

**TUGAS BESAR**  
**DESAIN DAN ANALISIS ALGORITMA**



**Dibuat Oleh:**

Friskadini Ismayanti	(1301198496)
Muhammad Faisal Amir	(1301198497)
Ridho Maulana Cahyudi	(1301198515)

**TELKOM UNIVERSITY**  
**FAKULTAS INFORMATIKA**

**2020**

## A. Latar Belakang Permasalahan

Salah satu pendekatan yang dapat dilakukan untuk menyelesaikan permasalahan convex hull dapat menggunakan algoritma divide and conquer. Sedangkan pendekatan untuk menyelesaikan masalah 8 puzzle problem adalah metode branch and bound, pengembangan dari Program Linear di mana beberapa atau semua variabel keputusannya harus berupa integer.

Pemrograman bulat dibutuhkan ketika keputusan harus dilakukan dalam bentuk bilangan bulat (bukan pecahan yang sering terjadi bila kita gunakan metode simpleks). Model matematis dari pemrograman bulat sebenarnya sama dengan model linear programming, dengan tambahan batasan bahwa variabelnya harus bilangan bulat.

Algoritma Divide and Conquer merupakan algoritma yang sangat populer di dunia Ilmu Komputer. Divide and Conquer merupakan algoritma yang berprinsip memecah-mecah permasalahan yang terlalu besar menjadi beberapa bagian kecil sehingga lebih mudah untuk diselesaikan. Langkah-langkah umum algoritma Divide and Conquer :

1. Divide : Membagi masalah menjadi beberapa upa-masalah yang memiliki kemiripan dengan masalah semula namun berukuran lebih kecil ( idealnya berukuran hampir sama ).
2. Conquer : Memecahkan ( menyelesaikan ) masing-masing upa-masalah ( secara rekursif ).
3. Combine : Menggabungkan solusi masing-masing upa-masalah sehingga membentuk solusi masalah semula.

Objek masalah yang di bagi adalah masukan (input) atau instances yang berukuran n: tabel (larik), matriks dan sebagainya, bergantung pada masalahnya. Tiap-tiap upa-masalah mempunyai karakteristik yang sama (the same type) dengan karakteristik masalah asal, sehingga metode Divide and Conquer lebih natural diungkapkan dalam skema rekursif. Sesuai dengan karakteristik pembagian dan pemecahan masalah tersebut maka algoritma ini dapat berjalan baik pada persoalan yang bertipe rekursif (perulangan dengan memanggil dirinya sendiri).

Dengan demikian, algoritma ini dapat diimplementasikan dengan cara iteratif ( perulangan biasa ), karena pada prinsipnya iteratif hampir sama dengan rekursif. Salah satu penggunaan algoritma ini yang paling populer adalah dalam hal pengolahan data yang bertipe array ( elemen larik ). Mengapa ? Karena pengolahan array pada umumnya selalu menggunakan prinsip rekursif atau iteratif. Penggunaan secara spesifik adalah untuk mencari nilai minimal dan maksimal serta untuk mengurutkan elemen array.

Dalam hal pengurutan ini ada empat macam algoritma pengurutan yang berdasar pada algoritma Divide and Conquer yaitu merge sort, insert sort, quick sort dan selection sort. Merge sort dan Quick sort mempunyai kompleksitas algoritma  $O(n^2 \log n)$ . Hal ini lebih baik jika dibandingkan dengan pengurutan biasa dengan menggunakan algoritma brute force.

Algoritma B&B (Branch and Bound) adalah salah satu algoritma yang digunakan untuk pencarian jalur. Contoh yang dibahas kali ini adalah mengenai pencarian jalur yang melalui semua titik dengan biaya terendah. Algoritma ini memiliki 2 prinsip, yaitu:

1. Algoritma ini akan melakukan perhitungan secara rekursif, akan memecah masalah kedalam masalah-masalah kecil, sambil tetap menghitung nilai terendah / terbaik. Proses ini dinamakan branching.
2. Jika branching diterapkan secara sendirian, maka hasilnya akan tetap mencari setiap kemungkinan yang ada. Untuk meningkatkan performa, algoritma ini akan melakukan pencatatan biaya minimum sebagai bound dalam setiap perhitungan, sehingga untuk calon hasil jawaban yang diperkirakan akan melebihi bound akan dibuang karena tidak mungkin akan mencapai nilai terbaik.

Metode Branch and Bound adalah sebuah teknik algoritma yang secara khusus mempelajari bagaimana caranya memperkecil Search Tree menjadi sekecil mungkin. Sesuai dengan namanya, metode ini terdiri dari 2 langkah yaitu :

1. Branch yang artinya membangun semua cabang tree yang mungkin menuju solusi.
2. Bound yang artinya menghitung node mana yang merupakan active node (E-node) dan node mana yang merupakan dead node (D-node) dengan menggunakan syarat batas constraint (kendala).

## B. Teori Dasar Strategi dan Masalah yang diselesaikan

- Algoritma Divide and Conquer

### *Pemecahan Masalah Convex Hull dengan Algoritma Divide and Conquer*

Pada penyelesaian masalah pencarian Convex Hull dengan menggunakan algoritma Divide and Conquer, hal ini dapat dipandang sebagai generalisasi dari algoritma pengurutan merge sort.

Permasalahan convex hull adalah sebuah permasalahan yang memiliki aplikasi terapan yang cukup banyak, seperti pada permasalahan grafika komputer, otomasi desain, pengenalan pola (pattern recognition) dan penelitian operasi. Divide and Conquer adalah metode pemecahan masalah yang bekerja dengan membagi masalah menjadi beberapa upa-masalah yang lebih kecil, kemudian menyelesaikan masing-masing upa-masalah tersebut secara independent dan akhirnya menggabungkan solusi masing-masing upa-masalah sehingga menjadi solusi dari masalah semula.

Algoritma Divide and Conquer merupakan salah satu solusi dalam penyelesaian masalah convex hull. Algoritma ini ternyata memiliki kompleksitas waktu yang cukup kecil dan efektif dalam menyelesaikan permasalahan ini (jika dibandingkan algoritma lain). Selain itu juga, algoritma ini dapat digeneralisasi untuk permasalahan convex hull yang berdimensi lebih dari 3.

- Algoritma Branch And Bound

### *8 Puzzle Problem*

8 puzzle terdiri dari delapan ubin bergerak bernomor yang diatur dalam bingkai 3x3. Satu sel bingkai selalu kosong sehingga memungkinkan untuk memindahkan ubin bernomor yang berdekatan ke sel kosong. Teka-teki seperti itu diilustrasikan dalam diagram berikut.

Program ini mengubah konfigurasi awal menjadi konfigurasi tujuan. Solusi untuk masalah ini adalah urutan langkah yang sesuai, seperti "pindahkan ubin 5 ke kanan, pindahkan ubin 7 ke kiri, pindahkan ubin 6 ke bawah, dll".

Untuk memecahkan masalah menggunakan sistem produksi, kita harus menentukan basis data global, aturan, dan strategi kontrol. Untuk 8 masalah teka-teki yang sesuai dengan tiga komponen ini. Elemen-elemen ini adalah status masalah, gerakan dan tujuan. Dalam masalah ini, setiap konfigurasi ubin adalah keadaan. Himpunan semua konfigurasi dalam ruang status masalah atau ruang masalah, hanya ada 3.62.880 konfigurasi berbeda dari 8 ubin

dan ruang kosong. Setelah status masalah diidentifikasi secara konseptual, kita harus membuat representasi komputer, atau deskripsi tentangnya. Deskripsi ini kemudian digunakan sebagai basis data sistem produksi. Untuk 8-puzzle, deskripsi lurus ke depan adalah array matriks angka 3X3. Basis data global awal adalah uraian kondisi awal masalah ini. Secara virtual segala jenis struktur data dapat digunakan untuk menggambarkan status.

Suatu langkah mengubah satu status masalah menjadi status lainnya. 8-puzzle diartikan sebagai memiliki langkah-langkah berikut untuk bergerak. Pindahkan ruang kosong (kosong) ke kiri, pindah kosong ke atas, pindah kosong ke kanan dan pindah kosong ke bawah. Gerakan ini dimodelkan oleh aturan produksi yang beroperasi pada deskripsi negara dengan cara yang tepat.

Aturan masing-masing memiliki prasyarat yang harus dipenuhi oleh deskripsi negara agar mereka berlaku untuk deskripsi negara itu. Jadi prasyarat untuk aturan yang terkait dengan "pindah kosong" berasal dari persyaratan bahwa ruang kosong belum harus berada di baris atas. Kondisi tujuan masalah membentuk dasar untuk kondisi penghentian sistem produksi. Strategi kontrol berulang kali menerapkan aturan untuk menyatakan deskripsi sampai deskripsi keadaan tujuan diproduksi. itu juga melacak aturan yang telah diterapkan sehingga dapat menyusunnya menjadi urutan yang mewakili solusi masalah. Solusi untuk masalah 8-teka diberikan pada gambar berikut.

## C. Penjelasan Data dan Skenario Percobaan

### 1. Penjelasan Data

#### a. Algoritma Divide and Conquer

```
int[] sampleData = {4, -3, 5, -2, -1, 2, 6, -2};
```

Data diatas merupakan data uji yang akan di coba dalam algoritma divide and conquer untuk penyelesaian algoritma convex hull. Data tersebut merupakan data random yang bisa diganti dengan angka berapa pun.

#### b. Algoritma Branch and Bound

```
int[][] initial = {{1, 8, 2}, {0, 4, 3}, {7, 6, 5}};  
int[][] goal = {{1, 2, 3}, {4, 5, 6}, {7, 8, 0}};
```

Data diatas merupakan data uji yang akan di coba dalam algoritma branch and bound untuk penyelesaian algoritma 8 puzzle problem. Data tersebut merupakan data random yang bisa diganti dengan angka berapa pun, sama seperti data uji pada algoritma divide and conquer.

### 2. Skenario Percobaan

#### a. Algoritma Divide and Conquer

Skenario
<ol style="list-style-type: none"><li>1. Pertama-tama lakukan pengurutan terhadap titik-titik dari himpunan S yang diberika berdasarkan koordinat absis-X, dengan kompleksitas waktu <math>O(n \log n)</math>.</li><li>2. Jika <math> S  \leq 3</math>, maka lakukan pencarian convex hull secara brute-force dengan kompleksitas waktu <math>O(1)</math>. (Basis).</li><li>3. Jika tidak, partisi himpunan titik-titik pada S menjadi 2 buah himpunan A dan B, dimana A terdiri dari setengah jumlah dari <math> S </math> dan titik dengan koordinat absis-X yang terendah dan B terdiri dari setengah dari jumlah <math> S </math> dan titik dengan koordinat absis-X terbesar.</li><li>4. Secara rekursif lakukan penghitungan terhadap <math>HA = \text{conv}(A)</math> dan <math>HB = \text{conv}(B)</math>.</li><li>5. Lakukan penggabungan (merge) terhadap kedua hull tersebut menjadi convex hull, H, dengan menghitung dan mencari upper dan lower tangents untuk HA dan HB dengan mengabaikan semua titik yang berada diantara dua buah tangen ini.</li></ol>

#### b. Algoritma Branch and Bound

Skenario
<ol style="list-style-type: none"><li>1. Masukkan simpul akar ke dalam antrian S. Jika simpul akar adalah simpul solusi yang ingin dicapai, maka solusi telah ditemukan. Pencarian selesai.</li><li>2. Jika antrian S kosong, maka solusi tidak ditemukan. Pencarian selesai.</li></ol>

3. Jika S tidak kosong, mata pilih dari antrian simpul yang memiliki *cost* paling kecil. Jika terdapat beberapa simpul dengan nilai *cost* yang minimal, maka pilih satu secara sembarang
4. Jika simpul yang dipilih adalah simpul solusi, maka solusi telah ditemukan. Pencarian selesai. Jika simpul yang dipilih bukan simpul solusi, maka bangkitkan anak-anak dari simpul tersebut. Jika simpul tidak memiliki anak, maka kembali ke langkah 2
5. Untuk setiap anak dari simpul yang dipilih, hitung *cost* dan masukkan anak-anak simpul tersebut ke dalam antrian S
6. Ulangi langkah 2

## D. Hasil Percobaan

### 1. Algoritma Divide and Conquer

#### a. Code Program

- Node.java

1	<code>package com.frogobox.branchbound;</code>
2	
3	<code>/**</code>
4	<code> * Created by Faisal Amir</code>
5	<code> * FrogoBox Inc License</code>
6	<code> * =====</code>
7	<code> * divide-conquer-branch-bound</code>
8	<code> * Copyright (C) 07/05/2020.</code>
9	<code> * All rights reserved</code>
10	<code> * -----</code>
11	<code> * Name      : Muhammad Faisal Amir</code>
12	<code> * E-mail    : faisalamircs@gmail.com</code>
13	<code> * Github    : github.com/amirisback</code>
14	<code> * LinkedIn  : linkedin.com/in/faisalamircs</code>
15	<code> * -----</code>
16	<code> * FrogoBox Software Industries</code>
17	<code> * com.frogobox.branchbound</code>
18	<code> */</code>
19	
20	<code>public class Node {</code>
21	
22	<code>    public Node parent;</code>
23	<code>    public int[][] matrix;</code>
24	
25	<code>    // Blank tile cordinates</code>
26	<code>    public int x, y;</code>
27	
28	<code>    // Number of misplaced tiles</code>
29	<code>    public int cost;</code>
30	
31	<code>    // The number of moves so far</code>
32	<code>    public int level;</code>
33	
34	<code>    public Node(int[][] matrix, int x, int y, int newX, int newY, int</code>
35	<code>level, Node parent) {</code>
36	<code>        this.parent = parent;</code>
37	<code>        this.matrix = new int[matrix.length][];</code>
38	<code>        for (int i = 0; i &lt; matrix.length; i++) {</code>
39	<code>            this.matrix[i] = matrix[i].clone();</code>
40	<code>        }</code>
41	<code>    // Swap value</code>



42	<code>this.matrix[x][y] = this.matrix[x][y] + this.matrix[newX][newY];</code>
43	<code>this.matrix[newX][newY] = this.matrix[x][y] - this.matrix[newX][newY];</code>
44	<code>this.matrix[x][y] = this.matrix[x][y] - this.matrix[newX][newY];</code>
45	
46	<code>this.cost = Integer.MAX_VALUE;</code>
47	<code>this.level = level;</code>
48	<code>this.x = newX;</code>
49	<code>this.y = newY;</code>
50	<code>}</code>
51	
52	<code>}</code>

- Algorithm.java

1	<code>package com.frogobox.branchbound;</code>
2	
3	<code>import java.util.ArrayList;</code>
4	<code>import java.util.List;</code>
5	<code>import java.util.PriorityQueue;</code>
6	
7	<code>/**</code>
8	<code> * Created by Faisal Amir</code>
9	<code> * FrogoBox Inc License</code>
10	<code> * =====</code>
11	<code> * divide-conquer-branch-bound</code>
12	<code> * Copyright (C) 07/05/2020.</code>
13	<code> * All rights reserved</code>
14	<code> * -----</code>
15	<code> * Name : Muhammad Faisal Amir</code>
16	<code> * E-mail : faisalamircs@gmail.com</code>
17	<code> * Github : github.com/amirisback</code>
18	<code> * LinkedIn : linkedin.com/in/faisalamircs</code>
19	<code> * -----</code>
20	<code> * FrogoBox Software Industries</code>
21	<code> * com.frogobox.branchbound</code>
22	<code> */</code>
23	<code>public class Algorithm {</code>
24	
25	<code>    public int dimension = 3;</code>
26	
27	<code>    // Bottom, Left, top, right</code>
28	<code>    int[] row = {1, 0, -1, 0};</code>
29	<code>    int[] col = {0, -1, 0, 1};</code>
30	
31	<code>    public int calculateCost(int[][] initial, int[][] goal) {</code>
32	<code>        int count = 0;</code>
33	<code>        int n = initial.length;</code>
34	<code>        for (int i = 0; i &lt; n; i++) {</code>

35	for (int j = 0; j < n; j++) {
36	if (initial[i][j] != 0 && initial[i][j] != goal[i][j]) {
37	count++;
38	}
39	}
40	}
41	return count;
42	}
43	
44	public void printMatrix(int[][] matrix) {
45	System.out.println("Resolve Puzzle");
46	for (int[] ints : matrix) {
47	for (int j = 0; j < matrix.length; j++) {
48	System.out.print(ints[j] + " ");
49	}
50	System.out.println();
51	}
52	}
53	
54	public boolean isSafe(int x, int y) {
55	return (x >= 0 && x < dimension && y >= 0 && y < dimension);
56	}
57	
58	public void printPath(Node root) {
59	if (root == null) {
60	return;
61	}
62	
63	printPath(root.parent);
64	printMatrix(root.matrix);
65	System.out.println();
66	
67	}
68	
69	public boolean isSolvable(int[][] matrix) {
70	int count = 0;
71	List<Integer> array = new ArrayList<Integer>();
72	
73	for (int[] ints : matrix) {
74	for (int j = 0; j < matrix.length; j++) {
75	array.add(ints[j]);
76	}
77	}
78	
79	Integer[] anotherArray = new Integer[array.size()];
80	array.toArray(anotherArray);
81	
82	for (int i = 0; i < anotherArray.length - 1; i++) {
83	for (int j = i + 1; j < anotherArray.length; j++) {

84	<b>if</b> (anotherArray[i] != 0 && anotherArray[j] != 0 &&
85	anotherArray[i] > anotherArray[j]) {
86	count++;
87	}
88	}
89	
90	<b>return</b> count % 2 == 0;
91	}
92	
93	<b>public void</b> solve( <b>int</b> [][] initial, <b>int</b> [][] goal, <b>int</b> x, <b>int</b> y) {
94	PriorityQueue<Node> pq = <b>new</b> PriorityQueue<Node>(1000, (a, b) ->
95	(a.cost + a.level) - (b.cost + b.level));
96	Node root = <b>new</b> Node(initial, x, y, x, y, 0, <b>null</b> );
97	root.cost = calculateCost(initial, goal);
98	pq.add(root);
99	<b>while</b> (!pq.isEmpty()) {
100	Node min = pq.poll();
101	<b>if</b> (min.cost == 0) {
102	printPath(min);
103	<b>return</b> ;
104	}
105	<b>for</b> ( <b>int</b> i = 0; i < 4; i++) {
106	<b>if</b> (isSafe(min.x + row[i], min.y + col[i])) {
107	Node child = <b>new</b> Node(min.matrix, min.x, min.y,
108	min.x + row[i], min.y + col[i], min.level + 1, min);
109	child.cost = calculateCost(child.matrix, goal);
110	pq.add(child);
111	}
112	}
113	}
114	
115	
116	<b>public static void</b> initGoal( <b>int</b> [][] goalState) {
117	<b>for</b> ( <b>int</b> [] ints : goalState) {
118	<b>for</b> ( <b>int</b> j = 0; j < goalState.length; j++) {
119	System.out.print(ints[j] + " ");
120	}
121	System.out.println();
122	}
123	}
124	
125	<b>public static void</b> initInitial( <b>int</b> [][] initialState) {
126	<b>for</b> ( <b>int</b> [] ints : initialState) {
127	<b>for</b> ( <b>int</b> j = 0; j < initialState.length; j++) {
128	System.out.print(ints[j] + " ");
129	}
130	System.out.println();

131	}
132	}
133	
134	<b>public static void</b> initState( <b>int</b> [][] initialState, <b>int</b> [][] goalState)
135	{
136	System.out.println("Initial State");
137	initInitial(initialState);
138	System.out.println();
139	System.out.println("Goal State");
140	initGoal(goalState);
141	}
142	}

- Main.java

1	<b>package</b> com.frogobox.branchbound;
2	
3	<b>import static</b> com.frogobox.branchbound.Algorithm.*;
4	
5	/**
6	* Created by Faisal Amir
7	* FrogoBox Inc License
8	* =====
9	* divide-conquer-branch-bound
10	* Copyright (C) 07/05/2020.
11	* All rights reserved
12	* -----
13	* Name : Muhammad Faisal Amir
14	* E-mail : faisalamircs@gmail.com
15	* Github : github.com/amirisback
16	* LinkedIn : linkedin.com/in/faisalamircs
17	* -----
18	* FrogoBox Software Industries
19	* com.frogobox.branchbound
20	*/
21	
22	
23	<b>public class</b> Main {
24	
25	<b>public static void</b> main(String[] args) {
26	
27	System.out.println("Java program to solve the 8 puzzle problem (using branch and bound algorithm)");
28	
29	// *** SAMPLE DATA ***
30	<b>int</b> [][] initial = {{1, 8, 2}, {0, 4, 3}, {7, 6, 5}};
31	<b>int</b> [][] goal = {{1, 2, 3}, {4, 5, 6}, {7, 8, 0}};
32	

33	");	System.out.println("-----
34		initState(initial, goal);
35	");	System.out.println("-----
36		
37		// White tile coordinate
38		int x = 1, y = 0;
39		
40		Algorithm branchBound = new Algorithm();
41		if (branchBound.isSolvable(initial)) {
42		branchBound.solve(initial, goal, x, y);
43		} else {
44	solve");	System.out.println("The given initial is impossible to
45		}
46		System.out.println("-----");
47		System.out.println("Finish Result : ");
48		initGoal(goal);
49		
50	}	
51		
52	}	

## b. Data (Sample Data)

```
// *** SAMPLE DATA ***
int[] sampleData = {4, -3, 5, -2, -1, 2, 6, -2};
```

## c. Screen Shot Hasil Program

```

C:\Program Files\Java\jdk-9.0.4\bin\java.exe "-javaagent:D:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2019.3.3\lib\idea_rt.jar=60436:D:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2019.3.3\bin" -Dfile.encoding=UTF-8 -classpath E:\PROJECT\PROJECT_JAVA\divide-conquer-branch-bound\out\production\divide-conquer-branch-bound com.frogbox.divideconquer.Main
Max sum is 11; it goes from 0 to 6
Max sum is 11; it goes from 0 to 6
Max sum is 11; it goes from 0 to 6
Max sum is 11
Algorithm #4 N = 10 time = 0 microsec
Algorithm #3 N = 10 time = 0 microsec
Algorithm #2 N = 10 time = 0 microsec
Algorithm #1 N = 10 time = 0 microsec
Algorithm #4 N = 100 time = 2 microsec
Algorithm #3 N = 100 time = 1 microsec
Algorithm #2 N = 100 time = 3 microsec
Algorithm #1 N = 100 time = 50 microsec
Algorithm #4 N = 1000 time = 30 microsec
Algorithm #3 N = 1000 time = 12 microsec
Algorithm #2 N = 1000 time = 198 microsec
Algorithm #1 N = 1000 time = 54513 microsec
Algorithm #4 N = 10000 time = 323 microsec
Algorithm #3 N = 10000 time = 125 microsec
Algorithm #2 N = 10000 time = 18310 microsec
Algorithm #1 N = 10000 time = 56512000 microsec
Algorithm #4 N = 100000 time = 3838 microsec
Algorithm #3 N = 100000 time = 1300 microsec
Algorithm #2 N = 100000 time = 1835000 microsec
Algorithm #4 N = 1000000 time = 42221 microsec
Algorithm #3 N = 1000000 time = 15335 microsec
Algorithm #2 N = 1000000 time = 210850000 microsec
Process finished with exit code 0
  
```

## 2. Algoritma Branch and Bound

### a. Code Program

- Algorithm.java

1	<code>package com.frogobox.divideconquer;</code>
2	
3	<code>import java.util.Random;</code>
4	
5	<code>/**</code>
6	<code> * Created by Faisal Amir</code>
7	<code> * FrogoBox Inc License</code>
8	<code> * =====</code>
9	<code> * divide-conquer-branch-bound</code>
10	<code> * Copyright (C) 07/05/2020.</code>
11	<code> * All rights reserved</code>
12	<code> * -----</code>
13	<code> * Name      : Muhammad Faisal Amir</code>
14	<code> * E-mail    : faisalamircs@gmail.com</code>
15	<code> * Github    : github.com/amirisback</code>
16	<code> * LinkedIn  : linkedin.com/in/faisalamircs</code>
17	<code> * -----</code>
18	<code> * FrogoBox Software Industries</code>
19	<code> * com.frogobox.divideconquer</code>
20	<code> */</code>
21	<code>public class Algorithm {</code>
22	
23	<code>    public static int seqStart = 0;</code>
24	<code>    public static int seqEnd = -1;</code>
25	<code>    private static Random rand = new Random();</code>
26	<code>    /**</code>
27	<code>     * Cubic maximum contiguous subsequence sum algorithm.</code>
28	<code>     * seqStart and seqEnd represent the actual best sequence.</code>
29	<code>     */</code>
30	
31	<code>    /**</code>
32	<code>     * Cubic maximum contiguous subsequence sum algorithm.</code>
33	<code>     * seqStart and seqEnd represent the actual best sequence.</code>
34	<code>     */</code>
35	<code>    public static int maxSubSum1(int[] a) {</code>
36	<code>        int maxSum = 0;</code>
37	
38	<code>        for (int i = 0; i &lt; a.length; i++)</code>
39	<code>            for (int j = i; j &lt; a.length; j++) {</code>
40	<code>                int thisSum = 0;</code>
41	
42	<code>                for (int k = i; k &lt;= j; k++)</code>
43	<code>                    thisSum += a[k];</code>
44	
45	<code>                if (thisSum &gt; maxSum) {</code>

46	maxSum = thisSum;
47	seqStart = i;
48	seqEnd = j;
49	}
50	}
51	
52	return maxSum;
53	}
54	
55	/**
56	* Quadratic maximum contiguous subsequence sum algorithm.
57	* seqStart and seqEnd represent the actual best sequence.
58	*/
59	public static int maxSubSum2(int[] a) {
60	int maxSum = 0;
61	
62	for (int i = 0; i < a.length; i++) {
63	int thisSum = 0;
64	for (int j = i; j < a.length; j++) {
65	thisSum += a[j];
66	
67	if (thisSum > maxSum) {
68	maxSum = thisSum;
69	seqStart = i;
70	seqEnd = j;
71	}
72	}
73	}
74	
75	return maxSum;
76	}
77	
78	/**
79	* Linear-time maximum contiguous subsequence sum algorithm.
80	* seqStart and seqEnd represent the actual best sequence.
81	*/
82	public static int maxSubSum3(int[] a) {
83	int maxSum = 0;
84	int thisSum = 0;
85	
86	for (int i = 0, j = 0; j < a.length; j++) {
87	thisSum += a[j];
88	
89	if (thisSum > maxSum) {
90	maxSum = thisSum;
91	seqStart = i;
92	seqEnd = j;
93	} else if (thisSum < 0) {
94	i = j + 1;
95	thisSum = 0;

96	}
97	}
98	
99	return maxSum;
100	}
101	
102	/**
103	* Recursive maximum contiguous subsequence sum algorithm.
104	* Finds maximum sum in subarray spanning a[left..right].
105	* Does not attempt to maintain actual best sequence.
106	*/
107	private static int maxSumRec(int[] a, int left, int right) {
108	int maxLeftBorderSum = 0, maxRightBorderSum = 0;
109	int leftBorderSum = 0, rightBorderSum = 0;
110	int center = (left + right) / 2;
111	
112	if (left == right) // Base case
113	return a[left] > 0 ? a[left] : 0;
114	
115	int maxLeftSum = maxSumRec(a, left, center);
116	int maxRightSum = maxSumRec(a, center + 1, right);
117	
118	for (int i = center; i >= left; i--) {
119	leftBorderSum += a[i];
120	if (leftBorderSum > maxLeftBorderSum)
121	maxLeftBorderSum = leftBorderSum;
122	}
123	
124	for (int i = center + 1; i <= right; i++) {
125	rightBorderSum += a[i];
126	if (rightBorderSum > maxRightBorderSum)
127	maxRightBorderSum = rightBorderSum;
128	}
129	
130	return max3(maxLeftSum, maxRightSum,
131	maxLeftBorderSum + maxRightBorderSum);
132	}
133	
134	/**
135	* Return maximum of three integers.
136	*/
137	private static int max3(int a, int b, int c) {
138	return a > b ? a > c ? a : c : b > c ? b : c;
139	}
140	
141	/**
142	* Driver for divide-and-conquer maximum contiguous
143	* subsequence sum algorithm.
144	*/
145	public static int maxSubSum4(int[] a) {



146	<code>return a.length &gt; 0 ? maxSumRec(a, 0, a.length - 1) : 0;</code>
147	<code>}</code>
148	
149	<code>public static void getTimingInfo(int n, int alg) {</code>
150	<code>int[] test = new int[n];</code>
151	
152	<code>long startTime = System.currentTimeMillis();</code>
153	<code>;</code>
154	<code>long totalTime = 0;</code>
155	
156	<code>int i;</code>
157	<code>for (i = 0; totalTime &lt; 4000; i++) {</code>
158	<code>for (int j = 0; j &lt; test.length; j++)</code>
159	<code>// *** SAMPLE DATA ***</code>
160	<code>test[j] = rand.nextInt(100) - 50;</code>
161	
162	<code>switch (alg) {</code>
163	<code>case 1:</code>
164	<code>maxSubSum1(test);</code>
165	<code>break;</code>
166	<code>case 2:</code>
167	<code>maxSubSum2(test);</code>
168	<code>break;</code>
169	<code>case 3:</code>
170	<code>maxSubSum3(test);</code>
171	<code>break;</code>
172	<code>case 4:</code>
173	<code>maxSubSum4(test);</code>
174	<code>break;</code>
175	<code>}</code>
176	
177	<code>totalTime = System.currentTimeMillis() - startTime;</code>
178	<code>}</code>
179	
180	<code>System.out.println("Algorithm #" + alg + "\t"</code>
181	<code>+ "N = " + test.length</code>
182	<code>+ "\ttime = " + (totalTime * 1000 / i) + " microsec");</code>
183	<code>}</code>
184	
185	<code>}</code>

- Main.java

1	<code>package com.frogobox.divideconquer;</code>
2	
3	<code>import static com.frogobox.divideconquer.Algorithm.*;</code>
4	
5	<code>/**</code>
6	<code> * Created by Faisal Amir</code>
7	<code> * FrogoBox Inc License</code>

8	<code>* =====</code>
9	<code>* divide-conquer-branch-bound</code>
10	<code>* Copyright (C) 07/05/2020.</code>
11	<code>* All rights reserved</code>
12	<code>* -----</code>
13	<code>* Name : Muhammad Faisal Amir</code>
14	<code>* E-mail : faisalamircs@gmail.com</code>
15	<code>* Github : github.com/amirisback</code>
16	<code>* LinkedIn : linkedin.com/in/faisalamircs</code>
17	<code>* -----</code>
18	<code>* Frogobox Software Industries</code>
19	<code>* com.frogobox.divideconquer</code>
20	<code>*/</code>
21	
22	<code>public final class Main {</code>
23	
24	<code>    public static void main(String[] args) {</code>
25	<code>        // *** SAMPLE DATA ***</code>
26	<code>        int[] sampleData = {4, -3, 5, -2, -1, 2, 6, -2};</code>
27	<code>        int maxSum;</code>
28	
29	<code>        maxSum = maxSubSum1(sampleData);</code>
30	<code>        System.out.println("Max sum is " + maxSum + "; it goes from " +</code>
31	<code>seqStart + " to " + seqEnd);</code>
32	<code>        maxSum = maxSubSum2(sampleData);</code>
33	<code>        System.out.println("Max sum is " + maxSum + "; it goes from " +</code>
34	<code>seqStart + " to " + seqEnd);</code>
35	<code>        maxSum = maxSubSum3(sampleData);</code>
36	<code>        System.out.println("Max sum is " + maxSum);</code>
37	
38	<code>        // Get some timing info</code>
39	<code>        for (int n = 10; n &lt;= 1000000; n *= 10) {</code>
40	<code>            for (int alg = 4; alg &gt;= 1; alg--) {</code>
41	<code>                if (alg == 1 &amp;&amp; n &gt; 50000)</code>
42	<code>                    continue;</code>
43	<code>                getTimingInfo(n, alg);</code>
44	<code>            }</code>
45	<code>        }</code>
46	
47	<code>    }</code>
48	<code>}</code>

## b. Data (Sample Data)

```
// *** SAMPLE DATA ***
int[][] initial = {{1, 8, 2}, {0, 4, 3}, {7, 6, 5}};
int[][] goal = {{1, 2, 3}, {4, 5, 6}, {7, 8, 0}};
```

## c. Screen Shot Hasil Program

The first screenshot shows the initial state and goal state of a puzzle. The initial state is a 3x3 grid with numbers 1, 8, 2 in the first row, 0, 4, 3 in the second row, and 7, 6, 5 in the third row. The goal state is a 3x3 grid with numbers 1, 2, 3 in the first row, 4, 5, 6 in the second row, and 7, 8, 0 in the third row. The program is titled "BranchBound" and is running in the IntelliJ IDEA IDE.

```
"C:\Program Files\Java\jdk-9.0.4\bin\java.exe" "-javaagent:D:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2019.3\lib\idea_rt.jar=59718:D:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2019.3\bin" -Dfile.encoding=UTF-8 -classpath E:\PROJECT\PROJECT_JAVA\divide-conquer-branch-bound\out\production\divide-conquer-branch-bound com.frogbox.branchbound.BranchBound  
Java program to solve the 8 puzzle problem (using branch and bound algorithm)  
-----  
Initial State  
1 8 2  
0 4 3  
7 6 5  
  
Goal State  
1 2 3  
4 5 6  
7 8 0  
-----  
Resolve Puzzle  
1 8 2  
0 4 3  
7 6 5  
  
Resolve Puzzle  
1 8 2  
4 0 3  
7 6 5
```

The second screenshot shows the step-by-step resolution of the puzzle. The program is titled "BranchBound" and is running in the IntelliJ IDEA IDE. The output shows the following steps:

```
Resolve Puzzle  
1 0 2  
4 8 3  
7 6 5  
  
Resolve Puzzle  
1 2 0  
4 8 3  
7 6 5  
  
Resolve Puzzle  
1 2 3  
4 8 0  
7 6 5  
  
Resolve Puzzle  
1 2 3  
4 8 5  
7 6 0  
  
Resolve Puzzle  
1 2 3  
4 8 5  
7 0 6
```

```

Run: BranchBound
7 0 6

Resolve Puzzle
1 2 3
4 0 5
7 8 6

Resolve Puzzle
1 2 3
4 5 0
7 8 6

Resolve Puzzle
1 2 3
4 5 6
7 8 0

-----
Finish Result :
1 2 3
4 5 6
7 8 0

Process finished with exit code 0

```

## Analisa Order Growth

Orders of Growth Order of Growth adalah Tingkat pertumbuhan waktu eksekusi algoritma jika ukuran input bertambah. Membandingkan Orders of Growth Algoritma A dan B merupakan algoritma untuk menyelesaikan permasalahan yang sama. Untuk input berukuran  $n$ , waktu eksekusi algoritma A adalah  $T_A(n)$  sedangkan waktu eksekusi algoritma B adalah  $T_B(n)$ . Orders of growth mana yang paling besar?

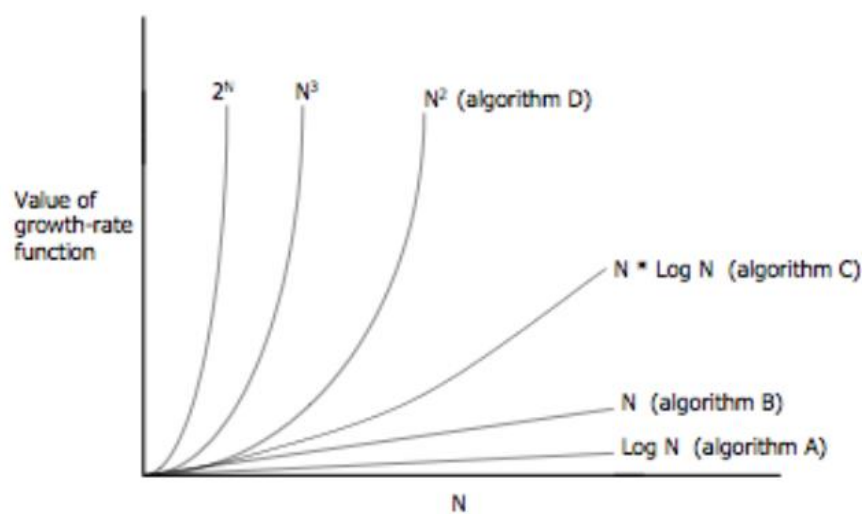
$$\lim_{n \rightarrow \infty} \frac{T_A(n)}{T_B(n)}$$

- 0 maka  $OoG T_A(n) < OoG T_B(n)$
- C maka  $OoG T_A(n) = OoG T_B(n)$
- $\infty$  maka  $OoG T_A(n) > OoG T_B(n)$

Kelas-Kelas Order of Growth Makin ke bawah, Order of Growth nya makin besar

C	constant
logN	logarithmic
N	linear
NlogN	
$N^2$	quadratic
$N^3$	cubic
$2^N$	exponential
N!	factorial

Grafik Kelas-Kelas Order of Growth



#### Sifat Order of Growth

- Misal  $T(n) = T_1(n) + T_2(n) + \dots + T_i(n)$   
Maka OoG  $T(n) = \max \text{OoG}(T_1(n), T_2(n), \dots, T_i(n))$
- Misal  $T(n) = cf(n)$   
Maka OoG  $T(n) = f(n)$

#### Kelas-kelas Order of Growth

- Waktu pelaksanaan algoritma adalah tetap, tidak bergantung pada ukuran input.
- $(\log n)$  Kompleksitas waktu logaritmik berarti laju pertumbuhan waktunya berjalan lebih lambat daripada pertumbuhan  $n$ .
- $(n)$  Bila  $n$  dijadikan dua kali semula, maka waktu pelaksanaan algoritma juga dua kali semula.

- $(n \log n)$  Bila  $n$  dijadikan dua kali semula, maka  $n \log n$  menjadi lebih dari dua kali semula (tetapi tidak terlalu banyak)
- $(n^2)$  Bila  $n$  dinaikkan menjadi dua kali semula, maka waktu pelaksanaan algoritma meningkat menjadi empat kali semula.
- $(n^3)$  Bila  $n$  dinaikkan menjadi dua kali semula, waktu pelaksanaan algoritma meningkat menjadi delapan kali semula.
- $(2^n)$  Bila  $n$  dijadikan dua kali semula, waktu pelaksanaan menjadi kuadrat kali semula.
- $(n!)$  Bila  $n$  dijadikan dua kali semula, maka waktu pelaksanaan algoritma menjadi faktorial dari  $2n$ .

## **E. Kesimpulan**

Untuk penyelesaian problem convex hull sangat cocok menggunakan algoritma divide and conquer, sedangkan untuk penyelesaian masalah 8 puzzle problem sangat cocok menggunakan algoritma branch and bound.

Hal tersebut di dasari oleh hasil percobaan di mana efektifitas dari waktu compiler sangat cepat dan masalah terpecahkan.

## F. Daftar Pustaka

Kirti\_Mangal. (2020, Mei 14). Retrieved from Geeks for Geeks: <https://www.geeksforgeeks.org/8-puzzle-problem-using-branch-and-bound/>

Raghavavaiah, G. V. (2020, Mei 14). *Blogspot*. Retrieved from artificialintelligence-notes: <http://artificialintelligence-notes.blogspot.com/2010/07/8-puzzle-problem.html>

Winata, R. (2020, Mei 14). *Rendy Winata Blog*. Retrieved from Blogspot: <http://rendywinata.blogspot.com/2015/01/algoritma-divide-and-conquer-pada-java.html>



# Lampiran

## Petunjuk Menjalankan Program

### a. Menggunakan IntelliJ IDEA JetBrains Community




















1. Ikuti tutorial resmi dari IntelliJ - <https://www.jetbrains.com/help/idea/creating-and-running-your-first-java-application.html>
2. Buka project ini
3. Algorithm Divide and Conquer
  - [root\_project]/divide-conquer-branch-bound/src/com/frogobox/divideconquer
  - run - Main.java
4. Algorithm Branch and Bound
  - [root\_project]/divide-conquer-branch-bound/src/com/frogobox/branchbound
  - run - Main.java

### b. Menggunakan Command Prompt

1. Buka CMD
2. Pergi ke folder tujuan
  - Algorithm Divide and Conquer  
[root\_project]/divide-conquer-branch-bound/src/com/frogobox/divideconquer
  - Algorithm Branch and Bound  
[root\_project]/divide-conquer-branch-bound/src/com/frogobox/branchbound
3. Run code dengan Javac dan Java
  - Algorithm Divide and Conquer  
\$ javac \*.java  
\$ java com.frogobox.divideconquer.Main
  - Algorithm Branch and Bound  
\$ javac \*.java  
\$ java com./frogobox.branchbound.Main

## Strukture Project

Link Project : <https://github.com/amirisback/divide-conquer-branch-bound>

- ▼  **divide-conquer-branch-bound** E:\PROJECT
  - >  .idea
  - >  docs
  - ▼  src
    - ▼  com.frogobox
      - ▼  branchbound
        -  Algorithm
        -  Main
        -  Node
      - ▼  divideconquer
        -  Algorithm
        -  Main
      - ▼  team
        -  TeamName
    - >  task
      -  .gitignore
      -  divide-conquer-branch-bound.iml
      -  README.md
    - >  External Libraries

# Kontribusi Kelompok

## 1. Friskadini Ismayanti

- Perumusan Masalah
- Penentuan Strategi Algoritma
- Pembuatan Program
- Penyelesaian Masalah
- Laporan
- Video

## 2. Muhammad Faisal Amir

- Perumusan Masalah
- Penentuan Strategi Algoritma
- Pembuatan Program
- Penyelesaian Masalah
- Laporan
- Video

## 3. Ridho Maulana Cahyudi

- Perumusan Masalah
- Penentuan Strategi Algoritma
- Pembuatan Program
- Penyelesaian Masalah
- Laporan
- Video

Semua anggota kelompok mengerjakan semua tugas secara bersama sama dan saling membantu.