Penerapan Teknik Dekomposisi Square Root dan Algoritma Mo's pada Rancangan Algoritma Studi Kasus: SPOJ Klasik Counting Diff-Pairs

Abdul Majid Hasani, Rully Soelaiman dan Fajar Baskoro
Departemen Informatika, Fakultas Teknologi Informasi dan Komunikasi, Institut Teknologi Sepuluh Nopember (ITS)
e-mail: fajar@if.its.ac.id

***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*. Pada penelitian ini akan dirancang penyelesaian masalah yang disampaikan pada paragraf pertama dengan menggunakan algoritma *Square Root Decomposition*, *Mo's Algorithm*, dan struktur data *Fenwick Tree*. Solusi yang didesain memiliki kompleksitas waktu $O((N+M) N K \log mv)$, dimana *N* adalah jumlah elemen pada baris sekuen yang diberikan *M* adalah jumlah operasi kueri, $\sqrt{N}$ adalah konstanta, *K* adalah jumlah langkah peyelesaian, dan log *mv* adalah kompleksitas *Fenwick Tree*. Algoritma yang didesain dapat menyelesaikan permasalahan yang diberikan dengan benar. Waktu eksekusi program yang mengimplementasi algoritma yang dirancang tidak melebihi batas waktu eksekusi program yang telah diberikan, yaitu 1.78 detik. Sehingga dapat disimpulkan algoritma yang didesain dapat menyelesaikan permasalahan yang diberikan.**

***Kata Kunci*—*Fenwick tree*, *Mo's Algorithm*, *Offline Query*, *Square Root Decomposition*.**

I. PENDAHULUAN

PERMASALAHAN yang diangkat adalah permasalahan optimasi kueri yang ada di situs penilaian SPOJ dengan kode soal *CP AIR*2 dan judul *Counting diff-pairs* [1]. Diberikan sebuah sekuen bilangan *A* dengan jumlah *N*, *M* baris kueri, dan selisih mutlak bernilai *k*. Ketentuan dari operasi kueri adalah mencari jumlah pasangan angka dalam jarak rentang tertentu di dalam sekuen bilangan *A* yang memiliki selisih mutlak sama dengan atau lebih dari *k*.

Selisih mutlak dua buah bilangan didapatkan dari pengurangan bilangan *a* dan *b* dalam rentang kueri. Bilangan *k* 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* [1].

Algoritma *Square Root Decomposition* dan *Mo's Algorithm* digunakan untuk menyelesaikan permasalahan yang sudah dijelaskan. Algoritma *Square Root Decomposition* akan membagi operasi kueri ke dalam blok-blok kecil yang mempengaruhi urutan pengerjaan operasi kueri yang dimasukan. Operasi kueri yang dimasukan akan terlebih dahulu ditentukan urutan pengerjaannya supaya mendapatkan jumlah langkah paling optimal. Penyelesaian operasi kueri akan menggunakan pendekatan *Offline Query*, yaitu setiap jawaban dari satu operasi kueri akan digunakan untuk menjawab operasi kueri selanjutnya. *Mo's Algorithm* menyelesaikan operasi kueri yang sudah diurutkan dengan cara mengganti batas kiri dan kanan sekarang menjadi batas kiri dan kanan operasi kueri yang sedang dijalankan. Struktur data *Fenwick Tree* digunakan untuk membuat waktu penyelesaian permasalahan lebih optimal [2]. Tujuan dari penelitian ini adalah untuk menyelesaikan permasalahan, menguji kebenaran dan performa dari algoritma yang didesain.

Tabel 1.
Tabel urutan masuk operasi kueri permasalahaan *Counting diff-pairs*

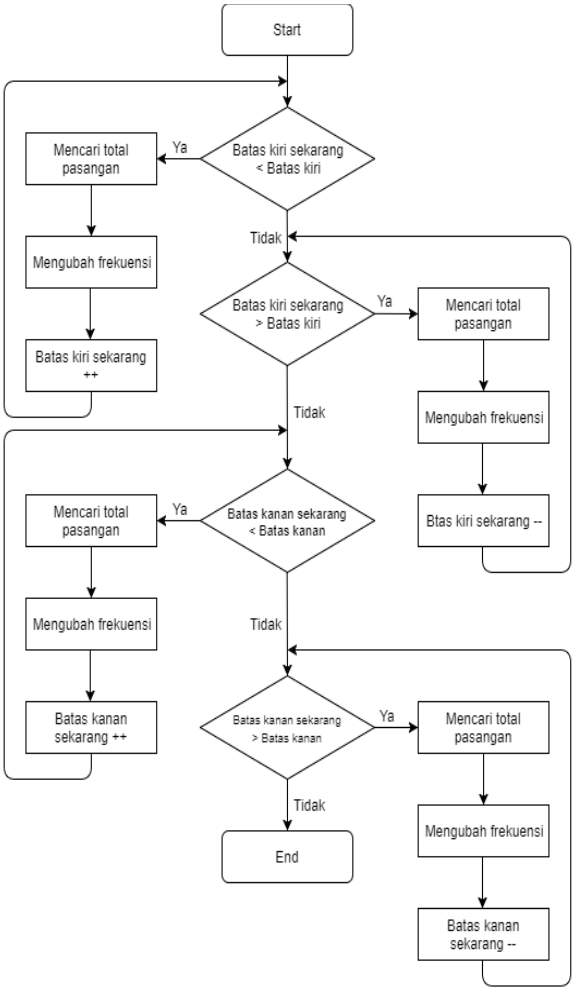
No	Batas Kiri	Batas Kanan
1	1	7
2	1	3
3	1	4
4	1	2
5	2	5
6	6	7
7	5	7
8	2	7
9	1	4
10	1	3

Tabel 2.
Tabel urutan pengerjaan operasi kueri permasalahaan *Counting diff-pairs*

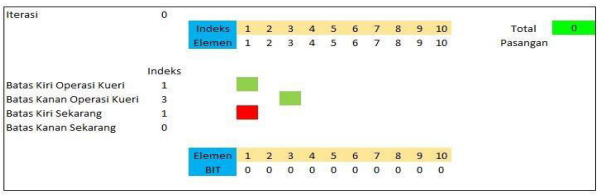
No	Urutan Masuk	Batas Kiri	Batas Kanan
1	2	1	3
2	4	1	2
3	10	1	3
4	3	1	4
5	5	2	5
6	9	1	4
7	1	1	7
8	8	2	7
9	6	6	7
10	7	5	7

II METODE PENYELESAIAN

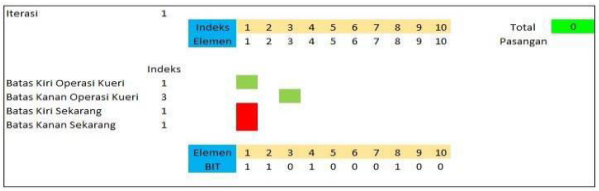
Perancangan algoritma yang optimal untuk menyelesaikan permasalahan yang diberikan dapat dilakukan dengan cara menyelesaikan permasalahan yang lebih sederhana dalam konteks yang sama. Diasumsikan



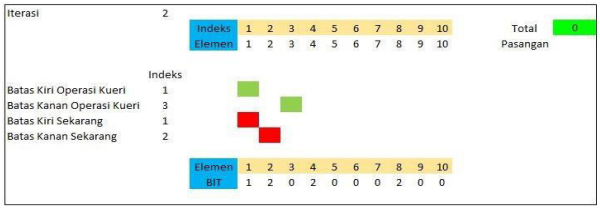
Gambar. 1. Alur kerja *Mo's Algorithm* pada penyelesaian permasalahan *Counting diff-pairs*



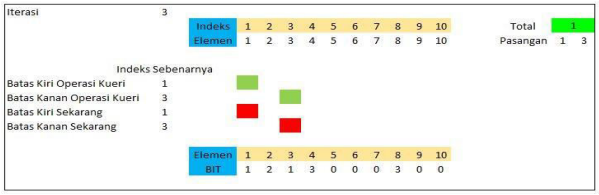
Gambar. 2. Visualisasi 1 penyelesaian contoh kasus uji *Counting diff-pairs*



Gambar. 3. Visualisasi 2 penyelesaian contoh kasus uji *Counting diff-pairs*



Gambar. 4. Visualisasi 3 penyelesaian contoh kasus uji *Counting diff-pairs*



Gambar. 5. Visualisasi 4 penyelesaian contoh kasus uji *Counting diff-pairs*

terdapat sebuah baris bilangan A = 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 10 operasi kueri, dan selisih absolut 2. Operasi kueri memiliki rentang yang berbeda-beda dan dapat dilihat pada Tabel 1.

Konstanta *Square Root Decomposition* yang digunakan untuk membagi operasi kueri ke dalam blok-blok kecil adalah $\sqrt{N}$. Batas kiri dan kanan dari setiap operasi kueri akan dibagi $\sqrt{N}$. Batas kiri dan kanan yang sudah dibagi oleh konstanta akan menjadi dasar pengurutan penyelesaian operasi kueri.

Penentuan konstanta dilakukan dengan cara uji coba dengan nilai konstanta yang variatif kemudian dipilih nilai konstanta yang menghasilkan waktu penyelesaian paling cepat [3]. Langkah selanjutnya adalah membuat urutan pengerjaan operasi kueri yang sudah dimasukkan. Pengurutan operasi kueri dilakukan untuk mengurangi jumlah langkah penyelesaian satu operasi kueri ke operasi kueri lainnya. Terdapat aturan *Mo's Algorithm* untuk pengurutan operasi kueri yang dapat dilihat sebagai berikut:

1. Operasi kueri prioritas pertama adalah semua operasi kueri yang ada di dalam blok terkecil dimulai dari indeks batas kiri yang paling kecil. Setelah blok pertama sudah diurutkan maka akan pindah ke blok selanjutnya.
2. Operasi kueri prioritas kedua adalah operasi kueri dengan batas kanan yang paling kecil.

Perubahan dari urutan masuk operasi kueri ke urutan pengerjaan operasi kueri dapat dilihat pada Tabel 2. Operasi kueri yang sudah diurutkan akan dikerjakan sesuai urutan pengerjaan dan setiap jawaban dari operasi kueri sebelumnya akan digunakan untuk menjawab operasi kueri selanjutnya. Hal tersebut membuat langkah pengerjaan operasi kueri menjadi lebih sedikit.

Langkah selanjutnya adalah melakukan operasi *Mo's Algorithm* untuk setiap operasi kueri. Terdapat batas kiri dan kanan sekarang pada *Mo's Algorithm* yang selalu berubah untuk setiap operasi kueri yang sedang dikerjakan. Alur kerja dari *Mo's Algorithm* dapat dilihat pada Gambar 1. Kegunaan dari *Mo's Algorithm* adalah untuk menemukan jumlah pasangan yang memiliki selisih absolut sama dengan atau lebih dari k [4].

Terdapat 4 kemungkinan dari *Mo's Algorithm* yang dapat dilihat sebagai berikut:

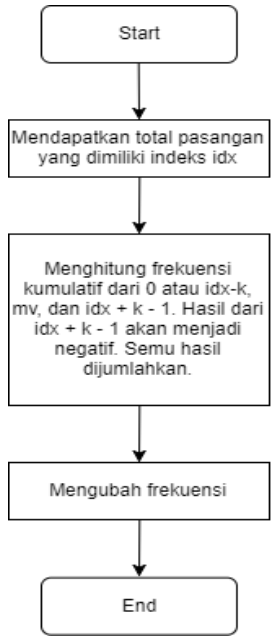
1. Batas kiri sekarang kurang dari batas kiri operasi kueri.
2. Batas kiri sekarang lebih dari batas kiri operasi kueri.
3. Batas kanan sekarang kurang dari batas kanan operasi kueri.
4. Batas kanan sekarang lebih dari batas kanan operasi kueri.

Setiap kemungkinan yang terjadi akan dilakukan penambahan/ pengurangan jumlah pasangan yang ada di dalam rentang operasi kueri sekarang, jumlah frekuensi bilangan yang ditunjuk oleh indeks sekarang di dalam *Fenwick Tree*, dan nilai dari batas kanan/kiri sekarang.

Penggunaan *Mo's Algorithm* akan mengurangi jumlah langkah pengerjaan operasi kueri yang sudah diurutkan jika dibandingkan dengan tidak menggunakan *Mo's Algorithm.*

Tabel 3.
Waktu maksimum, waktu minimum, waktu rata-rata, dan memori dari hasil uji coba *submission* ke situs penilaian SPOJ sebanyak 20 kali

Waktu Maksimum	1.85 detik
Waktu Minimum	1.78 detik
Waktu Rata-Rata	1.82 detik
Memori Maksimal	17 MB
Memori Minimal	17 MB
Memori Rata-Rata	17 MB

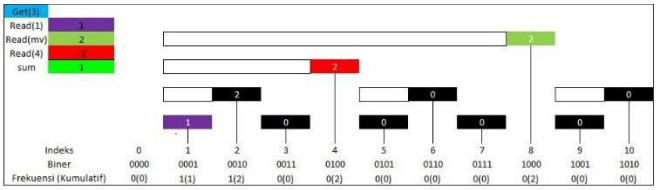


Gambar. 6. Alur kerja *Fenwick Tree* pada penyelesaian permasalahan *Counting diff-pairs*

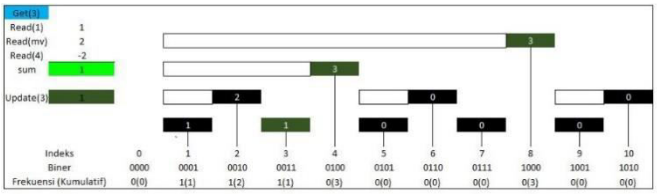
Langkah-langkah setiap iterasi *Mo's Algorithm* dalam pengerjaan operasi kueri dan perpindahan batas kiri dan kanan sekarang akan divisualisasikan dengan gambar. Contoh yang diambil adalah langkah-langkah *Mo's Algorithm* dalam mencari jumlah pasangan yang memiliki selisih absolut sama dengan atau lebih dari 2 pada operasi kueri dengan batas kiri 1 dan 3. Visualisasi setiap langkah penyelesaian dapat dilihat pada Gambar 2, 3, 4, dan 5.

Penggunaan struktur data *Fenwick Tree* bertujuan untuk menyimpan jumlah frekuensi elemen yang ada di dalam rentang operasi kueri sekarang. Setiap iterasi yang terjadi pada proses *Mo's Algorithm* akan memberikan efek terhadap *Fenwick Tree* yang dibentuk. Jumlah pasangan yang memiliki selisih absolut sama dengan atau lebih dari *k* didapatkan dengan cara menghitung frekuensi kumulatif untuk indeks yang ada di *Fenwick Tree*. Cara menghitung frekuensi kumulatif pada *Fenwick Tree* adalah dengan menjumlahkan frekuensi yang disimpan pada indeks tertentu dan indeks lainnya yang berhubungan dalam bentuk biner menggunakan *Least Significant One-bit* [5]. Selain menghitung frekuensi kumulatif, setiap perpindahan batas kiri dan kanan sekarang akan mempengaruhi jumlah frekuensi elemen yang ada di dalam *Fenwick Tree*.

Alur kerja pada *Fenwick Tree* dalam menentukan jumlah pasangan yang memilik selisih absolut sama dengan atau lebih dari *k* dapat dilihat pada Gambar 6. Cara menemukan

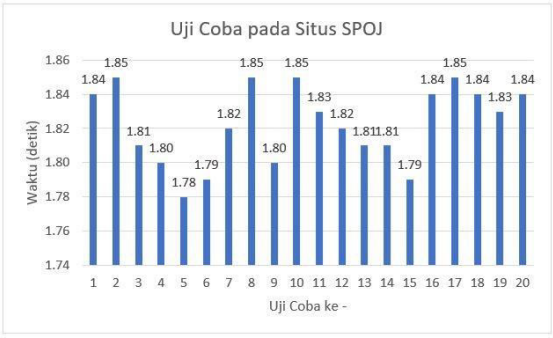


Gambar. 7. Visualisasi 1 *Fenwick Tree* penyelesaian contoh kasus uji *Counting diff-pairs*



Gambar. 8. Visualisasi 2 *Fenwick Tree* penyelesaian contoh kasus uji *Counting diff-pairs*

Gambar. 9. Hasil uji kebenaran dengan melakukan *submission* ke situs penilaian SPOJ



Gambar. 10. Hasil uji coba *submission* ke situs penilaian SPOJ sebanyak 20 kali

jumlah pasangan bilangan dengan selisih absolut sama dengan atau lebih dari *k* pada suatu indeks adalah mencari pasangan yang lebih kecil dan lebih besar dari indeks tersebut. Hal tersebut dilakukan dengan langkah-langkah sebagai berikut:

1. Menghitung frekuensi kumulatif dari pengurangan indeks dengan bilangan *k*. Hasil dari operasi ini adalah pasangan indeks yang memiliki nilai yang lebih kecil dari indeks.
2. Menghitung frekuensi kumulatif dari nilai maximum dan penjumlahan indeks dengan *k*-1. Hasil dari kedua operasi dikurangi untuk mendapatkan pasangan indeks yang memiliki nilai yang lebih besar dari indeks.

Hasil dari operasi yang sudah dijelaskan adalah total keseluruhan pasangan dari suatu indeks. Contoh visualisasi untuk mendapatkan pasangan dari indeks 3 dari *Fenwick Tree* dapat dilihat pada Gambar 7 dan 8. Jumlah frekuensi yang ada di dalam struktur data *Fenwick Tree* akan selalu berubah mengikuti perpindahan batas kiri dan kanan *Mo's Algorithm*.

III. UJI COBA DAN ANALISIS

Uji coba yang dilakukan pada permasalahan *Counting diff-pairs* adalah uji coba kebenaran pada situs penilaian SPOJ, pengaruh nilai konstanta *Square Root Decomposition* terhadap efisiensi waktu penyelesaian, dan pengaruh

struktur data *Fenwick Tree* terhadap efisiensi waktu dan memori penyelesaian.

A. Uji Coba Kebenaran

Uji coba kebenaran dilakukan dengan cara mengirimkan kode sumber ke situs penilaian SPOJ. Hasil uji coba kebenaran dan waktu eksekusi program saat pengumpulan kasus uji pada situs SPOJ ditunjukkan pada Gambar 9.

Tabel 4.
Tabel pengaruh nilai konstanta terhadap waktu proses system

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

Tabel 5.
Tabel pengaruh *Fenwick Tree* terhadap waktu proses system

No	*Q*	*BIT* [*x*]	*COU N T ER*[*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

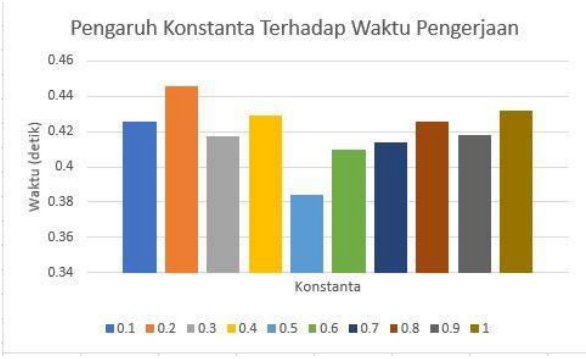
Berikutnya adalah pengujian performa dari algoritma yang dirancang dan diimplementasi dengan melakukan uji coba kebenaran pada situs penilaian SPOJ sebanyak 20 kali pengiriman untuk melihat variasi waktu dan memori yang dibutuhkan sistem penyelesaian. Hasil dari pengujian dapat dilihat pada grafik pada Gambar 10 dan Tabel 3.

B. Pengaruh Nilai Konstanta terhadap Efisiensi Waktu

Uji coba pengaruh nilai konstanta terhadap efisiensi waktu dilakukan untuk menemukan konstanta yang tepat agar mendapatkan waktu optimal penyelesaian permasalahan *Counting diff-pairs*. Nilai konstanta dibuat bervariasi antara0.1 sampai 1.0. Konstanta dijadikan nilai pangkat dari *N*. Uji coba menggunakan jumlah data sebanyak 500, jumlah operasi kueri sebanyak 1000, dan selisih absolut 20. Setiap kasus uji coba dihasilkan dari data *generator*. Waktu eksekusi dicatat dalam satuan detik. Hasil uji coba dapat dilihat pada Tabel 4 dan divisualisasikan dalam grafik pada Gambar 11. Dari hasil uji coba dapat disimpulkan konstanta 0.5 adalah konstanta paling efektif dengan waktu penyelesaian 0.384 detik.

C. Pengaruh Fenwick Tree terhadap Efisiensi Waktu dan Memori

Uji coba pengaruh struktur data *Fenwick Tree* terhadap



Gambar. 11. Grafik pengaruh nilai konstanta terhadap waktu proses system

Gambar. 12. Grafik pengaruh *Fenwick Tree* terhadap waktu proses system

efisiensi waktu dan memori dilakukan untuk membandingkan peforma sistem yang menggunakan struktur data *Fenwick Tree* dan tidak menggunakan struktur data *Fenwick Tree*. Peforma yang dibandingkan adalah waktu penyelesaian dan penggunaan memori. Uji coba menggunakan jumlah data sebanyak 350 dan selisih absolut 20. Jumlah operasi kueri bernilai variatif dengan rentang 100 sampai 2000. Setiap kasus uji coba dihasilkan dari data *generator*. Waktu eksekusi dicatat dalam satuan detik. Hasil uji coba dapat dilihat pada Tabel 5 dan divisualisasikan dengan grafik pada Gambar 12. Kolom *BIT* [*x*] adalah sistem dengan implementasi struktur data *Fenwick Tree*, kolom *COU N T ER* [*x*] adalah sistem yang tidak menggunakan implementasi struktur data *Fenwick Tree*, dan kolom *Q* adalah jumlah operasi kueri yang dimasukkan untuk diuji pada kedua sistem penyelesaian.

Sistem Peforma waktu yang dibutuhkan sistem dengan implementasi struktur data *Fenwick Tree* untuk menyelesaikan permasalahan lebih kecil daripada sistem yang tidak menggunakan *Fenwick Tree*. Penggunaan struktur data *Fenwick Tree* dapat mengurangi jumlah penggunaan memori karena terdapat perbedaan jumlah langkah pengerjaan operasi kueri. Selain itu dari hasil uji coba juga dapat disimpulkan bahwa sistem yang menggunakan struktur data *Fenwick Tree* menyelesaikan permasalahan lebih cepat daripada sistem yang tidak

menggunakan struktur data *Fenwick Tree*.

D. Analisis Kompleksitas

Kompleksitas waktu untuk menyelesaikan permasalahan *Counting diff-pairs* dengan mengimplementasi *Square Root Decomposition*, Mo's *Algorythm*, dan struktur data *Fenwick Tree adalah O((N + M) N K log mv). Rincian dari kompleksitas tersebut adalah*

1. *M* adalah jumlah operasi kueri.
2. *K* adalah jumlah langkah yang dibutuhkan untuk menyelesaikan operasi kueri.
3. *N* adalah konstanta *Square Root Decomposition.*
4. *mv* adalah nilai maksimum elemen dalam *Fenwick Tree*

Algoritma tersebut dapat menyelesaikan permasalahan yang diberikan.

IV. KESIMPULAN

Dari hasil uji coba yang telah dilakukan terhadap perancangan dan implementasi algoritma untuk menyelesaikan permasalahan klasik SPOJ CPAIR2 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 dibutuh√kan untuk seluruh proses adalah sebesar *O((N + M) N K log mv)*.

DAFTAR PUSTAKA

[1] Anonymous, "CPAIR2 - Counting diff-pairs - SPOJ," *www.spoj.com*. [Online]. Available: https://www.spoj.com/problems/CPAIR2/. [Accessed: 27-Dec-2017].

[2] Anonymous, "Binary Indexed Trees -TopCoder," *www.topcoder.com*. [Online]. Available: https://www.topcoder.com/community/data-science%0A/data-science-tutorials/binary-indexed-trees/. [Accessed: 26-Dec-2017].

[3] Anonymous, "Sqrt (or Square Root) Decomposition Technique — Set 1 (Introduction) - GeeksforGeeks," *www.geeksforgeeks.org*. [Online]. Available: https://www.geeksforgeeks.org/sqrt-square-root-decomposition%0A-technique-set-1-introduction/%0A. [Accessed: 27-Dec-2017].

[4] Quora, "How exactly is the square root decomposition of queries (also sometimes referred to as Mo's Algorithm), used for offline processing of queries?," *www.quora.com*. .

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