# Analisis Perbandingan Efisiensi Pencarian NIM Mahasiswa Teknik Informatika ITB dengan Metode *Binary Search* dan *Interpolation Search*

M Farrel Danendra Rachim - 13521048
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
13521048@std.stei.itb.ac.id

***Abstract*—Identitas mahasiswa merupakan sebuah komponen penting dalam lingkungan universitas, khususnya untuk kepentingan administrasi dan penilaian bagi para dosen. Nomor Induk Mahasiswa (NIM) adalah elemen unik yang dimiliki setiap mahasiswa dan berfungsi untuk membedakan setiap mahasiswa yang pernah mengikuti pendidikan di Teknik Informatika di Institut Teknologi Bandung (ITB). Data mahasiswa seperti nama dan jurusan lebih mudah dicari dengan menggunakan NIM. Algoritma *binary search* dan *interpolation search* dapat digunakan untuk menyelesaikan persoalan ini dengan mencari posisi NIM yang ingin dicari di dalam sebuah larik berisi NIM-NIM yang telah terurut menaik secara cepat dan efisien. Dalam makalah ini, penulis akan membandingkan algoritma mana yang lebih efisien dalam menentukan posisi sebuah NIM dalam suatu larik berisi NIM.**

***Keywords—NIM; Decrease and Conquer; Binary Search; Interpolation Search***

## I. PENDAHULUAN

Seiring bertambahnya jumlah mahasiswa yang melaksanakan pendidikan di universitas, pengelolaan data seorang mahasiswa juga harus lebih efisien dan cepat. Pengelolaan data mahasiswa yang efisien sangatlah penting untuk urusan penilaian bagi para dosen, khususnya dengan banyaknya ujian dan tugas yang dilalui oleh mahasiswa dalam satu semester. Selain itu, data mahasiswa perlu dikelola dengan baik untuk kepentingan administrasi, seperti pembiayaan UKM per semester, pelaksanaan evaluasi dan legalisasi akademik, dan aktivitas lainnya.

Pencarian Nomor Induk Mahasiswa (NIM) merupakan cara yang paling ampuh untuk memperoleh data mahasiswa dengan efisien, karena NIM dibuat unik untuk tiap mahasiswa. Dengan mencari NIM terlebih dahulu, pengguna juga dapat memperoleh data lain dari mahasiswa yang bermanfaat seperti nama, fakultas, jurusan, tahun ajaran awal, dan data lainnya. Pencarian NIM juga sangat berguna bagi kalangan mahasiswa sendiri – khususnya untuk tugas berkelompok yang kadang mengharuskan setiap anggota kelompok untuk mengetahui NIM sesama anggota kelompoknya agar dapat dicantumkan dalam hasil tugas tersebut.

Dengan meningkatnya kebutuhan untuk pencarian NIM yang mangkus, perlu diimplementasikan sebuah algoritma pencarian yang sesuai. Penulis menerapkan algoritma *Binary Search* dan *Interpolation Search* berdasarkan konsep *Decrease and Conquer* untuk menyelesaikan persoalan ini. Kedua algoritma ini dapat menghasilkan solusi yang optimal dan cepat dalam mencari NIM berdasarkan input dari pengguna, khususnya jika data berukuran besar.

| NIM Anggota 1 | NIM Anggota 2 | NIM Anggota 3 | NIM Anggota 4 | NIM Anggota 5 |
|---|---|---|---|---|
| 13569420 | 13542069 | 13569000 | 13569690 | 13513337 |
| 13521055 | 13521069 | 13521081 | 13521085 | 13521103 |
| 13521119 | 13521121 | 13521041 | 13521099 | |
| 13521061 | 13521065 | 13521097 | 13521109 | |
| 13521071 | 13521159 | 13521169 | 13521173 | |
| 13521043 | 13521087 | 13521149 | 13521157 | |
| 13521111 | 13521139 | 13521057 | 13521125 | |
| 13521115 | 13521131 | 13521133 | 13521161 | |
| 13521051 | 13521059 | 13521077 | 13521107 | |
| 13521095 | 13521129 | 13521151 | 13521171 | |
| 13521123 | 13521135 | 13521153 | 13521155 | |
| 13521083 | 13521075 | 13521091 | 13521063 | |
| 13521049 | 13521067 | 13521079 | 13521137 | |
| 13521045 | 13521089 | 13521093 | 13521101 | |
| 13521047 | 13521105 | 13521141 | 13521165 | |
| 13521073 | 13521127 | 13521143 | 13521145 | 13521163 |

Gambar 1. Pembagian Kelompok Berdasarkan NIM Mahasiswa (dokumentasi penulis)

## II. TEORI DASAR

### A. *Algoritma Decrease and Conquer*

Algoritma *decrease and conquer* adalah algoritma yang mereduksi persoalan menjadi dua bagian yang selanjutnya akan dipilih sebuah sub-persoalan untuk diproses dan diselesaikan selanjutnya.

Algoritma ini memiliki dua tahapan penting:

1. *Decrease*: Persoalan direduksi menjadi dua buah upa-persoalan yang lebih kecil.

2. *Conquer*: Akan dipilih sebuah upa-persoalan yang akan diselesaikan secara rekursif.

Algoritma *decrease and conquer* berbeda dengan algoritma *divide and conquer* yang memproses semua upa-persoalan dan menggabung (*combine*) semua solusi untuk setiap upa-persoalan tersebut. Algoritma *decrease and conquer* hanya memproses satu upa-persoalan saja.

Algoritma *decrease and conquer* memiliki tiga buah varian:

1. *Decrease by a constant*: Ukuran persoalan direduksi berdasarkan suatu konstanta yang sama untuk setiap iterasi algoritma yang berlangsung. Pada umumnya, penerapan varian algoritma ini memiliki konstanta yang bernilai 1. Beberapa contoh algoritma yang termasuk dalam varian ini adalah *selection sort* dan persoalan perpangkatan $a^n$.
2. *Decrease by a constant factor*: Ukuran persoalan direduksi berdasarkan nilai faktor konstanta yang sama untuk setiap iterasi algoritma yang berlangsung. Pada umumnya, penerapan varian algoritma ini memiliki faktor konstanta yang bernilai 2. Beberapa contoh algoritma yang termasuk dalam varian ini adalah *binary search* dan persoalan perkalian petani Rusia.
3. *Decrease by variable size*: Ukuran persoalan direduksi secara bervariasi untuk setiap iterasi algoritma yang berlangsung. Beberapa contoh algoritma yang termasuk dalam varian ini adalah *interpolation search* dan pencarian median.

## *B. Algoritma Binary Search*

Algoritma *binary search* merupakan algoritma pencarian yang berguna untuk mencari posisi sebuah elemen dalam sebuah larik yang sudah terurut, baik meningkat maupun menurun. Algoritma ini juga termasuk algoritma *decrease and conquer* yang mereduksi persoalan berdasarkan nilai faktor konstanta yang sama untuk setiap iterasi algoritma yang berlangsung. *Binary search* paling umum digunakan saat mencari sebuah koleksi data yang besar, terurut, kontigu, dan tidak memiliki struktur yang kompleks.

Algoritma *binary search* selalu mencari elemen di bagian tengah dari sebuah upalarik. Algoritma ini dapat diimplementasikan secara iteratif atau rekursif, namun dalam makalah ini penulis hanya akan menggunakan metode rekursif karena relasinya sebagai sebuah jenis algoritma *decrease and conquer*. Berikut cara kerja umum algoritma *binary search*:

1. Atur indeks “low” sebagai elemen pertama larik dan indeks “high” sebagai elemen terakhir larik.
2. Atur indeks “mid” sebagai rata-rata dari indeks “low” dan “high”:

$$\text{mid} = \text{low} + (\text{high} - \text{low}) / 2 \qquad (1)$$

3. Jika elemen di indeks “mid” adalah elemen tujuan, kembalikan indeks “mid” tersebut.
4. Jika nilai elemen “mid” lebih besar dari nilai elemen yang dicari (misalkan K), elemen K diselidiki di upalarik sebelah kiri elemen di tengah larik. Jika tidak ditemukan, elemen K dicari di upalarik sebelah kanan elemen di tengah larik.
5. Ulangi sampai pencarian cocok atau upalarik mencapai nol.

Algoritma *binary search* memiliki jumlah operasi perbandingan elemen larik sebanyak:

$$T(n) = \begin{cases} 0 & ,n = 0 \\ 1 + T(n/2) & ,n > 0 \end{cases}$$

Gambar 2. Relasi Banyak Operasi Perbandingan Elemen Larik dalam Algoritma *Binary Search* [2]

Dalam bentuk non-rekurens, relasi di atas dapat disederhanakan secara iteratif dengan cara:

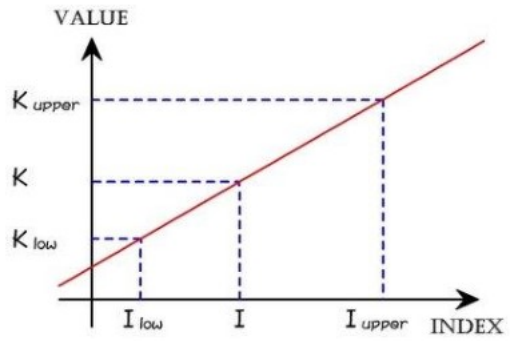
$$\begin{aligned} T(n) &= 1 + T(n/2) \\ &= 1 + (1 + T(n/4)) = 2 + T(n/4) \\ &= 2 + (1 + T(n/8)) = 3 + T(n/8) \\ &= \ldots \\ &= k + T(n/2^k) \end{aligned}$$

Jika diasumsikan ukuran larik adalah perpangkatan dua ($n = 2^k$ atau $k = {}^2\log n$), maka kompleksitas waktu algoritma *binary search* adalah $O({}^2\log n)$.

## *C. Algoritma Interpolation Search*

Algoritma *interpolation search* merupakan algoritma pencarian yang merupakan varian dari *binary search*. Algoritma ini bekerja dengan cara menyelidiki posisi dari nilai yang ingin dicari. Agar algoritma ini dapat bekerja, kumpulan data harus disusun secara terurut dan terdistribusi secara merata (setiap data bisa diperoleh dengan peluang yang sama). Ada beberapa jenis interpolasi, seperti *linear interpolation, polynomial interpolation, spline interpolation,* dan lain sebagainya. Dalam makalah ini, penulis menggunakan interpolasi linear.

Dalam sebuah koleksi data (misalnya sebuah larik) yang disusun secara terurut dan terdistribusi secara merata, hubungan antara nilai data dan indeks (posisi) data dapat divisualisasikan dengan graf berikut:




Gambar 3. Graf Hubungan Nilai Data dengan Indeks Data [2]

Nilai data dan indeks data berbanding lurus secara linear – semakin besar nilai data, semakin besar juga posisi data tersebut di dalam larik. Perbandingan kedua hal tersebut dapat dirumuskan sebagai berikut:

$$\frac{K - K_{low}}{K_{upper} - K_{low}} = \frac{I - I_{low}}{I_{upper} - I_{low}}$$

Gambar 4. Rumus Perbandingan Nilai Data dengan Indeks Data [2]

Adapun $I_{low}$ adalah indeks ujung kiri larik, $I_{upper}$ adalah indeks ujung kanan larik, $K_{low}$ adalah elemen minimum di dalam larik (pada indeks $I_{low}$), dan $K_{upper}$ adalah elemen maksimum di dalam larik (pada indeks $I_{upper}$).

Algoritma *binary search* memulai pemeriksaan dari nilai tengah sebuah koleksi data, sedangkan *interpolation search* mampu memulai pencarian dari lokasi yang berbeda berdasarkan input nilai pengguna. Posisi nilai elemen K dapat diprediksi dalam larik dengan rumus berikut:

$$I = I_{low} + (I_{upper} - I_{low}) \times \frac{K - K_{low}}{K_{upper} - K_{low}}$$

Gambar 5. Rumus Pencarian Indeks Nilai K [2]

Algoritma ini bekerja dengan cara berikut:

1. Hitung nilai I sesuai dengan rumus pada gambar 5.
2. Jika elemen cocok, kembalikan indeks item.
3. Jika elemen di tengah larik lebih besar dari elemen yang dicari (misalkan K), elemen K diselidiki di upalarik sebelah kiri elemen di tengah larik. Jika tidak ditemukan, elemen K dicari di upalarik sebelah kanan elemen di tengah larik.
4. Ulangi sampai pencarian cocok atau upalarik mencapai nol.

Algoritma *interpolation search* memiliki kompleksitas waktu O(n) dalam kasus terburuk (jika data terdistribusi secara sembarang), dan O($^2$log ($^2$log n)) dalam kasus terbaik (jika data di dalam larik terdistribusi secara merata).

## III. PEMODELAN ALGORITMA

### A. *Pemodelan Basis Data Mahasiswa*

Data yang digunakan untuk program ini dibuat dalam beberapa larik yang menyimpan berbagai data yang berbeda, merepresentasikan atribut-atribut mahasiswa dalam sebuah tabel pada basis data. Larik yang digunakan adalah:

- Larik penyimpan NIM yang sudah terurut membesar. Jika larik tidak terurut membesar, algoritma *binary search* dan *interpolation search* tidak akan bekerja dengan baik.
- Larik penyimpan nama mahasiswa yang berkorespondensi dengan urutan NIM dalam larik NIM
- Larik penyimpan kota asal mahasiswa yang berkorespondensi dengan urutan NIM dalam larik NIM
- Larik penyimpan IPK mahasiswa yang berkorespondensi dengan urutan NIM dalam larik NIM

Dalam percobaan ini, penulis menggunakan koleksi NIM yang memenuhi format 13521zzz. Artinya, mahasiswa terdaftar di ITB pada tahun ajaran 2021/2022 pada jurusan Teknik Informatika. Penulis akan mengambil sampel sebanyak 43 mahasiswa. Nama, alamat, dan IPK mahasiswa akan diasumsikan oleh penulis.

```
int[] nim = {13521046, 13521047, 13521048, 13521049,
             13521050, 13521051, 13521052, 13521053,
             13521054, 13521055, 13521056, 13521057, 13521058,
             13521059, 13521060, 13521061, 13521062, 13521063,
             13521064, 13521065, 13521066, 13521067, 13521068,
             13521069, 13521070, 13521071, 13521072, 13521073,
             13521074, 13521075, 13521076, 13521077, 13521078,
             13521079, 13521080, 13521081, 13521082, 13521083,
             13521084, 13521085, 13521086, 13521087, 13521088};

String[] namamhs = {"Muhammad Ali", "Bryan Ali", "Dewi Syifa", "Jonathan Osbert",
                    "Gita Hana", "Nabila Vania", "Wulandari Dinda", "Bintang Ariel",
                    "Ariel Imam", "Muhammad Bagus", "Fadhlan Rio", "Jessica Nadya",
                    "Sharon Aulia", "Henry Angga", "Dimas Kevin", "Hilman Denny",
                    "Michelle Sari", "Adrian Jonah", "Robby Iqbal", "Ansel Rendy",
                    "Maya Indah", "Tika Amanda", "Fitri Karina", "Rifqi Timothy",
                    "Debby Fani", "Rahmat Teguh", "Felix Rama", "Yoga Robby",
                    "Yudianto Gerry", "Gilang Yoga", "Dea Salsa", "Nadia Hani",
                    "Ken Linda", "Daniel Gideon", "Leo Fadlika", "Ricky Hardi",
                    "Kevin Jonathan", "Gita Bella", "Cindy Dewi", "Annisa Maria",
                    "Nurul Arif", "Jessica Cindy", "Michael Angga"};
```

Gambar 6. Contoh Larik Penyimpan NIM dan Nama Mahasiswa (dokumentasi penulis)

```java
double[] ipkmhs = {3.2,3.38,3.26,3.78,3.32,3.58,3.12,3.44,
                   3.55,3.93,3.59,3.55,3.38,3.27,3.23,3.15,
                   3.79,3.3,3.24,3.15,3.74,3.52,3.54,3.28,
                   3.95,3.07,3.23,3.58,3.49,3.08,3.46,3.48,
                   3.72,3.83,3.25,3.64,3.3,3.18,3.58,
                   3.32,3.25,3.16,3.36};

String[] kotaasalmhs = {"Jakarta Selatan", "Bandung", "Bandung", "Tangerang Selatan",
                        "Bandung", "Tangerang Selatan", "Pontianak", "Cilacap",
                        "Semarang", "Surabaya", "Padang", "Bandung",
                        "Jakarta Utara", "Tangerang", "Bandung", "Bandung",
                        "Bandung", "Bandung", "Jakarta Pusat", "Jakarta Selatan",
                        "Bandung", "Bandung", "Bandung", "Semarang",
                        "Surabaya", "Manado", "Banjarmasin", "Bekasi",
                        "Bekasi", "Jakarta Timur", "Jakarta Timur", "Jakarta Barat",
                        "Cilegon", "Bandung", "Tangerang Selatan", "Bandung",
                        "Bogor", "Cianjur", "Jakarta Selatan", "Bandung",
                        "Bandung", "Jakarta Selatan", "Bandung"};
```

Gambar 7. Contoh Larik Penyimpan IPK dan Kota Asal Mahasiswa (dokumentasi penulis)

## *B. Perancangan Program*

Penulis menggunakan algoritma *binary search* dan *interpolation search* untuk mencari posisi indeks NIM hasil input pengguna pada larik "nim". Lalu, dari indeks yang diperoleh, penulis juga akan memperoleh elemen dari larik "namamhs", "ipkmhs", dan "kotaasalmhs" yang berkorespondensi dengan indeks dari larik "nim". Akhirnya, kumpulan data tersebut, termasuk NIM dan posisi indeks, akan dicetak. Penulis menggunakan bahasa Java untuk membuat program ini.

Penulis mengimplementasikan fungsi "binarysearchalgorithm" yang mengimplementasikan algoritma *binary search* dan mengembalikan indeks dari "wantednim", serta "interpolationsearchalgorithm" yang mengimplementasikan algoritma *interpolation search* dan mengembalikan indeks dari "wantednim". Kedua metode ini memiliki cara kerja yang hampir sama, hanya saja cara mencari nilai "mid" untuk setiap proses rekursif berbeda untuk setiap fungsi. Metode "binarysearchalgorithm" memakai persamaan (1) untuk mencari nilai "mid", sedangkan metode "interpolationsearchalgorithm" memakai rumus pada gambar 5 untuk mencari nilai "mid". Selain itu, di "interpolation search" harus dicek jika "wantednim" di antara elemen "nim" di indeks "low" dan elemen "nim" di indeks "high" agar tidak terjadi *exception* indeks keluar dari *range* larik "nim". Kedua algoritma akan mengembalikan nilai -1 jika "wantednim" tidak ditemukan sama sekali dalam larik "nim".

Berikut implementasi fungsi "binarysearchalgorithm" dan "interpolationsearchalgorithm".

```java
public class binarysearch {

    public int binarysearchalgorithm(int nim[], int low, int high, int wantednim) {
        // Pre-condition: Larik nim sudah terurut membesar

        if (high >= low) {

            // Set mid value
            int mid = low + (high - low) / 2;

            // wantednim ditemukan
            if (nim[mid] == wantednim) {
                // indeks elemen larik yang bernilai = wantednim
                return mid;
            }

            // cari di upalarik kiri, di dalam larik nim[low..mid]
            if (nim[mid] > wantednim) {
                return binarysearchalgorithm(nim, low, mid - 1, wantednim);
            }

            // cari di upalarik kanan, di dalam larik nim[mid+1..high]
            return binarysearchalgorithm(nim, mid + 1, high, wantednim);
        }

        // wantednim tidak ditemukan
        return -1;
    }
}
```

Gambar 8. Implementasi Fungsi "binarysearchalgorithm" (dokumentasi penulis)

```java
public class interpolationsearch {

    public int interpolationsearchalgorithm(int nim[], int low, int high, int wantednim) {
        // Pre-condition: Larik nim sudah terurut membesar

        if (high >= low && wantednim >= nim[low] && wantednim <= nim[high]) {

            // Set mid value
            int mid = low + (((high - low) / (nim[high] - nim[low]))
                            * (wantednim - nim[low]));

            // wantednim ditemukan
            if (nim[mid] == wantednim) {
                // indeks elemen larik yang bernilai = wantednim
                return mid;
            }

            // cari di upalarik kiri, di dalam larik nim[low..mid]
            if (nim[mid] > wantednim) {
                return interpolationsearchalgorithm(nim, low, mid - 1, wantednim);
            }

            // cari di upalarik kanan, di dalam larik nim[mid+1..high]
            return interpolationsearchalgorithm(nim, mid + 1, high, wantednim);
        }

        // wantednim tidak ditemukan
        return -1;
    }
}
```

Gambar 9. Implementasi Fungsi "interpolationsearchalgorithm" (dokumentasi penulis)

## *C. Percobaan*

Pengguna program dapat menginput NIM pilihannya dengan kelas Scanner. Dari input tersebut, setiap algoritma akan mengeksekusi tugasnya. Untuk membandingkan kedua algoritma, penulis mengukur waktu eksekusi masing-masing algoritma menggunakan metode nanoTime() dari pustaka System bahasa Java. Waktu eksekusi hanya diukur saat masing-masing *binary search* dan *interpolation search* dilakukan pada larik "nim", dan tidak termasuk pencarian larik-larik lainnya. Program akan dilakukan dengan *Command Line Interface* (CLI).

```
binarysearch binary = new binarysearch();
interpolationsearch interpolation = new interpolationsearch();

Scanner sc = new Scanner(System.in);
System.out.print(s: "Masukkan NIM (13521xxx): ");
int x = sc.nextInt();
int datalength = nim.length;

long startTimeBinary = System.nanoTime();
int binaryresult = binary.binarysearchalgorithm(nim, low: 0, datalength-1, x);
long endTimeBinary = System.nanoTime();

long startTimeInterpolation = System.nanoTime();
int interpolationresult = interpolation.interpolationsearchalgorithm(nim, low: 0, datalength-1, x);
long endTimeInterpolation = System.nanoTime();

long timeBinary = endTimeBinary - startTimeBinary;
long timeInterpolation = endTimeInterpolation - startTimeInterpolation;

System.out.println(x: "=========================Binary Search=========================");
System.out.println(String.format(format: "Waktu eksekusi binary search: %d nanosecond", timeBinary));
System.out.println(String.format(format: "Indeks NIM: %d", binaryresult));
if (binaryresult != -1) {
System.out.println(String.format(format: "Nama mahasiswa: %s", namamhs[binaryresult]));
System.out.println(String.format(format: "IPK mahasiswa: %.2f", ipkmhs[binaryresult]));
System.out.println(String.format(format: "Kota asal mahasiswa: %s", kotaasalmhs[binaryresult]));
}

System.out.println(x: "=======================Interpolation Search=======================");
System.out.println(String.format(format: "Waktu eksekusi interpolation search: %d nanosecond", timeInterpolation));
System.out.println(String.format(format: "Indeks NIM: %d", interpolationresult));
if (interpolationresult != -1) {
System.out.println(String.format(format: "Nama mahasiswa: %s", namamhs[interpolationresult]));
System.out.println(String.format(format: "IPK mahasiswa: %.2f", ipkmhs[interpolationresult]));
System.out.println(String.format(format: "Kota asal mahasiswa: %s", kotaasalmhs[interpolationresult]));
}
```

Gambar 10. Program Utama (dokumentasi penulis)

Berdasarkan gambar di atas, hasil pencarian dan waktu eksekusi untuk setiap algoritma akan dicetak secara terpisah. Untuk menguji efisiensi algoritma *binary search* dan *interpolation search*, akan digunakan beberapa kasus uji berdasarkan posisi NIM di dalam larik "nim".

Untuk kasus pertama, penulis akan mencari NIM dengan posisi acak dalam larik "nim", misalkan NIM 13521057. Berikut adalah *output* dari program yang sudah dijalankan:

```
Masukkan NIM (13521xxx): 13521057
=========================Binary Search=========================
Waktu eksekusi binary search: 4600 nanosecond
Indeks NIM: 11
Nama mahasiswa: Jessica Nadya
IPK mahasiswa: 3.55
Kota asal mahasiswa: Bandung
=======================Interpolation Search=======================
Waktu eksekusi interpolation search: 1000 nanosecond
Indeks NIM: 11
Nama mahasiswa: Jessica Nadya
IPK mahasiswa: 3.55
Kota asal mahasiswa: Bandung
```

Gambar 11. Hasil Pencarian NIM 13521057 (dokumentasi penulis)

Kedua algoritma memberikan hasil data mahasiswa yang sama, karena keduanya juga mengembalikan posisi indeks NIM 13521057 yang sama. Namun, *interpolation search* memiliki waktu eksekusi yang lebih sedikit dibandingkan *binary search*. Hal ini sesuai harapan karena kompleksitas waktu *interpolation search* secara umum lebih mangkus daripada *binary search*.

Untuk kasus kedua, penulis akan mencari NIM yang terletak pada indeks pertama larik "nim", yakni 13521046.

```
Masukkan NIM (13521xxx): 13521046
=========================Binary Search=========================
Waktu eksekusi binary search: 9800 nanosecond
Indeks NIM: 0
Nama mahasiswa: Muhammad Ali
IPK mahasiswa: 3.20
Kota asal mahasiswa: Jakarta Selatan
=======================Interpolation Search=======================
Waktu eksekusi interpolation search: 2100 nanosecond
Indeks NIM: 0
Nama mahasiswa: Muhammad Ali
IPK mahasiswa: 3.20
Kota asal mahasiswa: Jakarta Selatan
```

Gambar 12. Hasil Pencarian NIM 13521046 (dokumentasi penulis)

Lagi-lagi, kedua algoritma memberikan indeks NIM dan hasil data mahasiswa yang sama, serta *interpolation search* memiliki waktu eksekusi yang lebih sedikit dibandingkan *binary search*. Namun, waktu eksekusi kedua algoritma meningkat dibandingkan kasus pertama, kira-kira sebesar dua kali lipat. Hal ini karena kedua algoritma memulai iterasi dari bagian tengah larik, bukan dari ujung larik. Akibatnya, semakin jauh elemen larik yang ingin dicari dari bagian tengah, semakin banyak waktu yang diperlukan untuk mencari elemen tersebut.

Untuk kasus ketiga, penulis akan mencari NIM yang terletak pada indeks terakhir larik "nim", yakni 13521088. Berikut adalah *output* dari program yang sudah dijalankan:

```
Masukkan NIM (13521xxx): 13521088
=========================Binary Search=========================
Waktu eksekusi binary search: 9500 nanosecond
Indeks NIM: 42
Nama mahasiswa: Michael Angga
IPK mahasiswa: 3.36
Kota asal mahasiswa: Bandung
=======================Interpolation Search=======================
Waktu eksekusi interpolation search: 2000 nanosecond
Indeks NIM: 42
Nama mahasiswa: Michael Angga
IPK mahasiswa: 3.36
Kota asal mahasiswa: Bandung
```

Gambar 13. Hasil Pencarian NIM 13521088 (dokumentasi penulis)

Seperti kasus pertama, kedua algoritma memberikan indeks NIM dan hasil data mahasiswa yang sama, serta *interpolation search* memiliki waktu eksekusi yang lebih sedikit dibandingkan *binary search*. Dan seperti kasus kedua, waktu eksekusi kedua algoritma meningkat dibandingkan kasus pertama, karena NIM yang dicari juga terletak di ujung larik, kali ini di ujung kanan.

Untuk kasus terakhir, penulis akan memilih sebuah NIM yang tidak terletak pada larik "nim", misal 13521100. Berikut adalah *output* dari program yang sudah dijalankan:

```
Masukkan NIM (13521xxx): 13521100
=========================Binary Search=========================
Waktu eksekusi binary search: 4600 nanosecond
Indeks NIM: -1
=======================Interpolation Search=======================
Waktu eksekusi interpolation search: 800 nanosecond
Indeks NIM: -1
```

Gambar 14. Hasil Pencarian NIM 13521100 (dokumentasi penulis)

Seperti yang diharapkan, karena NIM 13521100 tidak terdapat pada larik "nim", kedua algoritma mengembalikan indeks -1. Data mahasiswa juga tidak diperlihatkan jika nilai indeks dari hasil kedua algoritma tersebut sebesar -1. *Interpolation search* dapat menyimpulkan bahwa NIM tidak ada di dalam larik "nim" lebih cepat dibandingkan *binary search*.

## IV. KESIMPULAN

Algoritma pencarian menggunakan konsep *decrease and conquer* yang terdiri atas *binary search* dan *interpolation search* dapat melakukan pencarian posisi elemen dalam sebuah larik, dengan syarat larik tersebut sudah dalam kondisi terurut, dengan cepat dan efisien karena kedua algoritma tersebut membagi pekerjaannya menjadi dua buah upalarik. Hal ini sangat penting, khususnya jika ukuran larik sangat besar,

seperti larik yang menyimpan data puluhan ribu mahasiswa ITB. Namun, di sisi lain, algoritma pencarian *decrease and conquer* tidak bisa diimplementasikan di larik dengan nilai yang tidak terurut. Agar algoritma dapat bekerja dengan baik, nilai larik harus diurutkan terlebih dahulu, baik secara membesar maupun menurun. Untuk kasus seperti kumpulan data mahasiswa, pada kenyataannya, hampir semua data selalu terurut berdasarkan NIM secara membesar, sehingga kekurangan ini dapat diabaikan untuk pada makalah ini.

Berdasarkan hasil percobaan, *interpolation search* memiliki waktu eksekusi sekitar 4.5 kali lebih cepat dibandingkan *binary search*. Hal ini karena *interpolation search* memiliki kompleksitas waktu yang lebih efisien ($O(^2\log(^2\log n))$) dibandingkan *binary search* ($O(^2\log n)$). Tentu saja, larik yang penulis gunakan dalam percobaan hanya mengandung jumlah sampel data mahasiswa yang sangat minim. Jika larik berukuran ribuan, rasio antara waktu eksekusi *interpolation search* dan *binary search* akan semakin meningkat secara drastis. Selain itu, posisi elemen di dalam larik juga akan mempengaruhi waktu eksekusi program. Semakin jauh letak elemen dari bagian tengah larik, semakin besar waktu eksekusi program.

## VIDEO YOUTUBE

Berikut adalah video penjelasan singkat terhadap makalah ini: https://youtu.be/YdYyU7YWSlk

## UCAPAN TERIMA KASIH

Puji syukur penulis panjatkan ke pada Allah SWT, karena berkat rahmat-Nya, penulis dapat menyelesaikan makalah berjudul "Analisis Perbandingan Efisiensi Pencarian NIM Mahasiswa Teknik Informatika ITB dengan Metode *Binary Search* dan *Interpolation Search*". Makalah ini dapat diwujudkan berkat para dosen pengampu mata kuliah IF2211 Strategi Algoritma. Oleh karena itu penulis menyampaikan terima kasih kepada Dr. Nur Ulfa Maulidevi, S.T., M.Sc. sebagai dosen K02 atas ilmu dan bimbingan yang diberikan, serta teman-teman dan orang tua yang telah memberikan saran dan dukungan dalam pembuatan makalah ini.

## DAFTAR PUSTAKA

[1] https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Decrease-and-Conquer-2021-Bagian1.pdf (diakses pada 20 Mei 2023, 18.15 WIB)

[2] https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Decrease-and-Conquer-2021-Bagian2.pdf (diakses pada 20 Mei 2023, 18.15 WIB)

[3] https://www.tutorialspoint.com/data_structures_algorithms/interpolation_search_algorithm.htm (diakses pada 20 Mei 2023, 19.05 WIB)

[4] https://www.geeksforgeeks.org/interpolation-search/ (diakses pada 20 Mei 2023, 20.49 WIB)

[5] https://www.geeksforgeeks.org/binary-search/ (diakses pada 21 Mei 2023, 11.11 WIB)

[6] https://www.programiz.com/dsa/binary-search (diakses pada 21 Mei 2023, 11.12 WIB)

## PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 22 Mei 2023

M Farrel Danendra Rachim - 13521048