ANALISIS ALGORITMA INSERTION SORT, MERGE SORT DAN IMPLEMENTASINYA DALAM BAHASA PEMROGRAMAN C++

Arief Hendra Saptadi[1] ~ Desi Windi Sari[2]
Program Studi D-III Teknik Telekomunikasi
Akademi Teknik Telekomunikasi Sandhy Putra Purwokerto[1]
Jurusan Teknik Elektro Universitas Sriwijaya Palembang[2]

ariefhs@akatelsp.ac.id[1], desi.windisari.9@facebook.com[2]

ABSTRAK

Makalah ini mengetengahkan kajian implementasi dan performa proses pengurutan menggunakan dua algoritma yang berbeda, yaitu Insertion Sort dan Merge Sort. Pada tahap pertama, kedua algoritma tersebut diimplementasikan dalam bahasa C++ untuk mengurutkan sejumlah angka yang diketikkan oleh pengguna. Pada tahap kedua, kode sumber untuk kedua algoritma tersebut diubah untuk dapat mengurutkan angka yang dihasilkan secara acak dengan jumlah angka sebanyak permintaan dari pengguna. Untuk mengetahui seberapa baik performa dalam mengurutkan data, maka dalam tahap terakhir, kedua algoritma tersebut mengurutkan sejumlah angka acak dengan rentang jumlah yang sudah ditentukan dan hasilnya kemudian dibandingkan. Dari eksperimen yang sudah dilakukan, algoritma merge sort telah memperlihatkan performa yang lebih baik, khususnya untuk jumlah data yang banyak (> 10000). Adapun algoritma insertion sort memiliki keuntungan dalam hal kompleksitas algoritma yang lebih rendah terutama dalam kondisi *best case* dan karena tidak menggunakan rutin rekursi dalam proses pengurutan, maka tidak membutuhkan ruang penyimpanan atau memori sebanyak algoritma Merge Sort.

Kata kunci : Algoritma, *Insertion Sort*, *Merge Sort*, Performa, Bahasa C++

ABSTRACT

This paper presents the study of implementation and performance in sorting process, using two different algorithms, namely Insertion Sort and Merge Sort. In the first stage, the two algorithms were implemented in C++ language to sort several numbers typed by a user. In the second stage, the source codes for those algorithms were modified to enable sorting randomly generated numbers with the amount as requested by the user. To find out how well they performed in sorting out the data, therefore in the last stage the two algorithms were tested to sort random numbers with predetermined amount ranges and the results were compared. From the experiments performed, merge sort algorithm had shown a better performance, particularly in a large number of data (> 10000). Insertion sort algorithm has the advantage in lower complexity algorithm, notably in the best case condition and since it does not use recursion routines in sorting process, hence it does not require as much storage space or memory as needed by merge sort algorithm.

***Keywords**: Algorithm, Insertion Sort, Merge Sort, Performance, C++ language*

1. PENDAHULUAN

1. 1. Latar Belakang

Sorting atau pengurutan adalah proses menyusun elemen – elemen dari masukan awal acak menjadi keluaran akhir tertata dengan urutan tertentu[3]. Proses tersebut diimplementasikan dalam bermacam aplikasi. Contoh penerapannya antara lain berupa rincian transaksi sesuai urutan tanggal dan jam pada perbankan, daftar hadir yang diurutkan berdasarkan nomor induk dan daftar pustaka yang diurutkan sesuai abjad pengarang ataupun katalog buku di perpustakaan. Fungsi-fungsi statistik seperti median dan pembuatan kuartil data (*quarter*), desil dan percentil (*percentile*) mensyaratkan data untuk diurutkan terlebih dahulu.

Beberapa macam algoritma sorting telah dibuat karena proses tersebut sangat mendasar dan sering digunakan. Oleh karena itu, pemahaman atas algoritma – algoritma yang ada sangatlah berguna. Selain menjadi suatu

aplikasi yang berdiri sendiri, pengurutan juga biasanya menjadi suatu bagian dari algoritma yang lebih besar.

Permasalahan pengurutan (sorting problem) secara formal didefinisikan sebagai berikut[7]:

Input: Suatu urutan dari n bilangan,

Output: Suatu permutasi atau penyusunan kembali dari input sedemikian rupa sehingga pada tata urutan *ascending* (dari nilai kecil ke besar) atau pada tata urutan *descending* (dari nilai besar ke kecil).

Sebagai contoh jika diberikan masukan lima bilangan acak maka keluarannya adalah sebagaimana berikut ini:

Input:	2	5	4	1	6
	3	(n = 6)			
Output:	1	2	3	4	5
	6	(*ascending*)			
	6	5	4	3	2
	1	(*descending*)			

Data yang diurutkan tidak harus berupa angka, namun bisa saja *string*, misalnya:

Input:	Saptadi	Windisari	Desi
	Hendra	Arief	(n = 5)
Output:	Arief	Desi	Hendra
	Saptadi	Windisari	(*ascending*)
	Windisari	Saptadi	Hendra
	Desi	Arief	(*descending*)

Pengurutan juga bisa diterapkan secara tidak langsung pada sekelompok data. Misalkan untuk kelima data nama di atas, hendak diurutkan dari nama dengan jumlah karakter terkecil hingga nama dengan jumlah karakter terbanyak.

Sehingga keluarannya diperoleh:

Output:	Desi	Arief	Hendra
	(4)	(5)	(6)
	Saptadi	Windisari	
	(7)	(9)	(*ascending*)

Jadi dapat disimpulkan bahwa dalam pengurutan harus terdapat:

- data yang akan diurutkan dalam tipe yang sama atau setidaknya memperoleh perlakuan data yang sama
- aturan pengurutan yang jelas

Ada banyak metode pengurutan antara lain: bubble sort, bi-directional bubble sort, selection sort, shaker sort, insertion sort, in-place merge sort, double storage merge sort, comb sort 11, shell sort, heap sort, exchange sort, merge sort, quick sort, quick sort with bubblesort, enhange quick sort, fast quick sort, radix sort algorithm, swap sort, dan lain sebagainya[4].

Untuk membatasi luasnya pembahasan, maka dalam makalah ini hanya akan dibahas 2 metode, yaitu Insertion Sort dan Merge Sort. Pembahasan untuk tiap metode akan difokuskan pada cara kerja pengurutan beserta contohnya, analisa algoritma untuk kondisi terburuk (*worst case*), rata-rata (*average case*), terbaik (*best case*), implementasinya dalam bahasa C++ serta pengujian waktu eksekusi untuk kedua metode tersebut.

1.2 Tujuan Penulisan

Dari penjelasan diatas, tujuan yang ingin dicapai dari penelitian ini adalah :

1. Menerapkan algoritma merge sort dan insertion sort ke dalam bahasa C++.
2. Menguji dan membandingkan performa algoritma merge sort dan insertion sort dalam proses pengurutan.

1.3 Batasan Masalah

Adapun batasan permasalahan dalam penelitian ini adalah sebagai berikut:

1. Algoritma yang dikaji hanya insertion sort dan merge sort.
2. Implementasi algoritma menggunakan bahasa pemrograman C++.
3. Pengujian performa dilakukan dengan membandingkan waktu eksekusi pengurutan untuk tiap algoritma.
4. Pengujian dibatasi dan hanya dilakukan sesuai dengan spesifikasi perangkat keras dan perangkat lunak yang disebutkan.

2. KAJIAN PUSTAKA

2.1. Insertion Sort

Salah satu algoritma sorting yang paling sederhana adalah insertion sort. Insertion Sort disebut-sebut sebagai metode pertengahan. Artinya, metode ini memiliki kecepatan rata-rata antara metode primitif (bubble dan selection) dan modern (merge dan quick)[10]. Metode ini, didasarkan pada sebuah kunci yang diambil pada elemen ke-2 pada sebuah larik, lalu menyisipkan elemen tersebut jika kondisi percabangan terpenuhi. Metode penyisipan (insertion) bertujuan untuk menjadikan bagian sisi kiri larik terurutkan sampai dengan seluruh larik berhasil diurutkan.

2.1.1. Algoritma dan Pseudocode

Ide dari algoritma ini dapat dianalogikan seperti mengurutkan kartu.

Penjelasan berikut ini menerangkan bagaimana algoritma insertion sort bekerja dalam pengurutan kartu[7].

Gambar 1. Analogi Metode Insertion Sort

Anggaplah bahwa terdapat sebuah meja yang berisi setumpuk kartu. Meja ini melambangkan kondisi larik sebelum diurutkan. Langkah-langkah pengurutan adalah sebagai berikut:

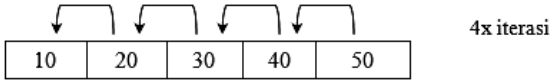
- Ambil kartu pertama dari meja, letakkan di tangan kiri.
- Ambil kartu kedua dari meja, bandingkan dengan kartu yang berada di tangan kiri, kemudian letakkan pada urutan yang sesuai setelah perbandingan.
- Ulangi proses hingga seluruh kartu pada meja telah diletakkan berurutan pada tangan kiri.

Kartu-kartu pada tangan kiri tersebut menunjukkan kondisi larik sesudah diurutkan. *Pseudocode* untuk algoritma insertion sort adalah sebagai berikut[6]:

```
insertionsort(data[],n)
  for(i = 1; i < n; i++)
     pindahkan seluruh elemen
     data[j] yang lebih besar
     daripada data[i] sebanyak
     satu posisi;
     geser data[ i ] pada posisi
     yang tepat;
```

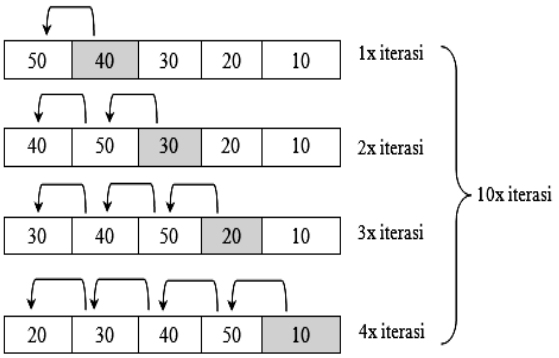
2.1.2. Kompleksitas Algoritma

Kondisi terbaik (*best case*) tercapai jika data telah terurut. Hanya satu perbandingan dilakukan untuk setiap posisi i, sehingga terdapat n – 1 perbandingan, atau O (n)[6].



Gambar 2. Kondisi Best Case pada Insertion Sort

Kondisi terburuk (*worst case*) tercapai jika data telah urut namun dengan urutan yang terbalik. Pada kasus ini, untuk setiap i, elemen data[i] lebih kecil dari elemen data[0], …, data[i-1], masing-masing dari elemen dipindahkan satu posisi[6].



Gambar 3. Kondisi Worst Case pada Insertion Sort

Untuk setiap iterasi i pada kalang for terluar, selalu ada perbandingan i, sehingga jumlah total perbandingan untuk seluruh iterasi pada kalang ini adalah[6]:

$$\sum_{i=1}^{n-1} i = 1+2+...+(n-1) = \frac{n(n-1)}{2} = O(n^2) \qquad (1)$$

2.2. Merge Sort

Merge sort adalah metode pengurutan yang menggunakan pola divide and conquer[10]. Strateginya adalah dengan membagi sekelompok data yang akan diurutkan menjadi beberapa kelompok kecil terdiri dari maksimal dua nilai untuk dibandingkan dan digabungkan lagi secara keseluruhan.

Langkah kerja dari Merge sort[9]:

1. Divide

   Memilah elemen – elemen dari rangkaian data menjadi dua bagian dan mengulangi pemilahan hingga satu elemen terdiri maksimal dua nilai.
2. Conquer

   Mengurutkan masing-masing elemen.
3. Kombinasi

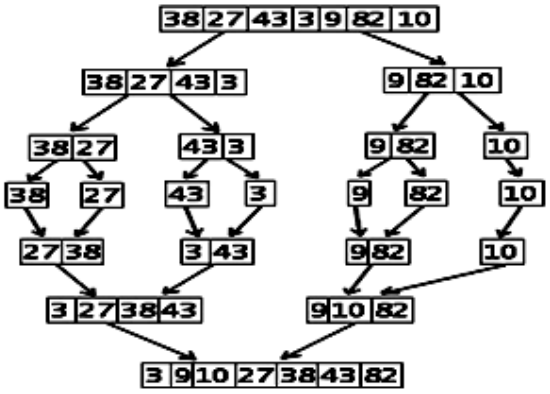
   Mengkombinasikan dua bagian tersebut secara rekursif untuk mendapatkan rangkaian data berurutan. Proses rekursi berhenti jika mencapai elemen dasar. Hal ini terjadi

bilamana bagian yang akan diurutkan menyisakan tepat satu elemen. Sisa pengurutan satu elemen tersebut menandakan bahwa bagian tersebut telah terurut sesuai rangkaian.

2.2.1. Algoritma dan Pseudocode

Algoritma Merge sort sebenarnya sederhana[9]: bagi larik menjadi dua sama besar, urutkan bagian pertama, urutkan bagian kedua, lalu gabungkan.

Sebagai contoh, jika terdapat data berupa 38, 27, 43, 3, 9,82, dan 10 maka ilustrasi pengurutannya adalah sebagai berikut:



Gambar 5. Ilustrasi Merge Sort

Pseudocode untuk merge sort[6] adalah sebagai berikut:

```
mergesort(data)
  if data memiliki setidaknya dua
  elemen
      mergesort (separuh kiri dari
      data);
      mergesort (separuh kanan
      dari data ;
  merge (kedua bagian ke dalam
  suatu urutan);
```

Sedangkan *pseudocode* untuk `merge` itu sendiri adalah:

```
merge (arrayl, pertama, terakhir)
  tengah = (pertama + terakhir) /
  2;
  il = 0;
  i2 = pertama;
  i3 = tengah + 1;
  while kedua sub larik dari
  array1 memiliki elemen
    if arrayl[i2] < arrayl[i3]
      temp[il++] = arrayl[i2++];
    else
      temp[il++] = arrayl[i3++];
  masukkan ke dalam temp sisa
  elemen dari arrayl;
  masukkan ke arrayl isi dari
  temp;
```

2.2.2. Kompleksitas Algoritma

Kompleksitas algoritma untuk larik dengan n elemen dan jumlah pergeseran (T) dihitung melalui relasi rekursif berikut ini[5]:

$$T(1) = 0 \quad (2a)$$

$$T(n) = 2T\left(\frac{n}{2}\right) + 2n \quad (2b)$$

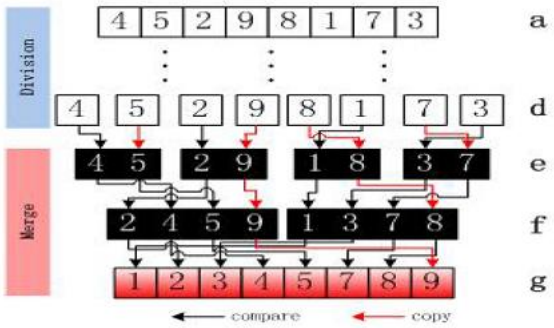
Adapun M(n) dihitung lewat cara berikut[5]:

$$\begin{aligned} T(n) &= 2\left(2T\left(\frac{n}{4}\right) + 2\left(\frac{n}{2}\right)\right) + 2n = 4T\left(\frac{n}{4}\right) + 4n \\ &= 4\left(2T\left(\frac{n}{8}\right) + 2\left(\frac{n}{4}\right)\right) + 4n = 8T\left(\frac{n}{8}\right) + 6n \\ &\vdots \\ &= 2^i T\left(\frac{n}{2^i}\right) + 2in \end{aligned} \quad (3)$$

Memilih i = log n sedemikian sehingga n = 2i, maka diperoleh[5]:

$$\begin{aligned} T(n) &= 2^i T\left(\frac{n}{2^i}\right) + 2in = nT(1) + 2n \cdot \log n \\ &= 2n \cdot \log n = O(n \log n) \end{aligned} \quad (4)$$

Kasus terburuk (*worst case*) terjadi bila selama pemanggilan fungsi rekursif `merge`, nilai terbesar dari setiap elemen terletak di larik yang berbeda[2]. Hal ini memaksa fungsi `merge` untuk melakukan pengurutan secara berpindah-pindah antar larik, sebagaimana digambarkan berikut:



Gambar 6. Kondisi *Worst Case* pada Merge Sort

Pada kondisi ini[2]:

$$T(n) = 2T\left(\frac{n}{2}\right) + n - 1 \quad (5a)$$

$$T(1) = 0 \quad (5b)$$

Kedua persamaan tersebut untuk selanjutnya diperluas seperti berikut[2]:

$$T(n) = 2\left(2T\left(\frac{n}{4}\right)+\frac{n}{2}-1\right)+n-1$$

$$T(n) = 4T\left(\frac{n}{8}\right)+n-2+n-1$$

$$T(n) = 4\left[2T\left(\frac{n}{8}\right)+\frac{n}{4}-1\right]+n-2+n-1$$

$$T(n) = 8T\left(\frac{n}{8}\right)+n-4+n-2+n-1 \quad (6)$$

Dengan mengenali pola yang ada, maka dapat dituliskan persamaan:

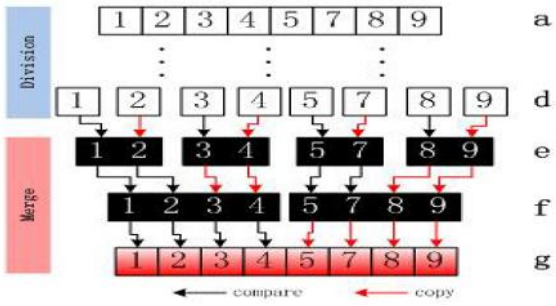
$$T(n) = 2^i T\left(\frac{n}{2^i}\right)+in-(2^i-1) \quad (7)$$

Dengan 2i = n dan I = n log n dan memasukkan nilai awal persamaan:

$$T(n) = n*0+n\log n-n+1 \quad (8)$$

Maka kompleksitas pada kondisi *worst case* adalah O (n log n)[2].

Kasus terbaik (*best case*) untuk metode ini dijumpai pada kondisi dimana elemen memiliki nilai terbesar yang lebih kecil dibandingkan dengan seluruh nilai pada elemen yang lain[7], sebagaimana berikut ini:



Gambar 7. Kondisi *Best Case* pada Merge Sort

Pada skenario ini hanya n/2 perbandingan dari elemen yang diperlukan. Menggunakan proses perhitungan yang sama sebagaimana dalam kasus terburuk, diperoleh[7]:

$$T(n) = 2T\left(\frac{n}{2}\right)+\frac{n}{2}$$

$$T(n) = 2^i T\left(\frac{n}{2^i}\right)+\frac{in}{2}$$

$$T(n) = n*0+\frac{n}{2}\log n \quad (9)$$

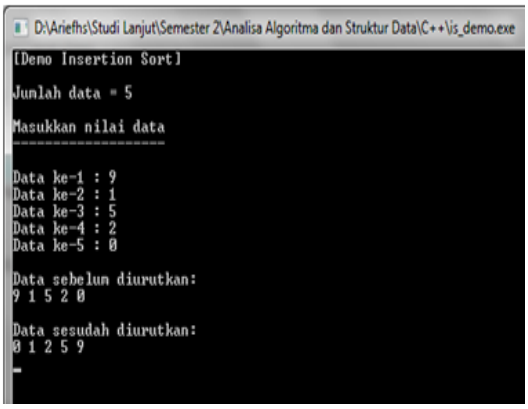
Dengan kata lain, diperoleh juga kompleksitas yang sama, O (n log n)[7].

3. IMPLEMENTASI

Implementasi dalam bahasa C++ untuk kedua algoritma tersebut dan tampilan ketika dijalankan adalah sebagai berikut:

3.1. Insertion Sort

```
void insertion_sort(int x[],int
panjang)
{
   int kunci,i;
   for(int j=1;j<panjang;j++)
   {
        kunci=x[j];
        i=j-1;
        while(x[i]>kunci&&i>=0)
        {
            x[i+1]=x[i];
            i--;
        }
        x[i+1]=kunci;
    }
}
```



Gambar 4. Tampilan Implementasi Insertion Sort dalam C++

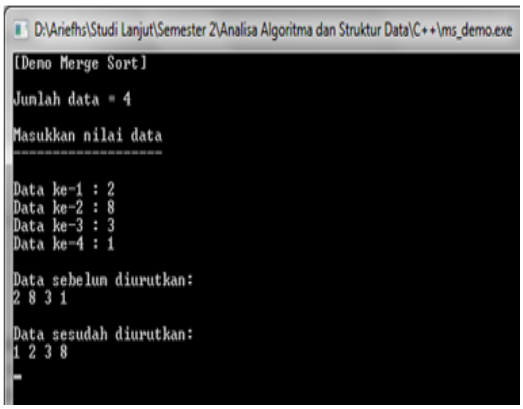
3.2. Merge Sort

```
void merge(int kiri,int tengah,
int kanan)
{
    int h,i,j,b[50],k;
    h=kiri;
    i=kiri;
    j=tengah+1;

    while((h<=tengah)&&
    (j<=kanan))
    {
        if(bilangan[h]<=
        bilangan[j])
        {
            b[i]=bilangan[h];
            h++;
        }
```

```
        else
        {
            b[i]=bilangan[j];
            j++;
        }
    i++;
    }
    if(h>tengah)
    {
        for(k=j;k<=kanan;k++)
        {
            b[i]=bilangan[k];
            i++;
        }
    }
    else
    {
        for(k=h;k<=tengah;k++)
        {
            b[i]=bilangan[k];
            i++;
        }
    }
    for(k=kiri;k<=kanan;k++)
        bilangan[k]=b[k];
}

void merge_sort(int kiri,
int kanan)
{
    int tengah;
    if(kiri<kanan)
    {
        tengah=(kiri+kanan)/2;
        merge_sort(kiri,tengah);
        merge_sort(tengah+1,
        kanan);
        merge(kiri,tengah,
        kanan);
    }
}
```



Gambar 5. Tampilan Implementasi Merge Sort dalam C++

4. PENGUJIAN

Tujuan pengujian ini adalah untuk mengetahui dan membandingkan kecepatan eksekusi perintah pengurutan data terhadap sekelompok data dengan rentang jumlah tertentu, baik pada algoritma insertion sort maupun merge sort.

4.1. Perangkat

Pengujian dilaksanakan dengan menggunakan compiler Bloodshed Dev-C++ versi 4.9.9.2 pada platform Windows 7 dan komputer *notebook* dengan spesifikasi:

a. Intel Pentium dual-core processor T2370 (1,73 GHz, 533 MHz FSB, 1 MB L2 cache).
b. Mobile Intel Graphic Media Accelerator X3100.
c. 2 GB DDR2-SDRAM.
d. 500 GB HDD.

4.2. Prosedur

Pengujian meliputi tahapan berikut ini:

a. Pengubahan Kode Sumber

Kode sumber diubah sehingga hanya berfungsi untuk menampilkan waktu eksekusi[9], lewat fungsi GetTickCount[13] sebagai berikut:

```
int mulai = GetTickCount();
<<fungsi yang diuji>>
int selesai = GetTickCount();
```

dan dengan menambahkan baris instruksi untuk mengisikan data secara acak[3], [2]:

```
srand(time(NULL));
for (int x=0; x<jumlah; x++)
{
    bilangan[x]=rand()%
    jumlah+1;
}
```

serta dengan menambahkan 2 buah pengarah *preprocessor*[1]:

```
#include <time.h>
#include <windows.h>
```

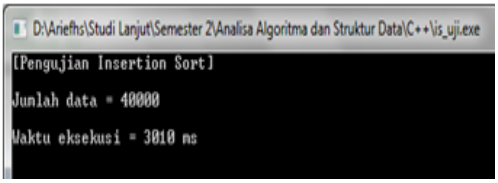
Waktu eksekusi untuk selanjutnya dapat diketahui dengan menggunakan selisih antara variabel selesai dan mulai[9], seperti berikut:

```
cout << endl <<
"Waktu eksekusi = " << (int)
(selesai - mulai) << " ms";
```

Seluruh fungsi untuk menampilkan keluaran kondisi larik, baik saat sebelum maupun sesudah diurutkan, dimatikan.

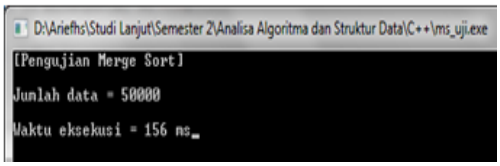
b. Eksekusi Program

Program untuk pengujian insertion sort maupun merge sort kemudian dieksekusi. Penguji memasukkan angka yang menunjukkan jumlah data yang akan diurutkan. Hasil eksekusi program pengujian insertion sort:



Gambar 9. Tampilan Pengujian Insertion Sort

Hasil eksekusi program pengujian merge sort adalah berikut ini:



Gambar 10. Tampilan Pengujian Merge Sort

c. Pencatatan Hasil

Data hasil pengujian selanjutnya dicatat dan dimasukkan ke dalam tabel.

d. Pengulangan

Yaitu mengulangi langkah b dan c di atas untuk tiap 10000 data dari 10000 hingga 100000.

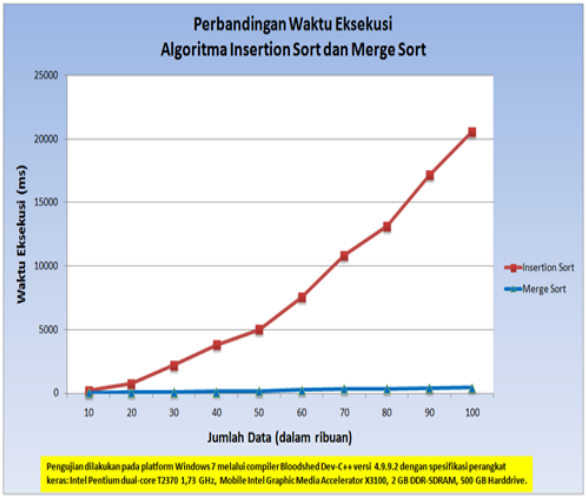
e. Hasil Akhir

Hasil akhir yang diperoleh dari pengujian adalah sebagaimana ditampilkan dalam Gambar 11.

5. KESIMPULAN

Dari hasil implementasi maupun pengujian yang telah dilakukan dapat ditarik beberapa kesimpulan sebagaimana berikut ini:

1. Dari hasil pengujian diketahui bahwa algoritma merge sort lebih cepat dibandingkan insertion sort untuk data yang lebih banyak, khususnya untuk jumlah data > 10000.
2. Kelemahan utama merge sort adalah algoritma ini membutuhkan setidaknya ruang atau memori dua kali lebih besar karena dilakukan secara rekursif dan memakai dua elemen terpisah.
3. Pada kasus *best case*, algoritma insertion sort lebih unggul daripada merge sort, sehubungan dengan kompleksitas yang lebih rendah yaitu nilai O(n) dibandingkan dengan O(n log n).
4. Pada kasus *average* maupun *worst case*, algoritma merge sort lebih unggul terhadap insertion sort, sehubungan dengan kompleksitas yang lebih rendah yaitu nilai O (n log n), dibandingkan dengan O (n^2).
5. Algoritma insertion sort secara teknis lebih mudah diterapkan dibandingkan dengan merge sort, berkaitan dengan panjangnya instruksi yang diperlukan.



Gambar 11. Hasil Akhir Pengujian Insertion Sort dan Merge Sort

Daftar Pustaka

[1] Anonim. *C++ Algorithms Sample Source Codes > Merge Sort*. http://www.cplusplus.happycodings.com /Algorithms/code17.html. Diakses pada 18 Mei 2011.

[2] Anonim. *Rand*. http://www.cplusplus.com/reference/clibrary/cstdlib/rand/. Diakses pada 17 Mei 2011.

[3] Anonim. *Srand*. http://www.cplusplus.com/reference/clibrary/cstdlib/srand/. Diakses pada 17 Mei 2011.

[4] Atrinawati, L. H. Analisis Kompleksitas algoritma untuk Berbagai Macam Metode Pencarian Nilai (*Searching*) dan Pengurutan Nilai (*Sorting*) pada Tabel. Program Studi Teknik Informatika. ITB. Bandung.

[5] Bingheng, W. 2008. *Merge Sort*. Dept. of Computer Science. Florida Institute of Technology. Florida, USA.

[6] Drozdek, A. 2001. *Data Structures and Algorithms in C++*. Brooks/Cole Thomson Learning. California. USA.

[7] Hibbler, R. 2008. *Merge Sort*. Dept. of Computer Science. Florida Institute of Technology. Florida, USA.

[8] Horman, T. H., Leiserson, C. E., Rivest, R. L., dan Stein, C. 2009. *Introduction to Algorithms*. The MIT Press. Cambridge. Massachusetts London. England.

[9] Horstmann, C. 2008. *C++ for Everyone*. Wiley Publishing. San Jose, USA.

[10] Karve, S. *Insertion Sort Example*. http://www.dreamincode.net/code/snippet279.htm. Diakses pada 18 Mei 2011.

[11] Kristanto, A. 2009. Algoritma dan Pemrograman dengan C++. Graha Ilmu. Yogyakarta.

[12] Liberty, J., Rao, S., Jones, B. 2008. *Sams Teach Yourself C++ in One Hour a Day*. Sams Publishing. Indiana, USA.

[13] MSDN. *GetTickCount Function*. http://msdn.microsoft.com/en-us/library/ms724408%28v=vs.85%29.aspx. Diakses pada 17 Mei 2011.

[14] Santosa, P.I. 2004. Struktur Data Menggunakan Turbo Pascal. Penerbit ANDI. Yogyakarta.

[15] Song, Q. 2008. *Merge Sort Algorithm*. Dept. of Computer Science. Florida Institute of Technology. Florida, USA.

[16] Yatini, I. dan Nasution, E. 2005. Algoritma dan Struktur Data. Graha Ilmu. Yogyakarta.