RUNNING TIME ALGORITMA SEARCHING DATA PADA ARRAY SATU DIMENSI MENGGUNAKAN JAVA NETBEANS

JOHNSON SIHOMBING
POLITEKNIK PIKSI GANESHA
johnson.sihombing@piksi-ganesha-online.ac.id

ABSTRACT

Data search algorithms are often used in any programming consisting of: sequential searching, binary searching and interpolation searching.
Search data can be done in an array, either one-dimensional array, two dimensional and even n-dimensional. For the research, the author only do the implementation of search algorithm on a one dimensional array that each data element is numeric type.
The search process is performed on consecutive and non-sequential data in the array. Unassisted data can be found in sequential searches. The opposite is true for binary search and interpolation search.
The purpose of implementing the algorithm is to show the difference of the result of the running time of the three search algorithms above, which can be visualized in graphical form.
The results are obtained after the authors perform the implementation of search algorithm by using a desktop-based Java software with editor or compiler Netbeans8.1.

Keywords: *searching, one dimensional array, running time, Netbeans*

A. PENDAHULUAN

Dalam kehidupan sehari-hari sebenarnya kita sering melakukan pencarian data. Sebagai contoh, jika kita menggunakan Kamus untuk mencari kata-kata dalam Bahasa Inggris yang belum diketahui terjemahannya dalam Bahasa Indonesia. Contoh lain saat kita menggunakan buku telepon untuk mencari nomor telepon teman atau kenalan dan masih banyak contoh yang lain.

Pencarian data sering juga disebut *table look-up* atau *storage and retrieval information* adalah suatu proses untuk mengumpulkan sejumlah informasi di dalam pengingat komputer dan kemudian mencari kembali informasi yang diperlukan secepat mungkin.

Algoritma pencarian (*searching algorithm*) adalah algoritma yang menerima sebuah argumen kunci dan dengan langkah-langkah tertentu akan mencari elemen data dengan kunci tersebut. Setelah proses pencarian dilaksanakan, akan diperoleh salah satu dari dua kemungkinan, yaitu data yang dicari ditemukan (*successful*) atau tidak ditemukan (*unsuccessful*).

B. KAJIAN PUSTAKA

1. Array

a. Pengertian *Array*

Array merupakan suatu kumpulan data terstruktur yang berupa sejumlah data sejenis (memiliki jenis data yang sama) yang jumlahnya tetap dan diberi suatu nama tertentu.

Sebuah *array* dapat dibayangkan sebagai sekumpulan kotak yang menyimpan sekumpulan elemen bertipe sama secara berurutan (*sequential*) di dalam memori komputer. *Array* juga dapat digambarkan sebagai elemen-elemen yang disusun secara vertikal sehingga dinamai tabel. Setiap elemen *array* data diacu melalui indeksnya.
Karena elemen disimpan secara berturutan, indeks *array* haruslah suatu tipe yang juga mempunyai keterurutan (memiliki suksesor dan ada predesesor), misalnya tipe *integer* atau karakter. Jika indeksnya adalah *integer* maka keterurutan indeks sesuai dengan urutan *integer*. Jika indeks array adalah karakter maka keterurutan indeks sesuai dengan urutan karakter. Tiap elemen *array* langsung diakses dengan menspesifikasikan nama *array*.
Array dapat berupa array 1 dimensi, 2 dimensi, dan bahkan n-dimensi.

2. Deklarasi Array

Untuk dapat mengakses array, maka terlebih dahulu ditentukan deklarasi terhadap suatu array yang berbentuk seperti berikut ini :

tipe_data nama_var_array[ukuran];

Keterangan :
tipe_data : menyatakan jenis tipe data dari elemen yang tersimpan di *array* (int, char, float, dan lain-lain).
nama_var_array : menyatakan nama variabel yang dipakai.
ukuran : menunjukkan jumlah maksimal elemen *array*.
Contoh : Int nilai[6].

3. *Array* Berdimensi Satu

a. Pengertian *Array* Berdimensi Satu

Merupakan sekumpulan item data yang disusun secara baik menjadi suatu rangkaian dan diacu atau ditunjuk oleh satu *identifier*. Contoh : Nilai = (56 42 89 65 48). . Item data individual dalam *array* bisa ditunjuk secara terpisah dengan menyatakan posisinya dalam array itu. Misalnya Nilai(1) menunjuk ke 56, Nilai(2) menunjuk ke 42, dan seterusnya.

Bilangan yang ditulis dalam tanda kurung menandakan posisi item individual dalam *array* (disebut juga *subscript* / indeks).
Variabel bisa digunakan sebagai *subscript*, misalnya Nilai(i). Jika i = 2 maka menunjuk ke Nilai(2) yaitu 42. Jika i = 4 maka menunjuk ke Nilai(4) yaitu 65. Sedangkan item data individual dalam suatu *array* sering disebut elemen**.**

b. Pemetaan di Memori

Memori komputer untuk pemetaan *array* dibentuk secara linier (*contunue*).
Memori memiliki alamat (*address*) untuk tempat menyimpan array.
Sebagai contoh dapat dilihat pada suatu komplek perumahan. Untuk identifikasi, maka rumah-rumah tersebut diberi nomor urut dalam pola yang tertentu (misalnya dari kecil ke besar).

Pemetaan di memori juga berfungsi sebagai identifikasi letak data, agar kelak data tersebut dapat diambil kembali, maka akan dengan tepat ditemui data tersebut.
Banyaknya alamat di memori tergantung dari jenis komputer yang digunakan.,Misalnya dari alamat 000000 hingga FFFFFF (dalam sistem bilangan Hexadesimal). Setiap alamat dapat ditempati oleh data sebesar satu *byte*.
Jika ingin memasukkan data baru, maka komputer akan mencatat dimana data itu disimpan.

Adapun formula untuk melakukan penghitungan alokasi memori array dapat dilihat di bawah ini :

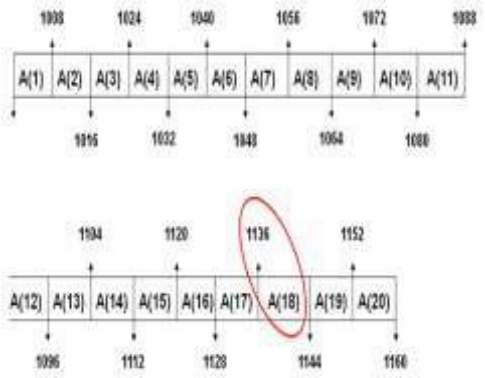
AD = B + (SK-1) * LD

Gambar 1. Penghiungan Alokasi Memori

Keterangan :
AD = Posisi alamat awal dari nilai data yang akan dicari
B = *Base Address*
SK = *Subscript* ke berapa yang akan dicari
LD = Lebarnya data yang dapat disimpan di setiap alamat memori
Contoh :
Jika pada suatu *array* memiliki 20 nilai data dalam variable yang ber-*subscript* A yang masing-masing memerlukan 8 *byte* data dan *base address*nya di 1000. Maka penentuan alamat memori untuk data A(18) adalah :

AD = B + (SK-1) * LD
AD = 1000 + (18-1) * 8
AD = 1000 + 17 * 8
AD = 1000 + 136
AD = 1136

Pemetaan alamat memori dari A(18) dapat direpresentasikan sebagai berikut :



Gambar 2. Pemetaan Alamat Array

Pada gambar di atas, nilai data A (18) dismpan pada alamat 1136 di memori komputer.

4. Metode *Searching* (Pencarian)

Metode pencarian data dapat dilakukan dengan dua cara yaitu pencarian internal (*internal searching*) dan pencarian eksternal (*external searching*). Pada pencarian internal, semua elemen yang diketahui berada dalam memori komputer. Pada pencarian eksternal, tidak semua elemen yang diketahui berada dalam memori komputer, tetapi ada sejumlah elemen yang tersimpan dalam penyimpan luar misalnya pita atau cakram magnetis.

Selain itu metode pencarian data juga dapat dikelompokkan menjadi pencarian statis (*static searching*) dan pencarian dinamis (*dynamic searching*). Pada pencarian statis, banyaknya elemen yang diketahui dianggap tetap. Sedangkan pada pencarian dinamis, banyaknya elemen yang diketahui bisa berubah-ubah yang disebabkan oleh penambahan atau penghapusan suatu elemen.
Ada tiga macam teknik pencarian yaitu pencarian sekuensial, pencarian biner.dan pencarian interpolasi. Perbedaan dari ketiga teknik ini terletak pada keadaan data. Pencarian sekuensial digunakan apabila data dalam keadaan acak atau tidak terurut. Sebaliknya, pencarian biner dan pencarian interpolasi digunakan pada data yang sudah dalam keadaan urut.

a. Pencarian Berurutan (*Sequential Searching*)

Pencarian berurutan sering disebut pencarian linear merupakan metode pencarian yang paling sederhana. Pencarian berurutan menggunakan prinsip sebagai berikut : data yang ada dibandingkan satu per satu secara berurutan dengan yang dicari sampai data tersebut ditemukan atau tidak ditemukan.

Pencarian terhadap elemen data *array* dapat dalam kondisi belum terurut dan sudah terurut.
Perbedaanya dapat dirincikan seperti berikut :

1) Data yang belum terurut :
   (a) Secara umum pencarian berjalan dengan relatif lambat.
   (b) Waktu pencarian sebanding dengan jumlah elemen *array*.
2) Data yang sudah terurut :
   (a) Dapat meningkatkan kinerja pencarian.
   (b) Karena dapat segera menyimpulkan bahwa data yang dicari tidak terdapat di dalam *array* bila ditemukan terhadap elemen data *array* yang dicari.

Contoh data yang tidak berurutan dan sudah berurutan :
Data tidak berurutan :

Tabel 1. Data Array tidak berurutan

13	16	14	21	76	15	43	37

Tabel 2. Data Array berurutan

13	14	15	16	21	37	43	76

Adapun struktur dari prosedur dapat ditulis sebagai berikut :

```
return jenis_data nama_prosedur(daftar_parameter) {
.....
nama_prosedur( ..... );
.....
}
```

Pada dasarnya, teknik ini hanya melakukan pengulangan dari 1 sampai dengan jumlah data. Pada setiap pengulangan, dibandingkan data ke-i dengan yang dicari. Apabila sama, berarti data telah ditemukan. Sebaliknya apabila sampai akhir pengulangan tidak ada data yang sama, berarti data tidak ada. Pada kasus yang paling buruk, untuk N elemen data harus dilakukan pencarian sebanyak N kali pula.
Algoritma pencarian berurutan dapat dituliskan sebagai berikut :

1. i := 0
2. ketemu := false

3. Selama (tidak ketemu) dan (i <= N) kerjakan baris 4
4. Jika (Data[i] = x) maka ketemu := true, jika tidak i := i + 1
5. Jika (ketemu) maka i adalah indeks dari elemen data yang dicari
6. Jika tidak, maka data tidak ditemukan.

Di bawah ini merupakan fungsi untuk mencari data menggunakan pencarian sekuensial :

```
int Cari_Sequential(int x)  {
int i := 0;
boolean ketemu := false;
while ((!ketemu) && (i < Max))  {
if (Data[i] == x)
ketemu := true;
else i++;
}
if (ketemu)
return i;
else
return -1;
}
```

Fungsi diatas akan mengembalikan indeks dari data yang dicari. Apabila data tidak ditemukan maka fungsi diatas akan mengembalikan nilai –1.

b. Pencarian Biner (*Binary Searching*)

Salah satu syarat agar pencarian biner dapat dilakukan adalah data sudah dalam keadaan urut. Dengan kata lain, apabila data belum dalam keadaan urut, maka pencarian biner tidak dapat dilakukan.

Dalam kehidupan sehari-hari, sebenarnya kita juga sering menggunakan pencarian biner. Misalnya saat ingin mencari suatu kata dalam kamus.

Langkah-langkah pengerjaan dari teknik pencarian biner dapat dijelaskan sebagai berikut :

1) Mula-mula diambil posisi awal 0 dan posisi akhir = N – 1
2) Menentukan posisi data tengah dengan rumus (posisi awal + posisi akhir) / 2.
3) Elemen Data yang dicari dibandingkan dengan elemen data tengah.
4) Jika lebih kecil, proses dilakukan kembali tetapi posisi akhir dianggap sama dengan posisi tengah –1.
5) Jika lebih besar, porses dilakukan kembali tetapi posisi awal dianggap sama dengan posisi tengah + 1. Demikian seterusnya sampai data tengah sama dengan yang dicari.

Contoh :

Misalnya ingin mencari data 17 pada sekumpulan elemen data berikut :

Tabel 3. Posisi Awal Pencarian Biner

3	9	11	12	15	17	20	23	31	35

awal tengah akhir

Mula-mula dicari data tengah, dengan rumus (0 + 9) / 2 = 4. Berarti data tengah adalah data ke-4, yaitu 15. Data yang dicari, yaitu 17, dibandingkan dengan data tengah ini. Karena 17 > 15 berarti proses dilanjutkan, tetapi kali ini posisi awal dianggap sama dengan posisi tengah + 1 atau 15.

Tabel 4. Posisi Tengah Pencarian Biner

3	9	11	12	15	17	20	23	31	35

awal tengah akhir

Data tengah yang baru didapat dengan rumus (5 + 9) / 2 = 7. Berarti data tengah yang baru adalah data ke-7, yaitu 23. Data yang dicari yaitu 17 dibandingkan dengan data tengah ini. Karena 17 < 23 berarti proses dilanjukkan, tetapi untuk posisi akhir dianggap sama dengan posisi tengah – 1 atau 20.

Tabel 5. Posisi Akhir Pencarian Biner

3	9	11	12	15	17	20	23	31	35

awal=tengah akhir

Data tengah yang baru didapat dengan rumus (5 + 6) / 2 = 5. Berarti data tengah yangbaru adalah data ke-5, yaitu 17. Data yang dicari dibandingkan dengan data tengah ini dan ternyata sama. Jadi data ditemukan pada indeks ke-5.

Pencarian biner ini akan berakhir jika data ditemukan atau posisi awal lebih besar daripada posisi akhir. Jika posisi sudah lebih besar daripada posisi akhir berarti data tidak ditemukan.

Untuk lebih jelasnya, perhatikan contoh pencarian data 16 pada data *array* diatas. Prosesnya hampir sama dengan pencarian data 17. Tetapi setelah posisi awal 5 dan posisi akhir 6, data tidak ditemukan dan 15 < 17, maka posisi akhir menjadi posisi tengah – 1, yaitu di posisi 4, sedangkan posisi awal = 5.

Tabel 6. Posisi Awal = Akhir

3	9	11	12	15	17	20	23	31	35

awal=akhir

Disini dapat dilihat bahwa posisi awal lebih besar daripada posisi akhir, yang artinya data tidak ditemukan.

Algoritma pencarian biner dapat dituliskan sebagai berikut :

1) L := 0
2) R := N – 1
3) ketemu := false
4) Selama (L <= R) dan (tidak ketemu) kerjakan baris 5 sampai dengan 8
5) m := (L + R) / 2
6) Jika (Data[m] = x) maka ketemu := true
7) Jika (x < Data[m]) maka R := m – 1
8) Jika (x > Data[m]) maka L := m + 1
9) Jika (ketemu) maka m adalah indeks dari data yang dicari. Jika tidak, maka data tidak ditemukan

Di bawah ini merupakan fungsi untuk mencari elemen data menggunakan pencarian biner.

```
int Cari_Biner(int x) {
int L = 0, R = Max-1, m;
boolean ketemu = false;
while ((L <= R) && (!ketemu)) {
m = (L + R) / 2;
if (Data[m] == x)
```

```
ketemu = true;
else if (x < data[m])
R = m - 1;
Else L = m + 1;
}
if (ketemu)
return m;
else
return -1;
}
```

Fungsi diatas akan mengembalikan indeks dari elemen data yang dicari. Apabila data tidak ditemukan, maka fungsi diatas akan mengembalikan nilai –1.

Jumlah pembandingan minimum pada pencarian biner adalah 1 kali, yaitu apabila data yang dicari tepat berada di tengah-tengah. Jumlah pembandingan maksimum yang dilakukan dengan pencarian biner dapat dicari menggunakan rumus logaritma, yaitu : $C = {}^{2}\log(N)$

c. Pencarian Interpolasi (*Interpolation Searching*)

Metode ini pada dasarnya sama dengan (varian) dari metode pencarian biner, kecuali dalam penentuan elemen data yang diperiksa berikutnya .

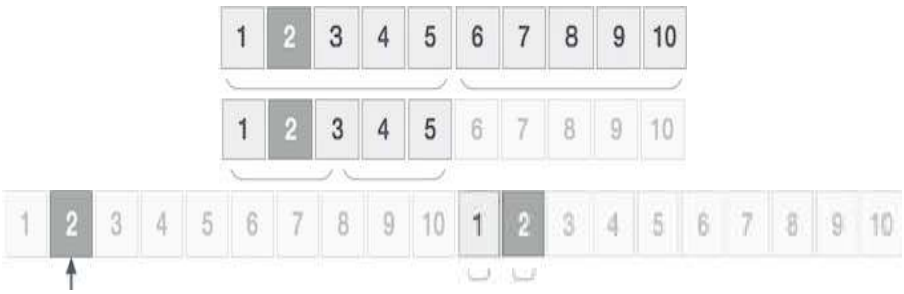
Sama seperti pencarian biner, metode pencarian interpolasi dilakukan terhadap elemen data *array* yang sudah terurut secara *ascending*.

Perbedaannya terleteak pada proses pencarian. Pada metode pencarian biner, pencarian selalu dilakukan bagi dua pada tiap prosesnya. Sedangkan pada metode interpolasi, proses pencarian terhadap elemen data bisa langusng dilakukan

Contoh : pencarian pada nomer telepon pada daftar buke telepon (*phone book)*. Misalnya nama data yang dicari berawalan huruf R, maka proses pencarian tidak akan dilakukan dari awal, namun langsung membuka 2/3 atau 3/4 dari tebal buku. Jadi, data yang dicari relatif terhadap jumlah data.

Dibawah ini adalah ilustrasi perbedaan proses pencarian elemen data dengan menggunakan metode pencarian biner dan metode pencarian interpolasi.

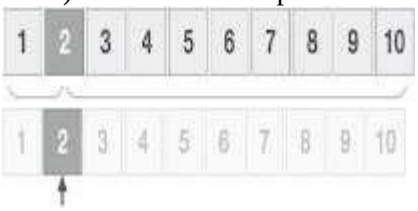
1) Pencarian Biner



Gambar 3. Posisi Awal Pencarian Biner

Jika pencarian elemen data yang diinginkan tidak ditemukan, maka sisa *array* dibagi dalam dua bagian (dilakukan perbandingan terhadap nilai elemen yang lebih rendah atau lebih tingi). Kemudian pencarian dilakukan di salah satu dari elemen *array* tersebut.

2) Pencarian Interpolasi



Gambar 4. Posisi Awal Pencarian Interpolasi

Pada pencarian interpolasi, pencarian elemen data tertentu dilakukan dengan menghitung posisi elemen secara probabilitas, yaitu menghitung posisi elemen data yang dicari dengan cara memasukkan ke rumus pencarian interpolasi yang dapat di tulis sebagai berikut :

$$Posisi = \frac{kunci - data[low]}{Data[high] - data[low]} x(high - low) + low$$

Keterangan :
Jika data[posisi] > data yang dicari, maka high = pos – 1. Jika data[posisi] < data yang dicari, maka high = pos + 1.
Teknik ini dilakukan terhadap data yang sudah terurut berdasarkan pencarian elemen data tertentu dengan perkiraan posisi data.
Adapun algoritma pencarian interpolasi dapat di lihat dibawah ini :

a. awal = 0
b. akhir = banyak Data Array – 1.
c. Masukkan data sesuai yang di inginkan (*input* bilangan).
d. Lakukan pencarian data dengan menggunakan rumus interpoasi :

posisi = ((cari_data – data[awal]) * (akhir - awal) + awal) / (data[akhir]-data[awal]).

e. Menenetukan posisi cari_data == data[posisi]
f. Jika sesuai dengan yang dicari / sama , maka data yang dicari telah ditemukan dan proses pencarian berhenti / selesai.
g. Jika tidak sesuaidengan yang dicari, maka akan dilakukanb perbandingan :

Jika (data[posisi] < cari_data, maka awal = posisi + 1.

h. Ulangi langkah 4 dan 5.
i. Jika data[posisi] > cari_data, maka tampilkan data tidak ditemukan, dan proses pencarian selesai.

Contoh :
Proses pencarian nilai 27 dan 49 pada sebuah *array* dengan bentuk seperti berikut :

Tabel 7. Posisi Dan Data Array
Pencarian Interpolasi

21	25	28	33	38	39	48	49	69

Cari data 27 :
Awal = 1, Akhir = 9
Cari data selama awal < Akhir

$$\text{Posisi} : [1+\frac{(27-21)}{(69-21)} x(9-1)] = 2$$

Data[2] =27 ? Tidak
Data[2] <27 ❼ posisi 3 =
"
Akhir = "9 <br" awal = "Posisi" ya = "">

$$\text{Posisi} : [3+\frac{(27-28)}{(69-28)}(9-3)] = 3$$

Data[3] = 27 ? Tidak
Data[3] <27 -1= "2," akhir ="posisi", awal ="3 <br" tidak.="">.
Hasil : Data tidak ditemukan karena awal > akhir

Cari data 49 :
Awal = 1, Akhir = 9
Cari data selama awal < Akhir

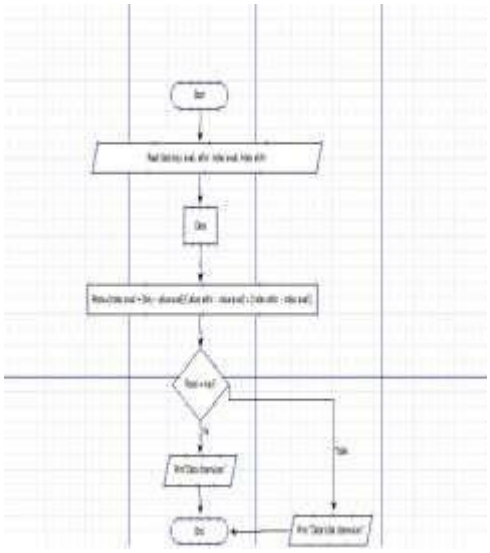
$$\text{Posisi} := [\,1 + \frac{(49\text{-}21)}{(69\text{-}21)}\ (9\text{-}1)\,] = 5{,}666 \cong 6$$

Data[6] = 49 ? Tidak
Data[6] < 49 , akhir = "9 <br" awal = "posisi" ya.= "">

$$\text{Posisi} := [\,7 + \frac{(49\text{-}48)}{(69\text{-}48)}\ (9\text{-}7)\,] = 7{,}0952 \cong 8$$

Data[8] = 49 ? Ya. Data ditemukan.

Adapun bentuk diagram alir (*flowchart)* untuk proses pencarian interpolasi dapat dilihat berikut ini :



Gambar 5. *Flowchart* Pencarian Interpolasi

C . HASIL DAN PEMBAHASAN

1. Pengujian Teknik Searching

Sesuai dengan judul *paper*/jurnal yang penulis teliti, maka proses pengujian terhadap ketiga teknik *searching* diatas dilakukan dengan membuat program aplikasi yang *output* nya adalah mencari perbedaan *running time* (waktu proses) untuk tiap proses searching tersebut. Untuk mengimplementasikan algoritma *searching*, penulis menggunakan *software* aplikasi berbasis Java dengan *editor* dan *compiler* adalah Nebeans IDE 8.1 sebagai dasar untuk menentukan proses pencarian.

Adapun hasil *running time* dari proses *searching* yang penulis lakukan, bukanlah suatu hasil yang mutlak. Hal tesebut dikarenakan hasil dari *running time* akan berbeda pada tiap jenis komputer. Dengan kata lain, *running time* akan bergantung pada kecepatan dan kecanggihan dari spesifikasi komputer yang digunakan. Semakin mutakhir spesifikasi dari komputer, maka semakin cepat pula *running time* nya.

2. Sekilas tentang Netbeans IDE 8.1

NetBeans adalah suatu *Integrated Development Environment* (IDE) berbasiskan Java dari Sun Microsystems yang berjalan di atas *Swing*. *Swing* merupakan sebuah teknologi Java untuk pengembangan aplikasi *desktop* yang dapat bejalan di berbagai macam *platforms* seperti Windows, Linux, Mac OS X and Solaris.

Netbeans merupakan *software development* yang *Open Source*, dengan kata lain software ini di bawah pengembangan bersama, bebas biaya. NetBeans merupakan sebuah proyek kode terbuka yang sukses dengan pengguna yang sangat luas, komunitas yang terus tumbuh, dan memiliki hampir 100 mitra. Sun Microsystems mendirikan proyek kode terbuka NetBeans pada bulan Juni 2000 dan terus menjadi sponsor utama.

Suatu IDE adalah lingkup pemrograman yang diintegrasikan ke dalam suatu aplikasi perangkat lunak yang menyediakan pembangun *Graphic User Interface* (GUI), suatu *text* atau kode *editor*, suatu *compiler* atau *interpreter* dan suatu *debugger*.

The NetBeans IDE adalah sebuah lingkungan pengembangan, yang merupakan sebuah *tools* untuk pemrogram menulis, mengkompilasi, mencari kesalahan dan menyebarkan program. Netbeans IDE ditulis dalam Java namun dapat mendukung bahasa pemrograman lain. Terdapat banyak modul untuk memperluas *Netbeans IDE*. *Netbeans IDE* adalah sebuah produk bebas dengan tanpa batasan bagaimana digunakan. *NetBeans IDE* mendukung pengembangan semua tipe aplikasi Java (*J2SE, web, EJB,* dan aplikasi *mobile*). Fitur lainnya adalah sistem proyek berbasis *Ant, kontrol versi*, dan *refactoring*.

3. Perangkat Keras Yang Digunakan

Adapun spesifikasi perangkat keras (*hardware*) yang digunakan penulis untuk membuat program aplikasi adalah sebagai berikut :

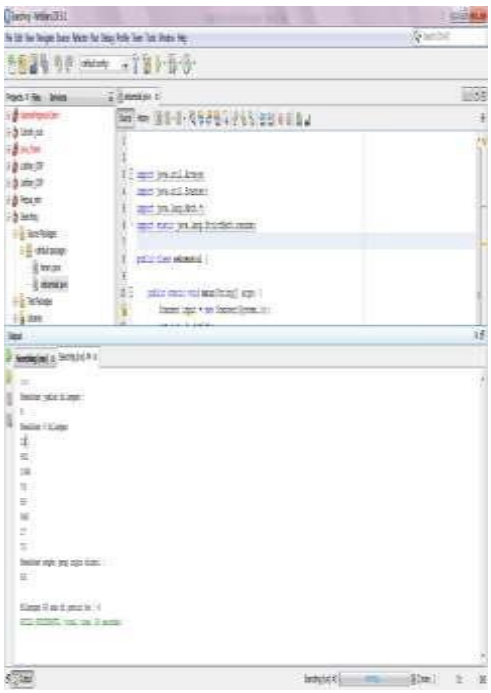
OS : Windows 10
Processor : AMD A6-5400K APU with Radeon™ HD Graphics 2.60 GHz
RAM : 4.00 GB
System type : 32-bit Operating System

Jumlah elemen data pada *array* yang di *input* adalah antara tiga hingga sepuluh data. Hal tersebut dibuat agar dapat dilihat perbedaan *running time* pada tiap teknik *searching* dengan jumlah data yang berbeda. Sehingga nantinya dapat dihasilkan sebuah grafik dari perbedaan *running time* pada tiap proses *searching*.

Berikut adalah tampilan dari beberapa ekesekusi program yang sudah di implementasikan ke dalam Netbeans 8.1 untuk tiap *searching* :

a. Running Time Sequential Search

Gambar 6. ***Running Time Sequential Searching*** **Dengan 5 Elemen Data**



Gambar 7. ***Running Time Sequential Searching*** **Dengan 8 Elemen Data**

Gambar 8. ***Running Time Binary Searching*** **Dengan 5 Elemen Data**

Gambar 9. ***Running Time Binary Searching*** **Dengan 8 Elemen Data**

Gambar 10. ***Running Time Interpolation Searching*** **Dengan 5 Elemen Data**

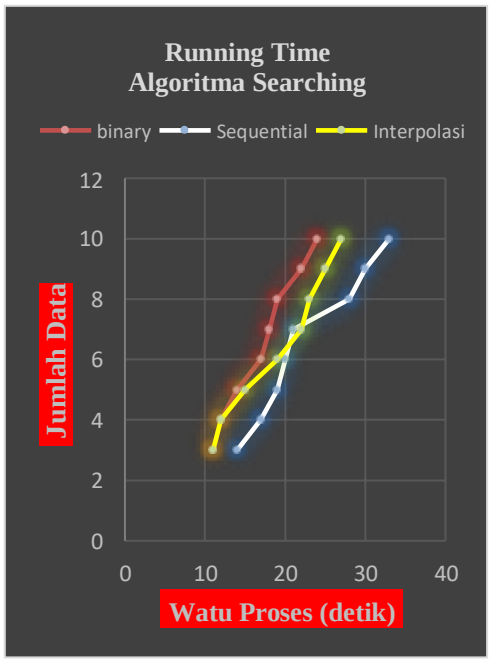
Gambar 11. ***Running Time Interpolation Searching*** **Dengan 8 Elemen Data**

Pada gambar-gambar di atas, terlihat bahwa ada sedikit perbedaan tiap teknik *searching* terhadap waktu proses pencarian data yang sudah di *input* ke dalam *array*. Untuk lebih jelasnya, bisa dilihat pada tabel dan grafik yang ada di bawah ini.

Tabel 7. *Running Time* Implementasi Algoritma *Searching*

No.	Jumlah Bilangan	Running Time (detik)		
		Sequential Search	Binary Search	Interpolaton Search
1	3	14	11	11
2	4	17	12	12
3	5	19	14	15
4	6	20	17	19
5	7	21	18	22
6	8	28	19	23
7	9	30	22	25
8	10	33	24	27

Dari tabel di atas, penulis membuat visualisasi dalam bentuk grafik di bawah ini.



Gambar 12. Grafik *Running Time* Impelementasi Algoritma *Searching*

D. KESIMPULAN

1. Kesimpulan

Setelah melakukan implementasi terhadap algoritma searching, penulis mendapat beberapa kesimpulan, antara lain :

a. Algoritma *Sequential Searching* merupakan metode pencarian data yang paling sederhana. Metode pencarian ini lebih mudah dan sederhana dibandingkan dengan *Binary Searching* dan *Intterpolation Searching*. Sehingga akan memudahkan pemrogram dalam menyelesaikan masalah
b. Algoritma *Sequential Searching* digunakan untuk mencari data pada *array* dengan elemen data yang tidak berurutan
c. Algoritma *Binary Searching* dan *Intterpolation Searching* digunakan untuk mencari data pada *array* dengan data yang sudah terurut.
d. *Running time* algoritma *Sequential Search* lebih efektif jika digunakan pada *array* dengan elemen data yang sedikit. Sedangkan hal sebaliknya terjadi pada algoritm *Binary Searching* dan *interpolation Searching.*

2. Saran

Seperti yang telah dibahas di sub bab Hasil dan Pembahasan, hasil *running time* yang didapatkan penulis bukanlah suatu nilai mutlak.
Running time akan lebih cepat didaptkan jika dilakukan pada spesifikasi computer yang lebih mutakhir.
Adapun saran-saran yang dapat ditulis adalah :

a. Pada tiap algoritma *searching*, buatlah algoritma sedmekian rupa sehingga lebih efektif, lebih cepat waktu proses dan tidak membutuhkan memori banyak.
b. Teliti dalam hal menghitung nilai tengah dari algoritma *Binary Searching*..
c. Menggunakan *software* yang lebih canggih dalam pengimplementasian di masa mendatang.

E. DAFTAR PUSTAKA

Utami, Ema, danSukrisno. 2005. *10 Langkah Belajar Logika danAlgoritma Menggunakan Bahasa C dan C++ di GNU/Linux.* Yogyakarta : Andi.

Hariyanto, Bambang. 2008. *Struktur Data*. Bandung : Penerbit Informatika.

Yatini Indra B, Nasution Erliansyah. 2006. *Algoritma Dan Struktur Data*. Yogyakarta : Penerbit Graha Ilmu.

Liem, Inggriani. (2003). *Catatan Kuliah Algoritma Dan Pemrograman*. Departemen Teknik Informatika, Institut Teknologi Bandung.

Munir, Rinaldi. (2003). *Matematika Diskrit*. Departemen Teknik Informatika, Institut Teknologi Bandung.

Inggriani Liem,2007. Diktat *Kuliah Dasar Pemrograman*. Departemen Teknik Informatika, Institut Teknologi BandungAhmad Haboush, Sami Qawasmeh. 2011 *Paralle sequential searching algorithm for unsorted array*. Jordan : Research Journal of Applied Sciences..

Aslam & Fell. 2005. *Analysis of Algorithms: Running Time.. CSU 2000 discrete Structures Fall*

Ficher, M.M., Getis,A. 2010. *Handbook of Applied Spatial Analysis Software Tools, Methods and Applications*. Berlin Heidelberg : Springer-Verlag

Sitorus, Lamhot , Sembiring David J.M. 2015. *Konsep dan implementasi struktur data dengan C++*. Yogyakarta: Andi