ pada $child$ kiri lebih dari atau sama dengan urutan yang dicari maka nilai yang dicari ada di segmen $child$ kiri tersebut. Sehingga pencarian akan dilanjutkan ke $child$ kiri. Sebaliknya jika nilai $banyakData$ pada $child$ kiri kurang dari urutan yang dicari maka nilai yang dicari berada pada segmen kanan. Pencarian akan dilanjutkan pada segmen $child$ kanan. Namun urutan yang dicari harus dikurangi dengan nilai $banyakData$ pada segmen kiri, karena pencarian dimulai dari segmen kanan dan meniadakan kemungkinan bahwa nilai yang dicari ada di segmen kiri.

Proses pencarian dilakukan secara rekursi dari $node$ ke $node$ hingga mencapai $leaf$. Data dengan indeks $leaf$ tersebut adalah data urutan yang dicari. Potongan $pseudocode$ fungsi $cari$ ditunjukkan oleh Gambar 3.16.

```
1: Require index, left, right, ke
2: if (left = right)
3:       return left
4: end if
5: if (seg_trees[2 * index].banyakData >= ke)
6:       return cari(2 * index, left, (left + right)/2, ke)
7: else
8:       return cari(2 * index + 1, (left + right)/2 + 1,
           right, seg_trees[2 * index].banyakData)
9: end if
```

Gambar 3.16: *Pseudocode* Fungsi *cari*

3.1.2.9 Menampilkan nilai *mean, median* dan *mode*

Untuk menampilkan hasil perhitungan nilai *mean, median* dan *mode* untuk setiap *query* sesuai dengan urutan masukan data, digunakan struktur data *struct valMO* yang menyimpan urutan masukan *query* beserta dengan blok dan batas kanan *query*. Nilai dari urutan data pada struct valMO kemudian digunakan sebagai indeks array variabel *answer* yang menyimpan nilai *mean*, *median* dan *mode*. Setelah semua *query* diselesaikan, variabel *answer* akan diiterasi berdasarkan indeksnya untuk menampilkan semua hasil secara terurut. Potongan *pseudocode* dapat dilihat pada Gambar 3.17 dan Gambar 3.18.

```
1: Require sorted ValMOs
2: for(tmp = 0 to tmp < Q)
3:       now ← valMOs[tmp].inputOrder
4:       ...
5:       answers[now] ← mean, median, mode
6: end for
```

Gambar 3.17: *Pseudocode* Menampilkan Hasil Sesuai Urutan Bagian 1

```
7: for tmp = 0 to tmp < Q
8:       print answers[now].mean
9:       print answers[now].median
10:       print answers[now].mode
11: end for
```

Gambar 3.18: $Pseudocode$ Menampilkan Hasil Sesuai Urutan Bagian 2

3.1.2.10 Desain Data *Generator*

Data *generator* digunakan untuk menghasilkan format masukan sesuai dengan deskripsi permasalahan seperti yang telah dijelaskan pada subbab 2.4. Data $generator$ memerlukan masukan N dan Q. Dari masukan tersebut akan dibuatkan data acak sebanyak N dan sepasang data $query$ sebanyak Q. *Pseudocode* dari data *generator* ditunjukkan pada Gambar 3.19 dan Gambar 3.20.

```
1: Input N
2: Input Q
3: Print N
4: for i = 0 to N − 1
5:       random ← random_number(0, 100000000)
6:       if (i < N − 1)
7:          Output random with space
8:       else
9:          Output random with enter
10:       end if
11: end for
12: Print Q
```

Gambar 3.19: *Pseudocode* Fungsi *Generator* Bagian 1

```
13: for j = 0 to Q − 1
14:       a = rand()%N
15:       b = rand()%N
16:       if (j < Q − 1)
17:          if (a > b)
18:                Print b a
19:          else
20:                Print a b
21:          end if
22:       else
23:          if (a > b)
24:                Print b a with enter
25:          else
26:                Print a b with enter
27:          end if
28:       end if
29: end for
```

Gambar 3.20: *Pseudocode* Fungsi *Generator* Bagian 2

(Halaman ini sengaja dikosongkan)

BAB IV

IMPLEMENTASI

Pada bab ini akan dibahas tentang implementasi yang dilakukan berdasarkan apa yang sudah dirancang pada bab sebelumnya.

4.1 Lingkungan Implementasi

Lingkungan implementasi dalam pembuatan Tugas Akhir ini meliputi perangkat keras dan perangkat lunak yang digunakan untuk melakukan penyelesaian dari permasalahan adalah sebagai berikut:

1. Perangkat Keras:
    - *Processor*: Intel(R) Core(TM) i7-4720HQ CPU @ 2.60 GHz 2.59 GHz.
    - *Random Access Memory*: 12 GB.
2. Perangkat Lunak:
    - *Operating System*: Windows 10 Professional 64 bit.
    - *IDE*: Eclipse C/C++ Developers Versi Oxygen.3a.
    - *Compiler*: g++ (TDM-GCC 4.8.1 32-bit).
    - Bahasa Pemrograman: C++.

4.2 Implementasi Penyelesaian Permasalahan DCEPCA09-MMM

Pada subbab ini akan dijelaskan mengenai implementasi dari penyelesaian permasalahan DCEPCA09-MMM yang telah dibuat pada bab sebelumnya.

4.2.1 Penggunaan *Library*, Konstanta, Variabel Global, *Struct* dan *Template*

Pada subsubbab ini akan dibahas penggunaan *library*, konstanta, *template*, *struct*, dan variabel global yang digunakan

dalam program. Potongan kode yang menyatakan penggunaan *library* dan konstanta dapat dilihat pada Kode Sumber 4.1.

```
1 #include <stdio.h>
2 #include <iostream>
3 #include <map>
4 #include <utility>
5 #include <algorithm>
6 #include <cmath>
7 #define left first
8 #define right second
9 using namespace std;
```

Kode Sumber 4.1: Penggunaan *Library* dan Konstanta pada Penyelesaian Permasalahan DCEPCA09-MMM

Untuk menyelesaikan permasalahan DCEPCA09-MMM dibutuhkan enam *library* yaitu $iostream$, map, $utility$, $algorithm$ dan $cmath$. $iostream$ digunakan untuk standar $input$ $output$ seperti cin atau $cout$. map digunakan untuk $template$ $libary$. $utility$ digunakan untuk melakukan operasi tipe data $pair$. $algorithm$ digunakan untuk fungsi $sort$. $cmath$ digunakan untuk melakukan operas matematika seperti $sqrt$.

Selain library, didefinisikan konstanta $left$ yang bernilai $first$ dan konstanta $right$ yang bernilai $second$. Konstanta ini digunakan untuk mempermudah pembacaan variabel pada tipe data $pair$ agar sesuai dengan konsep permasalahan.

Pada tabel berikut dijelaskan mengenai variabel-variabel global yang akan digunakan dalam implementasi program.

Tabel 4.1: Tabel Daftar Variabel Global Permasalahaan DCEPCA09-MMM Bagian 1

No	Nama Variabel	Tipe	Penjelasan
1	$data$	$int[\,]$	Digunakan untuk menyimpan elemen $array$ masukan.
2	$sorted$	$int[\,]$	Digunakan untuk menyimpan data terurut dari variabel $data$

Tabel 4.2: Tabel Daftar Variabel Global Permasalahaan DCEPCA09-MMM Bagian 2

No	Nama Variabel	Tipe	Penjelasan
3	val	$int[\,]$	Digunakan untuk menyimpan indeks posisi dari data dan urutannya.
4	$block$	int	Digunakan untuk menyimpan konstanta jumlah blok hasil perhitungan pada Algoritma Mo.
5	isi	int	Digunakan untuk menyimpan konstanta banyaknya data pada suatu blok hasil perhitungan pada Algoritma Mo.
6	n	int	Digunakan untuk menyimpan jumlah masukan elemen data
7	q	int	Digunakan untuk menyimpan jumlah masukan $query$
8	$jumlah$	int	Digunakan untuk menyimpan hasil perhitungan jumlah nilai unik pada variabel $data$
9	$comp$	$map < int, int >$	Digunakan untuk menyimpan pemetaan data dan urutannya pada kondisi data unik.
10	$query$	$pair < int, int >$	Digunakan untuk menyimpan batas kiri dan kanan dari $query$
11	$answers$	$struct[\,]$	Digunakan untuk menyimpan hasil jawaban $medan$, $median$ dan $mode$ dari $query$
12	$valMos$	$struct[\,]$	Digunakan untuk menyimpan nilai blok dari setiap $query$
13	seg_trees	$struct[\,]$	Digunakan untuk menyimpan hasil proses pengolahan data dengan menggunakan *segment tree*

Daftar variabel yang dijelaskan oleh Tabel 4.1 dan Tabel 4.2 ditunjukkan oleh Kode Sumber 4.2 dan 4.3.

```
1 int data[10005];
2 int sorted[10005];
3 int val[10005];
4 int block,isi;
5 int n, q, jumlah;
6 map<int,int> comp;
7 pair<int,int> query[10005];
```

Kode Sumber 4.2: Penggunaan Variabel Global pada Penyelesaian Permasalahan DCEPCA09-MMM Bagian 1

```
1 answer answers[10005];
2 valMO valMOs[10005];
3 seg_tree seg_trees[40005];
```

Kode Sumber 4.3: Penggunaan Variabel Global pada Penyelesaian Permasalahan DCEPCA09-MMM Bagian 2

Terdapat tiga *struct* yang digunakan dalam program yaitu *struct* $answer$, $valMo$, dan seg_tree. *Struct* $answer$ digunakan untuk menyimpan hasil dari operasi $query$ yaitu $mean$, $median$ dan $mode$. Potongan kode untuk *struct* $answer$ dapat dilihat di Kode Sumber 4.4.

```
1 struct answer{
2    long long mean;
3    long long median;
4    long long mode;
5 };
```

Kode Sumber 4.4: *Struct* $answer$ pada Penyelesaian Permasalahan DCEPCA09-MMM

Struct $valMo$ digunakan untuk menyimpan blok dari setiap masukan $query$. *Struct* $valMo$ berisi nilai blok, batas kanan dan urutan masukan $query$. Ketiga nilai tersebut yang akan digunakan untuk mengurutkan proses pengerjaan $query$ seperti

yang sudah dijelaskan pada subbab 3.1.2.4. Potongan kode untuk *struct* $valMo$ dapat dilihat di Kode Sumber 4.5.

```
1 struct valMO{
2    long long BlockNumber;
3    long long RValue;
4    long long inputOrder;
5 };
```

Kode Sumber 4.5: *Struct* $valMO$ pada Penyelesaian Permasalahan DCEPCA09-MMM

Struct seg_tree digunakan untuk menyimpan nilai-nilai agregasi untuk menyelesaikan $query$. *Struct* seg_tree berisi nilai banyaknya data, jumlah data, frekuensi data dan indeks data. Banyaknya data digunakan untuk menemukan nilai $median$. Jumlah data digunakan untuk menemukan $mean$. frekuensi dan indeks data digunakan untuk menemukan $mode$. Proses penggunaan nilai agregasi ini sudah dijelaskan pada subbab 3.1.2.6 dan 3.1.2.5. Potongan kode untuk *struct* seg_tree dapat dilihat di Kode Sumber 4.6.

```
1 struct seg_tree{
2    int indeksData;
3    int frekuensiData;
4    int banyakData;
5    long long jumlahData;
6 };
```

Kode Sumber 4.6: *Struct* seg_tree pada Penyelesaian Permasalahan DCEPCA09-MMM

Untuk mempercepat proses memasukkan data pada program, digunakan *template Fast IO*. Potongan kode untuk *template Fast IO* dapat dilihat di Kode Sumber 4.7.

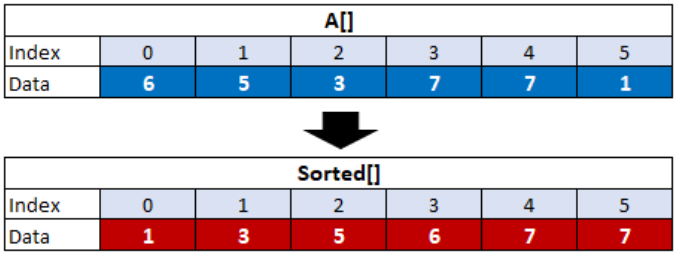
```
1 inline void scan_uint(int* p) {
2     static char c;
3     while ((c = getchar()) < '0');
4     for (*p = c-'0'; (c = getchar()) >= '0'; *p = *p*10+
      c-'0');
```

```
5 }
```

Kode Sumber 4.7: Penggunaan *Template Fast I/O* pada Penyelesaian Permasalahan DCEPCA09-MMM

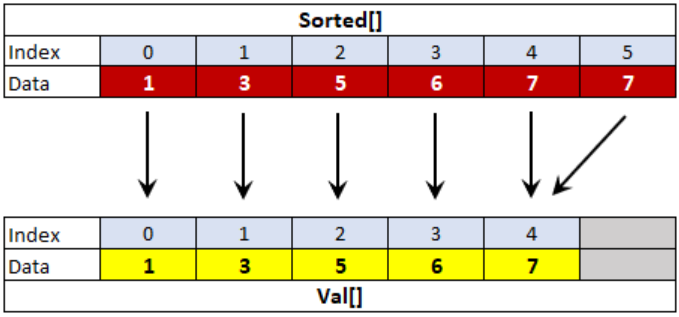
4.2.2 Implementasi Fungsi untuk Memadatkan *Array* dan Menentukan Indeks Posisi Data

Agar didapatkan data yang optimal untuk diolah pada Algoritma Mo dan *segment tree*. Data terlebih dahulu dipadatkan dan dicari indeks posisi datanya, seperti yang sudah dijelaskan pada subbab 3.1.2.2. Langkah yang dilakukan adalah dengan mengurutkan data masukan dan menyimpannya pada variabel baru. Pada implementasi akan digunakan variabel $sorted$ untuk menyimpan nilai hasil pengurutan. Seperti diilustrasikan pada Gambar 4.1 .

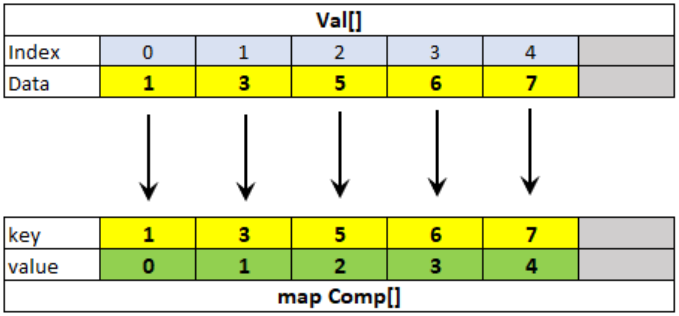


Gambar 4.1: Ilustrasi Pengurutan Data Masukan kedalam Variabel $sorted$

Setelah didapatkan data yang terurut selanjutnya akan dipadatkan dengan hanya menyimpan nilai unik pada variabel val sekaligus memetakan posisi data nilai tersebut pada variabel $comp$. Pada iterasi untuk memadatkan data tersebut, jumlah data unik juga akan dihitung dan disimpan pada variabel $jumlah$. Ilustrasi pemadatan data diilustrasikan pada Gambar 4.2 dan Gambar 4.3.



Gambar 4.2: Ilustrasi Pengurutan Data Masukan kedalam Variabel *sorted*



Gambar 4.3: Ilustrasi Pengurutan Data Masukan kedalam Variabel *sorted*

Dari hasil pemadatan didapatkan nilai data unik dan urutannya. Hasil data ini akan digunakan untuk memetakan data masukan awal. Sehingga akan didapatkan keseluruhan posisi nilai pada kondisi data terurut. Seperti yang dtunjukkan oleh Gambar 4.4.

data[]						
Index	0	1	2	3	4	5
Mapping	map[6]	map[5]	map[3]	map[7]	map[7]	map[1]
Data	3	2	1	4	4	0

Gambar 4.4: Ilustrasi Pengurutan Data Masukan kedalam Variabel *sorted*

Implementasi dari pemadatan data dan penentuan indeks posisi data dapat dilihat pada Kode Sumber 4.8.

```
1     for(int tmp=0;tmp<n;tmp++){
2         scan_uint(&data[tmp]);
3         sorted[tmp] = data[tmp];
4     }
5
6     sort(sorted,sorted+n);
7
8     val[0] = sorted[0];
9     comp[sorted[0]] = 0;
10    jumlah = 1;
11
12    for(int tmp=1;tmp<n;tmp++){
13        if(sorted[tmp]!=sorted[tmp-1]){
14            val[jumlah] = sorted[tmp];
15            comp[sorted[tmp]] = jumlah;
16            jumlah++;
17        }
18    }
19
20    for(int tmp=0;tmp<n;tmp++){
21        data[tmp] = comp[data[tmp]];
22    }
```

Kode Sumber 4.8: Implementasi Fungsi Memadatkan *Array* dan Menentukan Indeks Posisi Data

4.2.3 Implementasi Fungsi untuk Menghitung Konstanta Algoritma Mo

Untuk menyimpan konstanta Algoritma Mo yaitu jumlah blok dan isi blok digunakan variabel $blok$ dan isi. Untuk mendapatkan nilai $blok$ yang dilakukan adalah dengan mencari hasil akar kuadrat dari banyaknya data n. Apabila hasilnya bilangan desimal maka akan dibulatkan ke satuan terdekat diatasnya. Untuk mendapatkan nilai isi dilakukan dengan membagi banyaknya data dengan $blok$. Jika hasilnya bilangan desimal maka akan dibulatkan ke satuan terdekat diatasnya. Implementasi perhitungan konstanta Algoritma Mo dapat dilihat pada Kode Sumber 4.9.

```
1     block = sqrt(n);
2     if(block*block != n) block++;
3     isi = n/block;
4     if(n%block != 0) isi++;
```

Kode Sumber 4.9: Implementasi Fungsi Menghitung Konstanta Algoritma Mo

4.2.4 Implementasi Fungsi untuk Mengurutkan *Query*

Untuk mengurutkan $query$ diperlukan nilai blok, batas kanan dan urutan inputan dari setiap $query$, seperti yang dijabarkan pada subbab 3.1.2.4. Untuk mendapatkan nilai tersebut akan digunakan variabel $valMOs$ yang akan menyimpan nilai blok, batas kanan dan urutan ketika data masukan dikirim ke program. Nilai blok $query$ didapatkan dengan membagi batas kiri dengan konstanta isi Algoritma Mo. Selanjutnya variabel $valMos$ akan diurutkan menggunakan $library\ sort$ dengan parameter pengurutan fungsi $compare$. Implementasi pengurutan $query$ dapat dilihat pada Kode Sumber 4.10.

```
1     for(int tmp=0;tmp<q;tmp++){
2         scan_uint(&query[tmp].left);
3         scan_uint(&query[tmp].right);
```

```
4
5           valMOs[tmp].BlockNumber = query[tmp].left / isi;
6           valMOs[tmp].RValue = query[tmp].right;
7           valMOs[tmp].inputOrder = tmp;
8       }
9
10      sort(valMOs,valMOs+q,compare);
```

Kode Sumber 4.10: Implementasi Fungsi Mengurutkan *Query*

4.2.4.1 Implementasi Fungsi *compare*

Fungsi $compare$ digunakan sebagai parameter untuk mengurutkan data $valMos$ dengan menggunakan $library\ sort$. Sesuai dengan yang sudah dijelaskan pada subbab 3.1.2.4, prioritas pengurutan dimulai dari nilai blok, nilai batas kanan kemudian urutan masukan dari $query$. Fungsi *compare* diimplementasikan seperti pada Kode Sumber 4.11.

```
1 bool compare(valMO x, valMO y)
2 {
3     if (x.BlockNumber != y.BlockNumber)
4         return x.BlockNumber < y.BlockNumber;
5     if (x.RValue != y.RValue)
6         return x.RValue < y.RValue;
7     return x.inputOrder < y.inputOrder;
8 }
```

Kode Sumber 4.11: Implementasi Fungsi *compare*

4.2.5 Implementasi Fungsi *solveMMM*

Fungsi ini digunakan untuk mencari jawaban setiap operasi $query$. Menerapkan Algoritma Mo untuk melakukan perbaruan data pada *segment tree*. Pembaruan data dilakukan secara bertahap dengan menggeser batas kiri dan kanan hingga sesuai dengan batas pada $query$. Untuk menyimpan kondisi batas yang

sedang dilakukan digunakan variabel $currentL$ dan $currentR$. Variabel $currentL$ mula-mula diinisialisasi dengan nilai 0 dan variabel $currentR$ dengan nilai -1.

Kemudian dilakukan iterasi untuk setiap $query$ untuk merubah nilai dari batas $currentL$ dan $currentL$. Setiap perubahan nilai $currentL$ atau $currentR$ akan merubah nilai dari *segment tree*. Untuk merubah nilai tersebut digunakan fungsi $update$. Setelah nilai batasan $currentL$ dan $currentR$ sesuai dengan batasan $query$ maka hasil akan disimpan pada variabel $answers$.

Fungsi *solveMMM* diimplementasikan pada Kode Sumber 4.12 sesuai dengan *pseudocode* pada bagian 3.1.2.5.

```
1  void solveMMM(){
2     int currentL = 0, currentR = -1;
3
4     for(int tmp=0 ; tmp<q ; tmp++){
5        int now = valMOs[tmp].inputOrder;
6
7        while(currentR < query[now].right){
8           currentR++;
9           update(1,0,jumlah-1,data[currentR],1);
10       }
11       while(currentR > query[now].right){
12          update(1,0,jumlah-1,data[currentR],-1);
13          currentR--;
14       }
15       while(currentL < query[now].left){
16          update(1,0,jumlah-1,data[currentL],-1);
17          currentL++;
18       }
19       while(currentL > query[now].left){
20          currentL--;
21          update(1,0,jumlah-1,data[currentL],1);
22       }
23
24       answers[now].mean = seg_trees[1].jumlahData / (
       query[now].right - query[now].left + 1);
25
```

```
26          answers[now].median = findMedian();
27
28          answers[now].mode = val[seg_trees[1].indeksData];
29      }
30  }
```

Kode Sumber 4.12: Implementasi Fungsi *solveMMM*

4.2.6 Implementasi Fungsi *update*

Fungsi $update$ digunakan untuk melakukan pembaruan pada *segment tree* dengan menggunakan metode rekursif $bottom - up$. Fungsi $update$ memerlukan parameter masukan indeks $root$ (idx), batas kiri (l), batas kanan (r), data yang dimasukkan (ini) dan jenis operasi (value). Kondisi berhenti rekursi untuk fungsi $update$ adalah ketika nilai ini lebih kecil dari l atau ini lebih besar dari r yang berarti data berada diluar $rentangquery$. Rekursi juga akan berhenti ketika l sama dengan r yaitu ketika sudah mencapai $leaf$ dan akan memperbarui nilai pada $leaf$ tersebut. Rekursi akan dilakukan ke $child$ kiri dan $child$ kanan untuk memperbarui nilai agregasi karena perubahan data pada $leaf$.

Nilai yang diperbarui pada operasi $update$ adalah nilai-nilai agregasi pada variabel seg_tree yang meliputi nilai $indeksData$, $frekuensiData$, $banyakData$ dan $jumlahData$. Pada $leaf$ nilai $indeksData$ akan diisi dengan nilai parameter l. Nilai $frekuensiData$ dan $banyakData$ akan ditambahkan dengan nilai dari parameter $value$. Untuk nilai $jumlahData$ akan dikurangi atau ditambahkan dengan nilai data array pada indeks l dikali dengan nilai $value$. Nilai $value$ menentukan jenis operasi yang akan dilakukan, antara pengurangan atau penambahan data. Sehingga nilai $value$ hanya memiliki dua kemungkinan nilai yaitu -1 atau 1.

Pada $parent\ node$ nilai agregasi akan diperbarui berdasarkan nilai dari $child$-nya. Untuk nilai $frekuensiData$ dan

$indeksData$ akan mengambil nilai maksimal dari $child$-nya. Untuk nilai $banyakData$ dan $jumlahData$ akan mengambil jumlah nilai dari $child$-nya. Fungsi *update* diimplementasikan pada Kode Sumber 4.13 sesuai dengan *pseudocode* pada bagian 3.1.2.6.

```
1  void update(int idx,int l,int r,int ini,int value){
2      if(ini<l || ini>r) return;
3
4      if(l==r){
5          seg_trees[idx].indeksData = l;
6          seg_trees[idx].frekuensiData += value;
7          seg_trees[idx].banyakData += value;
8          seg_trees[idx].jumlahData += (val[l]*value);
9          return;
10     }
11
12     update(2*idx,l,(l+r)/2,ini,value);
13     update(2*idx+1,(l+r)/2+1,r,ini,value);
14
15     if(seg_trees[2*idx].frekuensiData > seg_trees[2*idx
       +1].frekuensiData){
16       seg_trees[idx] = seg_trees[2*idx];
17     }else{
18       seg_trees[idx] = seg_trees[2*idx+1];
19     }
20     seg_trees[idx].banyakData = seg_trees[2*idx].
       banyakData + seg_trees[2*idx+1].banyakData;
21     seg_trees[idx].jumlahData = seg_trees[2*idx].
       jumlahData + seg_trees[2*idx+1].jumlahData;
22     return;
23 }
```

Kode Sumber 4.13: Implementasi Fungsi *update*

4.2.7 Implementasi Fungsi *findMedian*

Fungsi $findMedian$ digunakan untuk mencari nilai median pada suatu $query$. Untuk mendapatkan nilai $median$ diperlukan

nilai banyaknya data pada rentang yang akan dicari nilai $median$-nya. Nilai banyak data didapatkan dari nilai agregasi banyak data pada $root$ *segment tree*. Nilai ini akan disimpan pada variabel $banyakbilangan$. Apabila nilai $banyakbilangan$ adalah ganjil maka akan dicari data yang berada tepat ditengah rentang data. Jika nilai $banyakbilangan$ adalah bilangan genap makan akan dicari dua nilai tengah pada rentang data.

Pencarian nilai tengah dilakukan dengan menggunakan fungsi $cari$ yang menggunakan nilai agregasi $banyakData$.Fungsi *findMedian* diimplementasikan pada Kode Sumber 4.14 sesuai dengan *pseudocode* pada bagian 3.1.2.7.

```
1  long long findMedian(){
2     int banyakbilangan = seg_trees[1].banyakData;
3     if(banyakbilangan % 2==1){
4        return val[cari(1,0,jumlah-1,banyakbilangan/2+1)];
5     }else{
6        int kiri = cari(1,0,jumlah-1,banyakbilangan/2);
7        int kanan = cari(1,0,jumlah-1,banyakbilangan/2 +
       1);
8        return (val[kiri] + val[kanan])/2;
9     }
10 }
```

Kode Sumber 4.14: Implementasi Fungsi *findMedian*

4.2.8 Implementasi Fungsi *cari*

Fungsi $cari$ digunakan untuk mencari data pada urutan tertentu. Pencarian dilakukan dengan menggunakan metode rekursi pada *seg_tree* dengan memanfaatkan nilai agregasi$banyakData$. Untuk menjalankan fungsi $cari$ diperlukan parameter masukan indeks $root$ (idx), batas kiri (l), batas kanan (r) dan urutan data yang dicaru (ke).

Kondisi rekursi akan berhenti ketika nilai l dan r sama atau ketika sudah mencapai $leaf$. Nilai dari $leaf$ yang akan

dikembalikan sebagai hasil dari fungsi *cari*. Pencarian dimulai dari *root* dan dilanjutkan ke *child* kiri jika nilai *banyakData* pada *child* kiri kurang dari atau sama dengan urutan yang dicari. Jika *banyakData* bernilai lebih maka akan dicari pada *child* kanan. Begitu seterusnya sampai menemui kondisi berhenti dari rekursi.

Fungsi *cari* diimplementasikan pada Kode Sumber 4.15 sesuai dengan *pseudocode* pada bagian 3.1.2.8.

```
1  int cari(int idx,int l,int r,int ke){
2      if(l==r){
3          return l;
4      }
5
6      if(seg_trees[2*idx].banyakData >= ke){
7          return cari(2*idx,l,(l+r)/2,ke);
8      }else{
9          return cari(2*idx+1,(l+r)/2+1,r,ke-seg_trees[2*
       idx].banyakData);
10     }
11 }
```

Kode Sumber 4.15: Implementasi Fungsi *cari*

4.2.9 Implementasi Fungsi untuk Menampilkan Hasil

Untuk menampilkan hasil akan dilakukan iterasi pada variabel *answers* sesuai dengan *pseudocode* pada subbab 3.1.2.9. Hasil akan ditampilkan dengan urutan mulai dari *mean*, *median* kemudian *mode*. Implementasi fungsi untuk menampilkan hasil dapat dilihat pada Kode Sumber 4.16.

```
1      for(int tmp=0;tmp<q;tmp++){
2          cout<<answers[tmp].mean<<" "<<answers[tmp].
       median<<" "<<answers[tmp].mode<<"\n";
3      }
```

Kode Sumber 4.16: Implementasi Fungsi untuk Menampilkan Hasil

4.2.10 Implementasi Data *Generator*

Untuk mendapatkan data tes sesuai dengan format masukan dibuat data *Generator* yang diimplementasikan pada Kode Sumber 4.17 sesuai dengan *pseudocode* pada subbab 3.1.2.10. Data *generator* memerlukan masukan jumlah data dan jumlah query, untuk selanjutnya dibuatkan secara acak data dan *query*-nya.

```
#include<cstdio>
#include<cstdlib>
#include<algorithm>
#include <time.h>
#include <random>
#include <chrono>

using namespace std;

int main()
{
   auto seed = chrono::high_resolution_clock::now().
    time_since_epoch().count();
   mt19937 gen(seed);
   uniform_int_distribution<> random_number(0,
    100000000);

   int N,Q;
   srand (seed);
   scanf("%d %d",&N,&Q);
   printf("%d\n",N);
   for(int i=0;i<N;i++)
   {
      int random = random_number(gen);
      if(i < N - 1)
         printf("%d ",random);
      else
         printf("%d\n",random);
   }
   printf("%d\n",Q);
```

```
29     for(int j=0;j<Q;j++)
30     {
31        int a = rand() % N;
32        int b = rand() % N;
33
34        if(j == Q - 1)
35        {
36           if(a > b)
37              printf("%d %d",b,a);
38           else
39              printf("%d %d",a,b);
40        }
41        else
42        {
43           if(a > b)
44              printf("%d %d\n",b,a);
45           else
46              printf("%d %d\n",a,b);
47        }
48     }
49  }
```

Kode Sumber 4.17: Implementasi Data Generator

(Halaman ini sengaja dikosongkan)

BAB V

UJI COBA DAN EVALUASI

Pada bab ini dijelaskan tentang uji coba dan evaluasi hasil implementasi yang dilakukan pada Tugas Akhir ini.

5.1 Lingkungan Uji Coba

Pada subbab ini akan dijelaskan tentang lingkungan uji coba yang dilakukan pada Tugas Akhir ini. Lingkungan uji coba dalam Tugas Akhir ini meliputi perangkat keras dan perangkat lunak yang digunakan untuk melakukan percobaan adalah sebagai berikut:

1. Bahasa Pemrograman: C++.
2. Perangkat Keras:
    - *Processor*: Intel(R) Core(TM) i7-4720HQ CPU @ 2.60 GHz 2.59 GHz.
    - *Random Access Memory*: 12 GB.
3. Perangkat Lunak:
    - *Operating System*: Windows 10 Professional 64 bit.
    - *IDE*: Eclipse C/C++ Developers Versi Oxygen.3a.
    - *Compiler*: g++ (TDM-GCC 4.8.1 32-bit).

5.2 Uji Coba Kebenaran

Pada subbab ini akan dijelaskan uji coba kebenaran yang dilakukan untuk menguji desain dan implementasi yang sudah dibuat untuk menyelesaikan Tugas Akhir ini.

Uji coba kebenaran dilakukan dengan melakukan analisis penyelesaian sebuah contoh kasus menggunakan pendekatan penyelesaian yang telah dijelaskan pada subbab 2.5. Kasus yang akan digunakan sebagai bahan uji kebenaran dalam analisis penyelesaian DCEPCA09-MMM menggunakan contoh kasus pada Gambar 5.1. Selain dengan contoh kasus, dilakukan pengumpulan berkas kode sumber hasil implementasi ke dalam

```
5
6 5 3 7 7
3
2 4
0 4
1 2
```

Gambar 5.1: Contoh Kasus Uji Permasalahan DCEPCA09-MMM

Array A : Data Masukan					
Indeks	0	1	2	3	4
Data	**6**	**5**	**3**	**7**	**7**

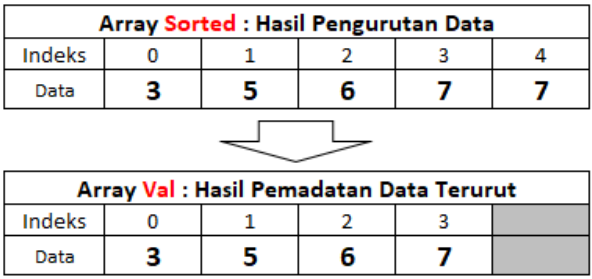
Array Sorted : Hasil Pengurutan Data					
Indeks	0	1	2	3	4
Data	**3**	**5**	**6**	**7**	**7**

Gambar 5.2: Pengurutan Data Masukan ke $Array$ Baru

situs penilaian daring SPOJ. Pengujian juga dilakukan dengan membandingkan hasil dari metode naif dengan metode yang diusulkan pada Tugas Akhir ini.

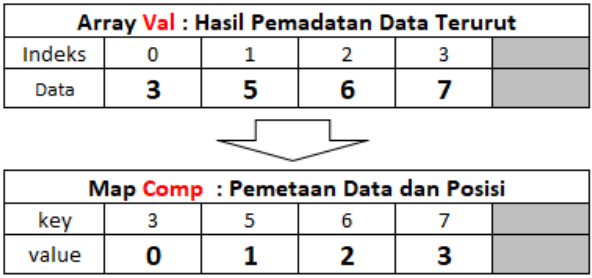
Proses pengerjaan dimulai ketika data masukan sudah diterima oleh program untuk dilakukan proses penentuan indeks posisi data seperti yang sudah dijelaskan pada subbab 3.1.2.2. Langkah pertama untuk melakukan proses penentuan indeks posisi data adalah mengurutkan data ke dalam $array$ baru. Seperti yang ditunjukkan pada Gambar 5.2.

Setelah data diurutkan data akan dipadatkan dengan menghilangkan nilai yang jumlahnya lebih dari satu, sehigga didapat data $array$ unik. Seperti yang ditunjukkan pada Gambar 5.3. Data hasil pemadatan yang sudah terurut akan digunakan untuk membuat pemetaan antara data dan posisinya. Untuk



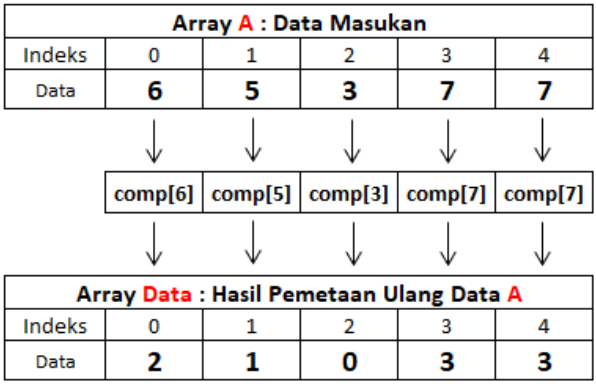
Gambar 5.3: Pemadatan Data Hasil Pengurutan ke $Array$ Baru

melakukan pemetaan digunakan tipe data map. Seperti ilustrasi yang ditunjukkan pada Gambar 5.4. Dapat dilihat bahwa key adalah data dan $value$ adalah posisi data dalam kondisi terurut dan unik. Sebagai contoh nilai 3 dalam kondisi terurut berada pada posisi pertama atau indeks ke-0. Nilai 6 berada pada urutan ke-3 atau indeks ke-2. Sedangkan nilai 7 dalam urutan terletak pada posisi ke-4 atau indeks ke-3.



Gambar 5.4: Pemetaan Data dan Posisi Data dari Hasil Pemadatan Data

Hasil pemetaan masing-masing data dan posisinya akan digunakan untuk memetakan ulang data masukan awal (A). Sehingga didapat data akhir adalah array posisi data masukan (A). Seperti yang ditunjukkan oleh Gambar 5.5.



Gambar 5.5: Pemetaan Ulang Data Masukan Awal Berdasarkan Map Posisi Data

Setelah selesai mengolah data masukan, langkah selanjutnya adalah mengolah operasi $query$ untuk diurutkan. Untuk dapat mengurutkan $query$ diperlukan tiga nilai yang menentukan urutan $query$ yaitu: nilai blok, batas kanan dan urutan masukan. Nilai blok didapatkan dengan membagi batas kiri dengan konstanta isi dari konstanta Algoritma Mo. Seperti yang sudah dijelaskan pada subbab 2.2.2, untuk menghitung jumlah blok dan isi setiap blok digunakan rumus:

$$blok = \sqrt{N} = \lceil \sqrt{5} \rceil = \lceil 2,24 \rceil = 3$$

$$isi = \frac{N}{blok} = \lceil \frac{5}{3} \rceil = \lceil 1,67 \rceil = 2$$

Sehingga didapatkan untuk contoh kasus ini terdapat 3 blok dengan masing-masing blok berisi 2 data kecuali untuk blok terakhir seperti yang ditunjukkan pada Gambar 5.6. Dapat dilihat bahwa blok 0 mencakup indeks 0 dan 1, blok 1 mencakup indeks 2 dan 3 sedangkan blok 2 hanya mencakup indeks 4.

Proses selanjutnya setelah didapatkan nilai konstanta $blok$

Blok	Blok 0		Blok 1		Blok 2
Indeks	0	1	2	3	4

Gambar 5.6: Hasil Pembagian Data menjadi 3 Blok

Tabel 5.1: Tabel Perhitungan Nilai Blok untuk Setiap $Query$

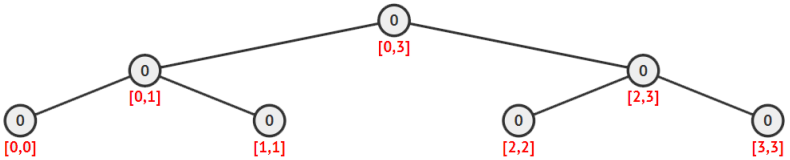
Urutan Masuk	Batas Kiri	Batas Kanan	Perhitungan	Blok
1	2	4	2/2	1
2	0	4	0/2	0
3	1	2	1/2	0

dan isi dari Algoritma Mo adalah menentukan nilai blok untuk setiap $query$ seperti yang ditunjukkan oleh Tabel 5.1. Nilai blok didapatkan dari hasil bagi batas kiri dengan isi, jika hasilnya desimal hanya diambil nilai bilangan bulatnya saja. Nilai blok ini akan digunakan sebagai parameter untuk mengurutkan $query$ beserta dengan nilai batas kanan dan urutan masuknya.

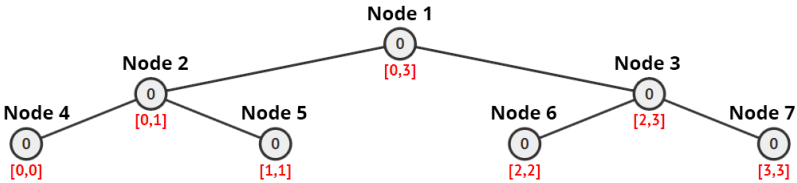
Setelah data diurutkan berdasarkan blok, batas kanan dan urutan masuk sesuai dengan penjelasan pada subbab 3.1.2.4 maka akan didapat hasil urutan pengerjaan pada tabel 5.2. Proses pengerjaan $query$ dilakukan berdasarkan hasil pengurutan $query$. Hasil pengurutan $query$ dimulai dari $query$ dengan urutan masuk ke-3 , ke-2 dan terakhir ke-1. Data yang digunakan

Tabel 5.2: Tabel Perhitungan Nilai Blok untuk Setiap $Query$

Urutan $Query$	Urutan Masuk	Batas Kiri	Batas Kanan	Blok
1	3	1	2	0
2	2	0	4	0
3	1	2	4	1



Gambar 5.7: Inisialisasi Nilai pada *Segment Tree*



Gambar 5.8: Penamaan $Node$ pada *Segment Tree* untuk Memudahkan Penjelasan

untuk proses $query$ adalah data hasil pemetaan indeks posisi seperti yang ditunjukkan pada Gambar 5.5.

Dengan mengimplementasikan Algoritma Mo seperti yang dijelaskan pada subbab 4.2.5 diperlukan variabel untuk menyimpan posisi terbaru dari batas kiri dan batas kanan. Posisi terbaru batas kiri dan batas kanan akan disimpan pada variabel $currentL$ dan $currentR$. Nilai variabel $currentL$ dan $currentR$ akan terus mengalami penambahan atau pengurangan hingga nilainya sesuai dengan batasan $query$. Untuk nilai awal $currentL$ adalah 0 dan $currentR$ adalah -1. Selain menginisiasi nilai $currentR$ dan $currentR$, juga dilakukan inisialisasi *segment tree* sebanyak data unik pada hasil pemadatan data dengan nilai setiap $node$-nya adalah 0 seperti yang ditunjukkan oleh Gambar 5.7. Dengan demikian operasi yang dilakukan setiap pergerseran nilai $currentL$ atau $currentR$ adalah operasi $update$ pada *segment tree*.

Untuk mempermudah penjelasan simulasi uji coba digunakan

penamaan $node$ seperti ditunjukkan pada Gambar 5.8. Proses pengerjaan $query$ akan dimulai dari $query$ dengan urutan masukan ke-3 yang memiliki batas kiri 1 dan batas kanan 2. Simulasi kondisi awal sebelum adanya pergeseran $currentL$ dan $currentR$ ditunjukkan oleh Gambar 5.9. Pergeseran akan dilakukan hingga mencapai batas $query$ kemudian berganti ke operasi $query$ selanjutnya.

CurrentR = -1

[currentL,currentR]	[CL]				
query [L,R]		L	R		
Indeks	0	1	2	3	4
Data	**2**	**1**	**0**	**3**	**3**
Array Data : Hasil Pemetaan Ulang Data A					

Gambar 5.9: Inisialisasi Nilai *currentL* dan *currentR*

Pergeseran akan dimulai dari $currentR$ karena nilai $currentR$ kurang dari batas kanan pada $query$. Pergeseran akan dilakukan satu persatu, mula-mula $currentR$ akan begeser satu nilai ke kanan, sehingga posisi $currentR$ berubah ke indeks 0. Karena nilai $currentR$ lebih kecil dari nilai batas kanan atau pergeserannya ke kanan maka operasi yang dilakukan adalah menambahkan elemen 2 pada *segment tree* sesuai dengan yang sudah dijelaskan pada 4.2.5. Ilustasi pergeseran ditunjukkan pada Gambar 5.10. Setiap pergeseran akan merubah nilai agregasi pada *segment tree* seperti yang sudah dijelaskan pada subbab 3.1.2.5 dan subbab 3.1.2.6.

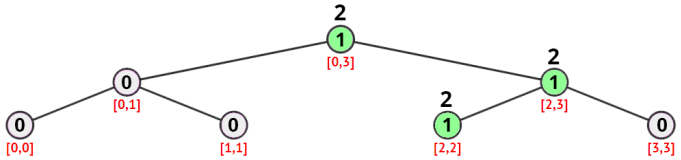
[currentL,currentR]	[CL,CR]				
query [L,R]		L	R		
Indeks	0	1	2	3	4
Data	**2**	**1**	**0**	**3**	**3**

Gambar 5.10: Pergeseran Nilai *currentR* dari indeks -1 ke Indeks 0

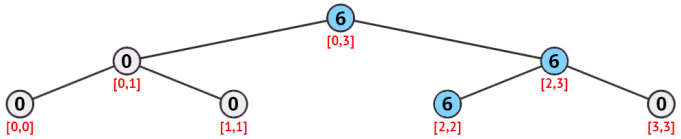
Penambahan data 2 pada *segment tree* merubah nilai agregasi:

1. Nilai frekuensi data menjadi 1 dan indeks data menjadi 2 pada *node* 6. Pada *node* 3 dan *node* 1 memiliki nilai frekuensi data 1 dan indeks data 2 diambil dari nilai maksimum frekuensi *child node*-nya. Jika nilai frekuensi sama diambil dari yang indeksnya lebih besar atau *child* kanan.
2. Nilai agregasi jumlah data berubah menjadi 6 pada *node* 6. Nilai 6 adalah data unik pada data terurut pada indeks ke 2. Pada *node* 3 dan 1 memiliki nilai 6 diambil dari hasil penjumlahan *node child*-nya.
3. Nilai agregasi banyak data berubah menjadi 1 pada *node* 6. Nilai agregasi banyak data pada *node* 3 dan *node* 1 menjadi 1 diambil dari hasil penjumlahan *child node*-nya.

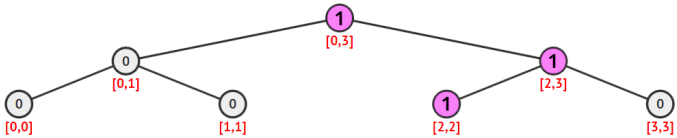
Ilustrasi perubahan data untuk operasi tersebut ditunjukkan pada Gambar 5.11, Gambar 5.12 dan Gambar 5.13.



Gambar 5.11: Perubahan Nilai Agregasi Frekuensi Data dan Indeks Data Karena Penambahan Data 2



Gambar 5.12: Perubahan Nilai Agregasi Jumlah Data Karena Penambahan Data 2



Gambar 5.13: Perubahan Nilai Agregasi Banyak Data Karena Penambahan Data 2

Karena nilai $currentR$ belum sesuai dengan batas kanan pada $query$. Pergeseran akan dilanjutkan dengan menggeser $currentR$ satu nilai ke kanan, sehingga posisi $currentR$ berubah ke indeks 1. Operasi yang dilakukan adalah menambahkan elemen 1 pada *segment tree*. Ilustasi pergeseran ditunjukkan pada Gambar 5.14.

[currentL,currentR]	[CL]	[CR]			
query [L,R]		L	R		
Indeks	0	1	2	3	4
Data	**2**	**1**	**0**	**3**	**3**

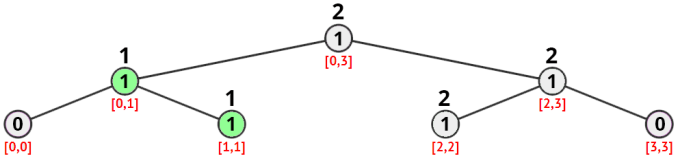
Gambar 5.14: Pergeseran Nilai *currentR* dari indeks 0 ke Indeks 1

Penambahan data 1 pada *segment tree* merubah nilai agregasi:

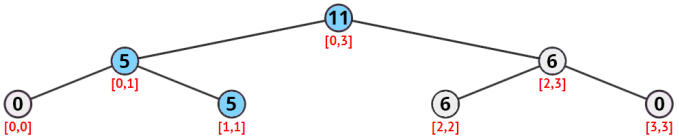
1. Nilai frekuensi data menjadi 1 dan indeks data menjadi 1 pada $node$ 5. Pada $node$ 2 memiliki nilai frekuensi data 1 dan indeks data 1 diambil dari nilai maksimum frekuensi $child\ node$-nya. Pada $node$ 1 data tidak berubah karena nilai agregasi pada $node$ 2 tidak lebih besar dari $node$ 3 sehingga $node$ 1 tetap mangambil data dari $node$ 3.
2. Nilai agregasi jumlah data berubah menjadi 5 pada $node$ 5. Nilai 5 adalah data unik pada data terurut pada indeks ke 1. Nilai agregasi jumlah data pada node 2 menjadi 5 dan $node$ 1 berubah menjadi 11 diambil dari hasil penjumlahan $child\ node$-nya.
3. Nilai agregasi banyak data berubah menjadi 1 pada $node$ 5. Nilai agregasi banyak data pada node 2 menjadi 1 dan $node$

1 berubah menjadi 2 diambil dari hasil penjumlahan *child node*-nya.

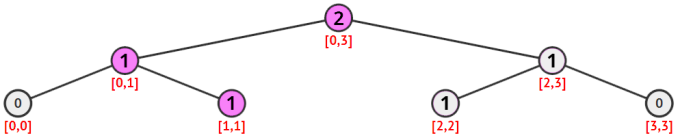
Ilustrasi perubahan data untuk operasi tersebut ditunjukkan pada Gambar 5.15, Gambar 5.16 dan Gambar 5.17.



Gambar 5.15: Perubahan Nilai Agregasi Frekuensi Data dan Indeks Data Karena Penambahan Data 1



Gambar 5.16: Perubahan Nilai Agregasi Jumlah Data Karena Penambahan Data 1



Gambar 5.17: Perubahan Nilai Agregasi Banyak Data Karena Penambahan Data 1

Karena nilai $currentR$ belum sesuai dengan batas kanan pada $query$. Pergeseran akan dilanjutkan dengan menggeser $currentR$ satu nilai ke kanan, sehingga posisi $currentR$ berubah ke indeks 2. Operasi yang dilakukan adalah menambahkan elemen 0 pada *segment tree*. Ilustasi pergeseran ditunjukkan pada Gambar 5.18.

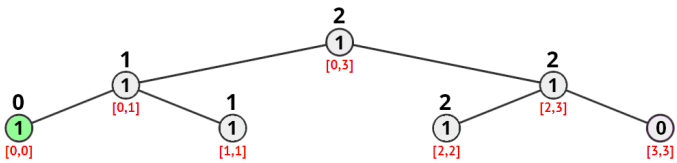
[currentL,currentR]	[CL]		[CR]		
query [L,R]		L	R		
Indeks	0	1	2	3	4
Data	**2**	**1**	**0**	**3**	**3**

Gambar 5.18: Pergeseran Nilai *currentR* dari indeks 1 ke Indeks 2

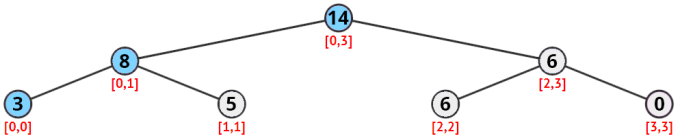
Penambahan data 0 pada *segment tree* merubah nilai agregasi:

1. Nilai frekuensi data menjadi 1 dan indeks data menjadi 0 pada *node* 4. Pada *node* 2 data tidak berubah karena nilai agregasi pada *node* 4 tidak lebih besar dari *node* 5 sehingga *node* 2 tetap mangambil data dari *node* 5 karena memiliki nilai indeks data yang lebih besar.
2. Nilai agregasi jumlah data berubah menjadi 3 pada *node* 4. Nilai 3 adalah data unik pada data terurut pada indeks ke 0. Nilai agregasi jumlah data pada *node* 2 berubah menjadi 8 dan *node* 1 berubah menjadi 14 diambil dari hasil penjumlahan *child node*-nya.
3. Nilai agregasi banyak data berubah menjadi 1 pada *node* 4. Nilai agregasi banyak data pada *node* 2 berubah menjadi 2 dan *node* 1 berubah menjadi 3 diambil dari hasil penjumlahan *child node*-nya.

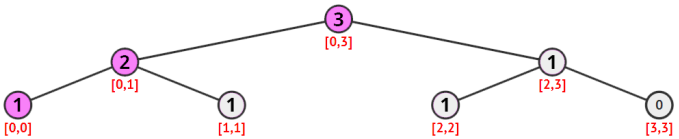
Ilustrasi perubahan data untuk operasi tersebut ditunjukkan pada Gambar 5.19, Gambar 5.20 dan Gambar 5.21.



Gambar 5.19: Perubahan Nilai Agregasi Frekuensi Data dan Indeks Data Karena Penambahan Data 0



Gambar 5.20: Perubahan Nilai Agregasi Jumlah Data Karena Penambahan Data 0



Gambar 5.21: Perubahan Nilai Agregasi Banyak Data Karena Penambahan Data 0

Karena nilai $currentR$ sudah sesuai dengan batas kanan pada $query$ maka selanjutnya akan dicek nilai dari $currentL$. Karena nilai $currentL$ lebih kecil dari nilai batas kiri maka pergeserannya dilakukan ke kanan dengan operasi mengurangi elemen 2 pada *segment tree*. Sehingga posisi $currentL$ berubah dari indeks 1 ke indeks 2. Ilustasi pergeseran ditunjukkan pada Gambar 5.22.

[currentL,currentR]		[CL]	[CR]		
query [L,R]		L	R		
Indeks	0	1	2	3	4
Data	**2**	**1**	**0**	**3**	**3**

Gambar 5.22: Pergeseran Nilai *currentL* dari indeks 0 ke Indeks 1

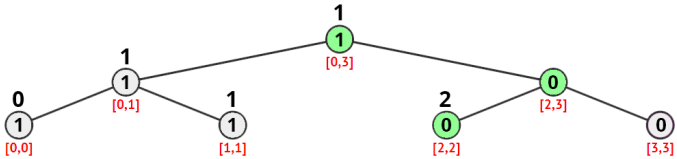
Pengurangan data 2 pada *segment tree* merubah nilai agregasi:

1. Nilai frekuensi data menjadi 0 pada $node$ 6. Nilai frekuensi pada $node$ 2 terjadi perubahan nilai menjadi 0 diambil dari nilai maksimum $child\ node$-nya. Pada $node$ 1

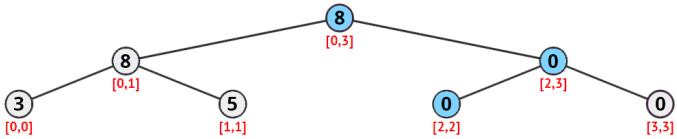
terjadi perubahan data karena nilai agregasi frekuensi pada $node$ 2 lebih besar dari $node$ 3 sehingga $node$ 1 mangambil data dari $node$ 2.

2. Nilai agregasi jumlah data berubah menjadi 0 pada $node$ 6. Nilai 0 didapatkan dari pengurangan nilai agregasinya sebelumnya dengan nilai 6. Nilai 6 adalah data unik pada data terurut indeks ke 2. Nilai agregasi jumlah data pada $node$ 3 berubah menjadi 0 dan $node$ 1 berubah menjadi 8 diambil dari hasil penjumlahan $child\ node$-nya.
3. Nilai agregasi banyak data berubah menjadi 0 pada $node$ 6. Nilai agregasi banyak data pada $node$ 3 berubah menjadi 0 dan $node$ 1 berubah menjadi 2 diambil dari hasil penjumlahan $child\ node$-nya.

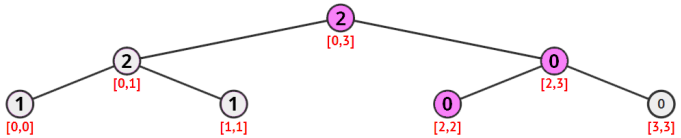
Ilustrasi perubahan data untuk operasi tersebut ditunjukkan pada Gambar 5.23, Gambar 5.24 dan Gambar 5.25.



Gambar 5.23: Perubahan Nilai Agregasi Frekuensi Data dan Indeks Data Karena Pengurangan Data 2



Gambar 5.24: Perubahan Nilai Agregasi Jumlah Data Karena Pengurangan Data 2



Gambar 5.25: Perubahan Nilai Agregasi Banyak Data Karena Pengurangan Data 2

Karena nilai $currentL$ dan $currentR$ sudah sesuai dengan batas kanan dan batas kiri pada $query$. Maka nilai-nilai agregasi pada *segment tree* dapat digunakan untuk menentukan nilai $mean$, $median$ dan $mode$ dari $query$ yang sedang diproses.

Untuk mendapatkan nilai $mean$ dilakukan dengan membagi nilai agregasi jumlah data pada $root$ *segment tree* dengan banyaknya data pada rentang $query$. Dari kondisi *segment tree* terakhir didapatkan nilai agregasi untuk jumlah data pada $root$ adalah 8. Banyaknya data didapatkan dari batas kanan dan kiri dari dari $query$. Dengan batasan $query$ 1 dan 2. Maka terdapat data sebanyak 2 data. Sehingga didapatkan $mean$ untuk data dengan rentang indeks 1 dan 2 adalah:

$$mean = \frac{8}{2} = 4$$

.

Nilai $median$ didapatkan dengan menentukan terlebih dahulu banyak data pada rentang $query$ antara ganjil atau genap. Jika genap maka akan terdapat dua nilai tengah untuk dijumlahkan kemudian dibagi 2. Jika genap maka dilakukan dengan mencari satu nilai tengah. Karena banyak data adalah 2 yang berarti genap, maka akan dicari dua nilai tengah. Nilai pertama yang dicari adalah data dengan urutan ke:

$$\frac{2}{2} = 1$$

Nilai kedua yang dicari adalah data dengan urutan ke:

$$\frac{2}{2} + 1 = 2$$

Untuk mencari data urutan tertentu akan digunakan nilai agregasi banyak data pada *segment tree* seperti yang dijelaskan pada subbab 3.1.2.8. Untuk mencari data urutan ke-1 pencarian dimulai dari $root$. Karena nilai $child$ kiri dari $root$ masih lebih dari urutan yang dicari pencarian dilanjutkan ke $node$ 2. Karena nilai $child$ kiri dari $node$ 2 masih sama dengan 1 sehingga pencarian dilanjutkan ke $node$ 4. Karena $node$ 4 merupakan $leaf$ maka pencarian berhenti dan nilai indeks dari $array$ yang diwakili $node$ 4 dikembalikan sebagai hasil pencarian. $Node$ 4 merepresentasikan indeks ke 0. Nilai data indeks ke-0 pada hasil pengurutan data adalah 3 seperti yang ditunjukkan pada Gambar 5.3. Sehingga didapatkan data urutan ke-1 adalah 3.

Untuk mencari data urutan ke-2 pencarian dimulai dari $root$. Karena nilai $child$ kiri dari $root$ masih sama dengan urutan yang dicari pencarian dilanjutkan ke $node$ 2. Karena nilai $child$ kiri $node$ 2 lebih kecil dari urutan yang dicari sehingga pencarian dilanjutkan ke $child$ kanan yaitu $node$ 5. Karena $node$ 5 merupakan $leaf$ maka pencarian berhenti dan nilai indeks dari $array$ yang diwakili $node$ 5 dikembalikan sebagai hasil pencarian. $Node$ 5 merepresentasikan indeks ke 1. Nilai data indeks ke-1 pada hasil pengurutan data adalah 5. Sehingga didapatkan data urutan ke-2 adalah 5.

Setelah mendapatkan data urutan ke-1 dan ke-2. Selanjutnya akan dijumlahkan dan dibagi 2. sehingga didapat $median$ untuk $query$ dengan batas 1 dan 2 adalah:

$$median = \frac{3+5}{2} = 4$$

Nilai $mode$ didapakan dari $nilai\ root$ pada agregasi frekuensi

dan indeks data. Frekuensi menyatakan nilai banyaknya kemunculan data sedangkan indeks data merepresentasikan indeks data yang memiliki nilai frekuensi tersebut. Nilai frekuensi dan indeks data pada $root$ adalah 1 dan 1. Didapatkan nilai $mode$ adalah data indeks ke-1 pada data terurut dengan kemunculan sebanyak satu kali. Nilai data indeks ke-1 pada hasil pengurutan data adalah 5. Maka didapatkan $mode$ untuk $query$ dengan batas 1 dan 2 adalah 5. Rangkuman hasil untuk $query$ batas 1 dan 2 ditunjukkan oleh Tabel 5.3.

Tabel 5.3: Tabel Hasil Perhitungan $Mean$, $Median$, $Mode$ pada $Query$ dengan Batas 1 dan 2

Urutan Masuk	Batas Kiri	Batas Kanan	Mean	Median	mode
3	1	2	4	4	5

Setelah mendapatkan nilai $mean$, $median$ dan $mode$ untuk operasi $query$ dengan batas kiri 1 dan batas kanan 2. Selanjutnya akan diproses untuk $query$ dengan batas kiri 0 dan batas kiri 4. Simulasi kondisi awal sebelum adanya pergeseran $currentL$ dan $currentR$ ditunjukkan oleh Gambar 5.26. Dapat dilihat bahwa $currentL$ dan $currentR$ menggunakan nilai dari operasi $query$ sebelumnya, hal ini untuk meminimalisasi pergeseran yang merupakan kelebihan Algoritma Mo.

[currentL,currentR]		[CL]	[CR]		
query [L,R]	L				R
Indeks	0	1	2	3	4
Data	**2**	**1**	**0**	**3**	**3**

Gambar 5.26: Pergantian Operasi Query Mengubah Batas Kanan dan Kiri menjadi 0 dan 4

Karena nilai $currentR$ belum sesuai dengan batas kanan pada

$query$. Pergeseran akan dilakukan dengan menggeser $currentR$ satu nilai ke kanan, sehingga posisi $currentR$ berubah ke indeks 3. Operasi yang dilakukan adalah menambahkan elemen 3 pada *segment tree*. Ilustasi pergeseran ditunjukkan pada Gambar 5.27.

[currentL,currentR]		[CL]		[CR]	
query [L,R]	L				R
Indeks	0	1	2	3	4
Data	**2**	**1**	**0**	**3**	**3**

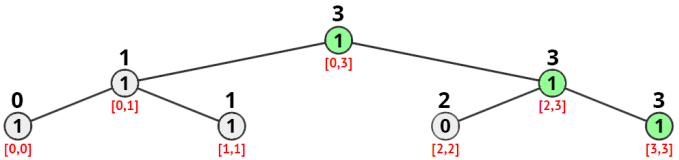
Gambar 5.27: Pergeseran Nilai *currentR* dari indeks 2 ke Indeks 3

Penambahan data 3 pada *segment tree* merubah nilai agregasi:

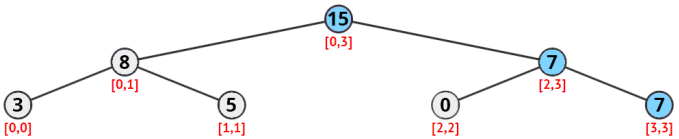
1. Nilai frekuensi data menjadi 1 dan indeks data menjadi 3 pada $node$ 7. Pada $node$ 3 dan $node$ 1 memiliki nilai frekuensi data 1 dan indeks data 3 diambil dari nilai maksimum frekuensi $child\ node$-nya.
2. Nilai agregasi jumlah data berubah menjadi 7 pada $node$ 7. Nilai 7 adalah data unik pada data terurut pada indeks ke 3. Pada $node$ 3 memiliki nilai jumlah data sebesar 7 sedangkan pada $node$ 1 memiliki nilai 15 diambil dari hasil penjumlahan $node\ child$-nya.
3. Nilai agregasi banyak data berubah menjadi 1 pada $node$ 7. Nilai agregasi banyak data pada $node$ 3 berubah menjadi 1 dan $node$ 1 menjadi 3 diambil dari hasil penjumlahan $child\ node$-nya.

Ilustrasi perubahan data untuk operasi tersebut ditunjukkan pada Gambar 5.28, Gambar 5.29 dan Gambar 5.30.

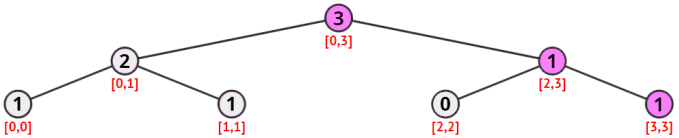
Karena nilai $currentR$ masih belum sesuai dengan batas kanan pada $query$. Pergeseran akan dilakukan dengan menggeser $currentR$ satu nilai ke kanan, sehingga posisi $currentR$ berubah ke indeks 4. Operasi yang dilakukan adalah menambahkan elemen 3 pada *segment tree*. Ilustasi pergeseran ditunjukkan pada Gambar 5.31.



Gambar 5.28: Perubahan Nilai Agregasi Frekuensi Data dan Indeks Data Karena Penambahan Data 3



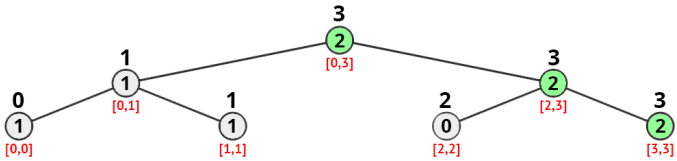
Gambar 5.29: Perubahan Nilai Agregasi Jumlah Data Karena Penambahan Data 3



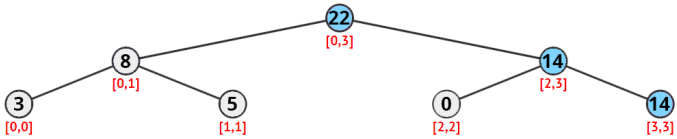
Gambar 5.30: Perubahan Nilai Agregasi Banyak Data Karena Penambahan Data 3

[currentL,currentR]		[CL]			[CR]
query [L,R]	L				R
Indeks	0	1	2	3	4
Data	**2**	**1**	**0**	**3**	**3**

Gambar 5.31: Pergeseran Nilai *currentR* dari indeks 3 ke Indeks 4



Gambar 5.32: Perubahan Nilai Agregasi Frekuensi Data dan Indeks Data Karena Penambahan Data 3



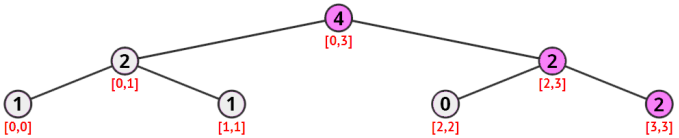
Gambar 5.33: Perubahan Nilai Agregasi Jumlah Data Karena Penambahan Data 3

Penambahan data 3 pada *segment tree* merubah nilai agregasi:

1. Nilai frekuensi data menjadi 2 pada $node$ 7. Pada $node$ 3 dan $node$ 1 memiliki nilai frekuensi data 2 dan indeks data 3 diambil dari nilai maksimum frekuensi $child\ node$-nya.
2. Nilai agregasi jumlah data berubah menjadi 14 pada $node$ 7. Nilai 14 adalah hasil penambahan nilai sebelumnya ditambah dengan data unik pada data terurut indeks ke 3 yaitu 7. Pada $node$ 3 memiliki nilai jumlah data sebesar 14 sedangkan pada $node$ 1 memiliki nilai 22 diambil dari hasil penjumlahan $node\ child$-nya.
3. Nilai agregasi banyak data berubah menjadi 2 pada $node$ 7. Nilai agregasi banyak data pada $node$ 3 berubah menjadi 2 dan $node$ 1 menjadi 4 diambil dari hasil penjumlahan $child\ node$-nya.

Ilustrasi perubahan data untuk operasi tersebut ditunjukkan pada Gambar 5.32, Gambar 5.33 dan Gambar 5.34.

Karena nilai $currentR$ sudah sesuai dengan batas kanan pada $query$ maka selanjutnya akan dicek nilai dari $currentL$. Karena



Gambar 5.34: Perubahan Nilai Agregasi Banyak Data Karena Penambahan Data 3

nilai $currentL$ lebih besar dari nilai batas kiri maka pergeserannya dilakukan ke kiri dengan operasi menambahkan elemen 2 pada *segment tree*. Sehingga posisi $currentL$ berubah dari indeks 1 ke indeks 0. Ilustasi pergeseran ditunjukkan pada Gambar 5.35.

[currentL,currentR]	[CL]				[CR]
query [L,R]	L				R
Indeks	0	1	2	3	4
Data	**2**	**1**	**0**	**3**	**3**

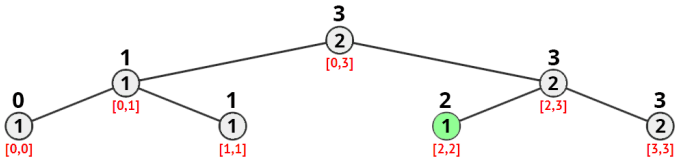
Gambar 5.35: Pergeseran Nilai *currentL* dari indeks 1 ke Indeks 0

Penambahan data 2 pada *segment tree* merubah nilai agregasi:

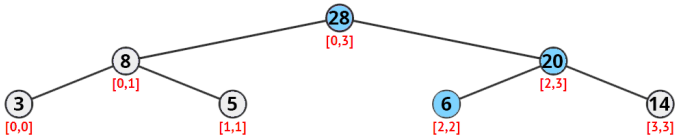
1. Nilai frekuensi data menjadi 1 pada $node$ 6. Pada $node$ 3 karena nilai frekuensi $node$ 7 lebih besar dari $node$ 6 maka diambil yang lebih besar yaitu node 7.
2. Nilai agregasi jumlah data berubah menjadi 6 pada $node$ 6. Nilai 6 adalah data unik pada urutan indeks ke 2. Pada $node$ 3 memiliki nilai jumlah data sebesar 20 sedangkan pada $node$ 1 memiliki nilai 28 diambil dari hasil penjumlahan $node\ child$-nya.
3. Nilai agregasi banyak data berubah menjadi 1 pada $node$ 6. Nilai agregasi banyak data pada $node$ 3 berubah menjadi 3 dan $node$ 1 menjadi 5 diambil dari hasil penjumlahan $child\ node$-nya.

Ilustrasi perubahan data untuk operasi tersebut ditunjukkan pada

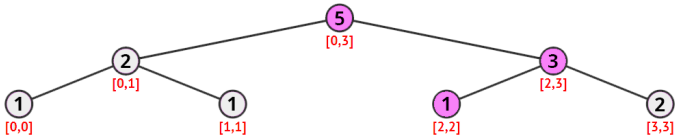
Gambar 5.36, Gambar 5.37 dan Gambar 5.38.



Gambar 5.36: Perubahan Nilai Agregasi Frekuensi Data dan Indeks Data Karena Penambahan Data 2



Gambar 5.37: Perubahan Nilai Agregasi Jumlah Data Karena Penambahan Data 2



Gambar 5.38: Perubahan Nilai Agregasi Banyak Data Karena Penambahan Data 2

Karena nilai $currentL$ dan $currentR$ sudah sesuai dengan batas kanan dan batas kiri pada $query$. Maka nilai-nilai agregasi pada *segment tree* dapat digunakan untuk menentukan nilai $mean$, $median$ dan $mode$ dari $query$ yang sedang diproses.

Untuk mendapatkan nilai $mean$ dilakukan dengan membagi nilai agregasi jumlah data pada $root$ *segment tree* dengan banyaknya data pada rentang $query$. Dari kondisi *segment tree*

terakhir didapatkan nilai agregasi untuk jumlah data pada $root$ adalah 28. Banyaknya data didapatkan dari batas kanan dan kiri dari dari $query$. Dengan batasan $query$ 0 dan 4. Maka terdapat data sebanyak 5 data. Sehingga didapatkan $mean$ untuk data dengan rentang indeks 0 dan 4 adalah:

$$mean = \frac{28}{5} = \lfloor 5,6 \rfloor = 5$$

.

Nilai $median$ didapatkan dengan menentukan terlebih dahulu banyak data pada rentang $query$ antara ganjil atau genap. Jika genap maka akan terdapat dua nilai tengah untuk dijumlahkan kemudian dibagi 2. Jika genap maka dilakukan dengan mencari satu nilai tengah. Karena banyak data adalah 5 yang berarti ganjil, maka akan dicari satu nilai tengah. Nilai tengah yang dicari adalah data dengan urutan ke:

$$\frac{5}{2} + 1 = \lfloor 2,5 \rfloor + 1 = 3$$

Untuk mencari data urutan tertentu akan digunakan nilai agregasi banyak data pada *segment tree* seperti yang dijelaskan pada subbab 3.1.2.8. Untuk mencari data urutan ke-3 pencarian dimulai dari $root$. Karena nilai $child$ kiri dari $root$ lebih kecil dari urutan yang dicari maka pencarian dilanjutkan ke $child$ kanan yaitu $node$ 3. Urutan pencarian diperbarui menjadi 1 karena sudah terdapat dua data pada $child$ kiri dari $root$. Dari $node$ 3 dilakukan pengecekan untuk $child$ kiri didapati nilainya sama dengan 1 maka pencarian dilanjutkan ke $node$ tersebut yaitu $node$ 6. Karena $node$ 6 merupakan $leaf$ maka pencarian berhenti dan nilai indeks dari $array$ yang diwakili $node$ 6 dikembalikan sebagai hasil pencarian. $Node$ 6 merepresentasikan indeks ke 2. Nilai data indeks ke-2 pada hasil pengurutan data adalah 6. Sehingga didapatkan $median$ untuk

$query$ dengan batas 0 dan 4 adalah 6.

Nilai $mode$ didapakan dari $nilai\ root$ pada agregasi frekuensi dan indeks data. Frekuensi menyatakan nilai banyaknya kemunculan data sedangkan indeks data merepresentasikan indeks data yang memiliki nilai frekuensi tersebut. Nilai frekuensi dan indeks data pada $root$ adalah 2 dan 3. Didapatkan nilai $mode$ adalah data indeks ke-3 pada data terurut dengan kemunculan sebanyak dua kali. Nilai data indeks ke-3 pada hasil pengurutan data adalah 7. Maka didapatkan $mode$ untuk $query$ dengan batas 0 dan 4 adalah 7. Rangkuman hasil untuk $query$ batas 0 dan 4 ditunjukkan oleh Tabel 5.4.

Tabel 5.4: Tabel Hasil Perhitungan $Mean$, $Median$, $Mode$ pada $Query$ dengan Batas 1 dan 2

Urutan Masuk	Batas Kiri	Batas Kanan	Mean	Median	mode
2	0	4	5	6	7

Setelah mendapatkan nilai $mean$, $median$ dan $mode$ untuk operasi $query$ dengan batas kiri 0 dan batas kanan 4. Selanjutnya akan diproses untuk $query$ dengan batas kiri 2 dan batas kiri 4. Simulasi kondisi awal sebelum adanya pergeseran $currentL$ dan $currentR$ ditunjukkan oleh Gambar 5.39.

[currentL,currentR]	[CL]				[CR]
query [L,R]			L		R
Indeks	0	1	2	3	4
Data	**2**	**1**	**0**	**3**	**3**

Gambar 5.39: Pergantian Operasi Query Mengubah Batas Kanan dan Kiri menjadi 2 dan 4

Karena nilai $currentR$ sudah sesuai dengan batas kanan pada $query$ maka tidak perlu dilakukan pergeseran pada

$currentR$. Sedangkan pada $currentL$ memiliki nilai lebih kecil dari batas kiri maka pergeserannya dilakukan ke kanan dengan operasi mengurangi elemen 2 pada *segment tree*. Sehingga posisi $currentL$ berubah dari indeks 0 ke indeks 1. Ilustasi pergeseran ditunjukkan pada Gambar 5.40.

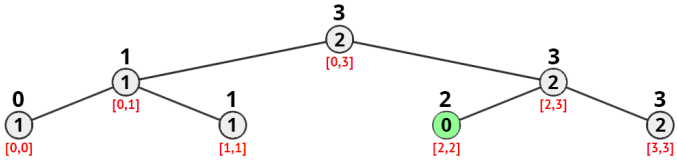
[currentL,currentR]		[CL]			[CR]
query [L,R]			L		R
Indeks	0	1	2	3	4
Data	**2**	**1**	**0**	**3**	**3**

Gambar 5.40: Pergeseran Nilai *currentL* dari indeks 0 ke Indeks 1

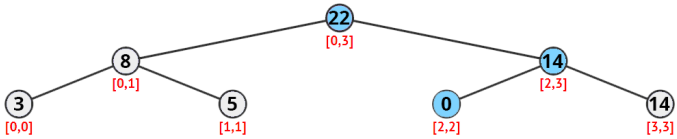
Pengurangan data 2 pada *segment tree* merubah nilai agregasi:

1. Nilai frekuensi data menjadi 0 pada $node$ 6. Pada $node$ 3 tidak terjadi perubahan data karena nilai agregasi frekuensi pada $node$ 7 lebih besar dari $node$ 6 sehingga $node$ 3 tetap mangambil data dari $node$ 7.
2. Nilai agregasi jumlah data berubah menjadi 0 pada $node$ 6. Nilai 0 didapatkan dari pengurangan nilai agregasinya sebelumnya dengan nilai 6. Nilai 6 adalah data unik pada data terurut indeks ke 2. Nilai agregasi jumlah data pada $node$ 3 berubah menjadi 14 dan $node$ 1 menjadi 22 diambil dari hasil penjumlahan $child$ $node$-nya.
3. Nilai agregasi banyak data berubah menjadi 0 pada $node$ 6. Nilai agregasi banyak data pada $node$ 3 berubah menjadi 2 dan $node$ 1 menjadi 4 diambil dari hasil penjumlahan $child$ $node$-nya.

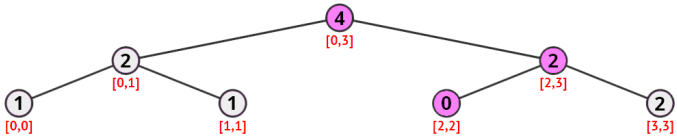
Ilustrasi perubahan data untuk operasi tersebut ditunjukkan pada Gambar 5.41, Gambar 5.42 dan Gambar 5.43.



Gambar 5.41: Perubahan Nilai Agregasi Frekuensi Data dan Indeks Data Karena Pengurangan Data 2



Gambar 5.42: Perubahan Nilai Agregasi Jumlah Data Karena Pengurangan Data 2



Gambar 5.43: Perubahan Nilai Agregasi Banyak Data Karena Pengurangan Data 2

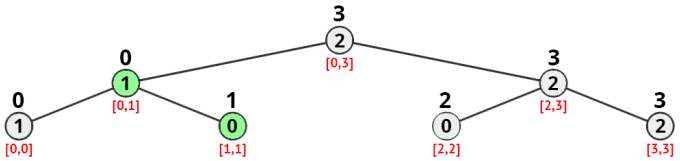
Karena nilai $currentL$ masih belum sesuai dengan batas kiri pada $query$. Pergeseran akan dilakukan dengan menggeser $currentL$ satu nilai ke kanan, sehingga posisi $currentL$ berubah ke indeks 2. Operasi yang dilakukan adalah mengurangi elemen 1 pada *segment tree*. Ilustasi pergeseran ditunjukkan pada Gambar 5.44.

Pengurangan data 1 pada *segment tree* merubah nilai agregasi:

1. Nilai frekuensi data menjadi 0 pada $node$ 5. Pada $node$ 2 terjadi perubahan data karena nilai agregasi frekuensi pada

[currentL,currentR]			[CL]		[CR]
query [L,R]			L		R
Indeks	0	1	2	3	4
Data	**2**	**1**	**0**	**3**	**3**

Gambar 5.44: Pergeseran Nilai *currentL* dari indeks 1 ke Indeks 2



Gambar 5.45: Perubahan Nilai Agregasi Frekuensi Data dan Indeks Data Karena Pengurangan Data 1

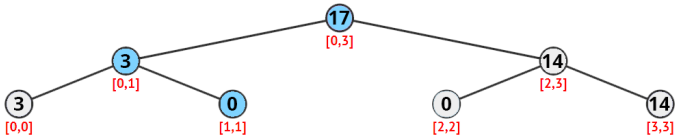
$node$ 4 lebih besar dari $node$ 5 sehingga $node$ 2 mangambil data dari $node$ 4.

2. Nilai agregasi jumlah data berubah menjadi 0 pada $node$ 5. Nilai 0 didapatkan dari pengurangan nilai agregasinya sebelumnya dengan nilai 5. Nilai 5 adalah data unik pada data terurut indeks ke 1. Nilai agregasi jumlah data pada $node$ 2 berubah menjadi 3 pada $node$ 1 menjadi 17 diambil dari hasil penjumlahan $child$ $node$-nya.
3. Nilai agregasi banyak data berubah menjadi 0 pada $node$ 5. Nilai agregasi banyak data pada $node$ 2 berubah menjadi 1 dan $node$ 1 menjadi 3 diambil dari hasil penjumlahan $child$ $node$-nya.

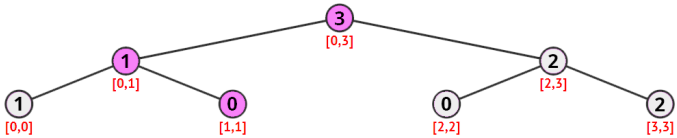
Ilustrasi perubahan data untuk operasi tersebut ditunjukkan pada Gambar 5.45, Gambar 5.46 dan Gambar 5.47.

Karena nilai $currentL$ dan $currentR$ sudah sesuai dengan batas kanan dan batas kiri pada $query$. Maka nilai-nilai agregasi pada *segment tree* dapat digunakan untuk menentukan nilai $mean$, $median$ dan $mode$ dari $query$ yang sedang diproses.

Untuk mendapatkan nilai $mean$ dilakukan dengan membagi



Gambar 5.46: Perubahan Nilai Agregasi Jumlah Data Karena Pengurangan Data 1



Gambar 5.47: Perubahan Nilai Agregasi Banyak Data Karena Pengurangan Data 1

nilai agregasi jumlah data pada *root segment tree* dengan banyaknya data pada rentang $query$. Dari kondisi *segment tree* terakhir didapatkan nilai agregasi untuk jumlah data pada $root$ adalah 17. Banyaknya data didapatkan dari batas kanan dan kiri dari dari $query$. Dengan batasan $query$ 2 dan 4. Maka terdapat data sebanyak 3 data. Sehingga didapatkan $mean$ untuk data dengan rentang indeks 2 dan 4 adalah:

$$mean = \frac{17}{3} = \lfloor 5,7 \rfloor = 5$$

.

Nilai $median$ didapatkan dengan menentukan terlebih dahulu banyak data pada rentang $query$ antara ganjil atau genap. Jika genap maka akan terdapat dua nilai tengah untuk dijumlahkan kemudian dibagi 2. Jika genap maka dilakukan dengan mencari satu nilai tengah. Karena banyak data adalah 3 yang berarti ganjil, maka akan dicari satu nilai tengah. Nilai

tengah yang dicari adalah data dengan urutan ke:

$$\frac{3}{2} + 1 = \lfloor 1,5 \rfloor + 1 = 2$$

Untuk mencari data urutan tertentu akan digunakan nilai agregasi banyak data pada *segment tree* seperti yang dijelaskan pada subbab 3.1.2.8. Untuk mencari data urutan ke-2 pencarian dimulai dari $root$. Karena nilai $child$ kiri dari $root$ lebih kecil dari urutan yang dicari sehingga pencarian dilanjutkan ke $child$ kanan yaitu $node$ 3. Pada $node$ 3 urutan selanjutnya yang dicari adalah urutan ke 2 karena $child$ kiri sebelumnya sudah memiliki nilai 1. Karena nilai $child$ kiri dari $node$ 3 lebih kecil dari 2 pencarian dilanjutkan ke $node$ 7. Karena $node$ 7 merupakan $leaf$ maka pencarian berhenti dan nilai indeks dari $array$ yang diwakili $node$ 7 dikembalikan sebagai hasil pencarian. $Node$ 7 merepresentasikan indeks ke 3. Nilai data indeks ke-3 pada hasil pengurutan data adalah 7. Sehingga didapatkan $median$ untuk $query$ dengan batas 2 dan 4 adalah 7.

Nilai $mode$ didapakan dari $nilai\ root$ pada agregasi frekuensi dan indeks data. Frekuensi menyatakan nilai banyaknya kemunculan data sedangkan indeks data merepresentasikan indeks data yang memiliki nilai frekuensi tersebut. Nilai frekuensi dan indeks data pada $root$ adalah 2 dan 3. Didapatkan nilai $mode$ adalah data indeks ke-3 pada data terurut dengan kemunculan sebanyak dua kali. Nilai data indeks ke-3 pada hasil pengurutan data adalah 7. Maka didapatkan $mode$ untuk $query$ dengan batas 2 dan 4 adalah 7. Rangkuman hasil untuk $query$ batas 2 dan 4 ditunjukkan oleh Tabel 5.5.

Semua $query$ sudah selesai diproses dan menghasilkan data seperti ditunjukkan pada Tabel 5.9. Untuk menguji kebenaran dari hasil perhitungan program akan dibandingkan dengan berhitungan manual sesuai dengan batasan-batasan pada permasalahan DCEPCA09-MMM.

Tabel 5.5: Tabel Hasil Perhitungan $Mean$, $Median$, $Mode$ pada $Query$ dengan Batas 2 dan 4

Urutan Masuk	Batas Kiri	Batas Kanan	Mean	Median	mode
1	2	4	5	7	7

Tabel 5.6: Tabel Hasil Perhitungan $Mean$, $Median$, $Mode$ pada Semua $Query$

Urutan Masuk	Batas Kiri	Batas Kanan	Mean	Median	mode
3	1	2	4	4	5
2	0	4	5	6	7
1	2	4	5	7	7

Tabel 5.7: Tabel Hasil Perhitungan $Mean$ Secara Manual

Urutan Masuk	Batas Kiri	Batas Kanan	Data Terurut	Mean	Pembulatan
1	2	4	$\{3, 7, 7\}$	$17/3 = 5,67$	5
2	0	4	$\{3, 5, 6, 7, 7\}$	$28/5$=$5,6$	5
3	1	2	$\{3, 5\}$	$8/2$=4	4

Tabel 5.8: Tabel Hasil Perhitungan $Median$ Secara Manual

Urutan Masuk	Batas Kiri	Batas Kanan	Data Terurut	Median
1	2	4	$\{3, 7, 7\}$	7
2	0	4	$\{3, 5, 6, 7, 7\}$	6
3	1	2	$\{3, 5\}$	$(3 + 5)/2 = 4$

Tabel 5.9: Tabel Hasil Perhitungan *Mode* Secara Manual

Urutan Masuk	Batas Kiri	Batas Kanan	Data Terurut	Mode	Maks. Mode
1	2	4	$\{3, 7, 7\}$	7	7
2	0	4	$\{3, 5, 6, 7, 7\}$	7	7
3	1	2	$\{3, 5\}$	3 dan 5	5

```
5
6 5 3 7 7
3
2 4
0 4
1 2
5 7 7
5 6 7
4 4 5
```

Gambar 5.48: Hasil Luaran Program Menggunakan Data Uji Coba

Dari hasil perhitungan oleh program dibandingkan dengan hasil perhitungan secara manual didapatkan hasil yang sama. selain itu juga dilakukan pengecekan terhadap urutan keluaran pada program seperti ditunjukkan pada Gambar 5.48. Hasilnya didapatkan urutan keluaran yang juga sesuai dengan urutan masukan. Sehingga dapat disimpulkan bahwa Algoritma Mo dan *segment tree* yang didesain dan diimplementasikan pada penyelesaian kasus uji coba dapat menyelesaikan permasalahan DCEPCA09-MMM.

Selanjutnya dilakukan juga uji coba kebenaran dengan mengirimkan kode sumber program ke dalam situs penilaian daring SPOJ. Permasalahan yang diselesaikan adalah DCEPCA09-MMM. Hasil uji kebenaran dan waktu eksekusi program pada saat pengumpulan kasus uji pada situs SPOJ ditunjukkan pada Gambar 5.49.

Dari Gambar 5.49, dapat dilihat bahwa kode sumber yang

ID	DATE	PROBLEM	RESULT	TIME	MEM	LANG
21968126	2018-07-10 12:16:02	MMM	accepted edit ideone it	0.08	17M	CPP

Gambar 5.49: Hasil Uji Coba pada Situs Penilaian SPOJ

telah dikirim mendapatkan hasil keluaran *Accepted*. Waktu yang dibutuhkan sistem untuk menyelesaikan soal DCEPCA09-MMM adalah $0,08$ detik dan memori yang dibutuhkan adalah 17 MB. Hal ini membuktikan bahwa implementasi yang dilakukan telah berhasil menyelesaikan permasalahan DCEPCA09-MMM dengan batasan-batasan yang telah ditetapkan oleh *problem setter* situs penilaian daring SPOJ.

Uji coba membandingkan hasil dengan menggunakan metode naif dilakukan dengan jumlah $query$ sebanyak 10 dan jumlah data sebanyak 500. Metode naif yang digunakan adalah dengan menduplikasi data yang berada dalam batasan $query$ pada $array$ baru untuk kemudian diurutkan. Untuk mendapatkan $mean$, $array$ diiterasi mulai dari indeks ke-0 untuk mendapatkan jumlah data kemudian dibagi dengan banyak data. Untuk mendapatkan $median$, akan diambil dari indeks ke tengah dari data array hasil pengurutan. Untuk mendapatkan $mode$, $array$ hasil pengurutan akan diiterasi untuk menemukan frekuensi maksimum dengan cara membandingkan data ke-i dengan data ke-i+1, jika data sama maka nilai frekuensi data tersebut akan bertambah kemudian dibandingkan dengan frekuensi $mode$. Jika frekuensi data lebih besar atau sama dengan frekuensi $mode$ maka data akan menjadi nilai $mode$ yang baru.

Hasil uji coba perbandingan dapat dilihat pada Tabel 5.10, Tabel 5.11 dan Tabel 5.12. Dari 10 $query$ pada percobaan perbandingan nilai $mean$, $median$ dan $mode$ untuk kedua metode menghasilkan nilai yang sama. Sehingga dapat disimpulkan bahwa metode yang diusulkan menggunakan Algoritma Mo dan struktur data *segment tree* dapat menyelesikan

Tabel 5.10: Hasil Uji Coba Perbandingan Nilai *Mean* Hasil Metode Naif dengan Metode Algoritma Mo Menggunakan *Segment Tree*

Query			Mean	
No	L	R	Naif	Mo
1	24	408	49434594	49434594
2	206	492	49924560	49924560
3	363	430	52178756	52178756
4	234	281	52290862	52290862
5	91	105	37536589	37536589
6	245	418	51386095	51386095
7	374	426	54696415	54696415
8	230	371	50223091	50223091
9	56	352	48666986	48666986
10	46	315	48043550	48043550

Tabel 5.11: Hasil Uji Coba Perbandingan Nilai *Median* Hasil Metode Naif dengan Metode Algoritma Mo Menggunakan *Segment Tree*

Query			Median	
No	L	R	Naif	Mo
1	24	408	49554702	49554702
2	206	492	49573357	49573357
3	363	430	49378795	49378795
4	234	281	55217657	55217657
5	91	105	34002233	34002233
6	245	418	50931900	50931900
7	374	426	50919008	50919008
8	230	371	50527367	50527367
9	56	352	49691936	49691936
10	46	315	49375594	49375594

Tabel 5.12: Hasil Uji Coba Perbandingan Nilai *Mode* Hasil Metode Naif dengan Metode Algoritma Mo Menggunakan *Segment Tree*

Query			Mode	
No	L	R	Naif	Mo
1	24	408	99579073	99579073
2	206	492	99455156	99455156
3	363	430	98666402	98666402
4	234	281	99455156	99455156
5	91	105	82460443	82460443
6	245	418	99455156	99455156
7	374	426	98153596	98153596
8	230	371	99455156	99455156
9	56	352	99579073	99579073
10	46	315	99579073	99579073

permasalahan $mean$, $median$ dan $mode$ dengan benar.

5.3 Uji Coba Kinerja

Kompleksitas waktu untuk menyelesaikan permasalahan DCEPCA09-MMM dengan mengimplementasi Algoritma Mo dan struktur data *Segment Tree* adalah $\mathcal{O}((N + M)\sqrt{N} \log N)$. Rincian dari kompleksitas yang sudah disebutkan:

- N adalah jumlah elemen yang ada di dalam baris angka.
- M adalah jumlah operasi kueri yang akan dikerjakan.
- $(N+M)\sqrt{N}$ adalah jumlah pergereran indeks $query$ untuk keseluruhan operasi.
- $\log N$ adalah kompleksitas untuk operasi $update$ pada *segment tree*.

Dari rincian tersebut dapat dilakukan 3 uji coba kinerja penyelesaian permasalahan dengan menggunakan pengaruh jumlah eleman data N dan jumlah $query$ Q. Jumlah keseluruhan

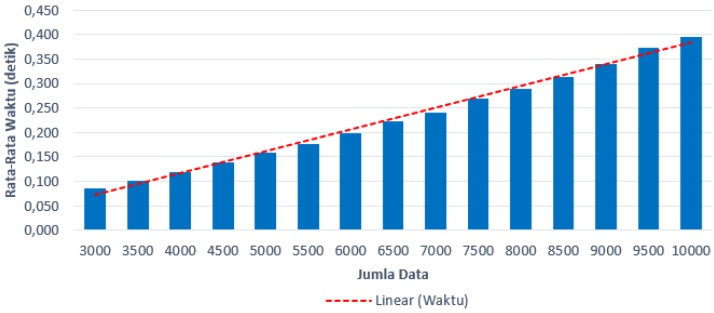
Tabel 5.13: Tabel Pengaruh Jumlah Data Terhadap Kinerja Waktu Pemrosesan

No	Jumlah Data	Rata-Rata Waktu (detik)
1	3000	0,085
2	3500	0,102
3	4000	0,119
4	4500	0,138
5	5000	0,158
6	5500	0,177
7	6000	0,199
8	6500	0,222
9	7000	0,241
10	7500	0,268
11	8000	0,289
12	8500	0,313
13	9000	0,340
14	9500	0,372
15	10000	0,396

operasi $update$ ditentukan urutan pengerjaan $query$, oleh karena itu akan dilakukan perbandingan kinerja dengan pengurutan dan tanpa pengurutan $query$.

5.3.1 Pengaruh Nilai Jumlah Banyak Data terhadap Efisiensi Waktu Pemrosesan

Pada uji coba ini terdapat 15 variasi jumlah data dengan nilai antara 3000 sampai 10000. Nilai selisih jumlah data untuk setiap uji coba sebesar 500 dan jumlah operasi $query$ dengan nilai konstan 10000. Banyak data ujicoba sebanyak 5 untuk masing-masing variasi. Setiap kasus uji coba dihasilkan dari fungsi Data *Generator* yang sudah didesain pada subbab



Gambar 5.50: Grafik Pengaruh Jumlah Data Terhadap Kinerja Waktu Pemrosesan

3.1.2.10. Untuk setiap waktu eksekusi akan tercatat dalam satuan detik dan diambil nilai rata-rata dari hasil uji coba. Hasil uji coba dari percobaan dapat dilihat pada Tabel 5.13 dan diilustrasikan dalam grafik pada Gambar 5.50.

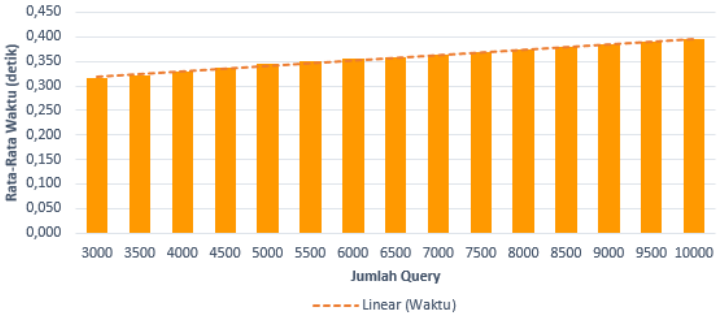
Dari hasil uji coba tersebut didapatkan pertumbuhan fungsi waktu linear terhadap pertambahan jumlah data. Hal ini disebabkan karena semakin banyak jumlah data maka akan semakin banyak rentang data untuk dilakukan $query$. Sehingga dengan jumlah $query$ yang sama didapatkan pertumbuhan fungsi yang linear dengan rata-rata peningkatan waktu sebesar 0,022 detik.

5.3.2 Pengaruh Nilai Jumlah *Query* terhadap Efisiensi Waktu Pemrosesan

Pada uji coba ini terdapat 15 variasi jumlah $query$ dengan nilai antara 400 sampai 6000 operasi *query*. Nilai selisih jumlah *query* untuk setiap uji coba sebesar 500 dan jumlah data dengan nilai konstan 10000. Banyak data ujicoba sebanyak 5 untuk masing-masing variasi. Setiap kasus uji coba dihasilkan dari fungsi Data *Generator* yang sudah didesain pada subbab

Tabel 5.14: Tabel Pengaruh Jumlah Query Terhadap Kinerja Waktu Pemrosesan

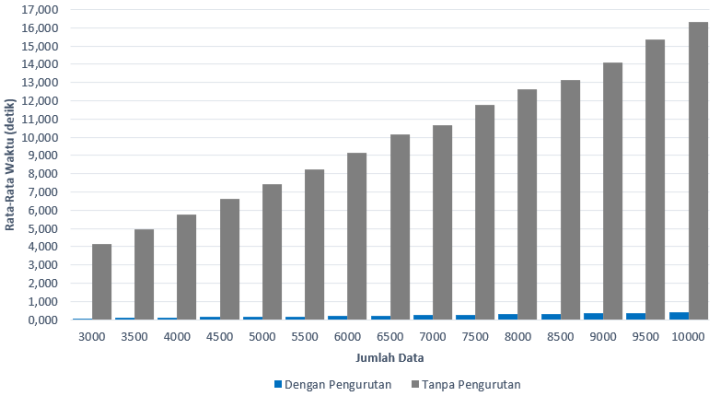
No	Jumlah *Query*	Rata-Rata Waktu (detik)
1	3000	0,315
2	3500	0,322
3	4000	0,329
4	4500	0,337
5	5000	0,345
6	5500	0,350
7	6000	0,355
8	6500	0,358
9	7000	0,364
10	7500	0,370
11	8000	0,375
12	8500	0,379
13	9000	0,384
14	9500	0,389
15	10000	0,395



Gambar 5.51: Grafik Pengaruh Jumlah $Query$ Terhadap Kinerja Waktu Pemrosesan

3.1.2.10. Untuk setiap waktu eksekusi akan tercatat dalam satuan detik dan diambil nilai rata-rata dari hasil uji coba. Hasil uji coba dari percobaan dapat dilihat pada Tabel 5.14 dan diilustrasikan dalam grafik pada Gambar 5.51.

Dari hasil uji coba tersebut didapatkan pertumbuhan fungsi waktu linear terhadap pertambahan jumlah $query$. Dengan peningkatan yang landai jika dibandingkan dengan hasil uji coba pada pengaruh jumlah data. Hal ini disebabkan karena semakin banyak jumlah $query$ tidak membuat operasi pergeseran data bertambah secara signifikan. Dengan jumlah data yang sama maka rentang $query$ tidak bertambah. Dengan jumlah $query$ semakin banyak akan semakin banyak operasi yang mengalami $overlaping$. Sehingga dengan jumlah data yang sama yaitu 10000 data, didapatkan pertumbuhan fungsi linear dengan peningkatan waktu rata-rata 0,006 detik.



Gambar 5.52: Grafik Perbandingan Hasil Waktu Menggunakan Pengurutan $Query$ dan Tanpa Pengurutan $Query$ pada Kasus Perubahan Jumlah Data

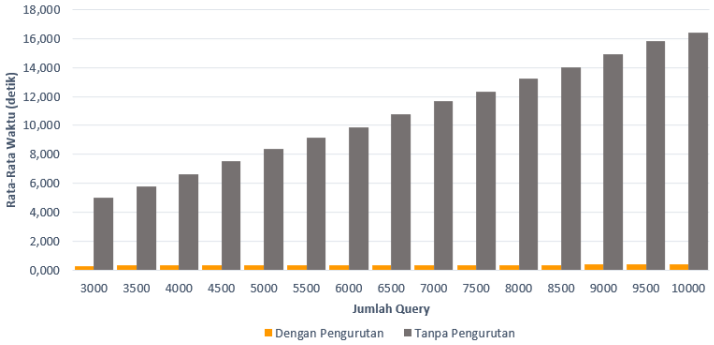
5.3.3 Pengaruh Pengurutan *Query* pada Algortima Mo terhadap Efisiensi Waktu Pemrosesan

Pada uji coba ini akan dilakukan percobaan untuk menguji pengaruh pengurutan $query$ pada Algortima Mo terhadap performa program. Uji coba dilakukan dengan dua skenario: peningkatan jumlah data dan peningkatan jumlah $query$. Pada skenario peningkatan jumlah data digunakan data yang sama pada ujicoba sebelumnya pada subbab 5.3.1 dibandingkan dengan waktu pengerjaan tanpa menggunakan pengurutan $query$. Sedangkan pada skenario peningkatan jumlah $query$ digunakan data ujicoba seperti pada subbab 5.3.2. Untuk setiap waktu eksekusi akan tercatat dalam satuan detik. Hasil uji coba dari percobaan dengan peningkatan jumlah data dapat dilihat pada Tabel 5.15. Untuk hasil uji coba dari percobaan dengan peningkatan jumlah $query$ dapat dilihat pada Tabel 5.16. Ilustrasi hasil perbandingan dapat dilihat pada Gambar 5.52 dan Gambar 5.53.

Dari hasil uji coba peningkatan jumlah data didapatkan

Tabel 5.15: Tabel Perbandingan Kinerja Waktu Pemrosesan dengan dan Tanpa Pengurutan $Query$ pada Kasus Perubahan Jumlah Data

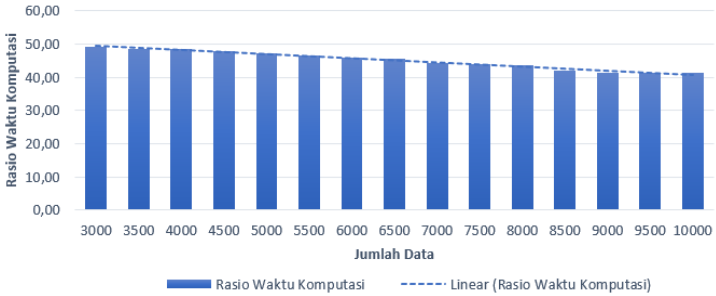
No	Jumlah Data	Dengan Pengurutan (detik)	Tanpa Pengurutan (detik)	Rasio Peningkatan
1	3000	0,085	4,165	49,24
2	3500	0,102	4,967	48,70
3	4000	0,119	5,777	48,63
4	4500	0,138	6,642	47,99
5	5000	0,158	7,424	47,10
6	5500	0,177	8,227	46,48
7	6000	0,199	9,168	45,98
8	6500	0,222	10,151	45,77
9	7000	0,241	10,656	44,25
10	7500	0,268	11,779	43,95
11	8000	0,289	12,619	43,69
12	8500	0,313	13,144	42,02
13	9000	0,340	14,097	41,41
14	9500	0,372	15,373	41,28
15	10000	0,396	16,339	41,28



Gambar 5.53: Grafik Perbandingan Hasil Waktu Menggunakan Pengurutan $Query$ dan Tanpa Pengurutan $Query$ pada Kasus Perubahan Jumlah $Query$

Tabel 5.16: Tabel Perbandingan Kinerja Waktu Pemrosesan dengan dan Tanpa Pengurutan $Query$ pada Kasus Perubahan Jumlah $Query$

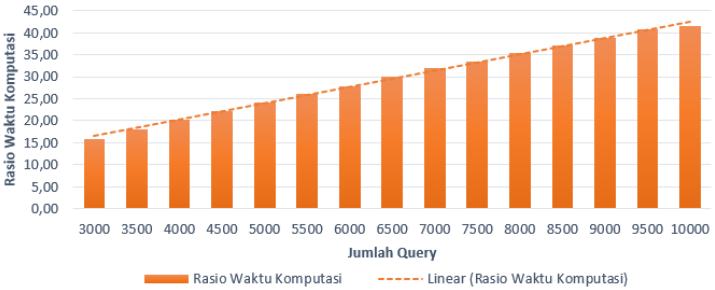
No	Jumlah *Query*	Dengan Pengurutan (detik)	Tanpa Pengurutan (detik)	Rasio Peningkatan
1	3000	0,315	4,983	15,81
2	3500	0,322	5,776	17,96
3	4000	0,329	6,634	20,16
4	4500	0,337	7,503	22,29
5	5000	0,345	8,370	24,26
6	5500	0,350	9,155	26,14
7	6000	0,355	9,841	27,74
8	6500	0,358	10,750	30,01
9	7000	0,364	11,659	32,05
10	7500	0,370	12,347	33,39
11	8000	0,375	13,242	35,35
12	8500	0,379	14,022	37,00
13	9000	0,384	14,889	38,81
14	9500	0,389	15,819	40,65
15	10000	0,395	16,390	41,49



Gambar 5.54: Grafik Rasio Waktu Menggunakan Pengurutan $Query$ dan Tanpa Pengurutan $Query$ pada Kasus Perubahan Jumlah Data

peningkatan waktu yang cukup besar. Sebagai contoh pada uji coba dengan jumlah data 3000 dan jumlah $query$ 10000 didapatkan hasil 0,085 detik untuk program dengan pengurutan dan 4,165 detik tanpa pengurutan. Contoh lain pada percobaan dengan jumlah data 10000 dan jumlah $query$ 10000 didapatkan hasil 0,396 detik untuk program dengan pengurutan dan 16,339 detik tanpa pengurutan. Rasio peningkatan waktu mengalami tren penurunan jika jumlah data semakin besar dengan jumlah $query$ tetap, seperti ditunjukkan oleh Gambar 5.54. Hal ini disebabkan karena bertambahnya rentang data yang mengakibatkan berkurangnya jumlah operasi yang *overlapping* walaupun $query$ sudah diurutkan.

Pada hasil uji coba peningkatan jumlah $query$ didapatkan peningkatan waktu yang signifikan. Sebagai contoh pada uji coba dengan jumlah data 10000 dan jumlah $query$ 3000 didapatkan hasil 0,315 detik untuk program dengan pengurutan dan 4,983 detik tanpa pengurutan. Contoh lain pada percobaan dengan jumlah data 10000 dan jumlah $query$ 10000 didapatkan hasil 0,395 detik untuk program dengan pengurutan dan 16,390 detik tanpa pengurutan. Rasio peningkatan waktu pada kasus penambahan jumlah $query$ adalah linear, seperti ditunjukkan



Gambar 5.55: Grafik Rasio Waktu Menggunakan Pengurutan $Query$ dan Tanpa Pengurutan $Query$ pada Kasus Perubahan Jumlah $Query$

oleh Gambar 5.55. Hal ini disebabkan karena bertambahnya jumlah operasi yang *overlapping* ketika $query$ sudah diurutkan. Dari hasil uji coba dapat disimpulkan bahwa pengurutan $query$ berhasil mengoptimasi waktu pemrosesan.

BAB VI

KESIMPULAN DAN SARAN

Pada bab ini dijelaskan mengenai kesimpulan dari hasil uji coba yang telah dilakukan dan saran mengenai hal-hal yang masih dapat dikembangkan dari Tugas Akhir ini.

6.1 Kesimpulan

Dari hasil uji coba yang telah dilakukan terhadap implementasi penyelesaian permasalahan DCEPCA09-MMM dapat diambil kesimpulan sebagai berikut:

1. Implementasi Algoritma Mo dengan menggunakan struktur data *segment tree* dapat menyelesaikan permasalahan SPOJ klasik DCEPCA09-MMM dengan benar.
2. Rancangan struktur data *segment tree* dengan menggunakan nilai agregasi jumlah data, banyak data dan frekuensi maksimum pada Algoritma Mo untuk mendapatkan nilai *mean, median* dan *mode* dapat menyelesaikan permasalahan SPOJ klasik DCEPCA09-MMM.
3. Kompleksitas algoritma $\mathcal{O}((M + N)\sqrt{N}logN)$ hasil implementasi Algoritma Mo dan *segment tree* dapat menyelesaikan soal dan memenuhi batasan waktu pada permasalahan SPOJ klasik DCEPCA09-MMM.
4. Waktu yang dibutuhkan program untuk menyelesaikan permasalahan SPOJ klasik DCEPCA09-MMM adalah $0,08$ detik dengan penggunaan memori sebesar 17 MB.
5. Peningkatan jumlah data dengan jumlah *query* tetap sebesar 10000 pada implementasi algoritma penyelesaian permasalahan SPOJ klasik DCEPCA09-MMM menyebabkan pertumbuhan waktu komputasi secara linear dengan rata-rata peningkatan sebesar 0,022 detik.
6. Peningkatan jumlah *query* dengan jumlah data tetap sebesar 10000 pada implementasi algoritma penyelesaian

permasalahan SPOJ klasik DCEPCA09-MMM menyebabkan pertumbuhan waktu komputasi secara linear dengan rata-rata peningkatan sebesar 0,006 detik.

7. Pengurutan *query* pada Algoritma Mo berhasil meningkatkan performa komputasi secara signifikan. Pada kasus uji coba peningkatan jumlah data rata-rata rasio peningkatan performa waktu komputasi sebesar 45,9 kali dan mengalami tren penurunan seiring pertumbuhan jumlah data. Sedangkan pada kasus uji coba peningkatan jumlah *query*, performa waktu komputasi meningkat secara linear seiring pertumbuhan jumlah *query*.

6.2 Saran

Pada Tugas Akhir kali ini tentunya terdapat kekurangan serta nilai-nilai yang dapat penulis ambil. Berikut adalah saran-saran yang dapat diambil melalui Tugas Akhir ini:

1. Algoritma penyelesaian DCEPCA09-MMM masih dapat dioptimasi untuk mendapatkan waktu penyelesaian yang lebih baik.
2. Pendekatan menggunakan Algoritma Mo dengan menggunakan struktur data yang lain dapat dicoba diaplikasikan untuk menyelesaikan permasalahan *range query* pada DCEPCA09-MMM.

DAFTAR PUSTAKA

[1] M. I. Hasan, *POKOK-POKOK MATERI STATISTIK 1*, 2nd ed. Jakarta: Bumi Aksara, 2013.

[2] "How exactly is the square root decomposition of queries (also sometimes referred to as Mo's Algorithm), used for offline processing of queries? - Quora," 14 April 2018. [Daring]. Tersedia pada: https://www.quora.com/How-exactly-is-the-square-root-decomposition-of-queries-also-sometimes-referred-to-as-Mos-Algorithm-used-for-offline-processing-of-queries. [Diakses: 14 April 2018].

[3] "Mo's algorithm - Mike Koltsov," 21 April 2018. [Daring]. Tersedia pada: https://www.hackerearth.com/fr/practice/notes/mos-algorithm/. [Diakses: 21 April 2018].

[4] "Mo's Algorithm," 21 April 2018. [Daring]. Tersedia pada: http://codeforces.com/blog/entry/7383. [Diakses: 21 April 2018].

[5] "SPOJ.com - Problem DCEPCA09," 15 Maret 2018. [Daring]. Tersedia pada: http://www.spoj.com/problems/DCEPCA09/. [Diakses: 15 Maret 2018].

[6] anudeep2011, "MO's Algorithm (Query square root decomposition)," Dec. 2014, 02 Mei 2018. [Daring]. Tersedia pada: https://blog.anudeep2011.com/mos-algorithm/. [Diakses: 02 Mei 2018].

[7] "Segment Trees Tutorials & Notes | Data Structures," 20 Mei 2018. [Daring]. Tersedia pada: https://www.hackerearth.com/fr/practice/data-structures/advanced-data-structures/segment-trees/tutorial/. [Diakses: 20 Mei 2018].

[8] "Segment Tree | Set 1 (Sum of given range)," Jan. 2013, 20 Mei 2018.

[9] W. Gozali, “Kupas Kode: Tentang Segment Tree: Variasi Nilai Agregat,” Jul. 2013.

[10] “Efficiently design Insert, Delete and Median queries on a set,” Jul. 2016, 19 Mei 2018. [Daring]. Tersedia pada: https://www.geeksforgeeks.org/efficiently-design-insert-delete-median-queries-set/. [Diakses: 19 Mei 2018].

LAMPIRAN A

KODE SUMBER PENYELESAIAN DCEPCA09-MMM

```
#include <stdio.h>
#include <iostream>
#include <map>
#include <utility>
#include <algorithm>
#include <cmath>
#define left first
#define right second
using namespace std;

int data[10005];
int sorted[10005];
int val[10005];
int block,isi;
int n, q, jumlah;
map<int,int> comp;
pair<int,int> query[10005];

struct valMO{
   long long BlockNumber;
   long long RValue;
   long long inputOrder;
};

struct answer{
   long long mean;
   long long median;
   long long mode;
};

struct seg_tree{
   int indeksData;
   int frekuensiData;
   int banyakData;
   long long jumlahData;
};
```

Kode Sumber A.1: Kode Sumber Lengap Penyelesaian DCEPCA09-MMM Bagian 1

```
37 answer answers[10005];
38 valMO valMOs[10005];
39 seg_tree seg_trees[40005];
40
41 inline void scan_uint(int* p) {
42     static char c;
43     while ((c = getchar()) < '0');
44     for (*p = c-'0'; (c = getchar()) >= '0'; *p = *p
       *10+c-'0');
45 }
46
47 bool compare(valMO x, valMO y)
48 {
49    if (x.BlockNumber != y.BlockNumber)
50       return x.BlockNumber < y.BlockNumber;
51    if (x.RValue != y.RValue)
52       return x.RValue < y.RValue;
53    return x.inputOrder < y.inputOrder;
54 }
55
56 int cari(int idx,int l,int r,int ke){
57     if(l==r){
58         return l;
59     }
60
61     if(seg_trees[2*idx].banyakData >= ke){
62         return cari(2*idx,l,(l+r)/2,ke);
63     }else{
64         return cari(2*idx+1,(l+r)/2+1,r,ke-seg_trees[2*
       idx].banyakData);
65     }
66 }
67
68 void update(int idx,int l,int r,int ini,int value){
69     if(ini<l || ini>r) return;
70
71     if(l==r){
```

Kode Sumber A.2: Kode Sumber Lengap Penyelesaian DCEPCA09-MMM Bagian 2

```
72          seg_trees[idx].indeksData = l;
73          seg_trees[idx].frekuensiData += value;
74          seg_trees[idx].banyakData += value;
75          seg_trees[idx].jumlahData += (val[l]*value);
76          return;
77      }
78
79      update(2*idx,l,(l+r)/2,ini,value);
80      update(2*idx+1,(l+r)/2+1,r,ini,value);
81
82      if(seg_trees[2*idx].frekuensiData > seg_trees[2*idx
        +1].frekuensiData){
83        seg_trees[idx] = seg_trees[2*idx];
84      }else{
85        seg_trees[idx] = seg_trees[2*idx+1];
86      }
87      seg_trees[idx].banyakData = seg_trees[2*idx].
        banyakData + seg_trees[2*idx+1].banyakData;
88      seg_trees[idx].jumlahData = seg_trees[2*idx].
        jumlahData + seg_trees[2*idx+1].jumlahData;
89      return;
90  }
91
92  long long findMedian(){
93     int banyakbilangan = seg_trees[1].banyakData;
94     if(banyakbilangan % 2==1){
95        return val[cari(1,0,jumlah-1,banyakbilangan/2+1)
        ];
96     }else{
97        int kiri = cari(1,0,jumlah-1,banyakbilangan/2);
98        int kanan = cari(1,0,jumlah-1,banyakbilangan/2 +
        1);
99        return (val[kiri] + val[kanan])/2;
100    }
101 }
102
103 void solveMMM(){
```

Kode Sumber A.3: Kode Sumber Lengap Penyelesaian DCEPCA09-MMM Bagian 3

```
104     int currentL = 0, currentR = -1;
105
106     for(int tmp=0 ; tmp<q ; tmp++){
107        int now = valMOs[tmp].inputOrder;
108
109        while(currentR < query[now].right){
110           currentR++;
111           update(1,0,jumlah-1,data[currentR],1);
112        }
113        while(currentR > query[now].right){
114           update(1,0,jumlah-1,data[currentR],-1);
115           currentR--;
116        }
117        while(currentL < query[now].left){
118           update(1,0,jumlah-1,data[currentL],-1);
119           currentL++;
120        }
121        while(currentL > query[now].left){
122           currentL--;
123           update(1,0,jumlah-1,data[currentL],1);
124        }
125
126        answers[now].mean = seg_trees[1].jumlahData / (
       query[now].right - query[now].left + 1);
127
128        answers[now].median = findMedian();
129
130        answers[now].mode = val[seg_trees[1].indeksData];
131     }
132  }
133
134  int main(){
135      scan_uint(&n);
136
137      for(int tmp=0;tmp<n;tmp++){
138          scan_uint(&data[tmp]);
139          sorted[tmp] = data[tmp];
```

Kode Sumber A.4: Kode Sumber Lengap Penyelesaian DCEPCA09-MMM Bagian 4

```
140     }
141
142     sort(sorted,sorted+n);
143
144     val[0] = sorted[0];
145     comp[sorted[0]] = 0;
146     jumlah = 1;
147
148     for(int tmp=1;tmp<n;tmp++){
149         if(sorted[tmp]!=sorted[tmp-1]){
150             val[jumlah] = sorted[tmp];
151             comp[sorted[tmp]] = jumlah;
152             jumlah++;
153         }
154     }
155
156     for(int tmp=0;tmp<n;tmp++){
157         data[tmp] = comp[data[tmp]];
158     }
159
160     block = sqrt(n);
161     if(block*block != n) block++;
162     isi = n/block;
163     if(n%block != 0) isi++;
164
165     scan_uint(&q);
166
167     for(int tmp=0;tmp<q;tmp++){
168         scan_uint(&query[tmp].left);
169         scan_uint(&query[tmp].right);
170
171         valMOs[tmp].BlockNumber = query[tmp].left / isi
    ;
172         valMOs[tmp].RValue = query[tmp].right;
173         valMOs[tmp].inputOrder = tmp;
174     }
175
176     sort(valMOs,valMOs+q,compare);
```

Kode Sumber A.5: Kode Sumber Lengap Penyelesaian DCEPCA09-MMM Bagian 5

```
177     solveMMM();
178
179     for(int tmp=0;tmp<q;tmp++){
180         cout<<answers[tmp].mean<<" "<<answers[tmp].
        median<<" "<<answers[tmp].mode<<"\n";
181     }
182 }
```

Kode Sumber A.6: Kode Sumber Lengap Penyelesaian DCEPCA09-MMM Bagian 6

LAMPIRAN B

DATA UJI COBA KEBENARAN PADA PERBANDINGAN METODE NAIF

83667300 57286354 31691771 61400619 48432169 7752119 47349229 98285867 24530753 87871798 3169628 30769973 44689161 62603292 40012815 2746950 37602274 3793356 62257371 53630153 48788218 16933253 94471907 45469006 86294317 89851360 78763379 87851764 87734575 5466745 12231586 59074290 76791477 37475962 63223535 40041397 15735418 61739073 91667243 38687407 30112046 37516497 71547605 82259853 36734132 57834890 57455810 39194544 71282632 67143306 22480287 3654318 80616529 44109453 31508305 66973535 78219377 1544683 9916957 1522701 73166720 27860954 5101242 71680916 27669873 18723239 52874373 98721724 72546619 88014423 5827488 99099906 50381340 92653721 63165114 13020142 81446244 82679145 68454751 59216008 62154007 26025316 30850415 30653808 95776546 71423853 17244787 71387359 69842655 54702957 27411882 3271763 56263461 20367145 12653383 32698306 37907171 18341007 82460443 61508435 38256377 12533131 16747637 65829997 70208346 34002233 64605042 39947395 98655522 1629334 19303642 35851758 27047988 28145588 13369881 99579073 81595001 8399210 56087306 33909613 3530587 22460493 4835388 30275823 85636526 18915159 39375445 13490143 50810529 30716713 65913453 80644259 26501453 85749477 26794209 51743334 81173343 84156736 98099260 4118296 95538419 71801538 5097455 33216604 5087337 91332247 82518669 67254343 17617375 32587199 29871630 47946605 22356245 25458706 22865902 33736277 90805910 32604508 69084907 98119943 78167781 49554702 51326828 78101036 84873247 4783697 62094092 53066835 73070307 26527896 49922920 51484293 98881756 68382527 85173841 93792906 73602862 17079173 36858822 65393077 15545944 23347661 19142784 8284540 26376809

53909174 59102524 8555321 8351479 17873320 29900069
85554449 44745719 73267372 90696041 24112353 77669550
17918360 31805116 10001845 11552522 22101451 70935549
98521368 76495667 49196487 79245004 89049144 69659702
14960609 12108535 66882932 19170769 16146155 34586459
57749950 70667989 65880213 75741002 26965966 7694179
40270868 61429826 23022957 41330818 1168810 65652710
49573357 62996753 22359941 39610367 46114385 6367061
78170980 60485549 44432087 50109942 16887335 59318633
13927852 42943447 92927429 67886450 40921906 59168631
65010103 55429407 65081671 27316283 49691936 73677101
97157531 60885364 30050468 94537452 13796702 58883577
18484608 93790864 55005908 37641746 7785544 23254078
30909614 14926117 78867554 54308635 64574826 21076458
96134309 80804843 27734991 69101347 99455156 94359901
27538624 3274961 1592963 72704467 40128168 60355358
95623505 95762038 72060431 41800343 71962788 7984789
11190076 81862188 96194011 13297828 80273093 16746520
26026994 4512319 87571437 11445497 33991992 47934483
98049566 12335303 67480609 76804337 32037392 24836632
66808051 53394520 19896357 50944792 97562296 62244184
46837301 10300970 59530083 29434237 14586668 85692594
98333754 28715426 83073863 92756722 27640662 86689829
10065031 12351888 13142229 14394238 7856337 29469920
53090669 79281853 51747705 68431358 56810274 67650639
44740159 42516606 92564701 80449596 39938774 54394966
96936468 9099400 40175684 10557295 76654853 19333913
55981372 58652115 93995740 40662773 56960623 85945188
36710436 28753552 47498451 44317527 73257393 16048320
18136444 30360765 73307895 62702554 14526390 44159641
69426011 4712502 10688204 65173575 98666402 36238535
86314800 47632933 4854549 49187066 98026025 45851555
79785386 65550541 50919008 57128461 42672013 18396089

21310376 16111872 37466058 70979781 37173343 66942193
92635756 36917542 86391561 29044251 22745212 46563191
91084274 49013423 97472769 13031854 45463573 84163353
98153596 47925040 18755756 91023541 72466212 63416530
75882900 12033629 51275210 43116019 81628956 49570524
75298318 73310075 30498643 74734309 59444170 5726223
95257450 9057699 61577781 50322647 20962454 69271816
91686311 24487684 29733970 10130090 82577815 44410001
35689427 63528850 63432546 53345411 15331446 74795044
58782031 18190524 17704024 15627795 40851934 47173854
91866370 11015689 66601588 75009413 46969302 26134580
24834692 63201588 17913952 49748111 74830270 73898332
70494141 36699531 46291061 19814495 17717352 11295545
65801026 89131896 12168718 37036083 89149675 27207099
96532666 32303639 97388196 35888642 73553707 89672049
87099546 97278210 85155082 10480935 53424300 62629997
5995739 77562585 93819925 30256912 12578586 21799563
76191601 54565403 15661875 50526606 36792939 32022104
13086120 12555704 97084714 942211 93791088 23170618
88868722 4982051 56607063

(Halaman ini sengaja dikosongkan)

LAMPIRAN C

HASIL UJI COBA KINERJA

Tabel C.1: Hasil Uji Coba Waktu Komputasi pada Kasus Perubahan Jumlah Data dengan Jumlah *Query* Tetap

No	Jumlah Data	Data 1	Data 2	Data 3	Data 4	Data 5	Rata-Rata (detik)
1	3000	0,084	0,085	0,085	0,084	0,085	0,085
2	3500	0,102	0,103	0,102	0,102	0,101	0,102
3	4000	0,119	0,118	0,118	0,119	0,120	0,119
4	4500	0,139	0,138	0,138	0,139	0,138	0,138
5	5000	0,157	0,158	0,157	0,158	0,158	0,158
6	5500	0,177	0,176	0,178	0,177	0,177	0,177
7	6000	0,199	0,200	0,199	0,200	0,199	0,199
8	6500	0,222	0,221	0,222	0,223	0,221	0,222
9	7000	0,240	0,241	0,242	0,240	0,241	0,241
10	7500	0,269	0,267	0,267	0,268	0,269	0,268
11	8000	0,289	0,290	0,288	0,289	0,288	0,289
12	8500	0,312	0,313	0,314	0,312	0,313	0,313
13	9000	0,340	0,341	0,341	0,340	0,340	0,340
14	9500	0,372	0,372	0,373	0,373	0,372	0,372
15	10000	0,395	0,396	0,396	0,396	0,396	0,396

Tabel C.2: Hasil Uji Coba Waktu Komputasi pada Kasus Perubahan Jumlah *Query* dengan Jumlah Data Tetap

No	Jumlah Query	Data 1	Data 2	Data 3	Data 4	Data 5	Rata-Rata
1	3000	0,313	0,313	0,317	0,317	0,316	0,315
2	3500	0,321	0,321	0,322	0,322	0,322	0,322
3	4000	0,329	0,329	0,328	0,331	0,328	0,329
4	4500	0,336	0,339	0,336	0,336	0,336	0,337
5	5000	0,345	0,344	0,344	0,346	0,346	0,345
6	5500	0,350	0,349	0,351	0,350	0,351	0,350
7	6000	0,354	0,355	0,355	0,355	0,355	0,355
8	6500	0,358	0,359	0,358	0,358	0,358	0,358
9	7000	0,366	0,364	0,363	0,363	0,363	0,364
10	7500	0,370	0,370	0,368	0,371	0,370	0,370
11	8000	0,374	0,374	0,375	0,375	0,375	0,375
12	8500	0,378	0,379	0,379	0,379	0,380	0,379
13	9000	0,383	0,384	0,384	0,384	0,383	0,384
14	9500	0,389	0,390	0,388	0,389	0,390	0,389
15	10000	0,396	0,395	0,394	0,395	0,395	0,395

Tabel C.3: Hasil Uji Coba Waktu Komputasi Tanpa Pengurutan Query pada Kasus Perubahan Jumlah Data dengan Jumlah *Query* Tetap

No	Jumlah Data	Data 1	Data 2	Data 3	Data 4	Data 5	Rata-Rata
1	3000	4,150	4,173	4,169	4,174	4,161	4,165
2	3500	4,950	4,957	4,955	4,990	4,984	4,967
3	4000	5,816	5,777	5,704	5,837	5,753	5,777
4	4500	6,560	6,740	6,583	6,653	6,675	6,642
5	5000	7,393	7,419	7,402	7,558	7,346	7,424
6	5500	8,193	8,171	8,335	8,220	8,216	8,227
7	6000	9,163	9,266	9,118	9,038	9,253	9,168
8	6500	10,282	10,078	10,060	10,219	10,118	10,151
9	7000	10,706	10,643	10,640	10,596	10,694	10,656
10	7500	11,820	11,824	11,798	11,734	11,718	11,779
11	8000	12,685	12,653	12,584	12,589	12,583	12,619
12	8500	13,090	13,226	13,144	13,145	13,117	13,144
13	9000	14,017	13,998	14,215	14,152	14,103	14,097
14	9500	15,463	15,185	15,351	15,472	15,393	15,373
15	10000	16,340	16,338	16,337	16,342	16,340	16,339

Tabel C.4: Hasil Uji Coba Waktu Komputasi Tanpa Pengurutan Query pada Kasus Perubahan Jumlah *Query* dengan Jumlah Data Tetap

No	Jumlah Query	Data 1	Data 2	Data 3	Data 4	Data 5	Rata-Rata
1	3000	4,998	4,906	5,030	4,936	5,044	4,983
2	3500	5,776	5,767	5,777	5,860	5,698	5,776
3	4000	6,693	6,585	6,484	6,744	6,664	6,634
4	4500	7,413	7,533	7,474	7,545	7,551	7,503
5	5000	8,360	8,473	8,292	8,219	8,504	8,370
6	5500	9,224	9,147	9,201	9,049	9,153	9,155
7	6000	9,779	9,910	9,666	9,937	9,911	9,841
8	6500	10,609	10,831	10,731	10,858	10,722	10,750
9	7000	11,749	11,741	11,535	11,581	11,687	11,659
10	7500	12,356	12,277	12,371	12,283	12,450	12,347
11	8000	13,317	13,208	13,299	13,209	13,177	13,242
12	8500	14,008	14,130	13,953	13,953	14,064	14,022
13	9000	14,797	14,873	14,959	15,057	14,758	14,889
14	9500	15,794	16,040	15,897	15,587	15,778	15,819
15	10000	16,382	16,388	16,402	16,393	16,387	16,390

BIODATA PENULIS

Miftakhul Akhyar, anak pertama dari 2 bersaudara lahir di Gresik tanggal 28 April 1995. Penulis telah menempuh pendidikan formal MI Tarbiyatul Aulad (2003-2008), SMP Negeri 1 Cerme (2008-2010), dan SMA Negeri 1 Cerme (2010-2013). Penulis pernah menempuh kuliah program sarjana selama satu tahun di Universitas Brawijaya (UB) jurusan Informatika, belum lulus dari UB penulis melanjutkan studi kuliah program sarjana di Jurusan Informatika ITS.

Selama kuliah di Informatika ITS, penulis mengambil bidang minat Algoritma Pemrograman (AP). Selama menempuh perkuliahan penulis juga aktif di kegiatan organisasi diantaranya menjadi staff Departemen Kesejahteraan Mahasiswa HMTC ITS 2015/2016, staff Departemen Riset dan Pengembangan Teknologi BEM FTIf ITS 2015/2016, ketua Departemen Riset dan Pengembangan Teknologi BEM FTIf ITS 2016/2017. Selain aktif berorganisasi penulis juga aktif mengembangkan diri melalui proyek-proyek pengembangan perangkat lunak diantaranya : Sistem POA Polres Surabaya, Ujian Online Sekolah Surabaya 2017, PPDB SMA/SMK Jawa Timur 2017, PPDB SMA/SMK Jawa Timur 2018, maintenance Web Simpati KDEI Taipei, Monitoring Dewatering System PT Pama Persada, e-recruitment PT Badak NGL dan pengembangan $platform$ investasi $p2p$ danaprospera. Penulis dapat dihubungi melalui surel di `mifta28.official@gmail.com`.