Penggunaan Metode Binary Search Pada Translator Bahasa Indonesia – Bahasa Jawa

Dewi Martina Andayani, Mike Yuliana, Tri Budi Santoso
Laboratorium Sinyal, Politeknik Elektronika Negeri Surabaya
Kampus PENS-ITS, Keputih, Sukolilo, Surabaya.
Email: tribudi@eepis-its.edu

Abstrak

Di propinsi Jawa Tengah dan Jawa Timur, pelajaran Bahasa Daerah, dalam hal ini Bahasa Jawa merupakan pelajaran yang tidak menarik dan cenderung sulit dipahami bagi para pelajar. Sementara itu disisi lain seringkali dibicarakan masalah pelestarian tradisi dan budaya, bahkan memaksa kita berselisih paham dengan negara tetangga. Paper ini merupakan hasil penelitian dari kami berupa pembuatan sebuah software Aplikasi Kamus Elektronik Bilingual Indonesia-Jawa dengan maksud untuk mengenalkan perbendaharaan kata dalam bahasa Jawa sehari-hari yang telah banyak digunakan. Perbendaharaan tersebut dapat diperoleh dari buku Pepak Bahasa Jawa, selain itu tentu saja berdasarkan survey yang akan dilakukan untuk mencari kata-kata apa saja yang banyak digunakan oleh masyarakat Jawa untuk berkomunikasi.

Dalam paper ini dibahas sebuah hasil penelitian tentang software penterjemah dari bahasa Indonesia ke Bahas Jawa. Pembuatan tampilan menggunakan pemrograman Java dan dalam sistem navigasinya dibuat sederhana mungkin agar mudah dalam pengaksesannya. Pada paper ini, diberikan bambaran bahwa pengguna hanya perlu memasukkan input berupa text dengan mengetik melalui keyboard komputer dan software akan menerjemahkannya ke dalam bahasa Jawa.

Diharapkan software ini dapat menarik minat pembaca, khususnya para pelajar SD dan SMP, dan pada akhirnya mampu memberikan kontribusi dalam upaya pelestarian budaya asli Indonesia.

Kata Kunci: *kamus elektronik, binary search, bubble sort, insertion sort*

1. PENDAHULUAN

Kamus berperan penting dalam pembelajaran bahasa karena dapat meningkatkan pengetahuan akan kosa kata. Penggunaan kamus mungkin hanya minimal pada saat berbicara, namun penting pada saat membaca dan menulis[1]. Oleh karena itu, pada Proyek Akhir ini dibuat sebuah *software* kamus sehingga memudahkan pengguna dalam mengaksesnya.

Saat ini telah banyak diperoleh berbagai macam bentuk kamus, namun kebanyakan kamus tersebut menerjemahkan bahasa Indonesia ke dalam bahasa asing, misalnya Inggris, Jepang, Prancis dan lain-lain. Sangat jarang ditemui Kamus untuk menerjemahkan bahasa Indonesia ke dalam bahasa Daerah, dalam hal ini bahasa Jawa karena mungkin dianggap kurang penting oleh sebagian masyarakat. Salah satu bentuk buku pembelajaran Bahasa Jawa yang terkenal adalah Pepak yang juga berisi terjemahan kata ke dalam bahasa Jawa, namun tidak dapat dipungkiri bahwa minat pembaca sangat rendah dikarenakan membaca adalah kegiatan yang kurang praktis untuk dilakukan. Sehingga dengan kamus elektronik ini diharapkan pengguna dapat dengan praktis memahami terjemahan kata dengan cara yang mudah tanpa harus membolak-balik lembaran buku.

2. TEORI PENUNJANG

2.1 Kamus Elektronik

Dalam penelitian ini, yang dimaksudkan sebagai kamus elektronik adalah kamus yang berupa piranti lunak dan bisa diinstal ke komputer[1]. Dalam hal ini, kamus elektronik yang tidak bisa diinstal ke komputer, misalnya kamus yang dipasarkan oleh *AlfaLink* tidak termasuk dalam pembahasan. Alasan utamanya adalah karena kamus tersebut menggunakan alat tertentu yang harganya cukup mahal dan tidak bisa diinstal ke komputer, sehingga penyebarannya sangat terbatas. Terdapat dua versi kamus elektronik *bilingual* Inggris-Indonesia yang cukup luas tersedia, yaitu *Linguist Version 1.0* dan *Indict Version 2.0*.

2.2 Metode Bubble Sort

Bubble Sort merupakan cara pengurutan yang sederhana. Konsep dari ide dasarnya adalah seperti "gelembung air" untuk elemen struktur data yang semestinya berada pada posisi awal. Cara kerjanya adalah dengan berulang-ulang melakukan traversal (proses *looping*) terhadap elemen-elemen struktur data yang belum diurutkan. Di dalam traversal tersebut, nilai dari dua elemen struktur data dibandingkan. Jika ternyata urutannya tidak sesuai dengan "pesanan", maka dilakukan pertukaran (*swap*).

Misalnya untuk larik dengan elemen:

7	3	4	1	6

Gambar 1. Larik Array 5 elemen

Akan diurutkan dengan algoritma *bubble sort* secara terurut menaik (*ascending*).

Proses pembandingan di dalam setiap *looping* digambarkan dengan warna kuning. Jika ternyata diperlukan pertukaran nilai, maka akan ditulis "tukar" di bawah elemen larik yang dibandingkan tersebut. Hasil penukaran digambarkan di tabel simulasi berikutnya. Untuk membedakan dengan elemen tabel yang lain, bagian tabel yang telah diurutkan digambarkan dengan warna biru muda.

Perhatikan bahwa nilai '1' yang merupakan nilai terkecil di dalam larik seolah-olah mengapung mengikuti proses *looping*. Pengapungan ini terus berlangsung hingga menemukan elemen tabel yang nilainya lebih kecil lagi dari elemen tabel tersebut. Inilah yang dimaksud dengan "efek gelembung" di dalam *bubble sort.*

Looping pertama:

7	3	4	1	6

7	3	4	1	6

tukar

7	3	1	4	6

tukar

7	1	3	4	6

tukar

1	7	3	4	6

Looping kedua:

1	7	3	4	6

1	7	3	4	6

1	7	3	4	6

tukar

1	3	7	4	6

Looping ketiga:

1	3	7	4	6

1	3	7	4	6

tukar

1	3	4	7	6

Looping keempat:

1	3	4	7	6

tukar

1	3	4	6	7

Gambar 2. Proses Simulasi Bubble Sort

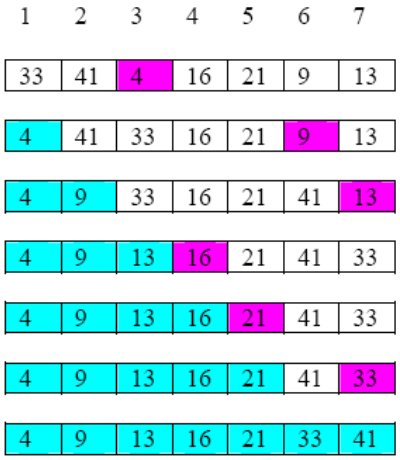
2.3 Selection Sort

Untuk lebih jelasnya, perhatikanlah simulasi *Selection Sort Ascending* berikut dengan menggunakan larik.

33	41	4	16	21	9	13

Gambar 3. Larik Array 7 elemen

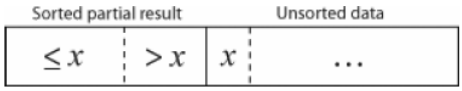
Dalam satu kali *pass*, ditentukan elemen yang paling kecil di dalam bagian list yang belum urut. Elemen yang paling kecil ini, diwarnai merah muda. Untuk bagian larik yang telah diurutkan diberi warna biru muda.



Gambar 4. Proses Simulasi Selection Sort

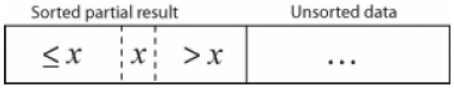
2.4 Insertion Sort

Cara kerja *insertion sort* sebagaimana namanya. Pertama-tama, dilakukan iterasi, dimana di setiap iterasi *insertion sort* memindahkan nilai elemen, kemudian menyisipkannya berulang-ulang sampai ke tempat yang tepat. Begitu seterusnya dilakukan. Dari proses iterasi, seperti biasa, terbentuklah bagian yang telah di-*sorting* dan bagian yang belum. Perhatikan gambar:



Gambar 5. Sebelum penyisipan

Dalam gambar, *insertion sort* sedang menangani elemen tabel yang ditandai *x*. Dengan satu kali *pass*, dilakukan penyisipan berulang-ulang dengan elemenelemen sebelumnya sampai ditemukan elemen dengan nilai lebih kecil atau sama dengan *x*. Perhatikan gambar:



Gambar 6. Sesudah penyisipan

Algoritma pengurutan dengan penyisipan ini kurang lebih dua kali lebih cepat dibandingkan dengan algoritma pengurutan gelembung dan hampir 40% lebih cepat dibandingkan algoritma pengurutan dengan seleksi.

2.5 Binary Search

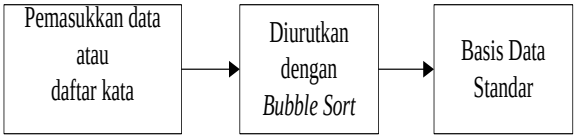
Binary Search adalah algoritma pencarian yang lebih efisien daripada algorima *Sequential Search*. Hal ini dikarenakan algoritma ini tidak perlu menjelajahi setiap elemen dari tabel. Kerugiannya adalah algoritma ini hanya bisa digunakan pada tabel yang elemennya sudah terurut baik menaik maupun menurun.

Pada intinya, algoritma ini menggunakan prinsip *divide and conquer*, dimana sebuah masalah atau tujuan diselesaikan dengan cara mempartisi masalah menjadi bagian yang lebih kecil. Algoritma ini membagi sebuah tabel menjadi dua dan memproses satu bagian dari tabel itu saja.

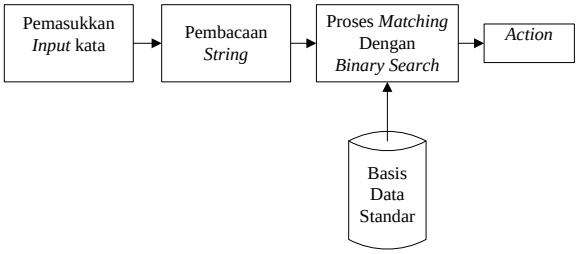
Algoritma ini bekerja dengan cara memilih *record* dengan indeks tengah dari tabel dan membandingkannya dengan *record* yang hendak dicari. Jika *record* tersebut lebih rendah atau lebih tinggi, maka tabel tersebut dibagi dua dan bagian tabel yang bersesuaian akan diproses kembali secara rekursif.

3. METODOLOGI

3.1 Perencanaan Sistem



Gambar 7. Proses Pembuatan Standar Kata dalam Kamus



Gambar 8. Proses Pencarian Padanan Kata

Perancangan sistem didasarkan pada pembuatan basis data (database) standar yang berisi kumpulan kata-kata. Kemudian basis data tersebut akan dirutkan menggunakan metode *sorting Bubble Sort*, *Selection Sort* dan *Insertion Sort*. Sehingga akan tercipta basis data baru dengan kumpulan kata yang telah terurut seperti kamus. Basis data tersebut akan memudahkan pengolahan *string* pada tahap pencarían

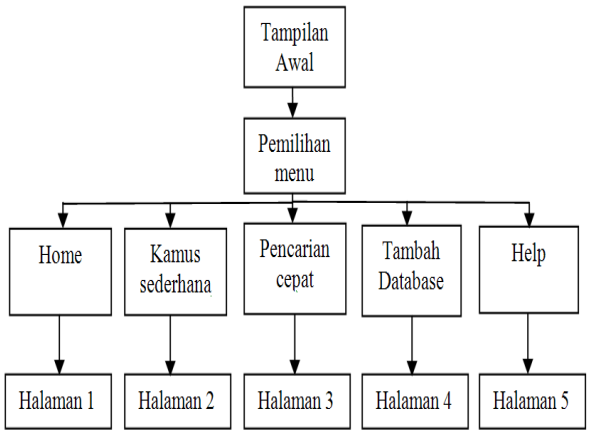
kata, karena digunakan metode pencarian *Binary* yang mengharuskan data (yang diolah) sudah dalam kondisi terurut. Kesemuanya itu akan dihubungkan ke dalam satu *interface* yang dibuat menggunakan Netbeans.

3.2 Pembuatan Software

3.2.1 Perancangan Basis Data Standar

Basis data berfungsi sebagai tempat penyimpanan kata. Karena dalam hal ini dibuat *software* penerjemah, sehingga akan dibutuhkan banyak sekali kata baik dalam bahasa Indonesia (sebagai masukan) dan kata dalam bahasa Jawa (sebagai keluaran). Basis data yang dibuat disini tidak menggunakan pemrograman khusus untuk *database*, namun hanya memanfaatkan file txt yang disimpan pada *notepad*. Sehingga nantinya akan ada perintah yang mengintegrasikan file teks ke dalam software. Pada kasus ini, basis data dibagi menjadi dua macam yaitu basis data yang berisi kata-kata yang belum diurutkan (disimpan dengan nama kamusku.txt) dan basis data yang berisi kata-kata yang sudah diurutkan (disimpan dengan nama kamusurut.txt).

3.2.2 Perancangan Interface



Gambar 9. Alur Desain Sistem

3.2.3 Pengurutan dengan Bubble Sort

Langkah pertama pengurutan data secara urut naik dengan metode ini adalah membandingkan harga data pertama dengan data kedua. Jika harga data pertama lebih besar maka posisinya ditukar dengan data kedua. Kemudian data kedua dibandingkan dengan data ketiga, ditukar posisinya jika data kedua lebih besar dari data ketiga. Selanjutnya data ketiga dibandingkan dengan data keempat, ditukar posisinya jika data ketiga lebih besar dari data keempat. Proses seperti ini akan diulang secara terus menerus hingga semua data selesai dibandingkan. Hasil pada langkah pertama adalah menempatkan data terbesar pada urutan terakhir. Data terbesar pada urutan terakhir tersebut tidak perlu diproses lagi pada langkah selanjutnya.
Pada langkah kedua, data pertama dibandingkan dengan data kedua. Jika harga data pertama lebih besar maka posisinya ditukar dengan data kedua. Kemudian data kedua dibandingkan dengan data ketiga, ditukar posisinya jika data kedua lebih besar dari data ketiga. Selanjutnya data ketiga dibandingkan dengan data keempat, ditukar posisinya jika data ketiga lebih besar dari data keempat. Proses seperti ini akan diulang secara terus menerus hingga semua data selesai dibandingkan. Hasil pada langkah kedua adalah menempatkan data terbesar pada urutan terakhir. Sehingga data terbesar yang diperoleh tersebut akan menempati urutan kedua dari belakang dari keseuruhan data hasil pengurutan. Data terbesar pada urutan terakhir tersebut tidak perlu diproses lagi pada langkah selanjutnya.

3.2.4 Pengurutan dengan Selection Sort

Prosedur pengurutan data dengan metode *selection sort* cukup sederhana dan mudah dipahami. Untuk memperoleh data dalam kondisi urut naik, maka prosedur yang dilakukan pada metode ini dapat dijelaskan sebagai berikut. Pada langkah pertama, data terkecil harus dicari dari seluruh data dan kemudian ditempatkan pada posisi urutan pertama. Langkah kedua adalah mencari data terkecil kedua dari seluruh data kecuali data pertama, dan kemudian ditempatkan pada posisi urutan kedua. Langkah ketiga adalah mencari data terkecil ketiga dari seluruh data kecuali data pertama dan kedua, dan kemudian ditempatkan pada posisi urutan ketiga. Demikian secara terus menerus akan diulangi proses tersebut hingga semua data akan menempati posisinya secara tepat sehingga menghasilkan sekumpulan data yang urut.

Proses pada langkah pertama dilakukan dengan cara mengambil data pertama dan kemudian dibandingkan dengan data kedua. Bila harga data pertama lebih besar dari harga data pada urutan kedua, maka data pertama ditukar dengan data kedua. Jika data kedua lebih besar dari data pertama maka proses dilanjutkan untuk membandingkan data pertama dengan data ketiga. Proses tersebut dilanjutkan hingga data pertama selesai dibandingkan dengan seluruh data yang ada, sehingga nantinya akan diperoleh harga data terkecil. Hasilnya kemudian akan ditempatkan pada urutan pertama. Karena data terkecil telah menempati lokasi yang benar, maka selanjutnya data pertama tidak perlu diproses lagi.

Pada langkah kedua, data kedua dibandingkan dengan data ketiga. Bila data kedua lebih besar dari urutan ketiga, maka keduanya ditukar tempat. Jika data ketiga lebih besar, maka proses dilanjutkan untuk membandingkan data kedua dengan data keempat. Jika data kedua lebih besar, maka ditukar posisinya dengan data keempat. Proses tersebut diteruskan hingga data kedua selesai dibandingkan dengan seluruh data yang ada, sehingga akan diperoleh data terkecil. Hasilnya kemudian akan ditempatkan pada urutan kedua. Karena data terkecil kedua telah menempati posisi yang benar, maka selanjutnya data kedua tidak perlu diproses lagi. Pada langkah selanjutnya diulangi untuk memperoleh data terkecil ketiga, keempat, kelima dan seterusnya sehingga semua data menempati lokasi yang tepat. Pada akhirnya diperoleh data dalam keadaan terurut menaik.

3.2.5 Pengurutan dengan Insertion Sort

Secara umum proses pengurutan data dengan metode *insertion sort* merupakan kebalikan dari proses yang dilakukan dalam metode *selection sort*. Dalam metode insertion sort, hanya akan diperhatikan satu elemen data dari sumber dan harus memperhatikan semua elemen data dari larik tujuan (*one source multiple destination*).

Pengurutan dengan *metode insertion sort* dimulai dengan membandingkan harga data pada urutan kedua terhadap harga data pertama. Jika data kedua lebih besar, maka ditukar posisinya dengan data pertama. Pada langkah berikutnya data pada urutan ketiga dibandingkan dengan data pada urutan kedua. Jika data ketiga lebih besar, maka ditukar posisinya dengan data kedua. Kemudian kedua data tersebut dibandingkan denga data pertama, posisi keduanya ditukar jika data urutan ketiga lebih besar. Berikutnya data keempat dibandingkan terhadap data-data urut yang telah diproses yaitu data pertama, kedua dan ketiga. Dalam metode *insertion sort*, proses membandingkan harga data baru (data yang sedang diproses) terhadap data-data lain yang telah urut akan dihentikan jika data tesebut telah menempati posisi yang semestinya sehingga data-data yang telah diproses akan urut.

3.2.6 Pencarian Kata dengan Binary Searching

Binary Search adalah algoritma pencarian yang lebih efisien daripada algorima *Sequential Search*. Hal ini dikarenakan algoritma ini tidak perlu menjelajahi setiap elemen dari tabel. Kerugiannya adalah algoritma ini hanya bisa digunakan pada tabel yang elemennya sudah terurut baik menaik maupun menurun.

Pada intinya, algoritma ini menggunakan prinsip *divide and conquer*, dimana sebuah masalah atau tujuan diselesaikan dengan cara mempartisi masalah menjadi bagian yang lebih kecil. Algoritma ini membagi sebuah tabel menjadi dua dan memproses satu bagian dari tabel itu saja.

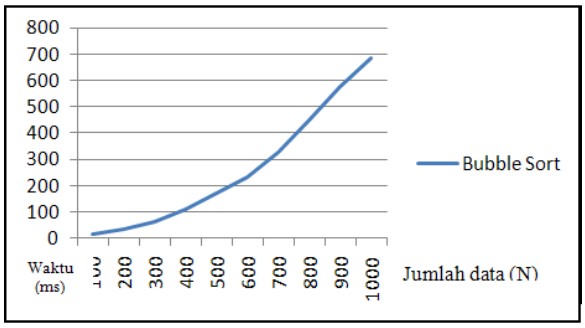
Algoritma ini bekerja dengan cara memilih *record* dengan indeks tengah dari tabel dan membandingkannya dengan *record* yang hendak dicari. Jika *record* tersebut lebih rendah atau lebih tinggi, maka tabel tersebut dibagi dua dan bagian tabel yang bersesuaian akan diproses kembali secara rekursif.

4. HASIL DAN ANALISA

4.1 Pendahuluan

Pengujian dilakukan pada metode sorting dan searching yang digunakan. Dimana pengujian meliputi efisiensi tiap metode terhadap fungsi waktu dan banyaknya data.

4.2 Efisiensi Bubble Sort



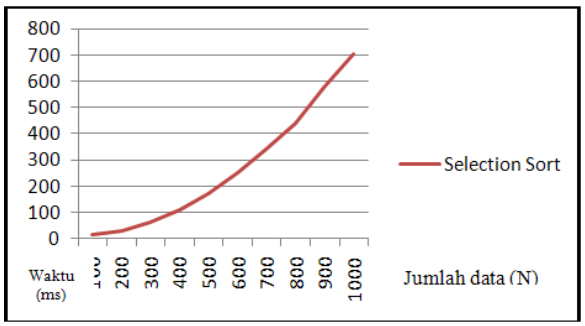
Gambar 10. Grafik efisiensi Bubble Sort

Berikut adalah tabel yang menunjukkan rata-rata waktu yang dibutuhkan saat eksekusi terhadap 1000 data.

Tabel 1. Tabel Rata-Rata Waktu

Data	Percobaan ke- (ms)										Rata-Rata (ms)
	1	2	3	4	5	6	7	8	9	10	
Urut	672	672	672	672	672	687	703	656	672	672	675
Acak	687	704	687	704	688	766	703	703	687	703	703

4.3 Efisiensi Selection Sort



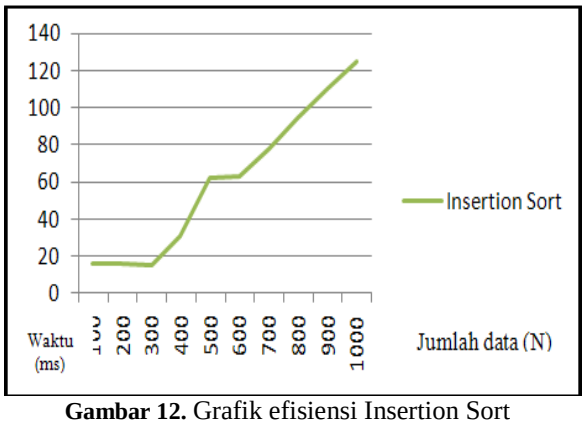
Gambar 11. Grafik efisiensi Selection Sort

Berikut adalah tabel yang menunjukkan rata-rata waktu yang dibutuhkan saat eksekusi terhadap 1000 data.

Tabel 2. Tabel Rata-Rata Waktu

Data	Percobaan ke- (ms)										Rata-Rata (ms)
	1	2	3	4	5	6	7	8	9	10	
Urut	688	703	703	687	688	688	718	703	687	688	695
Acak	688	796	890	828	797	688	703	687	688	703	746

4.4 Efisiensi Insertion Sort



Gambar 12. Grafik efisiensi Insertion Sort

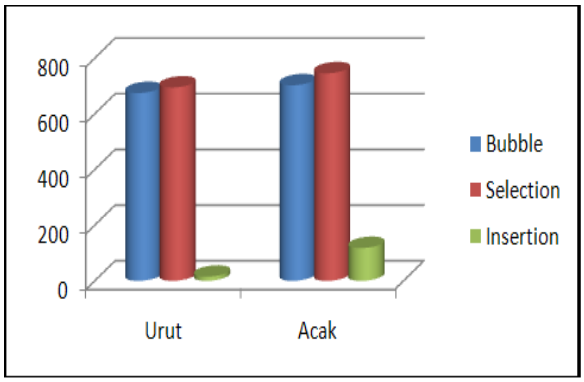
Berikut adalah table yang menunjukkan rata-rata waktu yang dibutuhkan saat eksekusi terhadap 1000 data.

Tabel 3. Tabel Rata-Rata Waktu

Data	Percobaan ke- (ms)										Rata-Rata (ms)
	1	2	3	4	5	6	7	8	9	10	
Urut	15	15	16	16	16	16	16	15	16	16	15
Acak	125	125	125	125	110	110	109	125	125	109	118

4.5 Perbandingan Kompleksitas Waktu

Setelah memperoleh rata-rata kecepatan *sorting* dari tiga buah metode *sorting* yaitu *Bubble Sort*, *Selection Sort* dan *Insertion Sort*, maka dapat diperoleh tabel Hasil eksekusi rata-rata ketiga metode tersebut, sebagai berikut.



Gambar 13. Grafik perbandingan

4.6 Pengujian Proses Searching

4.6.1 Pengujian kinerja Binary Search

Dalam *binary search*, pertama kali kita membandingkan *item* dalam posisi tengah dari *array*. Jika cocok, kita dapat mengembalikan secepatnya. Jika data yang dicari lebih kecil dari data tengah, maka *item* yang dicari berada pada setengah *array* kecil, jika lebih besar berada pada setengah *array* besar. Dengan demikian kita mengulangi prosedur ini pada setengah bagian *array* kecil.

Setiap langkah dari algoritma membagi blok-blok menjadi setengah dimana dalam blok-blok itu akan dicari *item* yang sesuai dengan *key* (data yang dicari). Kita dapat membagi n *item* menjadi dua, paling banyak $\log_2 n$ kali. Berarti dalam kasus ini:

$\log_2 1000 = \log 1000 : \log 2 = 3 : 0.3 = 10$

- 1000:2 = 500
- 500:2 = 250
- 250:2 = 125
- 125:2 = 62.5 ~ 62 (pembulatan ke atas)
- 62.5:2 = 31.25 ~ 31
- 31.25:2 = 15.625 ~ 15
- 15.625:2 = 7.8125 ~ 7
- 7.8125:2 = 3.90625 ~ 3
- 3.90625:2 = 1.953125 ~ 1
- 1.953125:2 = 0.9765625 ~ 0

Jika cacah data dalam vektor adalah ganjil, maka titik tengah interval akan membagi vektor tersebut menjadi dua bagian yang persis sama. Sebaliknya, jika cacah data dalam vektor adalah genap, maka titik tengah interval akan membagi

vektor tersebut menjadi dua bagian dimana salah satu bagian akan mempunyai cacah data lebih banyak.

Tabel 4. Tabel Jumlah elemen

Item (N)	Jumlah Data (elemen)	
	Interval 1	Interval 2
1000	499	500
500	249	250
250	124	125
125	62	62
62	30	31
31	15	15
15	7	7
7	3	3
3	1	1

4.6.2 Perbandingan pengujian Teori dan Praktik

Dengan menggunakan metode *Binary search*, akan dicari kunci 232 dengan seluruh data sejumlah 1000.

Penyelesaian.

(1+1000)/2 = 500

Angka 500 lebih besar dari 232, maka pencarian dilakukan lagi.

(1+499)/2 = 250

Angka 250 lebih besar dari 232, maka pencarian dilakukan lagi.

(1+249)/2 = 125

Angka 125 lebih kecil dari 232, maka pencarian dilakukan lagi.

(126+249)/2 = 187

Angka 187 lebih kecil dari 232, maka pencarian dilakukan lagi.

(188+249)/2 = 218

Angka 218 lebih kecil dari 232, maka pencarian dilakukan lagi.

(219+249)/2 = 234

Angka 234 lebih besar dari 232, maka pencarian dilakukan lagi.

(219+233)/2 = 226

Angka 226 lebih kecil dari 232, maka pencarian dilakukan lagi.

(227+233)/2 = 230

Angka 230 lebih kecil dari 232, maka pencarian dilakukan lagi.

(231+233)/2 = 232

Sudah ditemukan

Pada praktik, juga akan dicari kunci 232 yaitu data ‘dangkal’.

Gambar 14. Tampilan Perangkat Lunak

Pada hasil *compiler* dapat diperoleh keterangan bahwa data yang dicari adalah ‘dangkal’ dan merupakan data ke-231, karena *database* yang disimpan merupakan kumpulan *array*, maka nilai terendah adalah 0 dan tertinggi adalah 999. Jadi, data 232 akan dibaca sebagai data 231. Sehingga dapat disimpulkan bahwa teori sama dengan praktik.

```
cari kiri
cari kiri
cari kanan
cari kanan
cari kanan
cari kiri
cari kanan
cari kanan
data yang anda cari ditemukan
data yang dicari:dangkal

waktu : duration 0:0:0:16
data ke-231
```

Gambar 15. Hasil *Compiler*

5. KESIMPULAN

Dari hasil pengujian dan analisa pada bab sebelumnya, maka dapat diambil beberapa kesimpulan sebagai berikut :

1. Pada kondisi urut selisih waktu eksekusi algoritma *Insertion Sort* 97 % lebih singkat dibandingkan dengan Algoritma *Selection Sort* dan *Bubble Sort*. Serta pada kondisi acak, selisih waktu eksekusi algoritma *Insertion Sort* 84 % singkat dibandingkan dengan Algoritma *Selection Sort* dan *Bubble Sort.*
2. Berdasarkan perhitungan yang dilakukan dengan formula $\log_2 n$, diperoleh bahwa dengan data 1000 maka data dapat dibagi menjadi dua paling banyak 10 kali pembagian.
3. Berdasarkan waktu eksekusi yang relatif lebih singkat dibanding metode *sorting* lain, *Insertion Sort* lebih efisien untuk digunakan.

6. DAFTAR PUSTAKA

[1] Cook, V. 2001. *Second Language Learning and Language Teaching*, edisi ketiga London: Oxford University Press.

[2] Jurianto, Drs., *Pengembangan Kamus Elektronik Akuntansi berbasis Korpus*, universitas Airlannga, 2007.

[3] Bambang Hariyanto, Ir., MT., *Esensi-esensi bahasa pemrograman Java, Informatika, Bandung, 2007*.

[4] Betha Sidik, Ir., *MySQL*, Informatika, Bandung, 2003.

[5] Wahyu Fahmy Wisudawan., *Kompleksitas Algoritma Sorting yang PopulerDipakai*, Program Studi Teknik Informatika ITB, Bandung

[6] Indrayana, Muhamad Ihsan Fauzi, "*Perbandingan Kecepatan/Waktu Komputasi Beberapa Algoritma Pengurutan (Sorting)"*, Institut Teknologi Bandung, Jl. Ganesha 10 Bandung

[7] Ronny, "*Studi Mengenai Perbandingan Sorting Algorithmics Dalam Pemrograman dan Kompleksitasnya"*, Teknik Informatika Institut Teknologi Bandung.