ITS
Institut
Teknologi
Sepuluh Nopember

TUGAS AKHIR - KI141502

PENERAPAN TEKNIK DEKOMPOSISI SQUARE ROOT DAN ALGORITMA MO'S PADA RANCANGAN ALGORITMA STUDI KASUS: SPOJ KLASIK COUNTING DIFF-PAIRS

ABDUL MAJID HASANI
NRP 5114100097

Dosen Pembimbing 1
Rully Soelaiman, S.Kom., M.Kom.

Dosen Pembimbing 2
Fajar Baskoro, S.Kom., M.T.

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

Halaman ini sengaja dikosongkan

TUGAS AKHIR - KI141502

PENERAPAN TEKNIK DEKOMPOSISI SQUARE ROOT DAN ALGORITMA MO'S PADA RANCANGAN ALGORITMA STUDI KASUS: SPOJ KLASIK COUNTING DIFF-PAIRS

ABDUL MAJID HASANI
NRP 5114100097

Dosen Pembimbing 1
Rully Soelaiman, S.Kom., M.Kom.

Dosen Pembimbing 2
Fajar Baskoro, S.Kom., M.T.

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

Halaman ini sengaja dikosongkan

UNDERGRADUATE THESES - KI141502

APPLICATION OF SQUARE ROOT DECOMPOSITION AND MO'S ALGORITHM ON ALGORITHM DESIGN OF CASE STUDY: CLASSIC SPOJ COUNTING DIFF-PAIRS

ABDUL MAJID HASANI
NRP 5114100097

Supervisor 1
Rully Soelaiman, S.Kom., M.Kom.

Supervisor 2
Fajar Baskoro, S.Kom., M.T.

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

Halaman ini sengaja dikosongkan

PENERAPAN TEKNIK DEKOMPOSISI SQUARE ROOT DAN ALGORITMA MO'S PADA RANCANGAN ALGORITMA STUDI KASUS: SPOJ KLASIK COUNTING DIFF-PAIRS

TUGAS AKHIR

Diajukan Guna Memenuhi Salah Satu Syarat
Memperoleh Gelar Sarjana Komputer
pada
Bidang Studi Algoritma Pemrograman
Program Studi S-1 Departemen Informatika
Fakultas Teknologi Informasi dan Komunikasi
Institut Teknologi Sepuluh Nopember

Oleh:
Abdul Majid Hasani
NRP: 5114100097

Disetujui oleh Dosen Pembimbing Tugas Akhir:

Rully Soelaiman, S.Kom., M.Kom. .....

NIP: 197002131994021001 (Pembimbing 1)

Fajar Baskoro, S.Kom., M.T. .....

NIP: 197404031999031002 (Pembimbing 2)

SURABAYA
JANUARI 2018

vii

Halaman ini sengaja dikosongkan

PENERAPAN TEKNIK DEKOMPOSISI SQUARE ROOT DAN ALGORITMA MO'S PADA RANCANGAN ALGORITMA STUDI KASUS: SPOJ KLASIK COUNTING DIFF-PAIRS

Nama	: ABDUL MAJID HASANI
NRP	: 5114100097
Departemen	: Informatika FTIK-ITS
Pembimbing I	: Rully Soelaiman, S.Kom., M.Kom.
Pembimbing II	: Fajar Baskoro, S.Kom., M.T.

Abstrak

Diberikan sebuah sekuen bilangan A dengan jumlah N, M baris kueri, dan selisih mutlak bernilai k. Terdapat operasi kueri untuk mencari jumlah pasangan angka dalam jarak tertentu di sekuen bilangan A yang memiliki selisih mutlak sama dengan atau lebih dari k.

Offline Query adalah penyelesaian kueri yang diberikan dengan cara menyelesaikan seluruh kueri secara bersamaan. Square Root Decomposition adalah teknik optimasi yang digunakan untuk membagi sebuah sekuen bilangan menjadi blok-blok dengan jumlah tertentu. Jumlah dari blok didapatkan dari melakukan operasi akar kuadrat terhadap jumlah bilangan sekuen tersebut. Mo's Algorithm adalah teknik optimasi kueri yang mengembangkan konsep Square Root Decomposition. Algoritma ini dapat mengurangi waktu kompleksitas untuk menjawab permasalahan dengan jumlah kueri banyak dan jarak yang variatif. Kueri yang diberikan akan disortir menjadi urutan tertentu sehingga jawaban dari kueri sebelumnya dapat digunakan untuk kueri selanjutnya karena jarak yang berdekatan.

Pada tugas akhir ini akan dirancang penyelesaian masalah yang disampaikan pada paragraf pertama dengan menggunakan teknik Mo's Algorithm dan Square Root Decomposition. Solusi

yang dikembangkan berjalan dengan kompleksitas waktu $\mathcal{O}((N+M)\sqrt{N}K \log mv)$*, yang mana* N *adalah jumlah bilangan di dalam sekuen bilangan,* M *adalah jumlah operasi kueri,* $\sqrt{N}$ *adalah konstanta,* K *adalah jumlah langkah pengerjaan, dan* $\log mv$ *adalah kompleksitas pada Fenwick Tree.*

Kata Kunci: fenwick tree, mo's algorithm, offline query, square root decomposition

APPLICATION OF SQUARE ROOT DECOMPOSITION AND MO'S ALGORITHM ON ALGORITHM DESIGN OF CASE STUDY: CLASSIC SPOJ COUNTING DIFF-PAIRS

Name	: ABDUL MAJID HASANI
NRP	: 5114100097
Major	: Informatics Department Faculty of IC-ITS
Supervisor I	: Rully Soelaiman, S.Kom., M.Kom.
Supervisor II	: Fajar Baskoro, S.Kom., M.T.

Abstract

Given a sequence of number A with total of N, M sequence of query, and absolute difference of k. There is operation of query for searching total of pair of number with absolute difference of k or greater than k.

Offline Query is technique for solving given query at the same time. Square Root Decomposition is optimization technique used for dividing a number sequence into certain number of blocks. The number of blocks are obtained from square root of total number in number sequence. Mo's Algorithm is query optimization technique that develop Square Root Decomposition concept. This algorithm reduce complexity time for solving number of query problem with variative range. Given query will be sorted into certain order so that the answer of previous query will be used for answering the next query because its range is near.

In this final project, the writer will design and analyze an algorithm to solve the problem that stated in the first paragraph using Mo's Algorithm technique and Square Root Decomposition. The solution will run in $\mathcal{O}((N+M)\sqrt{N}K \log mv)$ time complexity where N is the total number of given number sequence, M is the total of given queries, $\sqrt{N}$ is constant value, K is total steps, and $\log mv$ is Fenwick Tree's Complexity.

Keywords: fenwick tree, mo's algorithm, offline query, square root decomposition

KATA PENGANTAR

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

PENERAPAN TEKNIK DEKOMPOSISI SQUARE ROOT DAN ALGORITMA MO'S PADA RANCANGAN ALGORITMA STUDI KASUS: SPOJ KLASIK COUNTING DIFF-PAIRS.

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 Rossy Roostiningsih selaku ibu penulis yang selalu memberikan perhatian serta kasih sayang dan tidak pernah lelah menemani penulis.
- Bapak Yanus Suprayogi selaku bapak penulis yang selalu memberikan motivasi, segala prinsip tentang kehidupan, dan menjadi sosok 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.

- Bapak Fajar Baskoro, S.Kom., M.T. selaku Dosen Pembimbing yang telah memberikan bimbingan dan arahan dalam pengerjaan Tugas Akhir ini.
- Said Abil Choir dan Wardatil Ahadiyyah selaku kakak penulis yang menjadi orang yang selalu ada untuk mendengarkan curhatan dari penulis.
- Sri Wahyuningrum selaku bibi penulis yang selalu memberikan perhatian, motivasi, dan memasak makanan yang diinginkan oleh penulis.
- Saean Achmady selaku paman penulis yang mengizinkan penulis untuk menempati rumahnya selama masa perkuliahan.
- Mbak Muk selaku pembantu penulis yang selalu memberikan makan penulis saat lapar.
- Keluarga Besar Sofyan Haryanto yang selalu mencari penulis karena tidak pernah ke Malang.
- Rekan dan sahabat Dasar dan Terapan Komputasi yang selalu menjadi teman untuk berbagi ilmu dan mencari makan saat dini hari.
- Angkatan 2014 yang menjadi teman penulis selama masa perkuliahan.
- M. Arif Perdana Ariyansyah, Fauzan Fakhrul Arifin, dan teman-teman SMA penulis.
- 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, Januari 2018

Abdul Majid Hasani

Halaman ini sengaja dikosongkan

DAFTAR ISI

Halaman ini sengaja dikosongkan

DAFTAR TABEL

DAFTAR GAMBAR

DAFTAR KODE SUMBER

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

Tugas Akhir ini mengacu pada permasalahan klasik SPOJ *Counting diff-pairs*. Diberikan sebuah sekuen bilangan A dengan jumlah N, M baris kueri, dan selisih mutlak bernilai k. Terdapat ketentuan operasi kueri sebagai berikut:

- Kueri untuk mencari jumlah pasangan angka dalam jarak tertentu di dalam sekuen bilangan A yang memiliki selisih mutlak sama dengan atau lebih dari k.

Selisih mutlak dua buah *integer* didapatkan dari pengurangan bilangan a dan b dalam rentang kueri. Diberikan sebuah bilangan bulat k yang merupakan selisih mutlak yang sudah ditentukan. Berapakah jumlah pasangan angka dalam jarak tertentu di dalam sekuen bilangan A yang memiliki selisih mutlak sama dengan atau lebih dari k.

Untuk menyelesaikan permasalahan di atas, penulis akan menggunakan pendekatan solusi dengan teknik *Mo's Algorithm* dan *Square Root Decomposition*. Selain dapat menjawab pertanyaan dengan benar, waktu juga menjadi salah satu faktor penting untuk memberikan gambaran tentang performa dari algoritma yang dirancang.

Hasil dari Tugas Akhir ini diharapkan dapat memberikan gambaran mengenai performa algoritma dengan teknik *Mo's Algorithm* dan *Square Root Decomposition*.

1.2 Rumusan Masalah

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

1. Perancangan dan implementasi algoritma yang sesuai untuk menyelesaikan permasalahan klasik SPOJ *Counting diff-pairs* yang didasari oleh teknik *Mo's Algorithm* dan *Square Root Decomposition*.
2. Analisis performa algoritma yang telah dirancang untuk menyelesaikan permasalahan klasik SPOJ *Counting diff-pairs*.

1.3 Batasan Masalah

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

1. Implementasi algoritma menggunakan bahasa pemrograman C++.
2. Jumlah kasus uji dalam 1 kali percobaan maksimal 50.000 kasus.
3. Angka yang dimasukkan ke dalam *array A* berada di dalam rentang 1 sampai 100.000.
4. Batas waktu pada program dalam 1 kali percobaan di bawah 3 detik.
5. *Memory* yang dibutuhkan dalam 1 kali percobaan di bawah 1536 *Megabyte*.

1.4 Tujuan

Tujuan dari Tugas Akhir ini adalah sebagai berikut:

1. Menyelesaikan permasalahan klasik SPOJ *Counting diff-pairs* dengan algoritma yang telah dirancang dan diimplementasikan menggunakan teknik *Mo's Algorithm* dan *Square Root Decomposition*.
2. Menganalisis hasil kinerja penyelesaian permasalahan permasalahan klasik SPOJ *Counting diff-pairs*.

1.5 Manfaat

Manfaat dari Tugas Akhir ini adalah sebagai berikut:

1. Mengetahui pemanfaatan metode *Mo's Algorithm* dan *Square Root Decomposition* dalam menyelesaikan suatu permasalahan.
2. Melatih kemampuan analisis karakteristik permasalahan yang dapat diselesaikan dengan metode *Mo's Algorithm* dan *Square Root Decomposition*.

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 permasalahan optimisasi pada permasalahan klasik SPOJ *Counting diff-pairs* 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 cara-cara meng-optimasi kecepatan *compiler* C++, penggunaan struktur data yang tepat, dan algoritma yang terkait dengan permasalahan klasik SPOJ *Counting diff-pairs*. 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 klasik SPOJ *Counting diff-pairs*. Luaran dari tahap ini adalah algoritma dan struktur data yang digunakan dalam implementasi.
4. Implementasi perangkat lunak
   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 klasik SPOJ *Counting diff-pairs*. Luaran dari tahap ini adalah implementasi algoritma yang dibangun menggunakan bahasa pemrograman C++ dan menggunakan IDE Dev-C++
5. Uji coba dan evaluasi
   Tahap pengujian dan evaluasi adalah tahap untuk menguji *program* yang telah dibuat hasil implementasi algoritma pada permasalahan klasik SPOJ *Counting diff-pairs*. 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.

Halaman ini sengaja dikosongkan

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. Istilah yang digunakan dalam buku ini masuk di dalam *domain Binary Indexed Tree* dan *Offline Query*.

2.1.1 *Binary Indexed Tree*

Binary Indexed Tree adalah sebuah struktur data yang digunakan untuk membuat waktu berjalannya algoritma lebih cepat. *Binary Indexed Tree* dapat juga disebut sebagai *Fenwick Tree*. *Binary Indexed Tree* pertama kali digunakan untuk kompresi data. Selain itu, *Binary Indexed Tree* digunakan untuk menyimpan frenkuensi dan memanipulasi tabel frekuensi kumulatif dari suatu deret bilangan[2].

2.1.2 *Offline Query*

Offline Query adalah penyelesaian kueri yang diberikan dengan cara menyelesaikan seluruh kueri secara bersamaan. Kelebihan dari *Offline Query* adalah dapat mengubah urutan kueri yang dimasukkan untuk mendapatkan waktu yang optimal. Cara kerja

dari *Offline Query* adalah memanipulasi data hasil operasi kueri dan urutan kueri sebelum jawaban dari satu kueri dicetak.

2.2 Algoritma *Square Root Decomposition*

Pada subbab ini akan dibahas deskripsi algoritma *Square Root Decomposition*, permasalahan yang dapat dipecahkan oleh algoritma tersebut, dan langkah-langkah penyelesainnya.

2.2.1 Deskripsi Algoritma

Square Root Decomposition adalah salah satu algoritma optimasi operasi kueri yang sering digunakan untuk pemrograman kompetitif. Algoritma ini mengurangi kompleksitas waktu dengan faktor dari $\sqrt{n}$. Konsep utama dari algoritma ini adalah mengurai *array* yang diberikan menjadi bagian-bagian kecil dengan ukuran $\sqrt{n}$[1].

Sebagai contoh terdapat persoalan untuk mendapatkan total penjumlahan setiap elemen yang ada di suatu *array* A. Pendekatan pertama adalah dengan melakukan iterasi terhadap setiap elemen yang ada di dalam *array* A untuk mengetahui jumlahnya sehingga kompleksitas waktu untuk setiap operasi kuerinya akan menjadi $O(n)$ untuk batas kiri adalah elemen pertama dan batas kanan adalah n. Cara tersebut tidak efektif jika permasalahan yang dikerjakan memiliki nilai n yang besar.

2.2.1.1 Operasi Kueri Penjumlahan

Berikut adalah langkah-langkah penggunaan *Square Root Decomposition* dengan contoh permasalahan sederhana. Dimodelkan sebuah *array* A dengan n elemen dan operasi kueri berjumlah k yang memiliki rentang batas kiri dan kanan yang berbeda. Solusi yang diinginkan dari permasalahan tersebut adalah mencari jumlah dari elemen yang ada di dalam rentang

operasi kueri. Langkah awal untuk *Square Root Decomposition* adalah menghitung akar dari n lalu membagi *array* A ke dalam blok kecil dengan jumlah elemen untuk setiap blok adalah hasil dari $\sqrt{n}$. Langkah selanjutnya adalah menghitung jumlah nilai elemen yang ada di setiap blok dan menyimpan jumlah tersebut. Untuk setiap blok yang ada di dalam rentang batas kiri dan kanan operasi kueri maka tidak perlu menghitung iterasi satu per satu akan tetapi langsung menggunakan jumlah dari blok yang telah terhitung. Untuk lebih jelasnya dapat dilihat pada ilustrasi-ilustrasi yang sudah dibuat.

1. Terdapat *array* $A[9] = [1, 3, 5, 6, 1, 10, 87, 9, 32]$. Ilustrasi dapat dilihat pada Gambar 2.1.

Indeks	0	1	2	3	4	5	6	7	8
Elemen	1	3	5	6	1	10	87	9	32

Gambar 2.1 Ilustrasi *Array* A

2. Akar dari n adalah 3 maka *array* A dibagi menjadi 3 blok dengan masing-masing 3 elemen. Ilustrasi dapat dilihat pada Gambar 2.2.

Block - 0			Block - 1			Block - 2		
1	3	5	6	1	10	87	9	32

Indeks	0	1	2	3	4	5	6	7	8
Elemen	1	3	5	6	1	10	87	9	32

Gambar 2.2 Ilustrasi *Array* A yang Telah Terbagi Menjadi 3 Blok

3. Menghitung jumlah nilai elemen untuk setiap blok dan menyimpan hasil tersebut. Ilustrasi dapat dilihat pada Gambar 2.3.

Block - 0			Block - 1			Block - 2		
9			17			128		
1	3	5	6	1	10	87	9	32

Indeks	0	1	2	3	4	5	6	7	8
Elemen	1	3	5	6	1	10	87	9	32

Gambar 2.3 Ilustrasi *Array* A dan Jumlah Nilai Elemen per Blok

4. Model yang digambarkan dapat menyelesaikan operasi kueri penjumlahan dengan rentang batas kiri dan kanan dengan kompleksitas waktu $O(\sqrt{n})$. Contoh jika batas kirinya adalah 2 dan batas kanannya 6 maka hasil untuk setiap iterasi yang akan didapatkan adalah $5 + 17 + 87 = 109$. Ilustrasi dapat dilihat pada Gambar 2.4.

Block - 0			Block - 1			Block - 2		
9			17			128		
1	3	5	6	1	10	87	9	32

Indeks	0	1	2	3	4	5	6	7	8
Elemen	1	3	5	6	1	10	87	9	32

Gambar 2.4 Ilustrasi *Array* A dan Contoh Hasil Operasi Kueri

Kompleksitas waktu untuk operasi kueri dengan menggunakan algoritma *Square Root Decomposition* adalah $O(\sqrt{n})$ untuk batas kiri 0 dan batas kanan n. Sehingga dapat diketahui bahwa algoritma *Square Root Decomposition* lebih efektif untuk memecahkan contoh permasalahan tersebut.

2.2.1.2 Operasi Kueri *Update*

Langkah-langkah untuk operasi kueri yang mengubah elemen dari *array A* dapat dilihat sebagai berikut:

1. Jika ingin mengubah elemen indeks 3 menjadi 2 maka yang dilakukan adalah mengubah indeks tersebut dan mengubah jumlah dari blok yang memiliki indeks tersebut. Ilustrasi dapat dilihat pada Gambar 2.5.

Block - 0			Block - 1			Block - 2		
9			13			128		
1	3	5	2	1	10	87	9	32

Indeks	0	1	2	3	4	5	6	7	8
Elemen	1	3	5	2	1	10	87	9	32

Gambar 2.5 Ilustrasi *Array* A dan Contoh Hasil Operasi Kueri *Update*

Kompleksitas waktu untuk operasi kueri *Update* adalah $O(1)$.

2.2.2 Permasalahan *Ada and Unique Vegetable* pada SPOJ

Pada subbab ini akan dijelaskan mengenai permasalahan yang digunakan untuk membuktikan kebenaran dari algoritma *Square Root Decomposition* yang dirancang. Contoh permasalahan yang dapat diselesaikan dengan algoritma *Square Root Decomposition* adalah permasalahan pada situs penilaian $SPOJ$ dengan judul permasalahan *Ada and Unique Vegetable* dan kode soal $ADAUNIQ$[6].

Di dalam soal tersebut dideskripsikan terdapat seekor kumbang kecil bernama Ada yang sedang mengolah tumbuhan miliknya. Ada memiliki parit yang panjang dan penuh dengan beragam jenis tumbuhan. Ada ingin mengetahui jumlah jenis tumbuhan yang unik dan berbeda-beda untuk setiap ruas dari paritnya. Proses pengolahan tumbuhan adalah proses yang dinamis sehingga satu tumbuhan dapat berubah menjadi jenis tumbuhan lain. Pertanyaan

yang diangkat dalam permasalahan ini adalah bagaimana mencari jumlah dari jenis tumbuhan yang unik jika terdapat perubahan jenis tumbuhan pada parit milik Ada.

Terdapat 2 operasi untuk soal $ADAUNIQ$, yaitu:

1. Menemukan jumlah jenis tumbuhan yang unik untuk *sub array* dari *array* A dengan jarak antara l sampai r.
2. Mengubah jenis tanaman dari elemen dalam *array* A.

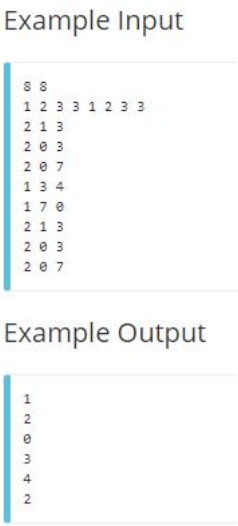
Format masukan untuk permasalahan *Ada and Unique Vegetable* dapat dilihat sebagai berikut:

1. Baris pertama adalah *integer* N dan *integer* Q. N untuk menyimpan jumlah tumbuhan yang dimiliki dan Q untuk menyimpan jumlah kueri yang diinginkan.
2. Baris kedua adalah elemen yang mengisi parit. Jumlah elemennya adalah N.
3. Baris ketiga sampai ke-Q adalah operasi kueri dengan format $t\ x\ y$.

Dua operasi kueri tersebut memiliki kriteria sebagai berikut:

1. Kueri satu untuk mengubah elemen di dalam *array* A. t adalah tipe operasi kueri yang selalu bernilai 1, x adalah posisi elemen yang ingin diubah menjadi y.
2. Kueri dua untuk mendapatkan jumlah jenis tumbuhan yang unik untuk jarak tertentu. t adalah tipe operasi kueri yang selalu bernilai 2, x adalah batas kiri dan y adalah batas kanan operasi kueri.

Format keluaran berisi sebuah nilai yang menunjukkan jumlah jenis tumbuhan yang unik dari bagian ruas parit yang dimiliki Ada untuk jarak tertentu. Contoh dari format masukan dan keluaran dapat dilihat pada Gambar 2.6.



Gambar 2.6 Contoh Masukan dan Luaran Permasalahan *Ada and Unique Vegetable*

Batasan-batasan pada permasalahan *Ada and Unique Vegetables* adalah sebagai berikut:

1. $1 \leq N \leq 200000$.
2. $1 \leq Q \leq 200000$.
3. $0 \leq A[i] \leq 200000$.
4. Jika x dan y adalah operasi kueri satu maka $0 \leq x \leq N$ dan $0 \leq y \leq 200000$.
5. Jika x dan y adalah operasi kueri dua maka $0 \leq x \leq y \leq N$.

2.2.3 Penyelesaian Permasahalan *Ada and Unique Vegetable*

Pada subbab ini akan dijelaskan bagaimana menyelesaikan permasalahan *Ada and Unique Vegetable* dengan penjelasan

permasalahan seperti pada subbab 3.1.1. Pada deskripsi soal tersebut diketahui bahwa terdapat dua operasi kueri, yaitu mencari jumlah tumbuhan unik yang dimiliki oleh Ada dan mengubah nilai dari tumbuhan yang ada di dalam *array*. Sebuah tumbuhan akan dikatakan unik jika elemen dari *array* tersebut hanya berjumlah 1 atau tidak sama dengan elemen lain yang ada dalam rentang yang sudah ditetapkan dari operasi kueri.
Operasi kueri akan dibagi menjadi blok-blok kecil dengan konstanta yang tepat untuk membuat waktu berjalannya solusi menjadi optimal. Hal ini juga memberikan efek terhadap penyortiran operasi kueri mana yang akan dikerjakan terlebih dahulu.
Jumlah dari setiap elemen yang ada di dalam operasi kueri akan disimpan di dalam sebuah *array* $counter$. Jika nilai dari $counter$ terhadap elemen tertentu di dalam rentang suatu operasi kueri berjumlah 1 maka total tumbuhan yang unik akan bertambah. Akan tetapi, jika nilai dari $counter$ terhadap elemen tertentu di dalam rentang suatu operasi kueri tidak berjumlah 1 maka total tumbuhan yang unik akan berkurang atau tetap.
Dikarenakan *array* yang menyimpan tumbuhan Ada dapat berubah maka terdapat satu *array* $temp$ untuk menyimpan nilai *array* tumbuhan yang sebenarnya. Untuk setiap operasi kueri akan dilakukan pengecekan apakah kueri tersebut berada di dalam kondisi yang sama dengan operasi kueri sebelumnya. Jika tidak dalam kondisi yang sama maka *array* akan diubah sesuai dengan kondisi saat operasi kueri dimasukkan. Hasil akhir dari sebuah operasi kueri akan disimpan di dalam sebuah *array* $answer[]$ dengan indeks sesuai urutan masuk operasi kueri.

2.3 *Mo's Algorithm*

Pada subbab ini akan dibahas deskripsi algoritma *Mo's Algorithm*, permasalahan yang dapat dipecahkan oleh algoritma tersebut, dan langkah-langkah penyelesainnya.

2.3.1 Deskripsi Algoritma

Mo's Algorithm adalah teknik optimasi kueri yang mengembangkan konsep *Square Root Decomposition*. Algoritma ini menjawab permasalahan operasi kueri berjumlah banyak dengan rentang yang berbeda-beda dan jumlah elemen dalam *array* yang banyak. Poin utama dari algoritma ini adalah mengurangi jumlah langkah pengerjaan operasi kueri dengan menggunakan kembali hasil kueri sebelumnya yang *overlapping*. Terdapat beberapa syarat yang harus dipenuhi untuk menerapkan algoritma *Mo's Algorithm*, yaitu:

1. Memiliki *array* dengan rentangan tertentu.
2. Menerapkan *Offline Query*.
3. Mendapatkan hasil dengan cara mengurangi/memasukkan satu nilai ke dalam jarak tertentu.

Sebagai contoh jika terdapat operasi kueri untuk menjumlahkan nilai elemen dalam rentang $[0, 100]$ dan $[50, 150]$ maka dengan menggunakan *Mo's Algorithm* akan mengurangi jumlah langkah pengerjaan dengan cara sebagai berikut:

1. Menyimpan hasil dari operasi kueri dengan rentang $[0, 100]$.
2. Membuat batas kiri dari operasi kueri sebelumya menjadi 50 dan mengurangi hasil operasi kueri $[0, 100]$ dengan hasil kueri $[0, 50]$. Hasil yang tersimpan adalah hasil operasi kueri $[50, 100]$.
3. Membuat batas kanan dari operasi kueri sebelumnyamenjadi 150 dan menambahkan hasil operasi kueri $[50, 100]$ dengan hasil kueri $[100, 150]$. Hasil yang tersimpan adalah hasil operasi kueri $[50, 150]$.

Tujuan selanjutnya dalam menggunakan *Mo's Algorithm* adalah menyortir urutan pengerjaan operasi kueri sehingga operasi kueri selanjutnya memiliki langkah minimum pengerjaan. Sebagai contoh jika terdapat tiga operasi kueri, yaitu $[1, 30]$, $[30, 50]$, dan

$[20, 40]$ maka langkah-langkah pengerjaannya adalah:

1. Menyelesaikan kueri $[1, 30]$.
2. Lalu menyelesaikan kueri $[20, 40]$ dengan menggunakan hasil dari kueri $[1, 30]$. Caranya adalah mengganti batas kiri dari 1 menjadi 20 dan batas kanan dari 30 menjadi 40. Jumlah langkah sekarang adalah 30.
3. Menyelesaikan kueri $[30, 50]$ dengan menggunakan hasil dari kueri $[20, 40]$. Caranya adalah mengganti batas kiri dari 20 menjadi 30 dan batas kanan dari 40 menjadi 50. Jumlah langkah sekarang adalah 50.

Untuk menyortir, terdapat beberapa aturan yang dapat dilihat sebagai berikut:

1. Operasi kueri prioritas pertama adalah semua operasi kueri yang ada di dalam blok terkecil (pembagian blok dapat dilihat pada subbab 2.2.1) dimulai dari indeks batas kiri yang paling kecil juga. Setelah blok terkecil sudah tersortir maka selanjutnya akan pindah ke blok selanjutnya.
2. Operasi kueri prioritas kedua adalah operasi kueri dengan batas kanan yang paling kecil.
3. Prioritas ketiga adalah operasi kueri yang dikerjakan adalah operasi kueri yang memiliki keadaan yang sama dengan operasi kueri sebelumnya. Maksud keadaan disini adalah operasi kueri tersebut berlaku untuk keadaan *array* yang belum diubah elemennya menjadi *array* baru.

Terdapat empat kemungkinan saat mengubah batas kiri dan kanan dari satu operasi kueri ke operasi kueri lainnya, yaitu:

1. Batas kiri 1 kurang dari batas kiri 2. Maka yang dilakukan adalah menambah nilai dari batas kiri 1 sampai memiliki nilai yang sama dengan batas kiri 2. Secara bersamaan mengurangi nilai-nilai elemen *array* yang memiliki indeks batas kiri dari total yang tersimpan.

2. Batas kiri 1 lebih dari batas kiri 2. Maka yang dilakukan adalah mengurangi nilai dari batas kiri 1 sampai memiliki nilai yang sama dengan batas kiri 2. Secara bersamaan menambah nilai-nilai elemen *array* yang memiliki indeks batas kiri dari total yang tersimpan.
3. Batas kanan 1 kurang dari batas kanan 2. Maka yang dilakukan adalah menambah nilai dari batas kanan 1 sampai memiliki nilai yang sama dengan batas kanan 2. Secara bersamaan menambah nilai-nilai elemen *array* yang memiliki indeks batas kanan dari total yang tersimpan.
4. Batas kanan 1 lebih dari batas kanan 2. Maka yang dilakukan adalah mengurangi nilai dari batas kanan 1 sampai memiliki nilai yang sama dengan batas kanan 2. Secara bersamaan mengurangi nilai-nilai elemen *array* yang memiliki indeks batas kanan dari total yang tersimpan.

Kompleksitas dari *Mo's Algorithm* adalah $O((n+q)\sqrt{n}k)$. n adalah jumlah elemen, q adalah jumlah dari operasi kueri, dan k adalah langkah yang dibutuhkan dalam menambah/mengurangi elemen untuk setiap kemungkinan.
Untuk setiap dua operasi kueri yang ada di dalam satu blok, batas kiri antara dua operasi kueri tersebut tidak akan berubah lebih dari $\sqrt{n}$ langkah. Lalu saat operasi kueri terakhir dari blok satu ke operasi kueri pertama dari blok yang berbeda tidak akan berubah lebih dari $2\sqrt{n}$. Batas kiri antara kueri satu dengan selanjutnya tidak akan berubah lebih dari $O(\sqrt{n})$. Untuk q operasi kueri, terdapat q langkah saat menambah/mengurangi elemen untuk setiap kemungkinan sebanyak $O(q\sqrt{n})$. Kompleksitas untuk mengubah batas kiri dari semua operasi kueri adalah $O(q\sqrt{n}k)$.
Batas kanan dari dua operasi kueri yang berkelanjutan dalam satu blok akan memiliki nilai yang sama besar atau bertambah sampai dengan n. Dikarenakan jumlah blok yang dimiliki adalah $\sqrt{n}$ maka waktu langkah saat menambah/mengurangi elemen untuk setiap kemungkinan sebanyak $O(n\sqrt{n})$. Kompleksitas untuk mengubah

batas kanan dari semua operasi kueri adalah $O(n\sqrt{n}k)$.
Kesimpulan dari uraian di atas adalah kompleksitas dari setiap operasi kueri yang ada di *Mo's Algorithm* adalah $O((n + q)\sqrt{n}k)$.

2.3.1.1 *Mo's Algorithm* dengan *Update*

Berikut akan dijelaskan bagaimana kondisi saat terdapat perubahan di *array* dalam *Mo's Algorithm*. Terdapat dua kemungkinan yang dapat dilihat seperti berikut:

1. Kondisi sekarang lebih dari kondisi operasi kueri yang sedang dijalankan. Maksud dari pernyataan tersebut adalah saat operasi kueri yang sedang dijalankan memiliki kondisi yang lebih dulu dari kondisi sekarang maka yang dilakukan adalah mengulang kembali kondisi tersebut dengan cara mengembalikan elemen *array* yang sama dengan kondisi saat operasi kueri tersebut dimasukkan ke dalam urutan operasi kueri. Secara bersamaan elemen-elemen yang diubah akan dilakukan pengecekan apakah elemen tersebut dapat merubah total jawaban sekarang atau tidak.
2. Kondisi sekarang kurang dari kondisi operasi kueri yang sedang dijalankan. Maksud dari pernyataan tersebut adalah saat operasi kueri yang sedang dijalankan memiliki kondisi yang lebih baru dari kondisi sekarang maka yang dilakukan adalah menambahkan kondisi tersebut dengan cara mengubah elemen *array* yang sama dengan kondisi saat operasi kueri tersebut dimasukkan ke dalam urutan operasi kueri. Secara bersamaan elemen-elemen yang diubah akan dilakukan pengecekan apakah elemen tersebut dapat merubah total jawaban sekarang atau tidak.

2.3.2 Permasalahan *Angry Siam* pada SPOJ

Pada subbab ini akan dijelaskan mengenai permasalahan yang digunakan untuk membuktikan kebenaran dari *Mo's Algorithm* yang dirancang. Contoh permasalahan yang dapat diselesaikan dengan *Mo's Algorithm* adalah permasalahan pada situs penilaian $SPOJ$ dengan judul permasalahan *Angry Siam* dan kode soal $HRSIAM$[5].
Di dalam soal tersebut dideskripsikan terdapat sebuah *array* A dengan N masalah. Setiap masalah memiliki tingkat kesulitan sehingga untuk elemen i memiliki kesulitan $A[i]$. $HrSiam$ adalah orang yang akan memecahkan masalah *array* A. Sementara itu, $HrSiam$ adalah orang yang tidak suka masalah dengan tingkat kesulitan yang mudah sehingga jika $HrSiam$ menemukan masalah yang sama dengan yang pernah $HrSiam$ temukan maka tingkat amarah dari $HrSiam$ akan bertambah. Untuk itu terdapat sebuah *array* yang menyimpan tingkat amarah dari $HrSiam$ bernama *array* $Angry$ dengan jumlah N elemen.

Jika $HrSiam$ mendapatkan sebuah masalah dengan tingkat kesulitan d maka amarah dari $HrSiam$ meningkat menjadi $d * Angry[1]$. Lalu jika $HrSiam$ menemukan kembali masalah dengan tingkat kesulitan d maka amarah dari $HrSiam$ bertambah $d * Angry[2]$ dan selanjutnya. Inti dari permasalahannya adalah jika $HrSiam$ bertemu masalah dengan tingkat kesulitan d untuk $i - th$ kali maka kemarahannya bertambah menjadi $d * Angry[i]$.

Terdapat dua operasi kueri untuk soal $HRSIAM$, yaitu:

1. Operasi kueri untuk menemukan jumlah total kemarahan $HrSiam$ untuk rentang yang telah ditentukan.
2. Operasi kueri untuk mengganti tingkat kesulitan dari permasalahan yang diberikan untuk $HrSiam$ pada indeks yang ditentukan dan tingkat kesulitan yang telah ditentukan juga.

Format masukan untuk permasalahan *Angry Siam* dapat dilihat sebagai berikut:

1. Baris pertama adalah *integer* N untuk menyimpan jumlah permasalahan yang dimiliki.
2. Baris kedua adalah *array* A dengan jumlah N elemen yang berisi tingkat kesulitan dari permasalahan.
3. Baris ketiga adalah *array* $Angry$ dengan jumlah N elemen yang berisi tingkat kemarahan dari $HrSiam$.
4. Baris keempat adalah *integer* Q untuk menyimpan jumlah kueri yang diinginkan.
5. Baris kelima sampai ke-Q adalah operasi kueri dengan format $t\ x\ y$.

Dua operasi kueri tersebut memiliki kriteria sebagai berikut:

1. Kueri satu untuk mendapatkan jumlah kemarahan dari $HrSiam$ untuk jarak tertentu. t adalah tipe operasi kueri, x adalah batas kiri dan y adalah batas kanan operasi kueri.
2. Kueri dua untuk mengubah elemen di dalam *array* A. t adalah tipe operasi kueri, x adalah posisi elemen yang ingin diubah menjadi y.

Format keluaran berisi sebuah nilai yang menunjukkan jumlah kemarahan dari $HrSiam$ untuk jarak tertentu. Contoh dari format masukan dan keluaran dapat dilihat pada Gambar 2.7.

```
Input:
8
1 2 3 3 1 1 2 3
1 2 3 4 5 6 7 8
2
1 3 6
1 1 5

Output:
12
14
```

Gambar 2.7 Contoh Masukan Dan Luaran Permasalahan *Angry Siam*

Batasan-batasan pada permasalahan *Angry Siam* adalah sebagai berikut:

1. $1 \leq N \leq 100000$.
2. $1 \leq Q \leq 100000$.
3. $1 \leq x \leq 100000$.
4. $1 \leq y \leq 100000$.
5. $1 \leq A[i] \leq 100000$.
6. $1 \leq Angry[i] \leq 100000$.
7. Jika x dan y adalah operasi kueri satu maka $1 \leq x \leq y \leq N$.

2.3.3 Penyelesaian Permasalahan *Angry Siam*

Pada subbab ini akan dijelaskan bagaimana menyelesaikan permasalahan *Angry Siam* dengan penjelasan permasalahan seperti pada subbab 3.2.1.

Pada deskripsi soal tersebut diketahui bahwa terdapat 2 operasi kueri, yaitu menemukan jumlah kemarahan dari $HrSiam$ untuk *sub array* dari *array* dengan jarak antara l sampai r dan mengubah tingkat kesulitan dari elemen dalam *array* A. Operasi kueri akan dibagi menjadi blok-blok kecil dengan konstanta yang tepat untuk membuat waktu berjalannya solusi menjadi optimal. Hal ini juga memiliki efek terhadap penyortiran operasi kueri mana yang akan dikerjakan dahulu.

Jumlah dari setiap elemen yang ada di dalam operasi kueri akan disimpan di dalam sebuah *array* $counter$. Nilai dari $counter$ akan menjadi indeks untuk mengetahui tingkat kemarahan $HrSiam$ dan akan dikalikan dengan nilai permasalahan tersebut. Untuk setiap operasi kueri, nilai elemen dari *array* $counter$ akan berubah-ubah sejalan dengan perubahan rentang yang dijalankan. Untuk setiap perubahan akan dilakukan operasi untuk menambah total kemarahan atau mengurangi total kemarahan dengan hasil perkalian *array* $angry$ yang memiliki indeks elemen *array* $counter$ dengan nilai permasalahan pada *array* permasalahan.

Dikarenakan *array* yang menyimpan permasalahan $HrSiam$ dapat berubah maka terdapat satu *array* $temp$ untuk menyimpan nilai *array* permasalahan yang sebenarnya. Untuk setiap operasi kueri akan dilakukan pengecekan apakah kueri tersebut berada di dalam kondisi yang sama dengan operasi kueri sebelumnya. Jika tidak dalam kondisi yang sama maka *array* akan diubah sesuai dengan kondisi saat operasi kueri dimasukkan. Hasil akhir dari sebuah operasi kueri akan disimpan di dalam sebuah *array* $answer[]$ dengan indeks sesuai urutan masuk operasi kueri.

2.4 Struktur Data *Fenwick Tree*

Struktur data *Fenwick Tree* atau yang juga dikenal sebagai *Binary Indexed Tree* (BIT) adalah struktur data yang berguna untuk mengimplementasi *dynamic cumulative frequency tables*[7].

2.4.1 Frekuensi Kumulatif Dinamis

Dynamic cumulative frequency tables atau tabel frekuensi kumulatif dinamis adalah tabel yang menyimpan frekuensi kumulatif dari sebuah baris angka. Frekuensi kumulatif adalah nilai frekuensi untuk indeks ke-i ditambah dengan jumlah frekuensi semua indeks sebelum indeks ke-i. Maksud dari dinamis disini adalah nilai yang ada di tabel akan berubah seiring dengan uji coba kasus yang berbeda. Untuk itu, dibutuhkan struktur data *Fenwick Tree* yang dapat menangani perubahan nilai frekuensi kumulatif dari baris angka.

Terdapat permisalan 11 murid yang memiliki hasil ujian yang beragam dan dapat dilihat pada $f = 2, 4, 5, 5, 6, 6, 6, 7, 7, 8, 9$ dimana semua hasil ujian bernilai *integer* antara $[1..10]$. Untuk frekuensi f dari setiap hasil ujian individu dan frekuensi kumulatif cf dari hasil ujian antara $[1..i]$ dapat dilihat pada Tabel 2.1.

Tabel 2.1 Contoh Tabel Frekuensi Kumulatif

Indeks	Frekuensi f	Frekuensi Kumulatif cf	Penjelasan
0	–	–	Indeks 0 diabaikan
1	0	0	$cf[1] = f[1] = 0$
2	1	1	$cf[2] = f[1] + f[2] = 0 + 1 = 1$
3	0	1	$cf[3] = f[1] + f[2] + f[3] = 0 + 1 + 0 = 1$
4	1	2	$cf[4] = cf[3] + f[4] = 1 + 1 = 2$
5	2	4	$cf[5] = cf[4] + f[5] = 2 + 2 = 4$
6	3	7	$cf[6] = cf[5] + f[6] = 4 + 3 = 7$
7	2	9	$cf[7] = cf[6] + f[7] = 7 + 2 = 9$
8	1	10	$cf[8] = cf[7] + f[8] = 9 + 1 = 10$
9	1	11	$cf[9] = cf[8] + f[9] = 10 + 1 = 11$
10	0	11	$cf[10] = cf[9] + f[10] = 11 + 0 = 11$

Tabel frekuensi kumulatif dapat menjadi solusi untuk kueri jumlah dengan jarak tertentu yang selanjutnya akan disebut dengan RSQ. Notasi dari RSQ dapat ditulis menjadi $RSQ(1, i) \forall i \in [1..n]$ dengan n adalah nilai tes terbesar. Dari Tabel 2.1 dapat dikatakan bahwa $n = 10$ dan beberapa contoh untuk RSQ adalah

$RSQ(1,1) = 0$, $RSQ(1,2) = 1$, $RSQ(1,6) = 7$. Untuk $i \neq 1$, rumus yang digunakan adalah $RSQ(i,j) = RSQ(1,j) - RSQ(1,i-1)$. Contohnya adalah $RSQ(4,6) = RSQ(1,6) - RSQ(1,3) = 7 - 1 = 6$.

Untuk frekuensi yang statik, kompleksitas untuk menghitung tabel frekuensi kumulatif dari Tabel 2.1 adalah $O(n)$ karena hanya menggunakan *looping*. Akan tetapi jika frekuensi dari elemen yang ada diubah menjadi lebih besar atau lebih kecil maka dibutuhkan struktur data yang dinamis.

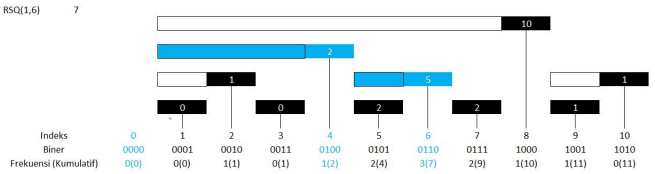
2.4.2 Implementasi *Fenwick Tree*

Fenwick Tree diimplementasi sebagai *array* yang memiliki indeks dalam bentuk *bits* untuk setiap elemennya. *Fenwick Tree* menggunakan *Least Significant One-bit*. Selanjutnya *Fenwick Tree* ditulis sebagai ft. Indeks i dalam *Fenwick Tree* bertanggung jawab untuk perubahan elemen yang memiliki indeks dengan rentang $[i - (i\&(-i))..i]$ dan untuk $ft[i]$ menyimpan frekuensi kumulatif untuk elemen dalam rentang $[i - (i\&(-i)) + 1, i - (i\&(-i)) + 2, i - (i\&(-i)) + 3, .., i]$.

2.4.2.1 Menghitung Frekuensi Kumulatif

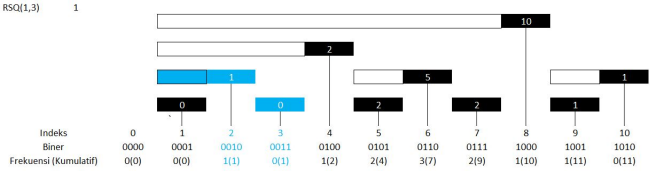
Dalam menghitung frekuensi kumulatif indeks b pada *Fenwick Tree* maka yang dilakukan adalah menjumlah elemen dari $ft[b], ft[b'], ft[b'']$ sampai b^i adalah 0. Deret ini didapatkan dengan cara mengurangi *Least Significant One-bit* menggunakan *bit manipulation expression*: $b' = b - (b\&(-b))$. Iterasi dari *bit manipulation* tersebut dapat menghilangkan *least significat one-bit* dari b secara efektif di setiap iterasi. *Integer* b memiliki kompleksitas $O(logb)$ bit dan $RSQ(b)$ memiliki kompleksitas $O(logn)$ jika $b = n$. Contoh operasi RSQ dalam *Fenwick Tree* dapat dilihat pada Gambar 2.8 dan Gambar 2.9.

Untuk Gambar 2.8, frekuensi kumulatif yang dicari adalah pada indeks 6, maka operasi yang dilakukan adalah $RSQ(6) = ft[6] + ft[4] = 5 + 2 = 7$. Indeks 4 dan 6 memiliki rentang $[1..4]$ dan $[5..6]$ secara berurutan. Dengan menggabungkan keduanya, maka didapatkan seluruh rentang antara $[1..6]$. Indeks 6, 4, dan 0 berhubungan di dalam bentuk biner karena: $b = 6_{10} = (1\underline{1}0)_2$ dapat ditransformasi menjadi $b' = 4_{10} = (\underline{1}00)_2$ dan selanjutnya menjadi $b'' = 0_{10} = (000)_2$.



Gambar 2.8 Ilustrasi *Fenwick Tree* Untuk Menemukan RSQ(1,6)

Untuk Gambar 2.9, frekuensi kumulatif yang dicari adalah pada indeks 3, maka operasi yang dilakukan adalah $RSQ(3) = ft[3] + ft[2] = 0 + 1 = 1$. Indeks 2 dan 3 memiliki rentang $[1..2]$ dan $[3]$ secara berurutan. Dengan menggabungkan keduanya, maka didapatkan seluruh rentang antara $[1..3]$.



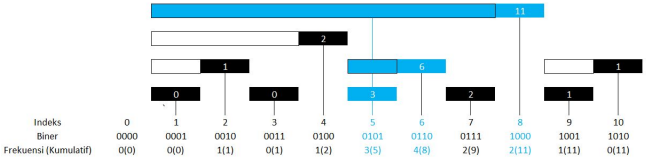
Gambar 2.9 Ilustrasi *Fenwick Tree* Untuk Menemukan RSQ(1,3)

Untuk mencari frekuensi kumulatif dengan rentang $[a..b]$ dimana $a \neq 1$ maka rumus yang digunakan adalah $rsq(a, b) = rsq(b) - rsq(a - 1)$. Operasi tersebut memiliki kompleksitas $O(2xlogb) \approx$

$O(logn)$ jika $b = n$.

2.4.2.2 Mengubah Elemen di Indeks k

Dalam mengubah elemen yang terdapt di indeks k pada *Fenwick Tree* dengan nilai v maka yang dilakukan adalah mengubah semua nilai dari $ft[k], ft[k'], ft[k''], ...$ sampai indeks k melebihi n, n adalah total elemen yang ada di *Fenwick Tree*. Baris indeks didapatkan dari iterasi *bit manipulation expression*: $k' = k + (k\&(-k))$. Kompleksitas dari operasi tersebut adalah $O(logn)$ sampai $k \geq n$. Contoh dari mengubah elemen yang ada di *Fenwick Tree* dapat dilihat pada Gambar 2.10.



Gambar 2.10 Ilustrasi *Fenwick Tree* Untuk Mengubah Elemen di Indeks 5 Menjadi 1

Dari Gambar 2.10 nilai indeks 5 akan diubah menjadi 1. Perubahan nilai tersebut akan merubah semua nilai $ft[k]$ yang berhubungan dengan k dalam bentuk biner. Indeks-indeks yang nilainya berubah dapat dilihat sebagai berikut: $k = 5_{10} = (101)_2, k' = (101)_2 + (00\underline{1})_2 = (110)_2 = 6_{10}$, dan $k'' = (110)_2 + (0\underline{1}0)_2 = (1000)_2 = 8_{10}$. dengan menggunakan operasi yang sudah diberikan di atas. Operasi tersebut hanya mencapai indeks 8 karena jika dilanjutkan: $k''' = 8_{10} = (1000)_2 + (0\underline{1}00)_2 = (1100)_2 = 12_{10}$. $k''' \geq n$. Untuk setiap indeks yang terhubungkan dalam bentuk biner akan diubah nilai dari frekuensi kumulatifnya.

2.5 Permasalahan *Counting diff-pairs* pada SPOJ

Permasalahan yang diangkat dalam Tugas Akhir ini bersumber dari *online judge* $SPOJ$ yaitu permasalahan *Counting diff-pairs* dengan kode soal $CPAIR2$[4]. Di dalam permasalahan ini diceritakan bahwa terdapat sebuah baris angka *integer* A dengan N jumlah elemen. Terdapat juga *integer* k untuk selisih absolut dan M jumlah kueri. Setiap kueri memiliki 2 *integer* l dan r untuk batas kanan dan batas kiri jarak kueri. Jawaban yang diinginkan adalah jumlah pasangan di dalam jarak l dan r yang memiliki selisih sama dengan atau lebih dari absolut k. Indeks dari permasalahan *Counting diff-pairs* dimulai dari 1. Uji kasus akan dibuat secara acak.

Format masukan untuk permasalahan *Counting diff-pairs* dapat dilihat sebagai berikut:

1. Baris pertama adalah *integer* N, *integer* M, dan *integer* k. N untuk menyimpan jumlah elemen dari *array* A, M untuk menyimpan jumlah operasi kueri, dan k untuk menyimpan selisih absolut yang diinginkan.
2. Baris kedua adalah *array* A dengan jumlah N elemen yang berisi angka yang menjadi sebuah baris.
3. Baris ketiga sampai ke-M adalah operasi kueri dengan format $l\ r$.

Format keluaran berisi sebuah nilai yang menunjukkan jumlah pasangan angka yang memiliki selisih absolut k atau selisih lebih dari k untuk jarak tertentu. Contoh dari format masukan dan keluaran dapat dilihat pada Gambar 2.11. Batasan-batasan pada permasalahan *Counting diff-pairs* adalah sebagai berikut:

1. $N \leq 100000$.
2. $M \leq 100000$.
3. $1 \leq A[i] \leq 100000$.
4. Jika i dan j adalah pasangan angka yang memiliki selisih absolut sama dengan atau lebih dari k maka $l \leq i < j \leq r$.

```
Input:
3 1 2
1 2 3
1 3
Output:
1
```

Gambar 2.11 Contoh Masukan dan Luaran Permasalahan *Counting diff-pairs*

2.6 Penyelesaian Permasahalan *Counting diff-pairs*

Pada subbab ini akan dijelaskan bagaimana menyelesaikan permasalahan *Counting diff-pairs* dengan penjelasan permasalahan seperti pada subbab 2.5.

Pada deskripsi soal tersebut diketahui bahwa terdapat operasi kueri untuk menemukan jumlah pasangan bilangan yang memiliki selisih absolut sama dengan atau lebih dari k untuk *sub array* dari *array* dengan jarak antara l sampai r. Operasi kueri akan dibagi menjadi blok-blok kecil dengan konstanta yang efektif untuk membuat waktu berjalannya solusi menjadi optimal. Hal ini juga memiliki efek terhadap penyortiran operasi kueri mana yang akan dikerjakan lebih dahulu.

Dalam penyelesaian dari permasalahan *Counting diff-pairs* akan digunakan struktur data *Fenwick Tree* untuk menyimpan frekuensi dari suatu bilangan yang ada di dalam rentang operasi kueri yang sedang dijalankan. Variabel yang digunakan untuk *Fenwick Tree* adalah $BIT[]$. Setiap perubahan yang terjadi dari rentang operasi kueri akan mengubah frekuensi suatu bilangan dalam rentang tersebut sehingga nilai dari *Fenwick Tree* $BIT[]$ di indeks dan semua indeks yang berhubungan dalam bentuk biner juga berubah. Untuk mengetahui jumlah angka yang memiliki selisih absolut sama dengan atau lebih dari bilangan dari indeks yang sedang dicek akan dilakukan penghitungan jumlah frekuensi kumulatif dari bilangan tersebut dengan cara membagi dua kemungkinan, yaitu:

1. Bilangan yang menjadi pasangan memiliki nilai lebih kecil

dari bilangan yang ada di indeks.
2. Bilangan yang menjadi pasangan memiliki nilai lebih besar dari bilangan yang ada di indeks.

Hasil dari setiap pengecekan akan disimpan dalam sebuah variabel *integer* $total$. Untuk setiap operasi kueri, nilai elemen dari *integer* $total$ akan berubah-ubah sejalan dengan perubahan rentang yang dijalankan. Hasil akhir dari sebuah operasi kueri akan disimpan di dalam sebuah *array* $answer[]$ dengan indeks sesuai urutan masuk operasi kueri.

Halaman ini sengaja dikosongkan

BAB III
DESAIN

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

3.1 Desain Penyelesaian Permasalahan *Ada and Unique Vegetable*

Pada subbab ini akan dijelaskan desain penyelesaian permasalahan *Ada and Unique Vegetable*

3.1.1 Definisi Umum Sistem

Sistem akan menerima masukan berupa dua buah bilangan N dan Q yang masing-masing mewakili panjang parit yang dimiliki oleh Ada dan jumlah operasi kueri yang akan dilakukan. Selanjutnya sistem menerima masukan panjang parit sejumlah n sebagai sebuah *array* A dan diikuti oleh Q baris berisi 3 buah bilangan t, x, dan y yang masing-masing mewakili jenis operasi kueri, batas kiri/indeks yang elemennya akan diubah, dan batas kanan/elemen baru dari *array* A. Semua masukan sesuai dengan format yang sudah ditentukan pada subbab 2.2.2 sebelumnya. Kemudian, sistem akan membagi Q jumlah operasi kueri menjadi blok-blok kecil sesuai dengan yang sudah dijelaskan pada subbab 2.2.1 dan sistem juga menyortir Q jumlah operasi kueri sesuai dengan ketentuan pada subbab 2.3.1. Kemungkinan pada subbab 2.3.1.1 juga diperhitungkan pada sistem karena terdapat operasi kueri yang dapat mengubah elemen yang ada di *array* A. Operasi kueri yang sudah disortir akan dikerjakan sesuai dengan urutannya dan untuk setiap jawaban dari satu operasi

kueri akan disimpan ke dalam sebuah *array* dan digunakan kembali untuk menjawab operasi kueri selanjutnya. Hasil keluaran dari sistem adalah jawaban dari operasi kueri yang mencari jumlah unik tumbuhan yang dimiliki Ada sesuai dengan urutan operasi kueri saat dimasukkan ke dalam sistem. Format keluaran sesuai dengan subbab 2.2.2. *Pseudocode* dari fungsi *main* ditunjukkan oleh Gambar 3.1 dan 3.2.

Masukan	Banyak elemen N, banyak operasi kueri Q, nilai setiap elemen $value[i]$, operasi kueri (t, x, y)
Keluaran	Jawaban setiap operasi kueri dengan tipe 2

```
main()
1 Input N
2 Input Q
3 blk_sz = 5000
4 for i = 1 to N
5     Input value[i]
6     temp[i] = value[i]
7 for i = 1 to Q
8     Input type
9     if type = 1
10        Input up[time].position
11        Input up[time].val
12        time++
13    else
14        Input arr[total_query].L
15        Input arr[total_query].R
16        arr[total_query].L++
17        arr[total_query].R++
18        arr[total_query].Number = total_query
```

Gambar 3.1 *Pseudocode* Fungsi *main* Bagian 1

```
19      arr[total_query].Time = time
20      total_query++
21 sort(arr, arr + total_query, compare)
22 solve()
```

Gambar 3.2 *Pseudocode* Fungsi *main* Bagian 2

3.1.2 Desain Algoritma

Sistem terdiri dari satu fungsi utama, yaitu fungsi *Solve* yang kemudian akan dijelaskan bagian-bagian yang ada di dalam fungsi tersebut, algoritma *Square Root Decomposition*, dan *Mo's Algorithm*. Pada subbab ini akan dijelaskan desain algoritma dari bagian-bagian fungsi tersebut dan algoritma yang digunakan.

3.1.2.1 Penentuan Konstanta *Square Root Decomposition*

Algoritma *Square Root Decomposition* bertugas untuk membagi Q jumlah operasi kueri menjadi blok-blok kecil. Caranya adalah dengan menentukan konstanta dan membagi batas kiri dan batas kanan dari setiap operasi kueri dengan konstanta tersebut. Penentuan konstanta dilakukan dengan cara uji coba dengan nilai konstanta yang variatif lalu dipilih nilai konstanta yang menghasilkan waktu penyelesaian paling cepat. Selanjutnya adalah melakukan penyortiran urutan pengerjaan operasi kueri yang sudah ditentukan menggunakan *std sort* dan fungsi *boolean compare*.

```
define_constanta()
1   blk_sz = 5000
```

Gambar 3.3 Konstanta *Square Root Decomposition*

Potongan desain algoritma *Square Root Decomposition* terdapat pada Gambar 3.3.

3.1.2.2 Desain *Mo's Algorithm*

Algoritma *Mo's Algorithm* bertugas untuk mengerjakan setiap operasi kueri yang sudah mendapatkan urutan pengerjaan dengan cara kerja yang sudah dijelaskan pada subbab 2.3.1. Setiap pergantian batas kiri dan kanan dari Q jumlah operasi kueri akan selalu dilakukan pengecekan perubahan elemen di setiap rentang dengan menggunakan fungsi *Add* dan *Remove*. Setiap pergantian kondisi akan dilakukan pemanggilan fungsi *Reset* dan *Change*. Potongan desain *Mo's Algorithm* terdapat pada Gambar 3.4 dan 3.5.

Masukan	Total operasi kueri, batas kiri dan kanan sekarang, nilai setiap elemen $value[i]$, batas kiri dan kanan operasi kueri, dan kondisi operasi kueri
Keluaran	Jawaban setiap operasi kueri

Mo's Algorithm()
1 **for** $i = 0$ **to** $total\_query$
2 **while** $currentL < left$
3 ...
4 **while** $currentL > left$
5 ...
6 **while** $currentR < right$
7 ...
8 **while** $currentR > right$
9 ...
10 **if** $currentTime > time$
11 ...

Gambar 3.4 Desain *Mo's Algorithm* Bagian 1

```
12    while currentTime > time
Mo's Algorithm()
13        ...
14    answer[i] = total
15 for i = 0 to total_query
16    output answer[i]
```

Gambar 3.5 Desain *Mo's Algorithm* Bagian 2

3.1.2.2.1 Desain Fungsi *Boolean compare*

Fungsi *boolean compare* digunakan untuk menjadi argumen std *sort* dengan ketentuan sebagai berikut:

1. Prioritas pertama adalah operasi kueri dengan batas kiri paling kecil di blok.
2. Prioritas kedua adalah operasi kueri dengan batas kanan paling kecil di blok.
3. Prioritas ketiga adalah operasi kueri yang memiliki kondisi paling awal.

Potongan desain fungsi *boolean compare* terdapat pada Gambar 3.6

Masukan	Kueri x, Kueri y
Keluaran	Argumen untuk fungsi $sort$

```
compare(x, y)
1 if x.L/blk_sz ≠ y.L/blk_sz
2    return x.L/blk_sz < y.L/blk_sz
3 if x.R/blk_sz ≠ y.R/blk_sz
4    return x.R/blk_sz < y.R/blk_sz
5 return x.Time < y.Time
```

Gambar 3.6 *Pseudocode* Fungsi *Boolean compare*

3.1.2.3 Desain Fungsi *solve*

Fungsi *solve* digunakan untuk menghitung total jenis tumbuhan unik yang dimiliki oleh Ada. Algoritma yang diterapkan di dalam Fungsi *Solve* terdiri dari beberapa fungsi yang digunakan untuk menyelesaikan permasalahan. Fungsi-fungsi tersebut adalah fungsi *Add*, *Remove*, *Change*, *Reset*. Potongan desain fungsi *Solve* terdapat pada Gambar 3.7 dan 3.8.

Masukan	Nilai setiap elemen $value[i]$, total operasi kueri, batas kiri dan kanan sekarang, batas kiri dan kanan operasi kueri, dan kondisi operasi kueri
Keluaran	Jawaban setiap operasi kueri dengan tipe 2

```
solve()
1 total = 0
2 currentL = 1
3 currentR = 0
4 for i = 0 to total_query
5   left = arr[i].L
6   right = arr[i].R
7   time = arr[i].Time
8   while currentL < left
9     remove(currentL)
10      currentL++
11    while currentL > left
12      add(currentL − 1)
13      currentL−−
14    while currentR < right
15      add(currentR + 1)
16      currentR++
```

Gambar 3.7 *Pseudocode* Fungsi *solve* Bagian 1

```
solve()
17    while currentR < right
18        remove(currentR)
19        currentR − −
20    if currentTime > time
21        reset()
22    while currentTime > time
23        change(currentTime)
24        currentTime + +
25    answer[arr[i].Number] = total
26 for i = 0 to total_query
27    output answer[i]
```

Gambar 3.8 *Pseudocode* Fungsi *solve* Bagian 2

3.1.2.3.1 Desain Fungsi *Add*

Fungsi *Add* digunakan untuk menandakan bahwa jumlah elemen yang ada di indeks tersebut ditambah 1 karena ada di dalam rentang operasi kueri sekarang. Jika jumlah elemen hanya 1 dalam rentang operasi kueri sekarang maka total jenis tumbuhan unik ditambah 1 sementara jika jumlahnya 2 maka total jenis tumbuhan unik dikurangi 1. Potongan desain fungsi *Add* terdapat pada Gambar 3.9.

Masukan	Indeks $index$
Keluaran	Total sementara untuk indeks $index$

```
Add(index)
1 counter[value[index]] + +
2 if counter[value[index]] == 1
3    total + +
4 else if counter[value[index]] == 2
5    total − −
```

Gambar 3.9 *Pseudocode* Fungsi *Add*

3.1.2.3.2 Desain Fungsi *Remove*

Fungsi *Remove* digunakan untuk menandakan bahwa jumlah elemen yang ada di indeks tersebut dikurangi 1 karena tidak ada di dalam rentang operasi kueri sekarang. Jika jumlah elemen hanya 1 dalam rentang operasi kueri sekarang maka total jenis tumbuhan unik ditambah 1 sementara jika jumlahnya 0 maka total jenis tumbuhan unik dikurangi 1. Potongan desain fungsi *Remove* terdapat pada Gambar 3.10.

Masukan	Indeks $index$
Keluaran	Total sementara untuk indeks $index$
Remove($index$) 1 $counter[value[index]] - -$ 2 **if** $counter[value[index]] == 1$ 3 $\quad total + +$ 4 **else if** $counter[value[index]] == 0$ 5 $\quad total - -$	

Gambar 3.10 *Pseudocode* Fungsi *Remove*

3.1.2.3.3 Desain Fungsi *Change*

Fungsi *Change* digunakan untuk mengubah kondisi *array* A pada kondisi yang sama dengan kondisi *array* A saat operasi kueri dimasukkan.

Masukan	Indeks $index$
Keluaran	Perubahan nilai dan total sementara
Change($index$) 1 **if** $up[index].position \geq currentL \wedge up[index].position \leq currentR$ 2 $\quad remove(up[index].position)$	

Gambar 3.11 *Pseudocode* Fungsi *Change* Bagian 1

Pada fungsi *Change* dilakukan juga fungsi *Add* dan *Remove* jika pergantian elemen terjadi di rentang operasi kueri yang sedang dijalankan. Potongan desain fungsi *Change* terdapat pada Gambar 3.11 dan 3.12.

```
Change(index)
3 value[up[index].position] = up[index].val
4 if up[index].position ≥ currentL ∧
up[index].position ≤ currentR
5   add(up[index].position)
```

Gambar 3.12 *Pseudocode* Fungsi *Change* Bagian 2

3.1.2.3.4 Desain Fungsi *Reset*

Fungsi *Reset* digunakan untuk mengembalikan kondisi *array A* pada kondisi saat awal dimasukkan ke dalam sistem. Pada fungsi *Reset* dilakukan juga fungsi *Add* dan fungsi *Remove* jika pergantian elemen terjadi pada rentang operasi kueri yang sedang dijalankan. Potongan desain fungsi *Reset* terdapat pada Gambar 3.13.

Masukan	Indeks $index$
Keluaran	Perubahan nilai dan total sementara

```
Reset()
1 currentTime = 0
2 for i = 1 to N
3   if value[i] ≠ temp[i]
4     if i ≥ currentL ∧ i ≤ currentR
5       remove(i)
6     value[i] = temp[i]
7     if i ≥ currentL ∧ i ≤ currentR
8       add(i)
```

Gambar 3.13 *Pseudocode* Fungsi *Reset*

3.2 Desain Penyelesaian Permasalahan *Angry Siam*

Pada subbab ini akan dijelaskan desain penyelesaian permasalahan *Angry Siam*

3.2.1 Definisi Umum Sistem

Sistem akan menerima masukan berupa bilangan N yang mewakili jumlah permasalahan yang akan diberikan kepada $HrSiam$. Selanjutnya sistem menerima masukan N jumlah elemen untuk *array* $A[i]$ untuk tingkat permasalahan dilanjutkan dengan N jumlah elemen untuk *array* $Angry$ untuk tingkat kemarahan $HrSiam$. Masukan keempat adalah bilangan Q yang mewakili jumlah operasi kueri yang akan dimasukkan. Selanjutnya adalah Q baris berisi 3 buah bilangan t, x, dan y yang masing masing mewakili jenis operasi kueri, batas kiri/indeks yang elemennya akan diubah, dan batas kanan/elemen baru dari *array* A. Semua masukan sesuai dengan format yang sudah ditentukan pada subbab 2.3.2 sebelumnya. Kemudian, sistem akan membagi Q jumlah operasi kueri menjadi blok-blok kecil sesuai dengan yang sudah dijelaskan pada subbab 2.2.1 dan sistem juga menyortir Q jumlah operasi kueri sesuai dengan ketentuan pada subbab 2.3.1. Kemungkinan pada subbab 2.3.1.1 juga diperhitungkan pada sistem karena terdapat operasi kueri yang dapat mengubah elemen yang ada di *array* A. Operasi kueri yang sudah disortir akan dikerjakan sesuai dengan urutannya dan untuk setiap jawaban dari satu operasi kueri akan disimpan ke dalam sebuah *array* dan digunakan kembali untuk menjawa operasi kueri selanjutnya. Hasil keluaran dari sistem adalah jawaban dari operasi kueri yang mencari jumlah kemarahan dari $HrSiam$ sesuai dengan urutan operasi kueri saat dimasukkan ke dalam sistem. Format keluaran sesuai dengan 2.3.2. *Pseudocode* dari fungsi *main* ditunjukkan oleh Gambar 3.14.

Masukan	Banyak elemen N, banyak operasi kueri Q, nilai setiap elemen $value[i]$, nilai setiap elemen $angry[i]$, operasi kueri (t, x, y)
Keluaran	Jawaban setiap operasi kueri dengan tipe 2

```
main()
1 Input N
2 blk_sz = 4000
3 for i = 1 to N
4    Input value[i]
5    temp[i] = value[i]
6 for i = 1 to N
7    Input angry[i]
8 Input Q
9 for i = 1 to Q
10   Input type
11   if type = 1
12      Input arr[total_query].L
13      Input arr[total_query].R
14      arr[total_query].Number = total_query
15      arr[total_query].Time = time
16      total_query++
17   else
18      Input up[time].position
19      Input up[time].val
20      time++
21 sort(arr, arr + total_query, compare)
22 solve()
```

Gambar 3.14 *Pseudocode* Fungsi *main*

3.2.2 Desain Algoritma

Sistem terdiri dari satu fungsi utama, yaitu fungsi *Solve* yang kemudian akan dijelaskan bagian-bagian yang ada di dalam fungsi tersebut, algoritma *Square Root Decomposition*, dan *Mo's Algorithm*. Pada subbab ini akan dijelaskan desain algoritma dari bagian-bagian fungsi tersebut dan algoritma yang digunakan.

3.2.2.1 Penentuan Konstanta *Square Root Decomposition*

Algoritma *Square Root Decomposition* bertugas untuk membagi Q jumlah operasi kueri menjadi blok-blok kecil. Caranya adalah dengan menentukan konstanta dan membagi batas kiri dan batas kanan dari setiap operasi kueri dengan konstanta tersebut. Penentuan konstanta dilakukan dengan cara uji coba dengan nilai konstanta yang variatif lalu dipilih nilai konstanta yang menghasilkan waktu penyelesaian paling cepat. Selanjutnya adalah melakukan penyortiran urutan pengerjaan operasi kueri yang sudah ditentukan menggunakan *std sort* dan fungsi *boolean compare*. Potongan desain algoritma *Square Root Decomposition* terdapat pada Gambar 3.15.

```
define_constanta()
1   blk_sz = 4000
```

Gambar 3.15 Konstanta *Square Root Decomposition*

3.2.2.2 Desain *Mo's Algorithm*

Algoritma *Mo's Algorithm* bertugas untuk mengerjakan setiap operasi kueri yang sudah mendapatkan urutan pengerjaan dengan cara kerja yang sudah dijelaskan pada subbab 2.3.1. Setiap

pergantian batas kiri dan kanan dari Q jumlah operasi kueri akan selalu dilakukan pengecekan perubahan elemen di setiap rentang dengan menggunakan fungsi *Add* dan *Remove*. Setiap pergantian kondisi akan dilakukan pemanggilan fungsi *Reset* dan *Change*. Potongan desain *Mo's Algorithm* terdapat pada Gambar 3.16.

Masukan	Total operasi kueri, batas kiri dan kanan sekarang, nilai setiap elemen $value[i]$, batas kiri dan kanan operasi kueri, dan kondisi operasi kueri
Keluaran	Jawaban setiap operasi kueri

Mo's Algorithm()
1 **for** $i = 0$ **to** $total\_query$
2 **while** $currentL < left$
3 ...
4 **while** $currentL > left$
5 ...
6 **while** $currentR < right$
7 ...
8 **while** $currentR > right$
9 ...
10 **if** $currentTime > time$
11 ...
12 **while** $currentTime > time$
13 ...
14 $answer[i] = total$
15 **for** $i = 0$ **to** $total\_query$
16 output $answer[i]$

Gambar 3.16 Desain *Mo's Algorithm*

3.2.2.2.1 Desain Fungsi *compare*

Fungsi *boolean compare* digunakan untuk menjadi argumen std

sort dengan ketentuan sebagai berikut:

1. Prioritas pertama adalah operasi kueri dengan batas kiri paling kecil di blok awal.
2. Prioritas kedua adalah operasi kueri dengan batas kanan paling kecil di blok awal.
3. Prioritas ketiga adalah operasi kueri yang memiliki kondisi paling awal.

Potongan desain fungsi *boolean Compare* terdapat pada Gambar 3.17

Masukan	Kueri x, Kueri y
Keluaran	Argumen untuk fungsi $sort$
compare(x, y) 1 **if** $x.L/blk\_sz \neq y.L/blk\_sz$ 2 **return** $x.L/blk\_sz < y.L/blk\_sz$ 3 **if** $x.R/blk\_sz \neq y.R/blk\_sz$ 4 **return** $x.R/blk\_sz < y.R/blk\_sz$ 5 **return** $x.Time < y.Time$	

Gambar 3.17 *Pseudocode* Fungsi *Boolean compare*

3.2.2.3 Desain Fungsi *solve*

Fungsi *solve* digunakan untuk menghitung total kemarahan dari $HrSiam$ untuk rentang operasi kueri yang sedang dijalankan. Algoritma yang diterapkan di dalam Fungsi *Solve* terdiri dari beberapa fungsi yang digunakan untuk menyelesaikan permasalahan. Fungsi-fungsi tersebut adalah fungsi *Add*, *Remove*, *Change*, *Reset*. Potongan desain fungsi *solve* terdapat pada Gambar 3.18.

Masukan	Nilai setiap elemen $value[i]$, total operasi kueri, batas kiri dan kanan sekarang, batas kiri dan kanan operasi kueri, dan kondisi operasi kueri
Keluaran	Jawaban setiap operasi kueri dengan tipe 1

```
solve()
1 total = 0
2 currentL = 1
3 currentR = 0
4 for i = 0 to total_query
5    left = arr[i].L
6    right = arr[i].R
7    time = arr[i].Time
8    while currentL < left
9       remove(currentL)
10         currentL++
11    while currentL > left
12         add(currentL − 1)
13         currentL−−
14    while currentR < right
15         add(currentR + 1)
16         currentR++
17    while currentR < right
18         remove(currentR)
19         currentR−−
20    if currentTime > time
21         reset()
22    while currentTime > time
23         change(currentTime)
24         currentTime++
25    answer[arr[i].Number] = total
26 for i = 0 to total_query
27    output answer[i]
```

Gambar 3.18 *Pseudocode* Fungsi *solve*

3.2.2.3.1 Desain Fungsi *Add*

Fungsi *Add* digunakan untuk menandakan bahwa jumlah elemen yang ada di indeks tersebut ditambah 1 karena ada di dalam rentang operasi kueri sekarang. Nilai yang berubah atau dapat dikatakan $counter$ akan menjadi indeks yang menunjukkan tingkat kemarahan dari $HrSiam$. Total kemarahan dari $HrSiam$ adalah hasil penjumlahan nilai permasalahan dikali dengan tingkat kemarahan $HrSiam$ yang menggunakan indeks $counter$. Potongan desain fungsi *Add* terdapat pada Gambar 3.19.

Masukan	Indeks $index$
Keluaran	Total sementara untuk indeks $index$
Add($index$) 1 $counter[value[index]]++$ 2 $total+ = angry[counter[value[index]]] * value[index]$	

Gambar 3.19 *Pseudocode* Fungsi *Add*

3.2.2.3.2 Desain Fungsi *Remove*

Fungsi *Remove* digunakan untuk menandakan bahwa jumlah elemen yang ada di indeks tersebut dikurangi 1 karena tidak di dalam rentang operasi kueri sekarang. Nilai yang berubah atau dapat dikatakan $counter$ akan menjadi indeks yang menunjukkan tingkat kemarahan dari $HrSiam$.

Masukan	Indeks $index$
Keluaran	Total sementara untuk indeks $index$
Remove($index$) 1 $total- = angry[counter[value[index]]] * value[index]$ 2 $counter[value[index]]--$	

Gambar 3.20 *Pseudocode* Fungsi *Remove*

Total kemarahan dari $HrSiam$ adalah hasil pengurangan nilai permasalahan dikali dengan tingkat kemarahan $HrSiam$ yang menggunakan indeks $counter$. Dalam fungsi *Remove*, yang dilakukan pertama kali adalah pengurangan dengan hasil perkalian dan mengurangi nilai $counter$ tersebut. Potongan desain fungsi *Remove* terdapat pada Gambar 3.20.

3.2.2.3.3 Desain Fungsi *Change*

Fungsi *Change* digunakan untuk mengubah kondisi *array* A pada kondisi yang sama dengan kondisi *array* A saat operasi kueri dimasukkan. Pada fungsi *Change* dilakukan juga fungsi *Add* dan fungsi *Remove* jika pergantian elemen terjadi pada rentang operasi kueri yang sedang dijalankan. Potongan desain fungsi *Change* terdapat pada Gambar 3.21.

Masukan	Indeks $index$
Keluaran	Perubahan nilai dan total sementara
Change($index$) 1 **if** $up[index].position \geq currentL \wedge up[index].position \leq currentR$ 2 $remove(up[index].position)$ 3 $value[up[index].position] = up[index].val$ 4 **if** $up[index].position \geq currentL \wedge up[index].position \leq currentR$ 5 $add(up[index].position)$	

Gambar 3.21 *Pseudocode* Fungsi *Change*

3.2.2.3.4 Desain Fungsi *Reset*

Fungsi *Reset* digunakan untuk mengembalikan kondisi *array* A pada kondisi saat awal dimasukkan ke dalam sistem. Pada fungsi *Reset* dilakukan juga fungsi *Add* dan fungsi *Remove* jika pergantian elemen terjadi pada rentang operasi kueri yang sedang dijalankan.

Potongan desain fungsi *Reset* terdapat pada Gambar 3.22.

Masukan	Indeks $index$
Keluaran	Perubahan nilai dan total sementara

```
Reset()
1 currentTime = 0
2 for i = 1 to N
3   if value[i] ≠ temp[i]
4     if i ≥ currentL ∧ i ≤ currentR
5       remove(i)
6     value[i] = temp[i]
7     if i ≥ currentL ∧ i ≤ currentR
8       add(i)
```

Gambar 3.22 *Pseudocode* Fungsi *Reset*

3.3 Desain Penyelesaian Permasalahan *Counting diff-pairs*

Pada subbab ini akan dijelaskan desain penyelesaian permasalahan *Counting diff-pairs*

3.3.1 Definisi Umum Sistem

Sistem akan menerima masukan berupa tiga buah bilangan N, M, dan k yang masing-masing mewakili N untuk menyimpan jumlah elemen dari *array A*, M untuk menyimpan jumlah operasi kueri, dan k untuk menyimpan selisih absolut yang diinginkan. Selanjutnya sistem menerima masukan bilangan sejumlah N sebagai sebuah *array A* dan diikuti oleh M baris berisi 2 buah bilangan l dan r yang masing-masing mewakili batas kiri dan batas kanan operasi kueri. Semua masukan sesuai dengan format yang sudah ditentukan pada subbab 2.5 sebelumnya. Kemudian, sistem akan membagi M jumlah operasi kueri menjadi blok-blok kecil sesuai dengan yang sudah dijelaskan pada subbab 2.2.1 dan

sistem juga menyortir M jumlah operasi kueri sesuai dengan ketentuan pada subbab 2.3.1. Operasi kueri yang sudah disortir akan dikerjakan sesuai dengan urutannya dan untuk setiap jawaban dari satu operasi kueri akan disimpan ke dalam sebuah *array* dan digunakan kembali untuk menjawab operasi kueri selanjutnya. Hasil keluaran dari sistem adalah jawaban dari operasi kueri yang mencari jumlah pasangan angka yang memiliki selisih absolut bernilai k atau selisih yang lebih besar dari k dalam rentang tertentu sesuai dengan urutan operasi kueri saat dimasukkan ke dalam sistem. Format keluaran sesuai dengan subbab 2.5. *Pseudocode* dari fungsi *main* ditunjukkan oleh Gambar 3.23 dan 3.24.

Masukan	Banyak elemen N, banyak operasi kueri M, selisih absolut k, nilai setiap elemen $value[i]$, dan operasi kueri (x, y)
Keluaran	Jawaban setiap operasi kueri

```
main()
1 Input N
2 Input M
3 Input k
4 blk_sz = √N
5 for i = 0 to N
6    Input value[i]
7 for i = 1 to M
8    Input arr[i].L
9    Input arr[i].R
10    arr[i].L − −
11    arr[i].R − −
12    arr[i].Number = i
```

Gambar 3.23 *Pseudocode* Fungsi *main* Bagian 1

```
13 sort(arr, arr + M, compare)
14 CPair()
```

Gambar 3.24 *Pseudocode* Fungsi *main* Bagian 2

3.3.2 Desain Algoritma

Sistem terdiri dari empat fungsi utama, yaitu fungsi *Update*, *Read*, *Get*, dan *Cpair* yang kemudian akan dijelaskan bagian-bagian yang ada di dalam fungsi tersebut, algoritma *Square Root Decomposition*, dan *Mo's Algorithm*. Selain itu, sistem juga menggunakan *Fenwick Tree* sebagai struktur data. Pada subbab ini akan dijelaskan desain algoritma dari fungsi, algoritma, dan struktur data yang digunakan.

3.3.2.1 Penentuan Konstanta *Square Root Decomposition*

Algoritma *Square Root Decomposition* bertugas untuk membagi M jumlah operasi kueri menjadi blok-blok kecil. Caranya adalah dengan menentukan konstanta dan membagi batas kiri dan batas kanan dari setiap operasi kueri dengan konstanta tersebut. Penentuan konstanta dilakukan dengan cara uji coba dengan nilai konstanta yang variatif lalu dipilih nilai konstanta yang menghasilkan waktu penyelesaian paling cepat. Selanjutnya adalah melakukan penyortiran urutan pengerjaan operasi kueri yang sudah ditentukan menggunakan $std\ sort$ dan fungsi *boolean compare*.

```
define_constanta()
1   blk_sz = √N
```

Gambar 3.25 Konstanta *Square Root Decomposition*

Potongan desain algoritma *Square Root Decomposition* terdapat pada Gambar 3.25

3.3.2.2 Desain Algoritma *Mo's Algorithm*

Algoritma *Mo's Algorithm* bertugas untuk mengerjakan setiap operasi kueri yang sudah mendapatkan urutan pengerjaan dengan cara kerja yang sudah dijelaskan pada subbab 2.3.1. Setiap pergantian batas kiri dan kanan dari M jumlah operasi kueri akan selalu dilakukan pengecekan perubahan elemen di setiap rentang dengan menggunakan fungsi *Get* dan untuk setiap elemen yang sedang dicek akan dilakukan perubahan pada *Fenwick Tree* $BIT[]$ dengan menggunkana fungsi *Update*.

Masukan	Total operasi kueri, batas kiri dan kanan sekarang, nilai setiap elemen $value[i]$, batas kiri dan kanan operasi kueri, dan kondisi operasi kueri
Keluaran	Jawaban setiap operasi kueri

Mo's Algorithm()
1 **for** $i = 0$ **to** M
2 **while** $currentL < left$
3 ...
4 **while** $currentL > left$
5 ...
6 **while** $currentR < right$
7 ...
8 **while** $currentR > right$
9 ...
10 $answer[i] = total$
11 **for** $i = 0$ **to** M
12 output $answer[i]$

Gambar 3.26 Desain *Mo's Algorithm*

Potongan desain *Mo's Algorithm* terdapat pada Gambar 3.26.

3.3.2.2.1 Desain Fungsi *compare*

Fungsi *boolean compare* digunakan untuk menjadi argumen std *sort* dengan ketentuan sebagai berikut:

1. Prioritas pertama adalah operasi kueri dengan batas kiri paling kecil di blok awal.
2. Prioritas kedua adalah operasi kueri dengan batas kanan paling kecil di blok awal.

Potongan desain fungsi *boolean Compare* terdapat pada Gambar 3.27.

Masukan	Kueri x, Kueri y
Keluaran	Argumen untuk fungsi $sort$
compare(x, y) 1 **if** $x.L/blk\_sz \neq y.L/blk\_sz$ 2 **return** $x.L/blk\_sz < y.L/blk\_sz$ 5 **return** $x.R/blk\_sz < y.R/blk\_sz$	

Gambar 3.27 *Pseudocode* Fungsi *Boolean compare*

3.3.2.3 Desain Fungsi *CPair*

Fungsi *CPair* digunakan untuk menghitung jumlah pasangan bilangan yang memiliki selisih absolut sama dengan atau lebih dari k untuk rentang operasi kueri yang sedang dijalankan. Fungsi *Cpair* mengimplementasi *Mo's Algorithm* dan di dalam fungsi tersebut juga terjadi pemanggilan-pemanggilan fungsi *Get* dan *Update*. Hasil dari fungsi ini adalah jawaban dari permasalahan *Counting diff-pairs*. Potongan desain fungsi *Cpair* terdapat pada Gambar 3.28.

Masukan	Nilai setiap elemen $value[i]$, total operasi kueri, batas kiri dan kanan sekarang, dan batas kiri dan kanan operasi kueri
Keluaran	Jawaban setiap operasi kueri

```
CPair()
1 total = 0
2 currentL = 0
3 currentR = −1
4 for i = 0 to M
5    while currentL < arr[i].L
6       total− = Get(value[currentL])
7       Update(value[currentL], −1)
8       currentL + +
9    while currentL > arr[i].L
10      total+ = Get(value[currentL − 1])
11      Update(value[currentL − 1], 1)
12      currentL − −
13   while currentR < right
14      total+ = Get(value[currentR + 1])
15      Update(value[currentR + 1], 1)
16      currentR + +
17   while currentR < right
18      total− = Get(value[currentR])
19      Update(value[currentR], −1)
20      currentR − −
21   answer[arr[i].Number] = total
22 for i = 0 to M
23   output answer[i]
```

Gambar 3.28 *Pseudocode* Fungsi *CPair*

3.3.2.4 Desain Fungsi *Update*

Fungsi *Update* digunakan untuk mengubah nilai yang terdapat di dalam *Fenwick Tree* $BIT[idx]$ dan semua indeks yang berkaitan dengan indeks idx dalam bentuk biner. Mengubah semua bilangan yang berkaitan dengan suatu bilangan dalam bentuk biner dapat dilihat pada subbab 2.4.2.2. Fungsi *Update* memiliki argumen *integer* idx dan *integer* val. idx mewakili indeks dan val mewakili bilangan yang akan mengubah nilai dari $BIT[idx]$. Potongan desain fungsi *Update* terdapat pada Gambar 3.29.

Masukan	Indeks idx dan nilai yang akan merubah nilai indeks $index$
Keluaran	Perubahan nilai indeks $index$ dan indeks yang berhubungan dalam bentuk biner
Update(idx, val) 1 **for each** idx **to** mv 2 $\quad BIT[idx] += val$ 3 $\quad idx += idx\& - idx$	

Gambar 3.29 *Pseudocode* Fungsi *Update*

3.3.2.5 Desain Fungsi *Read*

Fungsi *Read* digunakan untuk menghitung nilai frekuensi kumulatif dari *Fenwick Tree* $BIT[]$ dengan indeks idx. Penghitungan frekuensi kumulatif dilakukan dengan menjumlahkan nilai dari semua $BIT[]$ yang berkaitan dengan indeks idx dalam bentuk biner. Penjumlahan frekuensi kumulatif semua bilangan yang berkaitan dengan suatu bilangan dalam bentuk biner dapat dilihat pada subbab 2.4.2.1. Fungsi *Read* memiliki argumen *integer* idx yang mewakili indeks. Potongan desain fungsi *Read* terdapat pada Gambar 3.30.

Masukan	Indeks idx
Keluaran	Total frekuensi kumulatif untuk indeks idx
Read(idx) 1 $sum = 0$ 2 **for each** idx **to** 0 3 $sum+ = BIT[idx]$ 4 $idx- = idx\& - idx$ 5 **return** sum	

Gambar 3.30 *Pseudocode* Fungsi *Read*

3.3.2.6 Desain Fungsi *Get*

Fungsi *Get* digunakan untuk menghitung jumlah pasangan bilangan yang memiliki selisih absolut k atau selisih yang lebih dari k. Fungsi *Get* memiliki argumen *integer* idx yang mewakili indeks. Terdapat tiga langkah untuk menemukan jumlah pasangan bilangan yang memiliki selisih absolut k atau lebih besar dari k yang ada di dalam fungsi *Get* yang dapat dilihat sebagai berikut:

1. Menghitung jumlah frekuensi kumulatif pada $BIT[]$ dengan indeks $idx - k$. Hasil dari penghitungan ini adalah jumlah bilangan yang memiliki selisih absolut K dengan indeks yang sedang dilakukan pengecekan. Dalam langkah ini akan dipanggil fungsi *Read* yang sudah dijelaskan pada subbab 3.3.2.5.
2. Menghitung jumlah frekuensi kumulatif pada $BIT[]$ dengan indeks mv dan $idx+k-1$. Dalam langkah ini akan dipanggil fungsi *Read* yang sudah dijelaskan pada subbab 3.3.2.5. Hasil dari $Read(mv)$ akan dikurangi oleh $Read(idx+k-1)$. Hasil tersebut adalah jumlah bilangan yang memiliki selisih absolut sama dengan atau lebih dari k.
3. Hasil dari langkah pertama dan kedua akan dijumlahkan dan

menjadi jumlah pasangan bilangan yang memiliki selisih absolut sama dengan atau lebih dari k untuk indeks idx.

Langkah-langkah tersebut sudah dijelaskan pada subbab 2.6. Potongan desain fungsi *Read* terdapat pada Gambar 3.31.

Masukan	Indeks idx
Keluaran	Total pasangan dengan selisih absolut sama dengan atau lebih dari k
Get(idx) 1 $sum = 0$ 2 $y = max(0, idx - k)$ 3 $sum = Read(y)$ 4 $y = min(mv, idx + k - 1)$ 5 $sum+ = Read(mv) - Read(y)$ 6 **return** sum	

Gambar 3.31 *Pseudocode* Fungsi *Get*

3.3.2.7 Desain Data *Generator*

Data *generator* digunakan untuk menghasilkan format masukan sesuai dengan deskripsi permasalahan seperti yang telah dijelaskan pada subbab 2.5. *Pseudocode* dari data *generator* ditunjukkan pada Gambar 3.32 dan 3.33.

Masukan	Banyak elemen N, banyak operasi kueri M, selisih absolut K
Keluaran	Banyak elemen N, banyak operasi kueri M, selisih absolut K, data *random* untuk $arr[i]$, dan data *random* operasi kueri (x,y)

Gambar 3.32 *Pseudocode* Fungsi *Generator* Bagian 1

```
main()
1 Input N
2 Input M
3 Input K
4 Output N
5 Output M
6 Output K
7 for each i to N
8    random = rand()%100000
9    if random = 0
10       random++
11   if i < N − 1
12       Output random with space
13   else
14       Output random with enter
15 for each j to M
16   a = rand()%N
17   b = rand()%N
18   if a = 0
19       a++
20   if b = 0
21       b++
22   if j = M − 1
23       if a > b
24           Output b a
25       else
26           Output a b
27   else
28       if a > b
29           Output b a with enter
30       else
31           Output a b with enter
```

Gambar 3.33 *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 yang sudah ada 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*: Orwell Bloodshed Dev-C++ 5.11.
    - *Compiler*: g++ (TDM-GCC 4.8.1 32-bit).
    - Bahasa Pemrograman: C++.

4.2 Implementasi Penyelesaian Permasalahan *Ada and Unique Vegetable*

Pada subbab ini akan dijelaskan mengenai implementasi dari penyelesaian permasalahan *Ada and Unique Vegetable* yang telah dibuat pada subbab 3.1.

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

Pada subsubbab ini akan dibahas penggunaan *library*, konstanta, *template*, *struct*, dan variabel global yang digunakan dalam sistem. Potongan kode yang menyatakan penggunaan *library* dan konstanta dapat dilihat pada Kode Sumber 4.1.

```
1  #include <cmath>
2  #include <cstdio>
3  #include <algorithm>
4  #define n 200005
```

Kode Sumber 4.1 Penggunaan *Library* dan Konstanta pada Penyelesaian Permasalahan *Ada and Unique Vegetable*

Untuk menyelesaikan permasalahan *Ada and Unique Vegetable* dibutuhkan tiga *library* yaitu $cmath$, $cstudio$, dan $algorithm$. Didefinisikan konstanta n yang bernilai 200005 sebagai jumlah maksimal elemen untuk panjang parit dari Ada, jumlah operasi kueri yang dapat dimasukkan, dan besar maksimal nilai per tanaman Ada. Pada tabel berikut dijelaskan mengenai variabel-variabel global yang akan digunakan dalam implementasi program.

Tabel 4.1 Tabel Daftar Variabel Global Permaslahaan *Ada and Unique Vegetable* Bagian 1

No	Nama Variabel	Tipe	Penjelasan
1	$value$	$int[]$	Digunakan untuk menyimpan nilai setiap tumbuhan yang dimiliki oleh Ada
2	$temp$	$int[]$	Digunakan untuk menyimpan nilai awal setiap tumbuhan yang dimiliki oleh Ada

Tabel 4.2 Tabel Daftar Variabel Global Permasalahan *Ada and Unique Vegetable* Bagian 2

No	Nama Variabel	Tipe	Penjelasan
3	$counter$	$int[]$	Digunakan untuk menyimpan jumlah setiap tumbuhan yang ada di dalam rentang operasi kueri yang sedang dijalankan
4	$answer$	$int[]$	Digunakan untuk menyimpan jumlah unik dari tumbuhan yang ada di semua rentang operasi kueri
5	$total$	int	Digunakan untuk menyimpan jumlah sementara dari tumbuhan unik yang ada di rentang operasi kueri yang sedang dijalankan
6	N	int	Digunakan untuk menyimpan panjang parit yang dimiliki Ada
7	Q	int	Digunakan untuk menyimpan jumlah operasi kueri yang akan dijalankan
8	blk_sz	int	Digunakan untuk menyimpan nilai konstanta yang akan membagi operasi kueri menjadi blok-blok kecil
9	$currentL$	int	Digunakan untuk menyimpan nilai batas kiri dari *Mo's Algorithm*
10	$currentR$	int	Digunakan untuk menyimpan nilai batas kanan dari *Mo's Algorithm*
11	$currentTime$	int	Digunakan untuk menyimpan kondisi sekarang dari *Mo's Algorithm*

Tabel 4.3 Tabel Daftar Variabel Global Permasalahan *Ada and Unique Vegetable* Bagian 3

No	Nama Variabel	Tipe	Penjelasan
12	*type*	*int*	Digunakan untuk menyimpan tipe operasi kueri
13	*total_query*	*int*	Digunakan untuk menyimpan jumlah operasi kueri yang memiliki tipe kedua
14	*time*	*int*	Digunakan untuk menyimpan kondisi dari operasi kueri

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

```
1  int value[n], temp[n], counter[n], answer[n];
2  int total, N, Q, blk_sz, currentL, currentR;
3  int currentTime, type, total_query = 0, time = 0;
```

Kode Sumber 4.2 Penggunaan Variabel Global pada Penyelesaian Permasalahan *Ada and Unique Vegetable*

Untuk *template* yang digunakan adalah *template* untuk mempercepat proses memasukkan data ke sistem (*Fast IO*). Potongan kode untuk *template Fast IO* dapat dilihat di Kode Sumber 4.3 dan 4.4.

```
1  template<typename T>
2  T getNum(){
3    T res=0;
4    char c;
5    while(1) {
6      c=getchar();
7      if(c==' ' || c=='\n') continue;
```

Kode Sumber 4.3 Penggunaan *Template* pada Penyelesaian Permasalahan *Ada and Unique Vegetable* Bagian 1

```
1       else break;
2     }
3     res=c-'0';
4     while(1) {
5       c=getchar();
6       if(c>='0' && c<='9')
7         res=10*res+c-'0';
8       else break;
9     }
10    return res;
11  }
```

Kode Sumber 4.4 Penggunaan *Template* pada Penyelesaian Permasalahan *Ada and Unique Vegetable* Bagian 2

Terdapat dua *struct* dalam sistem yaitu *struct* $Query$ dan $Update$. *Struct* $Query$ digunakan untuk menyimpan batas kiri dan kanan, kondisi, dan urutan masuk operasi kueri. *Struct* $Query$ menyimpan semua operasi kueri yang bertipe 2 atau dapat dilihat pada subbab 2.2.2. Potongan kode untuk *struct* $Query$ dapat dilihat di Kode Sumber 4.5.

```
1  struct Query
2  {
3    int L,R, Time, Number;
4  } arr[n];
```

Kode Sumber 4.5 *Struct* $Query$ pada Penyelesaian Permasalahan *Ada and Unique Vegetable*

```
1  struct Update
2  {
3    int position, val;
4  } up[n];
```

Kode Sumber 4.6 *Struct* $Update$ pada Penyelesaian Permasalahan *Ada and Unique Vegetable*

Struct $Update$ digunakan untuk menyimpan operasi kueri yang bertipe 1. *Struct* $Update$ berisi indeks elemen yang akan diubah dan elemen baru. Potongan kode untuk *struct* $Update$ dapat dilihat di Kode Sumber 4.6.

4.2.2 Implementasi Fungsi *Boolean compare*

Fungsi *boolean compare* diimplementasikan sesuai dengan *pseudocode* pada subbab 3.1.2.2.1. Fungsi *boolean compare* yang diimplementasikan seperti pada Kode Sumber 4.7 digunakan untuk menjadi argumen *std sort* dengan ketentuan yang sudah dijelaskan sebelumnya.

```
1  bool compare(Query x, Query y)
2  {
3    if (x.L/blk_sz != y.L/blk_sz)
4      return x.L/blk_sz < y.L/blk_sz;
5    if (x.R/blk_sz != y.R/blk_sz)
6      return x.R/blk_sz < y.R/blk_sz;
7    return x.Time < y.Time;
8  }
```

Kode Sumber 4.7 Fungsi *Boolean compare* pada Penyelesaian Permasalahan *Ada and Unique Vegetable*

4.2.3 Implementasi Fungsi *main*

Fungsi *main* diimplementasikan sesuai dengan *pseudocode* pada subbab 3.1.1. Fungsi *main* yang diimplementasikan seperti pada Kode Sumber 4.8 dan 4.9 digunakan untuk membaca masukan dari sistem menggunakan *template Fast IO* dan menentukan konstanta untuk algoritma *Square Root Decomposition*. Untuk setiap operasi kueri bertipe 1 akan dimasukkan ke dalam *struct* $Update$ dan yang bertipe 2 akan dimasukkan ke dalam *struct* $Query$. Seluruh indeks yang digunakan untuk menunjukkan letak elemen pada *struct* akan ditambah 1 karena format masukkan dimulai dari indeks

0 sementara dalam sistem akan menggunakan indeks 1 sebagai awalan.

```
int main()
{
  N = getNum<int>();
  Q = getNum<int>();
  blk_sz = 5000;
  for(int i = 1; i <= N ; i++)
  {
    value[i] = getNum<int>();
    temp[i] = value[i];
  }
  for(int i = 1 ; i<= Q ; i++)
  {
    type = getNum<int>();;
    if(type == 1)
    {
      up[time].position=
        getNum<int>();
      up[time].val =
        getNum<int>();
      up[time].position++;
      time++;
    }
    else
    {
      arr[total_query].L =
        getNum<int>();
      arr[total_query].R =
        getNum<int>();
      arr[total_query].L++;
      arr[total_query].R++;
      arr[total_query].Number =
        total_query;
      arr[total_query].Time =
        time;
```

Kode Sumber 4.8 Fungsi *main* 1 pada Penyelesaian Permasalahan *Ada and Unique Vegetable*

```
1         total_query++;
2       }
3     }
4     sort(arr,arr+total_query,compare);
5     solve();
6   }
```

Kode Sumber 4.9 Fungsi *main* 2 pada Penyelesaian Permasalahan *Ada and Unique Vegetable*

4.2.4 Implementasi Fungsi *Add*

Fungsi *Add* diimplementasikan pada Kode Sumber 4.10 dan 4.11 sesuai dengan *pseudocode* pada subbab 3.1.2.3.1. Fungsi ini digunakan saat sebuah elemen pada indeks tertentu ada di dalam rentang operasi kueri yang sekarang sedang dicek sesuai dengan deskripsi desain yang telah dijelaskan sebelumnya.

```
1         counter[value[currentL]]++;
2         if(counter[value[currentL]] == 1)
3           total++;
4         else if (counter[value[currentL]] == 2)
5           total--;
```

Kode Sumber 4.10 Fungsi *Add* 1 pada Penyelesaian Permasalahan *Ada and Unique Vegetable*

```
1         counter[value[currentR]]++;
2         if(counter[value[currentR]] == 1)
3           total++;
4         else if (counter[value[currentR]] == 2)
5           total--;
```

Kode Sumber 4.11 Fungsi *Add* 2 pada Penyelesaian Permasalahan *Ada and Unique Vegetable*

4.2.5 Implementasi Fungsi *Remove*

Fungsi *Remove* diimplementasikan pada Kode Sumber 4.12 dan 4.13 sesuai dengan *pseudocode* pada subbab 3.1.2.3.2. Fungsi ini digunakan saat sebuah elemen pada indeks tertentu tidak di dalam rentang operasi kueri yang sekarang sedang dicek sesuai dengan deskripsi desain yang telah dijelaskan sebelumnya.

```
1         counter[value[currentL]]--;
2         if(counter[value[currentL]] == 1)
3           total++;
4         else if (counter[value[currentL]] == 0)
5           total--;
```

Kode Sumber 4.12 Fungsi *Remove* 1 pada Penyelesaian Permasalahan *Ada and Unique Vegetable*

```
1         counter[value[currentR]]--;
2         if(counter[value[currentR]] == 1)
3           total++;
4         else if (counter[value[currentR]] == 0)
5           total--;
```

Kode Sumber 4.13 Fungsi *Remove* 2 pada Penyelesaian Permasalahan *Ada and Unique Vegetable*

4.2.6 Implementasi Fungsi *Change*

Fungsi *Change* diimplementasikan pada Kode Sumber 4.14 dan 4.15 sesuai dengan *pseudocode* pada subbab 3.1.2.3.3. Fungsi ini digunakan jika kondisi sekarang berbeda dengan kondisi saat kueri dimasukkan sesuai dengan deskripsi desain yang telah dijelaskan sebelumnya.

```
1         if(up[currentTime].position >= currentL &&
```

Kode Sumber 4.14 Fungsi *Change* pada Penyelesaian Permasalahan *Ada and Unique Vegetable* Bagian 1

```
1           up[currentTime].position <= currentR)
2         {
3           counter[value[
4             up[currentTime].position]]--;
5           if(counter[value[
6             up[currentTime].position]] == 1)
7               total++;
8           else if (counter[value[
9             up[currentTime].position]] == 0)
10              total--;
11        }
12        value[up[currentTime].position] =
13          up[currentTime].val;
14        if(up[currentTime].position >= currentL &&
15          up[currentTime].position <= currentR)
16        {
17          counter[value[
18            up[currentTime].position]]++;
19          if(counter[value[
20            up[currentTime].position]] == 1)
21              total++;
22          else if (counter[value[
23            up[currentTime].position]] == 2)
24              total--;
25        }
```

Kode Sumber 4.15 Fungsi *Change* pada Penyelesaian Permasalahan *Ada and Unique Vegetable* Bagian 2

4.2.7 Implementasi Fungsi *Reset*

Fungsi *Reset* diimplementasikan pada Kode Sumber 4.16 sesuai dengan *pseudocode* pada subbab 3.1.2.3.4. Fungsi ini digunakan jika kondisi sekarang lebih baru dari kondisi saat kueri dimasukkan sehingga kondisi dari *array A* harus dikembalikan seperti semula sesuai dengan deskripsi desain yang telah dijelaskan sebelumnya.

```
1         currentTime = 0;
2         for(int i=1;i<=N;i++)
3         {
4           if(value[i] != temp[i])
5           {
6             if(i >= currentL && i <= currentR)
7             {
8               counter[value[i]]--;
9               if(counter[value[i]] == 1)
10                total++;
11              else if (counter[value[i]] == 0)
12                total--;
13            }
14            value[i] = temp[i];
15            if(i >= currentL && i <= currentR)
16            {
17              counter[value[i]]++;
18              if(counter[value[i]] == 1)
19                total++;
20              else if (counter[value[i]] == 2)
21                total--;
22            }
23          }
24        }
```

Kode Sumber 4.16 Fungsi *Reset* pada Penyelesaian Permasalahan *Ada and Unique Vegetable*

4.2.8 Implementasi Fungsi *Solve*

Fungsi *Solve* diimplementasikan pada Kode Sumber 4.17, 4.18, dan 4.19 sesuai dengan *pseudocode* pada subbab 3.1.2.3. Fungsi ini digunakan untuk menghitung total jenis tumbuhan unik dalam rentang operasi kueri. Fungsi *Solve* terdiri dari beberapa fungsi yang digunakan untuk menyelesaikan permasalahan. Fungsi-fungsi tersebut adalah fungsi *Add*, *Remove*, *Change*, *Reset* yang sudah dijelaskan pada bab-bab sebelumnya.

```
1  void solve()
2  {
3    total = 0;
4    currentL = 1;
5    currentR = 0;
6    for(int i = 0 ; i < total_query ; i++)
7    {
8      int left = arr[i].L;
9      int right = arr[i].R;
10     time = arr[i].Time;
11     while(currentL < left)
12     {
13       counter[value[currentL]]--;
14       if(counter[value[currentL]] == 1)
15         total++;
16       else if (counter[value[currentL]] == 0)
17         total--;
18       currentL++;
19     }
20     while(currentL > left)
21     {
22       currentL--;
23       counter[value[currentL]]++;
24       if(counter[value[currentL]] == 1)
25         total++;
26       else if (counter[value[currentL]] == 2)
27         total--;
28     }
29     while(currentR < right)
30     {
31       currentR++;
32       counter[value[currentR]]++;
33       if(counter[value[currentR]] == 1)
34         total++;
35       else if (counter[value[currentR]] == 2)
36         total--;
```

Kode Sumber 4.17 Fungsi *Solve* 1 pada Penyelesaian Permasalahan *Ada and Unique Vegetable*

```
    }
    while(currentR > right)
    {
      counter[value[currentR]]--;
      if(counter[value[currentR]] == 1)
        total++;
      else if (counter[value[currentR]] == 0)
        total--;
      currentR--;
    }
    if(currentTime > time)
    {
      currentTime = 0;
      for(int i=1;i<=N;i++)
      {
        if(value[i] != temp[i])
        {
          if(i >= currentL && i <= currentR)
          {
            counter[value[i]]--;
            if(counter[value[i]] == 1)
              total++;
            else if (counter[value[i]] == 0)
              total--;
          }
          value[i] = temp[i];
          if(i >= currentL && i <= currentR)
          {
            counter[value[i]]++;
            if(counter[value[i]] == 1)
              total++;
            else if (counter[value[i]] == 2)
              total--;
          }
        }
      }
    }
```

Kode Sumber 4.18 Fungsi *Solve* 2 pada Penyelesaian Permasalahan *Ada and Unique Vegetable*

```
      while(currentTime < time)
      {
        if(up[currentTime].position >= currentL &&
          up[currentTime].position <= currentR)
        {
          counter[value[
            up[currentTime].position]]--;
          if(counter[value[
            up[currentTime].position]] == 1)
              total++;
          else if (counter[value[
            up[currentTime].position]] == 0)
              total--;
        }
        value[up[currentTime].position] =
          up[currentTime].val;
        if(up[currentTime].position >= currentL &&
          up[currentTime].position <= currentR)
        {
          counter[value[
            up[currentTime].position]]++;
          if(counter[value[
            up[currentTime].position]] == 1)
              total++;
          else if (counter[value[
            up[currentTime].position]] == 2)
              total--;
        }
        currentTime++;
      }
      answer[arr[i].Number] = total;
    }
    for(int i = 0 ; i < total_query ; i++)
      printf("%d\n",answer[i]);
  }
```

Kode Sumber 4.19 Fungsi *Solve* 3 pada Penyelesaian Permasalahan *Ada and Unique Vegetable*

4.3 Implementasi Penyelesaian Permasalahan *Angry Siam*

Pada subbab ini akan dijelaskan mengenai implementasi dari penyelesaian permasalahan *Angry Siam* yang telah dibuat pada subbab 3.2.

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

Pada subbab ini akan dibahas penggunaan *library*, konstanta, *template*, *struct*, dan variabel global yang digunakan dalam sistem. Berikut adalah potongan kode yang menyatakan penggunaan *library* dan konstanta 4.20.

```
1  #include <cmath>
2  #include <cstdio>
3  #include <algorithm>
4  #define n 100005
```

Kode Sumber 4.20 *Library* dan Konstanta pada Penyelesaian Permasalahan *Angry Siam*

Untuk menyelesaikan permasalahan *Angry Siam* dibutuhkan tiga *library* yaitu $cmath$, $cstudio$, dan $algorithm$. Didefinisikan konstanta n yang bernilai 100005 sebagai jumlah maksimal elemen untuk permasalahan dan tingkat kemarahan dari $HrSiam$, jumlah operasi kueri yang dapat dimasukkan, dan besar maksimal nilai per permasalahan $HrSiam$. Pada tabel berikut dijelaskan mengenai variabel-variabel global yang akan digunakan dalam implementasi program.

Tabel 4.4 Tabel Daftar Variabel Global Permaslahaan *Angry Siam* Bagian 1

No	Nama Variabel	Tipe	Penjelasan
1	$value$	$longlong[]$	Digunakan untuk menyimpan nilai setiap permasalan yang akan diberikan ke $HrSiam$
2	$temp$	$longlong[]$	Digunakan untuk menyimpan nilai awal setiap permasalahn yang diberikan ke $HrSiam$
3	$counter$	$longlong[]$	Digunakan untuk menyimpan jumlah setiap permasalahan yang sama tingkat kesulitannya dan ada di dalam rentang operasi kueri yang sedang dijalankan
4	$answer$	$longlong[]$	Digunakan untuk menyimpan total kemarahan $HrSiam$ untuk setiap rentang operasi kueri
5	$total$	$longlong$	Digunakan untuk menyimpan jumlah sementara dari kemarahan $HrSiam$ dari rentang operasi kueri yang sedang dijalankan
6	N	$longlong$	Digunakan untuk menyimpan jumlah permasalahan yang akan diberikan ke $HrSiam$

Tabel 4.5 Tabel Daftar Variabel Global Permaslahaan *Angry Siam* Bagian 2

No	Nama Variabel	Tipe	Penjelasan
7	Q	$longlong$	Digunakan untuk menyimpan jumlah operasi kueri yang akan dijalankan
8	blk_sz	$longlong$	Digunakan untuk menyimpan nilai konstanta yang akan membagi operasi kueri menjadi blok-blok kecil
9	$currentL$	$longlong$	Digunakan untuk menyimpan nilai batas kiri dari *Mo's Algorithm*
10	$currentR$	$longlong$	Digunakan untuk menyimpan nilai batas kanan dari *Mo's Algorithm*
11	$currentTime$	$longlong$	Digunakan untuk menyimpan kondisi sekarang dari *Mo's Algorithm*
12	$type$	int	Digunakan untuk menyimpan tipe operasi kueri
13	$total\_query$	int	Digunakan untuk menyimpan jumlah operasi kueri yang memiliki tipe pertama
14	$time$	int	Digunakan untuk menyimpan kondisi dari operasi kueri

Daftar variabel yang dijelaskan oleh Tabel 4.4 dan Tabel 4.5 ditunjukkan oleh Kode Sumber 4.21.

```
long long value[n], temp[n], counter[n];
long long answer[n], angry[n], total, N, Q;
long long blk_sz, currentL, currentR, currentTime;
int type, total_query = 0, time = 0;
```

Kode Sumber 4.21 Variabel Global pada Penyelesaian Permasalahan *Angry Siam*

Untuk *template* yang digunakan adalah *template* untuk mempercepat proses memasukkan data ke sistem (*Fast IO*). Potongan kode untuk *template Fast IO* dapat dilihat di Kode Sumber 4.22.

```
template<typename T>
T getNum(){
  T res=0;
  char c;
  while(1) {
    c=getchar();
    if(c==' ' || c=='\n') continue;
    else break;
  }
  res=c-'0';
  while(1) {
    c=getchar();
    if(c>='0' && c<='9') res=10*res+c-'0';
    else break;
  }
  return res;
}
```

Kode Sumber 4.22 *Template* pada Penyelesaian Permasalahan *Angry Siam*

Terdapat dua *struct* dalam sistem yaitu *struct* $Query$ dan $Update$. *Struct* $Query$ digunakan untuk menyimpan batas kiri dan kanan,

kondisi, dan urutan masuk operasi kueri. *Struct Query* menyimpan semua operasi kueri yang bertipe 1 atau dapat dilihat pada subbab 2.3.2. Potongan kode untuk *struct Query* dapat dilihat di Kode Sumber 4.23.

```
1  struct Query
2  {
3    long long L,R, Time, Number;
4  } arr[n];
```

Kode Sumber 4.23 *Struct Query* pada Penyelesaian Permasalahan *Angry Siam*

Struct Update digunakan untuk menyimpan operasi kueri yang bertipe 2. *Struct Update* berisi indeks elemen yang akan diubah dan elemen baru. Potongan kode untuk *struct Update* dapat dilihat di Kode Sumber 4.24.

```
1  struct Update
2  {
3    long long position, val;
4  } up[n];
```

Kode Sumber 4.24 *Struct Update* pada Penyelesaian Permasalahan *Angry Siam*

4.3.2 Implementasi Fungsi *Boolean compare*

Fungsi *boolean compare* diimplementasikan sesuai dengan *pseudocode* pada subbab 3.2.2.2.1. Fungsi *boolean compare* yang diimplementasikan seperti pada Kode Sumber 4.25 digunakan untuk menjadi argumen *std sort* dengan ketentuan yang sudah dijelaskan sebelumnya.

```
1  bool compare(Query x, Query y)
2  {
3    if (x.L/blk_sz != y.L/blk_sz)
4      return x.L/blk_sz < y.L/blk_sz;
5    if (x.R/blk_sz != y.R/blk_sz)
6      return x.R/blk_sz < y.R/blk_sz;
7    return x.Time < y.Time;
8  }
```

Kode Sumber 4.25 Fungsi *Boolean compare* pada Penyelesaian Permasalahan *Angry Siam*

4.3.3 Implementasi Fungsi *main*

Fungsi *main* diimplementasikan sesuai dengan *pseudocode* pada subbab 3.2.1. Fungsi *main* yang diimplementasikan seperti pada Kode Sumber 4.26 dan 4.27 digunakan untuk membaca masukan dari sistem menggunakan *template Fast IO*, menentukan konstanta untuk algoritma *Square Root Decomposition*. Untuk setiap operasi kueri bertipe 1 akan dimasukkan ke dalam *struct* $Query$ dan yang bertipe 2 akan dimasukkan ke dalam *struct* $Update$.

```
1  int main()
2  {
3    N = getNum<LL>();
4    blk_sz = 4000;
5    for(int i = 1; i <= N ; i++)
6    {
7      value[i] = getNum<LL>();
8      temp[i] = value[i];
9    }
10   for(int i = 1; i <= N ; i++)
11   {
12     angry[i] = getNum<LL>();
13   }
```

Kode Sumber 4.26 Fungsi *main* 1 pada Penyelesaian Permasalahan *Angry Siam*

```
1     Q = getNum<LL>();
2     for(int i = 1 ; i<= Q ; i++)
3     {
4       type = getNum<LL>();
5       if(type == 1)
6       {
7         arr[total_query].L =
8           getNum<LL>();
9         arr[total_query].R =
10          getNum<LL>();
11        arr[total_query].Number =
12          total_query;
13        arr[total_query].Time =
14          time;
15        total_query++;
16      }
17      else
18      {
19        up[time].position =
20          getNum<LL>();
21        up[time].val =
22          getNum<LL>();;
23        time++;
24      }
25    }
26    sort(arr,arr+total_query,compare);
27    solve();
28  }
```

Kode Sumber 4.27 Fungsi *main* 2 pada Penyelesaian Permasalahan *Angry Siam*

4.3.4 Implementasi Fungsi *Add*

Fungsi *Add* diimplementasikan pada Kode Sumber 4.28 dan 4.29 sesuai dengan *pseudocode* pada subbab 3.2.2.3.1. Fungsi ini digunakan saat sebuah elemen pada indeks tertentu ada di dalam rentang operasi kueri yang sekarang sedang dicek sesuai dengan

deskripsi desain yang telah dijelaskan sebelumnya.

```
1         counter[value[currentL]]++;
2         total += angry[counter[value[currentL]]]
3           * value[currentL];
```

Kode Sumber 4.28 Fungsi *Add* 1 pada Penyelesaian Permasalahan *Angry Siam*

```
1         counter[value[currentR]]++;
2         total += angry[counter[value[currentR]]]
3           * value[currentR];
```

Kode Sumber 4.29 Fungsi *Add* 2 pada Penyelesaian Permasalahan *Angry Siam*

4.3.5 Implementasi Fungsi *Remove*

Fungsi *Remove* diimplementasikan pada Kode Sumber 4.30 dan 4.31 sesuai dengan *pseudocode* pada subbab 3.2.2.3.2. Fungsi ini digunakan saat sebuah elemen pada indeks tertentu tidak di dalam rentang operasi kueri yang sekarang sedang dicek sesuai dengan deskripsi desain yang telah dijelaskan sebelumnya.

```
1         total -= angry[counter[value[currentL]]]
2           * value[currentL];
3         counter[value[currentL]]--;
```

Kode Sumber 4.30 Fungsi *Remove* 1 pada Penyelesaian Permasalahan *Angry Siam*

```
1         total -= angry[counter[value[currentR]]]
2           * value[currentR];
3         counter[value[currentR]]--;
```

Kode Sumber 4.31 Fungsi *Remove* 2 pada Penyelesaian Permasalahan *Angry Siam*

4.3.6 Implementasi Fungsi *Change*

Fungsi *Change* diimplementasikan pada Kode Sumber 4.32 sesuai dengan *pseudocode* pada subbab 3.2.2.3.3. Fungsi ini digunakan jika kondisi sekarang berbeda dengan kondisi saat kueri dimasukkan sesuai dengan deskripsi desain yang telah dijelaskan sebelumnya.

```
1         if(up[currentTime].position
2           >= currentL
3             && up[currentTime].position
4               <= currentR)
5         {
6           total -= angry[counter[value[
7             up[currentTime].position]]]
8               * value[
9                 up[currentTime].position];
10          counter[value[
11            up[currentTime].position]]--;
12        }
13        value[up[currentTime].position] =
14          up[currentTime].val;
15        if(up[currentTime].position
16          >= currentL &&
17            up[currentTime].position
18              <= currentR)
19        {
20          counter[value[
21            up[currentTime].position]]++;
22          total += angry[counter[value[
23            up[currentTime].position]]]
24              * value[
25                up[currentTime].position];
26        }
```

Kode Sumber 4.32 Fungsi *Change* pada Penyelesaian Permasalahan *Angry Siam*

4.3.7 Implementasi Fungsi *Reset*

Fungsi *Reset* diimplementasikan pada Kode Sumber 4.33 sesuai dengan *pseudocode* pada subbab 3.2.2.3.4. Fungsi ini digunakan jika kondisi sekarang lebih baru dari kondisi saat kueri dimasukkan sehingga kondisi dari *array A* harus dikembalikan seperti semula sesuai dengan deskripsi desain yang telah dijelaskan sebelumnya.

```
1         currentTime = 0;
2         for(int i=1;i<=N;i++)
3         {
4           if(value[i] != temp[i])
5           {
6             if(i >= currentL &&
7               i <= currentR)
8             {
9               total -=
10                angry[counter[value[i]]]
11                  * value[i];
12              counter[value[i]]--;
13            }
14            value[i] = temp[i];
15            if(i >= currentL && i <= currentR)
16            {
17              counter[value[i]]++;
18              total +=
19                angry[counter[value[i]]]
20                  * value[i];
21            }
22          }
23        }
```

Kode Sumber 4.33 Fungsi *Reset* pada Penyelesaian Permasalahan *Angry Siam*

4.3.8 Implementasi Fungsi *Solve*

Fungsi *Solve* diimplementasikan pada Kode Sumber 4.34, 4.35, 4.36, dan 4.37 sesuai dengan *pseudocode* pada subbab 3.2.2.3. Fungsi ini digunakan untuk menghitung total jenis tumbuhan unik dalam rentang operasi kueri. Fungsi *Solve* terdiri dari beberapa fungsi yang digunakan untuk menyelesaikan permasalahan. Fungsi-fungsi tersebut adalah fungsi *Add*, *Remove*, *Change*, *Reset*.

```
1  void solve()
2  {
3    total = 0;
4    currentL = 1;
5    currentR = 0;
6    for(int i = 0 ; i < total_query ; i++)
7    {
8      int left = arr[i].L;
9      int right = arr[i].R;
10     time = arr[i].Time;
11     while(currentL < left)
12     {
13       total -= angry[counter[value[currentL]]]
14         * value[currentL];
15       counter[value[currentL]]--;
16       currentL++;
17     }
18     while(currentL > left)
19     {
20       currentL--;
21       counter[value[currentL]]++;
22       total += angry[counter[value[currentL]]]
23         * value[currentL];
24     }
```

Kode Sumber 4.34 Fungsi *Solve* 1 pada Penyelesaian Permasalahan *Angry Siam*

```
1       while(currentR < right)
2       {
3         currentR++;
4         counter[value[currentR]]++;
5         total += angry[counter[value[currentR]]]
6           * value[currentR];
7       }
8       while(currentR > right)
9       {
10        total -= angry[counter[value[currentR]]]
11          * value[currentR];
12        counter[value[currentR]]--;
13        currentR--;
14      }
15      if(currentTime > time)
16      {
17        currentTime = 0;
18        for(int i=1;i<=N;i++)
19        {
20          if(value[i] != temp[i])
21          {
22            if(i >= currentL &&
23              i <= currentR)
24            {
25              total -=
26                angry[counter[value[i]]]
27                  * value[i];
28              counter[value[i]]--;
29            }
30            value[i] = temp[i];
31            if(i >= currentL && i <= currentR)
32            {
33              counter[value[i]]++;
34              total +=
35                angry[counter[value[i]]]
36                  * value[i];
37            }
```

Kode Sumber 4.35 Fungsi *Solve* 2 pada Penyelesaian Permasalahan *Angry Siam*

```
1           }
2         }
3       }
4       while(currentTime < time)
5       {
6         if(up[currentTime].position
7           >= currentL
8             && up[currentTime].position
9               <= currentR)
10        {
11          total -= angry[counter[value[
12            up[currentTime].position]]]
13              * value[
14                up[currentTime].position];
15          counter[value[
16            up[currentTime].position]]--;
17        }
18        value[up[currentTime].position] =
19          up[currentTime].val;
20        if(up[currentTime].position
21          >= currentL &&
22            up[currentTime].position
23              <= currentR)
24        {
25          counter[value[
26            up[currentTime].position]]++;
27          total += angry[counter[value[
28            up[currentTime].position]]]
29              * value[
30                up[currentTime].position];
31        }
32        currentTime++;
33      }
34      answer[arr[i].Number] = total;
35    }
36    for(int i = 0 ; i < total_query ; i++)
```

Kode Sumber 4.36 Fungsi *Solve* 3 pada Penyelesaian Permasalahan *Angry Siam*

```
1       printf("%lld\n",answer[i]);
2   }
```

Kode Sumber 4.37 Fungsi *Solve* 4 pada Penyelesaian Permasalahan *Angry Siam*

4.4 Implementasi Penyelesaian Permasalahan *Counting diff-pairs*

Pada subbab ini akan dijelaskan mengenai implementasi dari penyelesaian permasalahan *Counting diff-pairs* yang telah dibuat pada subbab 3.3.

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

Pada subbab ini akan dibahas penggunaan *library*, konstanta, *template*, *struct*, dan variabel global yang digunakan dalam sistem. Potongan kode yang menyatakan penggunaan *library* dan konstanta dapat dilihat pada Kode Sumber 4.38.

```
1   #include <cmath>
2   #include <cstdio>
3   #include <algorithm>
4
5   using namespace std;
6   const int mv = 100000;
7   const int n = 50000;
```

Kode Sumber 4.38 *Library* dan Konstanta pada Penyelesaian Permasalahan *Counting diff-pairs*

Untuk menyelesaikan permasalahan *Angry Siam* dibutuhkan tiga *library* yaitu $cmath$, $cstudio$, dan $algorithm$. Didefinisikan konstanta n yang bernilai 50000 sebagai jumlah maksimal elemen untuk deret bilangan dan jumlah operasi kueri yang dapat

dimasukkan serta konstanta mv yang bernilai 100000 untuk besar maksimal nilai per elemen dalam deret bilangan. Pada tabel berikut dijelaskan mengenai variabel-variabel global yang akan digunakan dalam implementasi program.

Tabel 4.6 Tabel Daftar Variabel Global Permaslahaan *Counting diff-pairs* Bagian 1

No	Nama Variabel	Tipe	Penjelasan
1	$value$	$int[]$	Digunakan untuk menyimpan nilai setiap elemen yang ada di deret bilangan
2	BIT	$int[]$	Digunakan untuk menyimpan frekuensi dari setiap elemen yang ada di rentang operasi
3	N	int	Digunakan untuk menyimpan jumlah elemen yang ada di dalam deret bilangan
4	M	int	Digunakan untuk menyimpan jumlah operasi kueri
5	k	int	Digunakan untuk menyimpan nilai selisih absolut
6	blk_sz	int	Digunakan untuk menyimpan nilai konstanta yang akan membagi operasi kueri menjadi blok-blok kecil

Tabel 4.7 Tabel Daftar Variabel Global Permaslahaan *Counting diff-pairs* Bagian 2

No	Nama Variabel	Tipe	Penjelasan
7	$answer$	$longlong[]$	Digunakan untuk menyimpan jawaban dari setiap operasi kueri

Daftar variabel yang dijelaskan oleh Tabel 4.6 dan Tabel 4.7 ditunjukkan oleh Kode Sumber 4.39.

```
1  int value[n], BIT[mv + 5], N, M, k, blk_sz;
2  long long answer[n];
```

Kode Sumber 4.39 Variabel Global pada Penyelesaian Permasalahan *Counting diff-pairs*

Untuk *template* yang digunakan adalah *template* untuk mempercepat proses memasukkan data ke sistem (*Fast IO*). Potongan kode untuk *template Fast IO* dapat dilihat di Kode Sumber 4.40 dan 4.41.

```
 1  template<typename T>
 2  T getNum(){
 3    T res=0;
 4    char c;
 5    while(1) {
 6      c=getchar_unlocked();
 7      if(c==' ' || c=='\n') continue;
 8      else break;
 9    }
10    res=c-'0';
11    while(1) {
```

Kode Sumber 4.40 *Template* pada Penyelesaian Permasalahan *Counting diff-pairs* Bagian 1

```
1       c=getchar_unlocked();
2       if(c>='0' && c<='9') res=10*res+c-'0';
3       else break;
4     }
5     return res;
6   }
```

Kode Sumber 4.41 *Template* pada Penyelesaian Permasalahan *Counting diff-pairs* Bagian 2

Terdapat *struct* dalam sistem yaitu *struct Query*. *Struct Query* digunakan untuk menyimpan batas kiri dan kanan, dan urutan masuk operasi kueri. *Struct Query* menyimpan semua operasi kueri atau dapat dilihat pada subbab 2.5. Potongan kode untuk *struct Query* dapat dilihat di Kode Sumber 4.42.

```
1  struct Query
2  {
3    int L,R, Number;
4  } arr[n];
```

Kode Sumber 4.42 *Struct Query* pada Penyelesaian Permasalahan *Counting diff-pairs*

4.4.2 Implementasi Fungsi *Boolean compare*

Fungsi *boolean compare* diimplementasikan sesuai dengan *pseudocode* pada *paragraph* 3.3.2.2.1.

```
1  bool compare(Query x, Query y)
2  {
3    if (x.L/blk_sz != y.L/blk_sz)
4      return x.L/blk_sz < y.L/blk_sz;
5    return x.R/blk_sz < y.R/blk_sz;
6  }
```

Kode Sumber 4.43 Fungsi *Boolean compare* pada Penyelesaian Permasalahan *Counting diff-pairs*

Fungsi *boolean compare* yang diimplementasikan seperti pada Kode Sumber 4.43 digunakan untuk menjadi argumen *std sort* dengan ketentuan yang sudah dijelaskan sebelumnya.

4.4.3 Implementasi Fungsi *main*

Fungsi *main* diimplementasikan sesuai dengan *pseudocode* pada subsubbab 3.3.1. Fungsi *main* yang diimplementasikan seperti pada Kode Sumber 4.44 digunakan untuk membaca masukan dari sistem menggunakan *template Fast IO*, menentukan konstanta untuk algoritma *Square Root Decomposition*. Seluruh indeks yang digunakan untuk menunjukkan letak elemen pada *struct* akan dikurangi 1 karena format masukkan dimulai dari indeks 1 sementara dalam sistem akan menggunakan indeks 0 sebagai awalan.

```
 1  int main()
 2  {
 3    N = getNum<int>();
 4    M = getNum<int>();
 5    k = getNum<int>();
 6    blk_sz = (double) pow(N,(double)1/2);
 7    for(int i = 0; i < N ; i++)
 8      value[i] = getNum<int>();
 9    for(int i = 0 ; i< M ; i++)
10    {
11      arr[i].L = getNum<int>();
12      arr[i].R = getNum<int>();
13      arr[i].L --; arr[i].R --;
14      arr[i].Number = i;
15    }
16    sort(arr,arr+M,compare);
17    CPair();
18    return 0;
19  }
```

Kode Sumber 4.44 Fungsi *main* pada Penyelesaian Permasalahan *Counting diff-pairs*

4.4.4 Implementasi Fungsi *Update*

Fungsi *Update* diimplementasikan pada Kode Sumber 4.45 sesuai dengan *pseudocode* pada bagian 3.3.2.4. Fungsi ini digunakan untuk mengubah nilai yang terdapat di dalam *Fenwick Tree* $BIT[idx]$ dan semua indeks yang berkaitan dengan indeks idx dalam bentuk biner.

```
1  void Update(int idx, int val)
2  {
3    for(;idx <= mv;idx += idx & -idx)
4      BIT[idx] += val;
5
6  }
```

Kode Sumber 4.45 Fungsi *Update* pada Penyelesaian Permasalahan *Counting diff-pairs*

4.4.5 Implementasi Fungsi *Read*

Fungsi *Read* diimplementasikan pada Kode Sumber 4.46 sesuai dengan *pseudocode* pada bagian 3.3.2.5. Fungsi ini digunakan untuk menghitung nilai frekuensi kumulatif dari *Fenwick Tree* $BIT[]$ indeks idx dan semua bilangan yang berkaitan dengan idx dalam bentuk biner.

```
1  int Read(int idx)
2  {
3    int sum = 0;
4    for(;idx>0; idx -= idx & -idx)
5      sum += BIT[idx];
6    return sum;
7  }
```

Kode Sumber 4.46 Fungsi *Read* pada Penyelesaian Permasalahan *Counting diff-pairs*

4.4.6 Implementasi Fungsi *Get*

Fungsi *Get* diimplementasikan pada Kode Sumber 4.47 sesuai dengan *pseudocode* pada bagian 3.3.2.6. Fungsi ini digunakan untuk menghitung jumlah pasangan bilangan yang memiliki selisih absolut sama dengan atau lebih dari k.

```
1  int Get(int idx)
2  {
3    int sum = 0;
4    int y = max(0,idx-k);
5    sum = Read(y);
6    y = min(mv, idx + k - 1);
7    sum += Read(mv) - Read(y);
8    return sum;
9  }
```

Kode Sumber 4.47 Fungsi *Get* pada Penyelesaian Permasalahan *Counting diff-pairs*

4.4.7 Implementasi Fungsi *CPair*

Fungsi *CPair* diimplementasikan pada Kode Sumber 4.48 dan 4.49 sesuai dengan *pseudocode* pada bagian 3.3.2.3. Fungsi ini digunakan untuk mencari jawaban setiap operasi kueri. Fungsi *CPair* terdiri dari beberapa fungsi yang digunakan untuk menyelesaikan permasalahan. Fungsi-fungsi tersebut adalah fungsi *Update* dan *Get*.

```
1  void CPair()
2  {
3    long long total = 0;
4    int currentL = 0;
5    int currentR = -1;
6    for(int i = 0 ; i < M ; i++)
7    {
```

Kode Sumber 4.48 Fungsi *CPair* 1 pada Penyelesaian Permasalahan *Counting diff-pairs*

```
1       while(currentL < arr[i].L)
2       {
3         total-=Get(value[currentL]);
4         Update(value[currentL],-1);
5         currentL++;
6       }
7       while(currentL > arr[i].L)
8       {
9         total+=Get(value[currentL-1]);
10        Update(value[currentL-1],1);
11        currentL--;
12      }
13      while(currentR < arr[i].R)
14      {
15        total+=Get(value[currentR+1]);
16        Update(value[currentR+1],1);
17        currentR++;
18      }
19      while(currentR > arr[i].R)
20      {
21        total-=Get(value[currentR]);
22        Update(value[currentR],-1);
23        currentR--;
24      }
25      answer[arr[i].Number] = total;
26    }
27    for(int i = 0; i < M ; i++)
28      printf("%lld\n",answer[i]);
29  }
```

Kode Sumber 4.49 Fungsi *CPair* 2 pada Penyelesaian Permasalahan *Counting diff-pairs*

4.4.8 Implementasi Data *Generator*

Data *Generator* diimplementasikan pada Kode Sumber 4.50 sesuai dengan *pseudocode* pada subbab 3.3.2.7.

```
#include<cstdio>
#include<cstdlib>
#include<algorithm>

int main()
{
  int N,M,K;
  scanf("%d␣%d␣%d",&N,&M,&K);
  printf("%d␣%d␣%d\n",N,M,K);
  for(int i=0;i<N;i++)
  {
    int random = rand() % 100000;
    if(random == 0) random++;
    if(i < N - 1) printf("%d␣",random);
    else printf("%d\n",random);
  }
  for(int j=0;j<M;j++)
  {
    int a = rand() % N;
    int b = rand() % N;
    if(a == 0 ) a++;
    if(b == 0) b++;
    if(j == M - 1)
    {
      if(a > b) printf("%d␣%d",b,a);
      else printf("%d␣%d",a,b);
    }
    else
    {
      if(a > b) printf("%d␣%d\n",b,a);
      else printf("%d␣%d\n",a,b);
    }
  }
}
```

Kode Sumber 4.50 Data *Generator* pada Penyelesaian Permasalahan *Counting diff-pairs*

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. Bahasa yang digunakan pada uji coba ini adalah bahasa pemrograman C++. *Integrated Development Environment* atau *IDE* yang digunakan adalah Orwell Bloodshed Dev-C++ 5.11 dengan *compiler* g++ (TDM-GCC 4.8.1 32-bit). Sistem Operasi yang digunakan adalah Windows 10 Professional 64 bit.Perangkat keras yang digunakan adalah prosesor Intel(R) Core(TM) i7-4720HQ CPU @ 2.60 GHz 2.59 GHz. *Random Access Memory* atau *RAM* yang digunakan berukuran 12.00 GB.

5.2 Uji Coba Kebenaran

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

5.2.1 Uji Coba Kebenaran Penyelesaian Permasalahan *Ada and Unique Vegetable*

Uji coba kebenaran dilakukan dengan cara mengirimkan kode sumber program ke situs penilaian SPOJ. Hasil uji coba dengan waktu terbaik yang didapatkan dari situs penilaian SPOJ dapat dilihat pada Gambar 5.1.

Gambar 5.1 Hasil Uji Coba pada Situs SPOJ permasalahan *Ada and Unique Vegetable*

Dari Gambar 5.1, dapat dilihat bahwa kode sumber yang telah dikirim mendapatkan hasil keluaran *Accepted*. Waktu yang dibutuhkan sistem untuk menyelesaikan soal *Ada and Unique Vegetable* adalah 24.52 detik dan memori yang dibutuhkan adalah 10 MB. Selanjutnya akan dijelaskan langkah-langkah sistem dalam melakukan penyelesaian soal dan hasil dari uji coba sederhana akan dibandingkan dengan keluaran sistem. Uji coba kasus sederhana akan diselesaikan mengacu dengan langkah-langkah yang sudah dijelaskan pada subbab 2.2.3.

```
 1  10 10
 2  1 7 2 3 3 1 2 3 3 4
 3  2 0 7
 4  2 1 3
 5  2 0 4
 6  1 0 2
 7  2 2 5
 8  2 6 7
 9  1 7 5
10  2 2 7
11  2 0 4
12  2 0 3
```

Gambar 5.2 Format Masukan Uji Kebenaran *Ada and Unique Vegetable*

Kasus sederhana yang digunakan adalah sebuah kasus dimana *Ada*

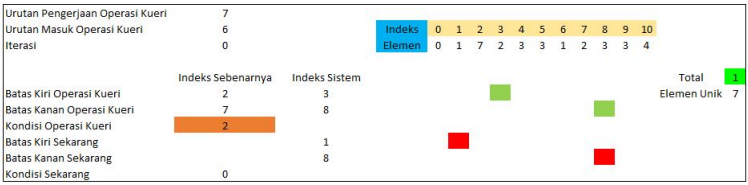
memiliki parit dengan panjang senilai 10 serta operasi kueri yang akan dijalankan berjumlah 10 dengan ketentuan operasi kueri tipe pertama berjumlah 2 dan kueri tipe kedua berjumlah 8. Seluruh operasi kueri akan memiliki rentang yang berbeda-beda. Format masukan dari masalah yang akan diselesaikan dapat dilihat pada Gambar 5.2. Langkah selanjutnya setelah memasukkan kasus uji coba sederhana seperti pada Gambar 5.2 adalah membuat urutan pengerjaan operasi kueri yang memiliki tipe kedua dengan menggunakan konstanta yang telah ditentukan dan aturan yang sudah dijelaskan pada deskripsi *Mo's Algorithm* di subbab 2.3.1. Perubahan dari urutan masuk operasi kueri ke urutan pengerjaan operasi kueri dapat dilihat pada Tabel 5.1.

Tabel 5.1 Tabel Urutan Pengerjaan Operasi Kueri Permasalahaan *Ada and Unique Vegetable*

No	Urutan Masuk	Batas Kiri	Batas Kanan	Kondisi
1	2	1	3	0
2	3	0	4	0
3	4	2	5	1
4	7	0	4	2
5	8	0	3	2
6	1	0	7	0
7	6	2	7	2
8	5	6	7	1

Konstanta yang digunakan dalam membagi operasi kueri masuk ke dalam blok-blok kecil adalah $N^{2/3}$. Nilai konstanta berbeda dengan nilai konstanta yang dijelaskan pada bagian 3.1.2.1 karena N yang digunakan pada kasus uji coba ini memiliki nilai yang kecil, yaitu 10. Jika nilai konstanta yang digunakan bernilai besar maka semua operasi kueri akan masuk dalam blok yang sama sehingga tidak

dapat disortir. Operasi kueri yang sudah disortir sesuai dengan peraturan dapat dikerjakan untuk menyelesaikan kasus uji coba ini. Terdapat perbedaan nilai yang digunakan antara batas kiri dan kanan masukan dengan batas kiri dan kanan sisem sehingga setiap masukan batas kiri dan kanan akan ditambah 1 seperti yang sudah dijelaskan pada subbab 4.2.3. Langkah-langkah dari *Mo's Algorithm* dalam pengerjaan operasi kueri akan divisualisasikan menggunakan gambar. Iterasi yang divisualisasikan adalah saat terjadi perpindahan batas kiri dan kanan sekarang ke batas kiri dan kanan satu operasi kueri dan perubahan kondisi dari operasi kueri.



Gambar 5.3 Visualisasi 1 Penyelesaian Contoh Kasus Uji *Ada and Unique Vegetable*

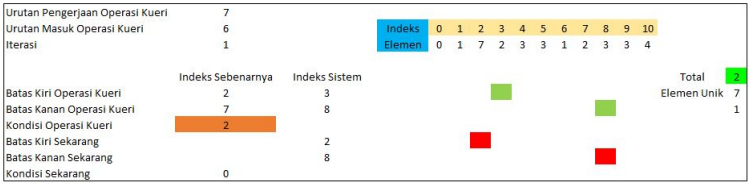
Gambar 5.3 adalah salah satu visualisasi bagian dari penyelesaian kasus uji coba untuk *Ada and Unique Vegetable*. Penjelasan dari gambar tersebut dapat dilihat sebagai berikut:

1. Urutan Pengerjaan Operasi Kueri adalah urutan waktu pengerjaan operasi kueri.
2. Urutan Masuk Operasi Kueri adalah urutan operasi kueri tersebut masuk dalam sistem.
3. Iterasi adalah waktu dalam pengerjaan satu operasi kueri.
4. Batas Kiri dan Kanan Operasi Kueri menunjukkan nilai batas kiri dan kanan operasi kueri yang sedang diuji. Terdapat Indeks Sebenarnya dan Indeks Sistem yang membedakan nilai yang digunakan waktu masukan dan nilai yang ada di

sistem.

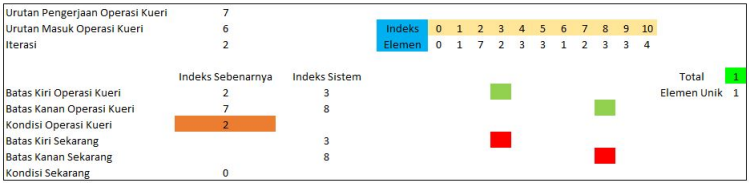
5. Kondisi Operasi Kueri adalah kondisi saat operasi kueri tersebut dimasukkan.
6. Batas Kiri dan Kanan Sekarang menunjukkan indeks dari *Mo's Algorithm*
7. Kondisi Sekarang adalah kondisi dari *array*.
8. Total menunjukkan elemen unik dalam satu rentang operasi kueri.
9. Elemen Unik adalah elemen yang tidak memiliki jumlah lebih dari 1 di dalam satu rentang batas kiri dan kanan sekarang.
10. Warna hijau menunjukkan batas kanan dan kiri operasi kueri sementara warna merah menunjukkan batas kiri dan kanan sekarang.

Gambar 5.3 menunjukkan kondisi iterasi ke-0 saat pengerjaan operasi kueri ke-7 untuk masukan ke-6.



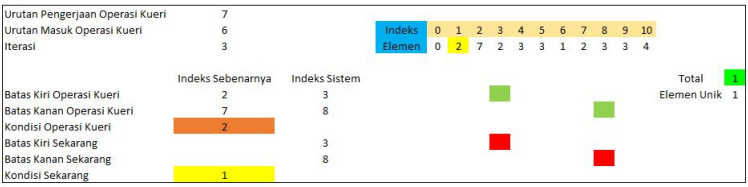
Gambar 5.4 Visualisasi 2 Penyelesaian Contoh Kasus Uji *Ada and Unique Vegetable*

Pada Gambar 5.4 menunjukkan indeks pada batas kiri sekarang mendekati batas kiri operasi kueri. Untuk iterasi ke-1 terjadi perubahan total elemen unik menjadi 2 karena elemen 1 hanya berjumlah 1 di dalam rentang batas kiri dan kanan sekarang.



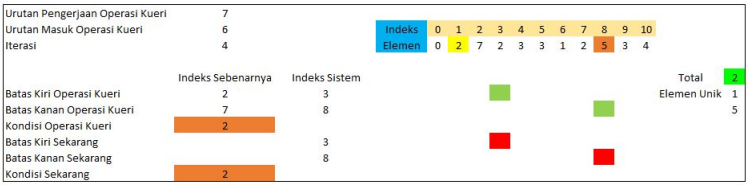
Gambar 5.5 Visualisasi 3 Penyelesaian Contoh Kasus Uji *Ada and Unique Vegetable*

Pada Gambar 5.5 menunjukkan batas kiri sekarang sudah sama dengan batas kiri operasi kueri. Perubahan total elemen unik berubah menjadi 1 karena elemen 7 sudah tidak di dalam rentang batas kiri dan kanan sekarang.



Gambar 5.6 Visualisasi 4 Penyelesaian Contoh Kasus Uji *Ada and Unique Vegetable*

Langkah selanjutnya adalah mengubah kondisi dari *array* sesuai dengan kondisi saat operasi kueri dimasukkan. Pada Gambar 5.6 menunjukkan perubahan nilai *array* indeks 1 dari 1 menjadi 2. Karena perubahan nilai elemen dari *array* tidak di dalam rentang batas kiri dan kanan sekarang maka total elemen unik tetap sama.



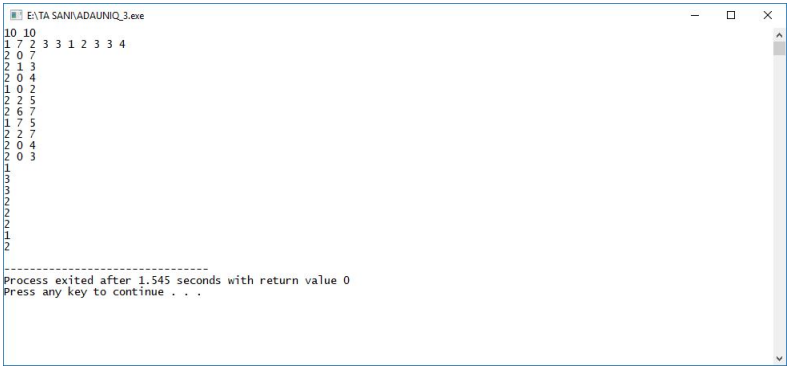
Gambar 5.7 Visualisasi 5 Penyelesaian Contoh Kasus Uji *Ada and Unique Vegetable*

Kondisi sekarang akan diubah sampai memiliki kondisi yang sama dengan kondisi saat operasi kueri masuk ke dalam sistem. Pada Gambar 5.7 menunjukkan perubahan nilai elemen *array* indeks 8 dari 3 menjadi 5. Perubahan tersebut membuat total elemen unik berubah menjadi 2 karena elemen bernilai 5 hanya berjumlah 1. Total elemen unik yang didapatkan pada pengerjaan operasi kueri dengan batas kiri sistem 3 dan batas kanan sistem 8 adalah 1. Langkah-langkah yang sudah divisualisasikan juga berlaku untuk semua operasi kueri yang akan diselesaikan oleh sistem.

Tabel 5.2 Tabel Hasil Pengerjaan Kasus Uji Coba Sederhana *Ada and Unique Vegetable*

No	Batas Kiri	Batas Kanan	Kondisi	Total
1	0	7	0	1
2	1	3	0	3
3	0	4	0	3
4	2	5	1	2
5	6	7	1	2
6	2	7	2	2
7	0	4	2	1
8	0	3	2	2

Hasil keluaran dari sistem adalah total elemen unik untuk setiap operasi kueri sesuai dengan urutan masuk operasi kueri ke dalam sistem. Hasil yang didapatkan untuk masukan operasi kueri dari Tabel 5.1 dapat dilihat pada Tabel 5.2. Setelah melakukan visualisasi, sistem penyelesaian dijalankan diberi masukan pada Gambar 5.2. Keluaran sitem telah sesuai dengan permasalahan yang diberikan. Angka yang tercetak pada Gambar 5.8 sama dengan hasil yang ada pada Tabel 5.2.



Gambar 5.8 Keluaran Sistem Penyelesaian Kasus Uji Coba Sederhana *Ada and Unique Vegetable*

Berdasarkan hasil uji coba tersebut, Algoritma *Square Root Decomposition* yang didesain dan dijalankan pada penyelesaian permasalahan kasus uji coba sederhana dapat menyelesaikan permasalahan *Ada and Unique Vegetable*. Selanjutnya, desain algoritma *Square Root Decomposition* yang telah dibuatakan digunakan untuk menyelesaikan permasalahan *Counting diff-pairs*.

5.2.2 Uji Coba Kebenaran Penyelesaian Permasalahan *Angry Siam*

Uji coba kebenaran dilakukan dengan cara mengirimkan kode sumber program ke situs penilaian SPOJ. Hasil uji coba dengan waktu terbaik yang didapatkan dari situs penilaian SPOJ dapat dilihat pada Gambar 5.9.

Gambar 5.9 Hasil Uji Coba pada Situs SPOJ permasalahan *Angry Siam*

Dari Gambar 5.9, dapat dilihat bahwa kode sumber yang telah dikirim mendapatkan hasil keluaran *Accepted*. Waktu yang dibutuhkan sistem untuk menyelesaikan soal *Angry Siam* adalah 4.76 detik dan memori yang dibutuhkan adalah 11 MB.

```
 1  10
 2  1 7 2 3 3 1 2 3 3 4
 3  1 2 3 4 5 6 7 8 9 10
 4  10
 5  1 1 7
 6  1 1 3
 7  1 1 4
 8  2 1 2
 9  1 2 5
10  1 6 7
11  2 7 5
12  1 2 7
13  1 1 4
14  1 1 3
```

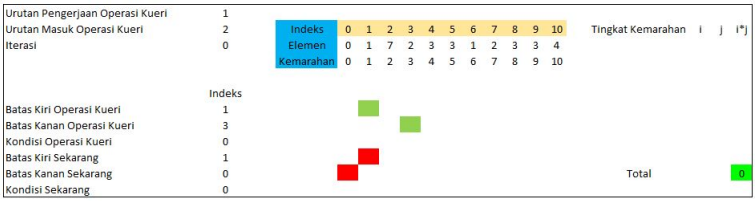
Gambar 5.10 Format Masukan Uji Kebenaran *Angry Siam*

Selanjutnya akan dijelaskan langkah-langkah sistem dalam melakukan penyelesaian soal dan hasil dari uji coba sederhana akan dibandingkan dengan keluaran sistem. Uji coba kasus sederhana akan diselesaikan mengacu dengan langkah-langkah yang sudah dijelaskan pada subbab 2.3.3. Kasus sederhana yang digunakan adalah sebuah kasus dimana $HrSiam$ memiliki 10 permasalahan dengan nilai permasalahan yang variatif serta operasi kueri yang akan dijalankan berjumlah 10 dengan ketentuan operasi kueri tipe pertama berjumlah 8 dan kueri tipe kedua berjumlah 2. Seluruh operasi kueri akan memiliki rentang yang berbeda-beda. Format masukan dari masalah yang akan diselesaikan dapat dilihat pada Gambar 5.10. Langkah selanjutnya setelah memasukkan kasus uji coba sederhana seperti pada Gambar 5.10 adalah membuat urutan pengerjaan operasi kueri yang memiliki tipe pertama dengan menggunakan konstanta yang telah ditentukan dan aturan yang sudah dijelaskan pada deskripsi *Mo's Algorithm* di subbab 2.3.1. Perubahan dari urutan masuk operasi kueri ke urutan pengerjaan operasi kueri dapat dilihat pada Tabel 5.3.

Tabel 5.3 Tabel Urutan Pengerjaan Operasi Kueri Permasalahaan *Angry Siam*

No	Urutan Masuk	Batas Kiri	Batas Kanan	Kondisi
1	2	1	3	0
2	8	1	3	2
3	1	1	7	0
4	3	1	4	0
5	4	2	5	1
6	6	2	7	2
7	7	1	4	2
8	5	6	7	1

Konstanta yang digunakan dalam membagi operasi kueri masuk ke dalam blok-blok kecil adalah $N^{2/3}$. Nilai konstanta berbeda dengan nilai konstanta yang dijelaskan pada bagian 3.2.2.1 karena N yang digunakan pada kasus uji coba ini memiliki nilai yang kecil, yaitu 10. Jika nilai konstanta yang digunakan bernilai besar maka semua operasi kueri akan masuk dalam blok yang sama sehingga tidak dapat disortir. Operasi kueri yang sudah disortir sesuai dengan peraturan dapat dikerjakan untuk menyelesaikan kasus uji coba ini. Langkah-langkah dari *Mo's Algorithm* dalam pengerjaan operasi kueri akan divisualisasikan menggunakan gambar. Iterasi yang divisualisasikan adalah saat terjadi perpindahan batas kiri dan kanan sekarang ke batas kiri dan kanan satu operasi kueri dan perubahan kondisi dari operasi kueri.



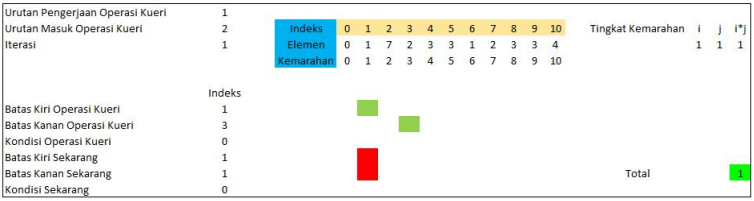
Gambar 5.11 Visualisasi 1 Penyelesaian Contoh Kasus Uji *Angry Siam*

Gambar 5.11 adalah salah satu visualisasi bagian dari penyelesaian kasus uji coba untuk *Ada and Unique Vegetable*. Penjelasan dari gambar tersebut dapat dilihat sebagai berikut:

1. Urutan Pengerjaan Operasi Kueri adalah urutan waktu pengerjaan operasi kueri.
2. Urutan Masuk Operasi Kueri adalah urutan operasi kueri tersebut masuk dalam sistem.
3. Iterasi adalah waktu dalam pengerjaan satu operasi kueri.
4. Batas Kiri dan Kanan Operasi Kueri menunjukkan nilai batas kiri dan kanan operasi kueri yang sedang diuji.

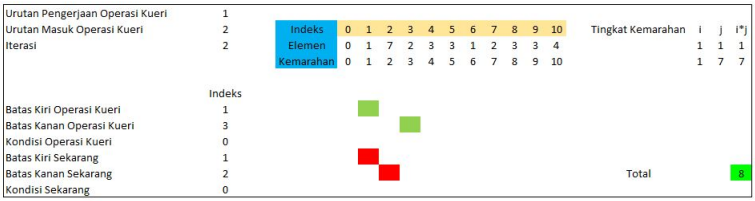
5. Kondisi Operasi Kueri adalah kondisi saat operasi kueri tersebut dimasukkan.
6. Batas Kiri dan Kanan Sekarang menunjukkan indeks dari *Mo's Algorithm*
7. Kondisi Sekarang adalah kondisi dari *array*.
8. Total menunjukkan tingkat kemerahan $HrSiam$ dalam satu rentang operasi kueri.
9. Tingkat Kemarahan adalah hasil perkalian antara kemarahan $HrSiam$ dan tingkat kesulitan dari permasalahan di dalam satu rentang batas kiri dan kanan sekarang.
10. Warna hijau menunjukkan batas kanan dan kiri operasi kueri sementara warna merah menunjukkan batas kiri dan kanan sekarang.

Gambar 5.11 menunjukkan kondisi saat iterasi ke-0 pengerjaan operasi kueri ke-1 untuk masukan ke-2.



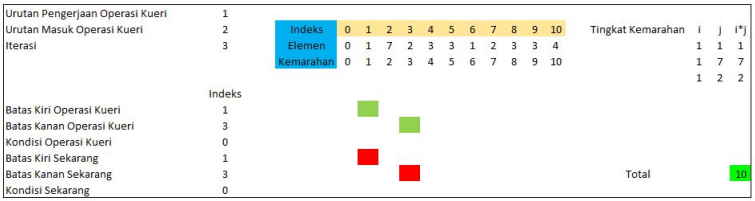
Gambar 5.12 Visualisasi 2 Penyelesaian Contoh Kasus Uji *Angry Siam*

Pada Gambar 5.12 menunjukkan indeks pada batas kiri sekarang sudah sama dengan batas kiri operasi kueri. Sementara untuk batas kanan sekarang akan mendekati batas kanan operasi kueri. Untuk iterasi ke-1 terjadi perubahan total kemarahan $HrSiam$ menjadi 1 karena elemen 1 baru pertama kali ditemui untuk rentang batas kiri dan kanan sekarang sehingga elemen 1 hanya dikalikan dengan 1.



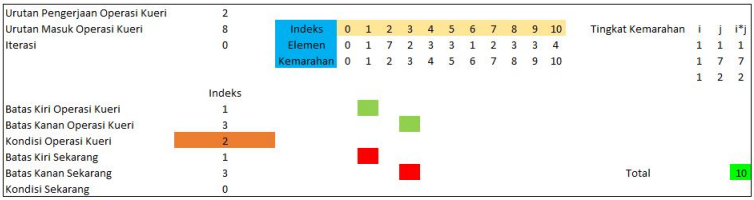
Gambar 5.13 Visualisasi 3 Penyelesaian Contoh Kasus Uji *Angry Siam*

Pada Gambar 5.13 menunjukkan perubahan total kemarahan $HrSiam$ menjadi 8 karena elemen 7 masuk ke dalam rentang batas kiri dan kanan sekarang. Sama halnya dengan elemen 1, elemen 7 baru ditemui pertama kali sehingga hanya dikali dengan nilai 1.



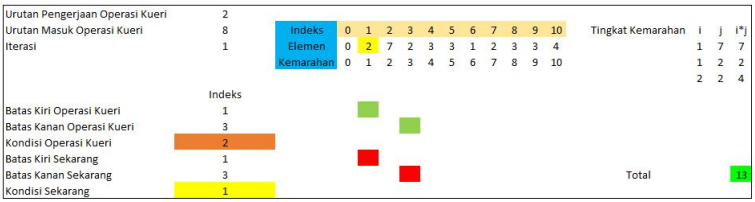
Gambar 5.14 Visualisasi 4 Penyelesaian Contoh Kasus Uji *Angry Siam*

Pada Gambar 5.14 menunjukkan perubahan total kemarahan $HrSiam$ menjadi 10 karena elemen 2 masuk ke dalam rentang batas kiri dan kanan sekarang. Elemen 2 baru ditemui pertama kali sehingga hanya dikali dengan nilai 1. Ini adalah akhir dari pengerjaan operasi kueri tersebut karena batas kiri dan kanan sekarang sudah sama dengan batas kiri dan kanan operasi kueri tersebut. Operasi kueri untuk batas kiri 1 dan batas kanan 3 memiliki jumlah kemarahan 10.



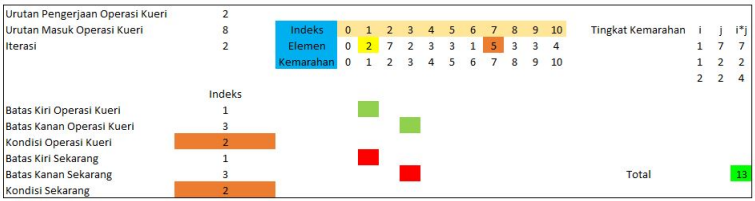
Gambar 5.15 Visualisasi 5 Penyelesaian Contoh Kasus Uji *Angry Siam*

Operasi kueri kedua yang dikerjakan ternyata memiliki batas kiri dan kanan yang sama dengan operasi kueri sebelumnya sehingga perpindahan batas kiri dan kanan tidak akan terjadi. Akan tetapi operasi kueri kedua memiliki perbedaan kondisi sehingga kondisi sekarang harus diubah hingga sama dengan kondisi operasi kueri yang sedang dikerjakan. Visualisasi dapat dilihat pada Gambar 5.15.



Gambar 5.16 Visualisasi 6 Penyelesaian Contoh Kasus Uji *Angry Siam*

Kondisi sekarang akan diubah sampai memiliki kondisi yang sama dengan kondisi saat operasi kueri masuk ke dalam sistem. Pada Gambar 5.16 menunjukkan perubahan nilai elemen *array* indeks 1 dari 1 menjadi 2. Perubahan tersebut membuat total kemarahan $HrSiam$ berubah menjadi 13 karena $HrSiam$ menemukan elemen bernilai 2 untuk kedua kalinya sehingga elemen 2 dikali dengan 2.



Gambar 5.17 Visualisasi 7 Penyelesaian Contoh Kasus Uji *Angry Siam*

Pada Gambar 5.17 menunjukkan perubahan nilai elemen *array* indeks 7 dari 2 menjadi 5. Perubahan tersebut tidak membuat total kemarahan $HrSiam$ berubah karena elemen tidak sedang di dalam rentang batas kiri dan kanan sekarang. Untuk operasi kueri dengan batas kiri 1, kanan 3, dan kondisi 2 memiliki total kemarahan 13. Langkah-langkah yang sudah divisualisasikan juga berlaku untuk semua operasi kueri yang akan diselesaikan oleh sistem. Hasil keluaran dari sistem adalah total kemarahan $HrSiam$ untuk setiap operasi kueri sesuai dengan urutan masuk operasi kueri ke dalam sistem. Hasil yang didapatkan untuk masukan operasi kueri dari Tabel 5.3 dapat dilihat pada Tabel 5.4

Tabel 5.4 Tabel Hasil Pengerjaan Kasus Uji Coba Sederhana *Angry Siam*

No	Batas Kiri	Batas Kanan	Kondisi	Total
1	1	7	0	25
2	1	3	0	10
3	1	4	0	13
4	2	5	1	18
5	6	7	1	3
6	2	7	2	24
7	1	4	2	16
8	1	3	2	13

Setelah melakukan visualisasi, sistem penyelesaian dijalankan diberi masukan pada Gambar 5.10. Keluaran sitem telah sesuai dengan permasalahan yang diberikan. Angka yang tercetak pada Gambar 5.18 sama dengan hasil yang ada pada Tabel 5.4.

Gambar 5.18 Keluaran Sistem Penyelesaian Kasus Uji Coba Sederhana *Angry Siam*

Berdasarkan hasil uji coba tersebut, *Mo's Algorithm* yang didesain dan dijalankan pada penyelesaian permasalahan kasus uji coba sederhana dapat menyelesaikan permasalahan *Angry Siam*. Selanjutnya, desain algoritma *Mo's Algorithm* yang telah dibuatakan digunakan untuk menyelesaikan permasalahan *Counting diff-pairs*.

5.2.3 Uji Coba Kebenaran Penyelesaian Permasalahan *Counting diff-pairs*

Uji coba kebenaran dilakukan dengan cara mengirimkan kode sumber program ke situs penilaian SPOJ. Hasil uji coba dengan waktu terbaik yang didapatkan dari situs penilaian SPOJ dapat dilihat pada Gambar 5.19.

Gambar 5.19 Hasil Uji Coba pada Situs SPOJ permasalahan *Counting diff-pairs*

Tabel 5.5 Tabel Hasil Uji Coba pada Situs SPOJ untuk Permasalahan *Counting diff-pairs*

No	Waktu
1	1.84
2	1.85
3	1.81
4	1.80
5	1.78
6	1.79
7	1.82
8	1.85
9	1.80
10	1.85
11	1.83
12	1.82
13	1.81
14	1.81
15	1.79
16	1.84
17	1.85
18	1.84
19	1.83
20	1.84

Dari Gambar 5.19, dapat dilihat bahwa kode sumber yang telah dikirim mendapatkan hasil keluaran *Accepted*. Waktu yang

dibutuhkan sistem untuk menyelesaikan soal *Counting diff-pairs* adalah 1.78 detik dan memori yang dibutuhkan adalah 17 MB. Selain itu, dilakukan pengujian sebanyak 20 kali pengiriman pada situs SPOJ untuk melihat variasi waktu dan memori yang dibutuhkan program. Hasil uji coba dapat dilihat pada Tabel 5.5. Dari hasil uji coba sebanyak 20 kali, seluruh kode sumber program mendapatkan keluaran *Accepted* dengan rincian sebagai berikut:

- Waktu minimum sebesar 1.78 detik.
- Waktu maksimum sebesar 1.85 detik.
- Waktu rata-rata sebesar 1.82 detik.
- Memori yang dibutuhkan program sebesar 17 MB.

Selanjutnya akan dijelaskan langkah-langkah sistem dalam melakukan penyelesaian soal dan hasil dari uji coba sederhana akan dibandingkan dengan keluaran sistem.

```
 1  10 10 2
 2  1 2 3 4 5 6 7 8 9 10
 3  1 7
 4  1 3
 5  1 4
 6  1 2
 7  2 5
 8  6 7
 9  5 7
10  2 7
11  1 4
12  1 3
```

Gambar 5.20 Format Masukan Uji Kebenaran *Counting diff-pairs*

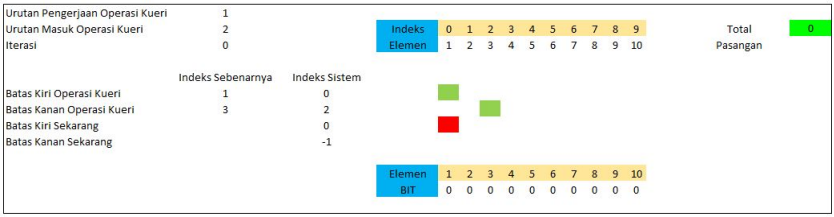
Uji coba kasus sederhana akan diselesaikan mengacu dengan langkah-langkah yang sudah dijelaskan pada subbab 2.6. Kasus sederhana yang digunakan adalah sebuah kasus dimana terdapat deret bilangan dengan jumlah elemen 10 dan operasi kueri berjumlah 10. Seluruh operasi kueri akan memiliki rentang yang berbeda-beda. Untuk selisih absolut adalah 2. Format masukan dari masalah yang akan diselesaikan dapat dilihat pada Gambar 5.20. Langkah selanjutnya setelah memasukkan kasus uji coba sederhana seperti pada Gambar 5.20 adalah membuat urutan pengerjaan operasi kueri dengan menggunakan konstanta yang telah ditentukan dan aturan yang sudah dijelaskan pada deskripsi *Mo's Algorithm* yang sudah dijelaskan pada subbab 2.3.1. Perubahan dari urutan masuk operasi kueri ke urutan pengerjaan operasi kueri dapat dilihat pada Tabel 5.6.

Tabel 5.6 Tabel Urutan Pengerjaan Operasi Kueri Permasalahaan *Counting diff-pairs*

No	Urutan Masuk	Batas Kiri	Batas Kanan
1	2	1	3
2	4	1	2
3	10	1	3
4	3	1	4
5	5	2	5
6	9	1	4
7	1	1	7
8	8	2	7
9	6	6	7
10	7	5	7

Konstanta yang digunakan dalam membagi operasi kueri masuk ke dalam blok-blok kecil adalah $N^{1/2}$. Operasi kueri yang

sudah disortir sesuai dengan peraturan dapat dikerjakan untuk menyelesaikan kasus uji coba ini. Terdapat perbedaan nilai yang digunakan antara batas kiri dan kanan masukan dengan batas kiri dan kanan sisem sehingga setiap masukan batas kiri dan kanan akan dikurangi 1 sesuai dengan subbab 4.4.3. Langkah-langkah dari *Mo's Algorithm* dalam pengerjaan operasi kueri akan divisualisasikan menggunakan gambar. Iterasi yang divisualisasikan adalah saat terjadi perpindahan batas kiri dan kanan sekarang ke batas kiri dan kanan satu operasi kueri dan perubahan elemen di dalam *Fenwick Tree*.



Gambar 5.21 Visualisasi 1 Penyelesaian Contoh Kasus Uji *Counting diff-pairs*

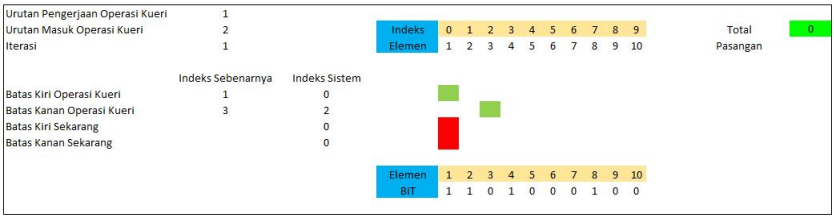
Gambar 5.21 adalah salah satu visualisasi bagian dari penyelesaian kasus uji coba untuk *Counting diff-pairs*. Penjelasan dari gambar tersebut dapat dilihat sebagai berikut:

1. Urutan Pengerjaan Operasi Kueri adalah urutan waktu pengerjaan operasi kueri.
2. Urutan Masuk Operasi Kueri adalah urutan operasi kueri tersebut masuk dalam sistem.
3. Iterasi adalah waktu dalam pengerjaan satu operasi kueri.
4. Batas Kiri dan Kanan Operasi Kueri menunjukkan nilai batas kiri dan kanan operasi kueri yang sedang diuji. Terdapat Indeks Sebenarnya dan Indeks Sistem yang membedakan nilai yang digunakan waktu masukan dan nilai yang ada di

sistem.

5. Batas Kiri dan Kanan Sekarang menunjukkan indeks dari *Mo's Algorithm*
6. Total menunjukkan elemen unik dalam satu rentang operasi kueri.
7. Pasangan menunjukkan pasangan bilangan yang memiliki selisih absolut sama dengan atau lebih besar dari yang diinginkan.
8. Warna hijau menunjukkan batas kanan dan kiri operasi kueri sementara warna merah menunjukkan batas kiri dan kanan sekarang.
9. Terdapat 2 deret bilangan yaitu deret bilangan untuk elemen yang ada di dalam *array* dan deret bilangan untuk elemen yang ada di dalam *BIT* / *Fenwick Tree*.

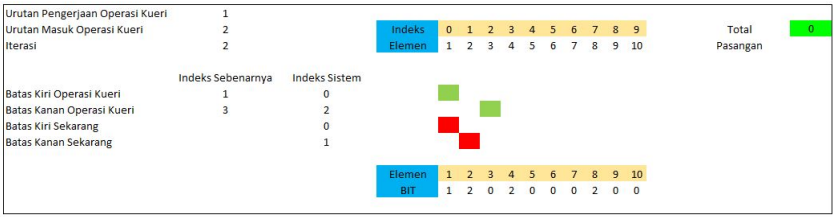
Gambar 5.21 menunjukkan kondisi iterasi ke-0 saat pengerjaan operasi kueri ke-1 untuk masukan ke-2.



Gambar 5.22 Visualisasi 2 Penyelesaian Contoh Kasus Uji *Counting diff-pairs*

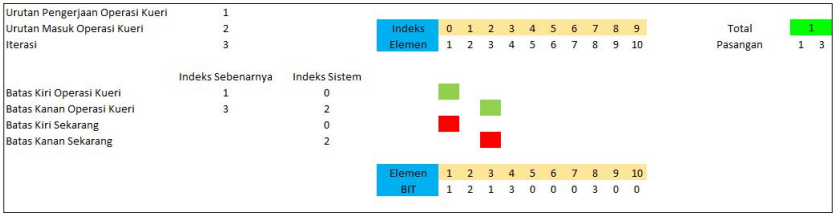
Pada Gambar 5.22 menunjukkan indeks pada batas kiri sekarang sudah sama dengan batas kiri operasi kueri. Sementara untuk batas kanan sekarang akan mendekati batas kanan operasi kueri. Untuk iterasi ke-1 tidak terjadi perubahan total pasangan bilangan dengan selisih absolut sama dengan atau lebih dari 2. Terjadi perubahan pada *Fenwick Tree* pada indeks elemen 1 karena elemen

1 ada di dalam rentang batas kiri dan kanan sekarang. Setiap perubahan elemen pada indeks tertentu di dalam *Fenwick Tree BIT* akan merubah semua indeks elemen yang berhubungan dalam bentuk biner / *least significat one-bit* yang sudah dijelaskan pada subbab 2.4.2.2. Perubahan elemen terjadi pada indeks elemen 1, 2, 4, 8, hingga nilai maksimal yang sudah ditentukan pada subbab 4.4.1.



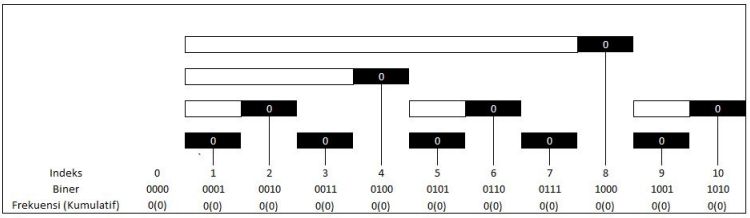
Gambar 5.23 Visualisasi 3 Penyelesaian Contoh Kasus Uji *Counting diff-pairs*

Pada Gambar 5.23 menunjukkan perubahan batas kanan sekarang akan tetapi masih tidak merubah total pasangan. Terjadi perubahan pada indeks elemen 2, 4, dan 8 pada *Fenwick Tree BIT* karena elemen 2 masuk ke dalam rentang batas kiri dan kanan sekarang.



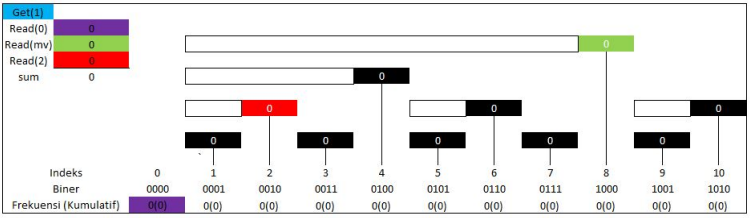
Gambar 5.24 Visualisasi 4 Penyelesaian Contoh Kasus Uji *Counting diff-pairs*

Pada Gambar 5.24 menunjukkan perubahan total pasangan yang memiliki selisih absolut sama dengan atau lebih dari 2 karena elemen 3 masuk ke dalam rentang batas kiri dan kanan sekarang sehingga terdapat 1 pasangan yaitu elemen 1 dan 3. Terjadi perubahan pada indeks elemen 3, 4, dan 8 pada *Fenwick Tree* BIT karena elemen 3 masuk ke dalam rentang batas kiri dan kanan sekarang.



Gambar 5.25 Visualisasi 1 *Fenwick Tree* Penyelesaian Contoh Kasus Uji *Counting diff-pairs*

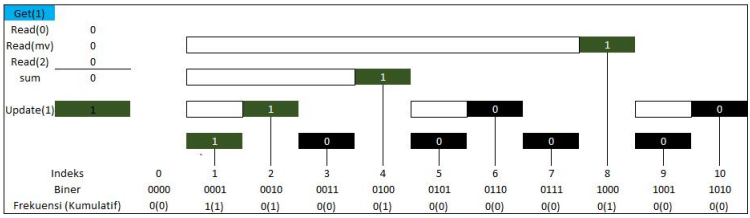
Pada Gambar 5.25 menunjukkan kondisi awal dari *Fenwick Tree* untuk penyelesaian permasalahan *Counting diff-pairs*. Setiap frekuensi yang ada masih bernilai 0 karena belum ada operasi kueri yang dijalankan.



Gambar 5.26 Visualisasi 2 *Fenwick Tree* Penyelesaian Contoh Kasus Uji *Counting diff-pairs*

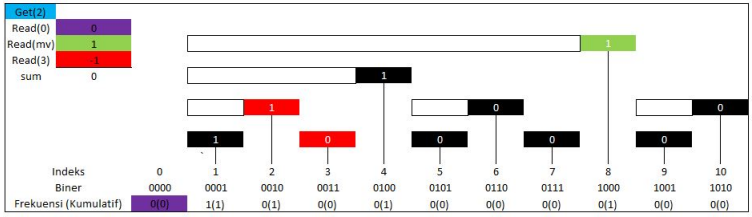
Pada Gambar 5.26 menunjukkan visualisasi fungsi $Get()$ dengan

indeks 1. Untuk setiap pemanggilan fungsi $Get()$ akan menghasilkan jumlah pasangan yang memiliki selisih absolut sama dengan atau lebih dari 2 dengan indeks 1. Fungsi $Get()$ akan memanggil fungsi $Read()$ sebanyak 3 kali untuk menghitung frekuensi kumulatif di bawah dan di atas indeks 1. Hasil yang didapatkan untuk fungsi $Get()$ adalah 0 karena elemen yang ada di dalam rentang batas kiri dan kanan sekarang hanya elemen bernilai 1.



Gambar 5.27 Visualisasi 3 *Fenwick Tree* Penyelesaian Contoh Kasus Uji *Counting diff-pairs*

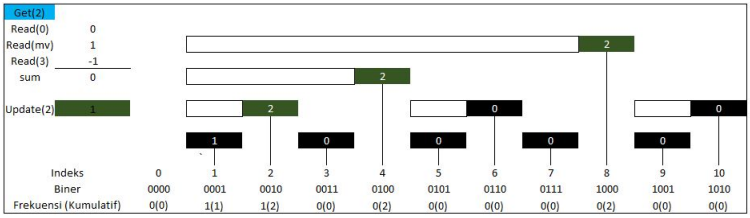
Pada Gambar 5.27 menunjukkan hasil fungsi $Update()$ dengan indeks 1. Untuk setiap indeks yang berkaitan dengan indeks 1 dalam bentuk biner akan diubah nilai frekuensi kumulatifnya.



Gambar 5.28 Visualisasi 4 *Fenwick Tree* Penyelesaian Contoh Kasus Uji *Counting diff-pairs*

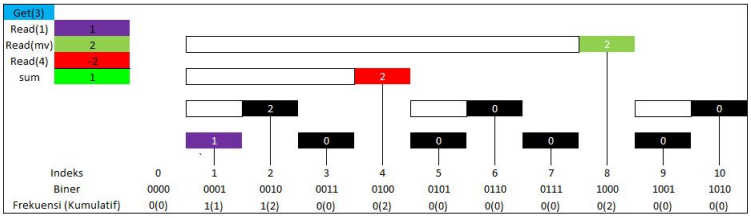
Pada Gambar 5.28 menunjukkan visualisasi fungsi $Get()$ dengan

indeks 2. Untuk setiap pemanggilan fungsi $Get()$ akan menghasilkan jumlah pasangan yang memiliki selisih absolut sama dengan atau lebih dari 2 dengan indeks 2. Hasil yang didapatkan untuk fungsi $Get()$ adalah 0 karena elemen yang ada di dalam rentang batas kiri dan kanan sekarang adalah 1 dan 2.



Gambar 5.29 Visualisasi 5 *Fenwick Tree* Penyelesaian Contoh Kasus Uji *Counting diff-pairs*

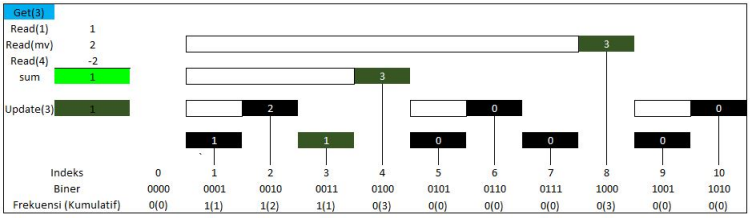
Pada Gambar 5.29 menunjukkan hasil fungsi $Update()$ dengan indeks 2. Untuk setiap indeks yang berkaitan dengan indeks 2 dalam bentuk biner akan diubah nilai frekuensi kumulatifnya.



Gambar 5.30 Visualisasi 6 *Fenwick Tree* Penyelesaian Contoh Kasus Uji *Counting diff-pairs*

Pada Gambar 5.30 menunjukkan visualisasi fungsi $Get()$ dengan indeks 3. Untuk setiap pemanggilan fungsi $Get()$ akan menghasilkan jumlah pasangan yang memiliki selisih absolut sama dengan atau lebih dari 2 dengan indeks 3. Hasil yang didapatkan

untuk fungsi $Get()$ adalah 1 karena terdapat elemen 1 dan 3 di dalam rentang batas kiri dan kanan sekarang.



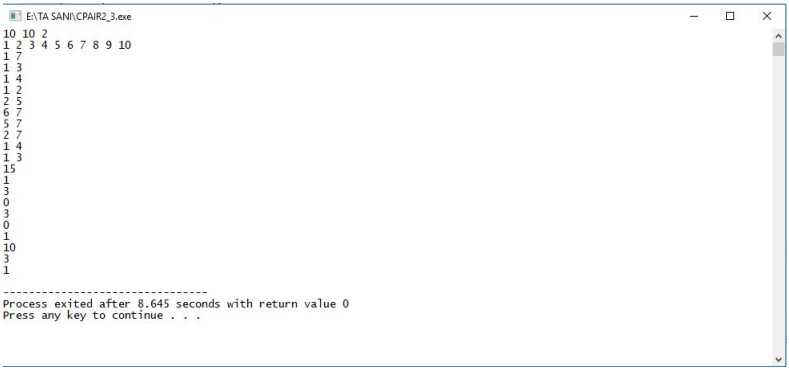
Gambar 5.31 Visualisasi 7 *Fenwick Tree* Penyelesaian Contoh Kasus Uji *Counting diff-pairs*

Pada Gambar 5.29 menunjukkan hasil fungsi $Update()$ dengan indeks 3. Untuk setiap indeks yang berkaitan dengan indeks 3 dalam bentuk biner akan diubah nilai frekuensi kumulatifnya. Langkah-langkah yang sudah divisualisasikan juga berlaku untuk semua operasi kueri yang akan diselesaikan oleh sistem.

Tabel 5.7 Tabel Hasil Pengerjaan Kasus Uji Coba Sederhana *Counting diff-pairs*

No	Batas Kiri	Batas Kanan	Total
1	1	7	15
2	1	3	1
3	1	4	3
4	1	2	0
5	2	5	3
6	6	7	0
7	5	7	1
8	2	7	10
9	1	4	3
10	1	3	1

Hasil keluaran dari sistem adalah total pasangan bilangan yang memiliki selisih absolut sama dengan atau lebih dari yang sudah dimasukkan untuk setiap operasi kueri sesuai dengan urutan masuk operasi kueri ke dalam sistem. Hasil yang didapatkan untuk masukan operasi kueri dari Tabel 5.3 dapat dilihat pada Tabel 5.7. Setelah melakukan visualisasi, sistem penyelesaian dijalankan diberi masukan pada Gambar 5.20. Keluaran sitem telah sesuai dengan permasalahan yang diberikan. Angka yang tercetak pada Gambar 5.32 sama dengan hasil yang ada pada Tabel 5.7.



Gambar 5.32 Keluaran Sistem Penyelesaian Kasus Uji Coba Sederhana *Counting diff-pairs*

Berdasarkan hasil uji coba tersebut, *Mo's Algorithm*, *Fenwick Tree* dan *Square Root Decomposition* yang didesain dan dijalankan pada penyelesaian permasalahan kasus uji coba sederhana dapat menyelesaikan permasalahan *Counting diff-pairs*.

5.3 Uji Coba Kinerja Penyelesaian Permasalahan *Counting diff-pairs*

Kompleksitas waktu untuk menyelesaikan permasalahan *Counting diff-pairs* dengan mengimplementasi *Square Root Decomposition*,

Mo's Algorithm, dan struktur data *Fenwick Tree* adalah $O((N + M)\sqrt{N}K \log mv)$. Rincian dari kompleksitas yang sudah disebutkan dapat dilihat sebagai berikut:

- N adalah jumlah elemen yang ada di dalam baris angka.
- M adalah jumlah operasi kueri yang akan dikerjakan.
- K adalah jumlah langkah yang dibutuhkan dalam menyelesaikan operasi kueri.
- $\sqrt{N}$ adalah konstanta *Square Root Decomposition*.
- mv adalah nilai maksimum dari angka yang menjadi indeks di dalam *Fenwick Tree*, yaitu 100.000.

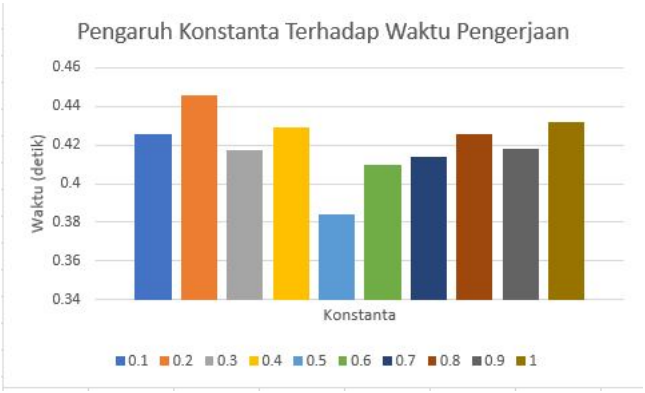
Dari rincian tersebut dapat dilakukan 2 uji coba kinerja penyelesaian permasalahan dengan menggunakan pengaruh nilai konstanta dan pengaruh penggunaan struktur data *Fenwick Tree*.

5.3.1 Pengaruh Nilai Konstanta terhadap Efisiensi Waktu Penyelesaian Permasalahan *Counting diff-pairs*

Tabel 5.8 Tabel Pengaruh Nilai Konstanta Terhadap Waktu Proses Sistem

No	Konstanta	Waktu (detik)
1	0.1	0.426
2	0.2	0.446
3	0.3	0.417
4	0.4	0.429
5	0.5	0.384
6	0.6	0.41
7	0.7	0.414
8	0.8	0.426
9	0.9	0.418
10	1.0	0.432

Pada uji coba ini nilai konstanta untuk membagi operasi kueri dibuat bervariasi antara 0.1 sampai 1.0. Nilai konstanta yang akan digunakan akan menjadi nilai pangkat dari N. Uji coba menggunakan jumlah data sebanyak 500, jumlah operasi kueri sebanyak 1000, dan selisih absolut bernilai 20. Setiap kasus uji coba dihasilkan dari fungsi *Generate* yang sudah didesain pada subbab 3.3.2.7. Untuk setiap waktu eksekusi akan tercatat dalam satuan detik. Hasil uji coba dari percobaan dapat dilihat pada Tabel 5.8 dan digambarkan dalam grafik pada Gambar 5.33.



Gambar 5.33 Grafik Pengaruh Nilai Konstanta Terhadap Waktu Proses Sistem

Dari hasil uji coba tersebut dapat disimpulkan bahwa konstanta dengan nilai 0.5 adalah konstanta paling efektif dengan waktu pengerjaan hanya 0.384 detik.

5.3.2 Pengaruh Struktur Data *Fenwick Tree* terhadap Efisiensi Waktu dan Memori Penyelesaian Permasalahan *Counting diff-pairs*

Pada uji coba ini akan membandingkan waktu penyelesaian permasalahan *Counting diff-pairs* dengan menggunakan struktur

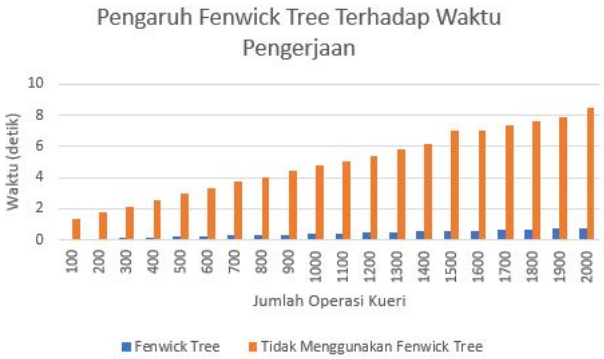
data *Fenwick Tree* dan tidak menggunakan struktur data tersebut. Uji coba menggunakan jumlah data sebanyak 350 dan selisih absolut bernilai 20.

Tabel 5.9 Tabel Pengaruh *Fenwick Tree* Terhadap Waktu Proses Sistem

No	Q	$BIT[x]$	$COUNTER[x]$
1	100	0.04	1.334
2	200	0.078	1.777
3	300	0.119	2.13
4	400	0.159	2.572
5	500	0.201	2.959
6	600	0.226	3.331
7	700	0.273	3.734
8	800	0.321	4.028
9	900	0.341	4.42
10	1000	0.386	4.771
11	1100	0.426	5.018
12	1200	0.486	5.403
13	1300	0.493	5.851
14	1400	0.536	6.139
15	1500	0.604	7.028
16	1600	0.611	7.006
17	1700	0.671	7.336
18	1800	0.684	7.623
19	1900	0.724	7.906
20	2000	0.759	845

Jumlah operasi kueri yang diuji akan bernilai variatif dari angka 100 sampai 2000. Setiap kasus uji coba dihasilkan dari fungsi *Generate* yang sudah didesain pada subbab 3.3.2.7. Untuk setiap waktu eksekusi akan tercatat dalam satuan detik. Hasil uji coba dari percobaan dapat dilihat pada Tabel 5.9 dan digambarkan

dalam grafik pada Gambar 5.34. Nama kolom untuk waktu pengerjaan sistem yang menggunakan *Fenwick Tree* adalah $BIT[x]$ sementara nama kolom untuk waktu pengerjaan sistem yang tidak menggunakan *Fenwick Tree* adalah $COUNTER[x]$.

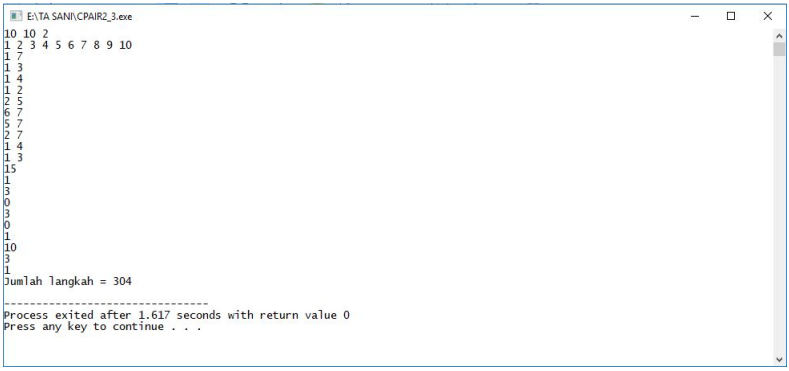


Gambar 5.34 Grafik Pengaruh *Fenwick Tree* Terhadap Waktu Proses Sistem

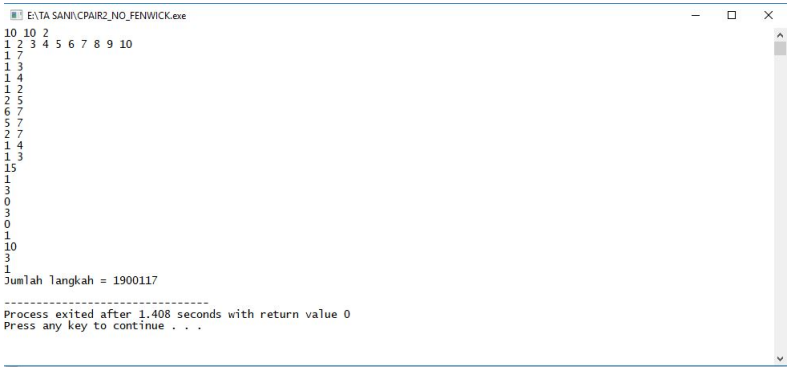
Dari hasil uji coba tersebut dapat disimpulkan bahwa sistem yang tidak menggunakan struktur data *Fenwick Tree* membutuhkan waktu lebih lama dan selalu meningkat untuk menyelesaikan permasalahan daripada sistem yang menggunakan struktur data *Fenwick Tree*.

Efek penggunaan struktur data *Fenwick Tree* terhadap jumlah memori yang dialokasi untuk penyelesaian permasalahan *Counting diff-pairs* adalah mengurangi memori yang digunakan. Setiap pencarian frekuensi kumulatif terhadap indeks tertentu akan membutuhkan langkah yang lebih sedikit jika menggunakan struktur data *Fenwick Tree* daripada tidak menggunakan struktur data *Fenwick Tree*. Jumlah langkah yang lebih sedikit membuat jumlah memori yang digunakan untuk menyelesaikan permasalahan akan lebih kecil. Perbandingan jumlah langkah tersebut dapat

dilihat pada Gambar 5.35 untuk sistem yang menggunakan struktur data *Fenwick Tree* dan Gambar 5.36 untuk sistem yang tidak menggunakan struktur data *Fenwick Tree*. Untuk masukan dari sistem sama seperti Gambar 5.20.



Gambar 5.35 Jumlah Langkah Pengerjaan Menggunakan *Fenwick Tree*



Gambar 5.36 Jumlah Langkah Pengerjaan Tidak Menggunakan *Fenwick Tree*

BAB VI
KESIMPULAN

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 *Counting diff-pairs* dapat diambil kesimpulan sebagai berikut:

1. Implementasi algoritma *Square Root Decomposition* dan *Mo's Algorithm* dapat menyelesaikan permasalahan *Counting diff-pairs* dengan benar.
2. Kompleksitas waktu yang dibutuhkan untuk seluruh proses adalah $O((N+M)\sqrt{N}K \log mv)$

6.2 Saran

Saran yang diberikan dalam pengembangan algoritma penyelesaian masalah *Counting diff-pairs* adalah melakukan optimasi *running time* agar mendapatkan waktu penyelesaian yang lebih baik.

Halaman ini sengaja dikosongkan

DAFTAR PUSTAKA

[1] **Sqrt (or Square Root) Decomposition Technique | Set 1 (Introduction) - GeeksforGeeks** [Online]. Available: https://www.geeksforgeeks.org/sqrt-square-root-decomposition-technique-set-1-introduction/. [Accessed 27-December-2017].

[2] **Binary Indexed Trees - TopCoder** [Online]. Available: https://www.topcoder.com/community/data-science/data-science-tutorials/binary-indexed-trees/. [Accessed 26-December-2017].

[3] **How exactly is the square root decomposition of queries (also sometimes referred to as Mo's Algorithm), used for offline processing of queries? - Quora** [Online]. Available: 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. [Accessed 27-December-2017].

[4] **CPAIR2 - Counting diff-pairs - SPOJ** [Online]. Available: https://www.spoj.com/problems/CPAIR2/. [Accessed 27-December-2017].

[5] **HRSIAM - Angry Siam - SPOJ** [Online]. Available: https://www.spoj.com/problems/HRSIAM/. [Accessed 27-December-2017].

[6] **ADAUNIQ - Ada and Unique Vegetable - SPOJ** [Online]. Available: https://www.spoj.com/problems/ADAUNIQ/. [Accessed 27-December-2017].

[7] S. Halim and F. Halim, **Competitive Programming 2**.Singapore, 2011.

Halaman ini sengaja dikosongkan

BIODATA PENULIS

Abdul Majid Hasani, lahir di Sidoarjo tanggal 29 Maret 1997. Penulis merupakan anak ketiga dari 3 bersaudara. Penulis telah menempuh pendidikan formal TK Hang Tuah X Juanda (2001 - 2003), SD Hang Tuah X Juanda (2003-2007,2008-2009), SD Negeri 5 Mataram (2007-2008), SMP Negeri 1 Sidoarjo (2009-2011), dan SMA Negeri 1 Sidoarjo (2011-2014). Penulis melanjutkan studi kuliah program sarjana di Jurusan Informatika ITS.

Selama kuliah di Informatika ITS, penulis mengambil bidang minat Algoritma Pemrograman (AP). Penulis pernah menjadi asisten dosen dan praktikum untuk mata kuliah Jaringan Komputer (2016) dan Matematika Informatika (2015). Selama menempuh perkuliahan penulis juga aktif di kegiatan organisasi dan kepanitiaan diantaranya menjadi staff Departemen Dalam Negeri HMTC ITS 2015, wakil koordinator Dana dan Usaha Schematics 2015, ketua Schematics 2016. Penulis dapat dihubungi melalui surel di
am.sani9050@gmail.com.