ANALISIS PERBANDINGAN METODE ALGORITMA QUICK SORT DAN MERGE SORT DALAM PENGURUTAN DATA TERHADAP JUMLAH LANGKAH DAN WAKTU

Yolanda Y.P Rumapea

Prodi Sistem Informasi, Universitas Methodist Indonesia
Jl. Hang Tuah no. 4 Medan

rumapeayolanda@gmail.com

Abstrak

Pada pengolahan data seringkali menjadi bagian yang krusial dalam proses pengurutan data (sorting), sehingga hal ini merupakan topik yang sangat penting untuk dapat diteliti lebih lanjut. Data yang harus diurutkan tentunya sangat bervariasi, bahkan cukup sulit untuk menentukan algoritma mana yang baik untuk situasi tertentu. Salah satu cara untuk memudahkan pemahaman algoritma pengurutan data adalah dengan menampilkan cara kerja dari algoritma secara langsung melalui animasi. Melalui animasi, pertunjukkan tahap-tahap proses pengurutan data dapat ditampilkan secara perlahan sehingga diharapkan dapat memberikan pemahaman yang lebih baik. Tujuan pengurutan data adalah untuk mempermudah proses pengolahan data seperti penyisipan data, penghapusan data, dan pencarian data.

Algoritma pengurutan data Quick Sort dan Merge Sort merupakan algoritma yang paling banyak digunakan dan dianggap yang paling cepat dan terbaik di dalam proses pengurutan data (sorting). Maka akan dianalisis dan dibandingkan jumlah langkah dan waktu (Run-Time) dari algoritma Quick Sort dan Merge Sort. Dalam hal ini, Bahasa pemrograman yang digunakan dalam perancangannya program intinya adalah Visual Basic 6.0 dengan Macromedia Flash 8.0 yang digunakan sebagai program pembuatan animasi. Pada hasil pengujian dan analisis diperoleh bahwa algoritma Quick Sort dan Merge Sort masing-masing memiliki kelebihan dan kekurangan pada waktu komputasi dan jumlah langkah. Banyak faktor yang mempengaruhi hal tersebut, salah satunya adalah faktor besar kecilnya input data, jenis input data dan juga penentuan nilai pivot (khusus pada algoritma Quick Sort).

Kata kunci : Quick Sort, Merge Sort, Jumlah Langkah , Run-Time

I. PENDAHULUAN

Struktur data sebagai salah satu cabang ilmu komputer saat ini mulai mengalami perkembangan. Penggunaan struktur data ini sangat efektif dalam menyelesaikan suatu persoalan. Pengurutan data (*sorting*) merupakan contoh yang baik untuk menunjukkan bahwa suatu persoalan bisa diselesaikan dengan sejumlah algoritma yang berbeda satu sama lain lengkap dengan kekurangan dan kelebihan. Pengurutan data secara umum bisa didefinisikan sebagai satu proses untuk menyusun kembali himpunan objek menggunakan aturan tertentu. Pengurutan data menjadi suatu bagian yang penting dalam pemrograman karena waktu yang diperlukan untuk melakukan proses pengurutan perlu dipertimbangkan. Pengurutan data juga digunakan dalam mengkompilasi program komputer dan juga memegang peran penting untuk mempercepat proses data yang harus dilakukan berulang kali. Data yang harus diurutkan tentunya sangat bervariasi baik dalam hal banyak maupun jenis data yang akan diurutkan. Bahkan cukup sulit untuk menentukan algoritma mana yang baik untuk situasi tertentu karena ada faktor yang mempengaruhi efektifitasnya.

Ada banyak metode dalam pengurutan data yang bertujuan untuk mempermudah pencarian data antara lain adalah *Quick Sort, Bubble Sort, Merge Sort, Seleksi, Shell Sort* dan sebagainya. Dalam penelitian ini metode yang akan dibahas adalah *Quick Sort* dan *Merge Sort* untuk mengetahui jumlah langkah serta waktu yang diperlukan untuk pengurutan data. Berdasarkan uraian di atas maka penulis membuat judul "*Analisis Algoritma untuk Membandingkan Pengurutan Data Quick Sort dan Merge Sort Terhadap Jumlah Langkah dan Waktu (Run Time)*"

II. PENGURUTAN DATA (*SORTING*)

Pengurutan data (ada juga yang menyebutnya sebagai pemilihan data) secara umum bisa didefinisikan sebagai suatu proses untuk menyusun kembali himpunan obyek menggunakan aturan tertentu. Secara umum ada dua jenis pengurutan data, yaitu pengurutan data secara urut naik (*ascending*) yaitu dari data yang nilainya paling kecil sampai data yang nilainya paling besar, atau pengurutan data secara urut turun (*descending*), yaitu dari data yang mempunyai nilai paling besar sampai paling kecil. Dalam hal pengurutan data yang bertipe *string* atau *char*, nilai data dikatakan lebih kecil atau lebih besar dari yang lain didasarkan pada urutan relatif (*collating sequence*) seperti yang telah dinyatakan dalam tabel ASCII. Keuntungan yang diperoleh dari data yang sudah dalam keadaan terurut adalah bahwa data mudah dicari (misalnya dalam buku telepon atau kamus bahasa), mudah untuk dibetulkan, dihapus, disisip, atau digabungkan, dan mudah mencek apakah ada data yang hilang (misalnya dalam tumpukan kartu bridge). Algoritma sangat ditentukan oleh struktur data yang digunakan, maka sejumlah algoritma pengurutan yang dijelaskan bisa diklasifikasikan menjadi dua kategori, yaitu pengurutan larik (*array*), dan pengurutan berkas masup urut (*Sequential Acces File*).

Kategori yang pertama disebut sebagai pengurutan secara internal, dan kategori kedua disebut dengan pengurutan eksternal. Dikatakan demikian karena larik (*array*) tersimpan dalam memori utama komputer yang mempunyai kecepatan tinggi, sedangkan berkas biasanya tersimpan dalam pengingat luar (tambahan), misalnya cakram, atau pita magnetis.

Contoh sederhana untuk membedakan dua kategori ini adalah pengurutan sejumlah kartu yang telah diberi nomor. Penyusunan kartu sebagai larik (*array*), misalkan

semua kartu terletak dihadapan pemakai sehingga semua terlihat dengan jelas nomornya, dan kartu bisa dimasup sendiri-sendiri.

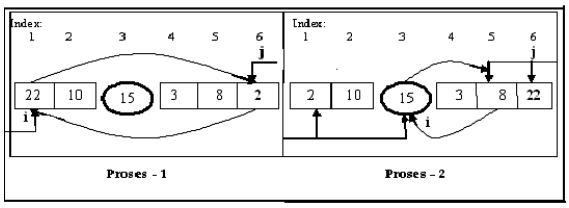
Penyusunan kartu sebagai sebuah berkas, agak berbeda yakni semua kartu harus ditumpukkan sehingga hanya satu kartu bagian atas saja yang bisa dilihat nomornya. Batasan ini akan membawa konsekuensi yang serius dalam pemilihan metode yang akan dibangun, tetapi mungkin tidak dapat dihindarkan karena jumlah kartu cukup banyak sehingga meja tidak cukup luas untuk meletakkan seluruh kartu di atasnya.

III. METODE QUICK SORT

Metode *Quick Sort* juga sering disebut dengan metode *Patition Exchange Sort*. Metode ini diperkenalkan oleh C.A.R Hoare. Untuk mempertinggi efektifitasnya, dalam metode ini jarak kedua elemen yang akan ditukarkan nilainya ditentukan cukup besar. Metode pengurutan *Quick Sort* dapat diimplementasikan dalam bentuk nonrekursif dan rekursif.

Misalkan N elemen dalam keadaan urut turun. Untuk mengurutkan N elemen tersebut dengan N/2 kali, yakni dengan pertama kali menukarkan elemen paling kiri dengan paling kanan, kemudian secara bertahap menuju ke elemen yang ada ditengah. Tetapi hal ini hanya bisa dilakukan jika tahu pasti bahwa urutannya adalah urut turun .

Secara garis besar metode ini dijelaskan sebagai berikut : misalkan ingin mengurutkan vektor A yang mempunyai 6 elemen. Pilih sembarang elemen dari vektor tersebut, biasanya elemen pertama, misalnya 15. Kemudian semua elemen tersebut disusun dengan menempatkan 15 pada posisi J sedemikian rupa sehingga elemen ke I sampai ke J – 1 mempunyai nilai lebih kecil dari 15 dan elemen ke J + 1 sampai ke N mempunyai nilai lebih besar dari 15. Dengan demikian ada dua buah subvektor, subvektor pertama nilai elemennya lebih kecil dari 15, subvektor kedua nilai elemennya lebih besar dari 15.



Gambar 1.Ilustrasi Metode Quick Sort.

Pada langkah berikutnya, proses di atas diulang pada kedua subvektor, sehingga akan mempunyai empat subvektor. Proses di atas diulang pada setiap subvektor sehingga seluruh vektor semua elemennya menjadi terurutkan. Untuk jelasnya perhitungan contoh berikut (ambil elemen pertama dan tempatkan pada posisinya).

A. Algoritma Quick Sort

[Digunakan untuk menempatkan elemen pertama pada posisi J sedemikian rupa sehingga elemen ke 1 sampai elemen ke J-1 selalu lebih kecil dari elemen tersebut, dan elemen J+1 sampai elemen ke N selalu lebih besar dari elemen tersebut.]

Langkah 1 Tentukan : I = Awal + 1.

Langkah 2 (Bergerak dari kiri ke kanan.)
Tambah nilai I dengan 1 selama A[I] < A[Awal].

Langkah 3 (Bergerak dari kanan ke kiri.)
Kurangi nilai I dengan 1 selama A[J] > A[Awal].

Langkah 4 Kerjakan langkah 5 sampai 7 selama I < J.

Langkah 5 Tukarkan nilai A[I] dengan A[J].

Langkah 6 (Bergerak dari kiri ke kanan.)
Tambah nilai I dengan 1 selama A[I] < A[Awal].

Langkah 7 (Bergerak dari kanan ke kiri.)
Kurangi nilai I dengan 1 selama A[J] < A[Awal].

Langkah 8 Tukarkan nilai A[I] dengan A[J].

Langkah 9 Selesai.

B. *Kompleksitas Algoritma Quick Sort*

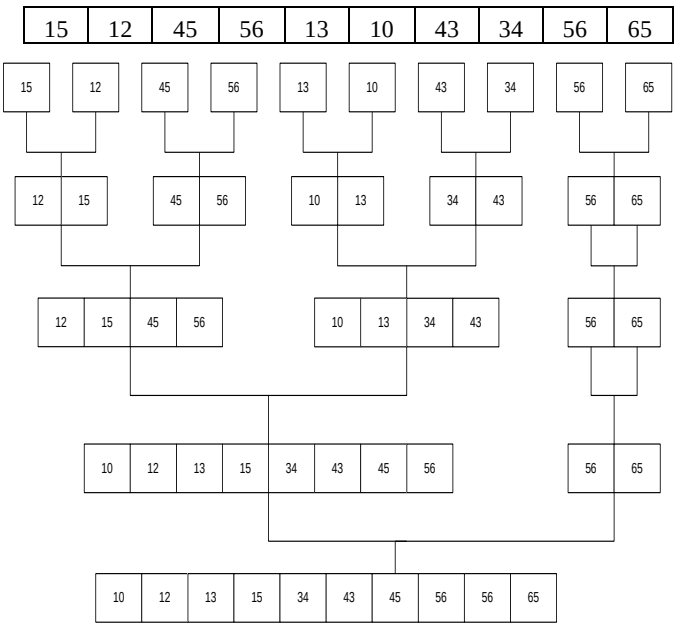
Terdapat 3 jenis kompleksitas waktu dari *Quick Sort*:

1. Kasus terburuk (*worst case*), yaitu terjadi bila terbentuk partisi dengan komposisi sub-masalah antara n – 1 elemen dan 0 elemen. Dengan demikian pemanggilan fungsi secara rekursif dengan *array* berukuran 0 akan langsung kembali, T(0) = Θ(1), sehingga berlaku :
$$T(n) = T(n) + cn = O(n^2)$$
2. Kasus terbaik (*best case*), yaitu terjadi bila terbentuk partisi dengan dengan komposisi seimbang, dengan ukuran masing-masing tidak lebih dari n/2. Sehingga didapat :
$$T(n) = 2T\left(\frac{n}{2}\right) + cn = na + cn\log n = O(n\log n)$$
3. Kasus rata-rata (*average case*), yaitu terjadi dari perimbangan pivot antara terbaik dan terburuk, yang dalam prakteknya lebih mendekati kasus terbaik ketimbang terburuk. Sehingga didapat :
$$T_{avg}(n) = O(n\log n)$$

IV. METODE MERGE SORT

Metode *Merge Sort* yang akan dijelaskan di bawah ini juga sering disebut dengan metode *Merge Sort* Dua Arah (*Two-Way Merge Sort*). Metode ini memanfaatkan keturunan yang diperoleh dari hasil *merge* dua buah vektor.

Secara singkat m*erge sort* ini bisa dijelaskan sebagai berikut. Vektor masukan dianggap yang mempunyai N elemen, dianggap N buah vektor yang masing-masing terdiri dari sebuah vektor. Untuk setiap pasang vektor lakukan proses *merge* sehingga akan memperoleh N/2 vektor yang masing-masing terdiri dari 2 elemen (jika N ganjil, akan terdapat sebuah vektor dengan 1 elemen). Kemudian dilakukan *merge* kembali untuk setiap pasang vektor, sehinga memperoleh N/4 buah vektor. Langkah ini diteruskan sampai memperoleh sebuah vektor yang sudah dalam keadaan terurut.



Pada prosedur yang akan disajikan di bawah ini, terdapat dua buah prosedur bantu. Prosedur pertama adalah prosedur *merge* yaitu prosedur yang digunakan untuk melakukan *merge* terhadap dua buah vektor dengan panjang tertentu.

Merge sort menggabungkan dua ide utama untuk meningkatkan *runtime*-nya:

1. *Array* kecil akan mengambil langkah-langkah untuk menyortir lebih sedikit dari *array* besar.
2. Lebih sedikit langkah yang diperlukan untuk membangun sebuah *array* terurut dari dua buah *array* terurut daripada dari dua buah *array* tak terurut.

A. *Algoritma Merge Sort*

Langkah 0 Baca elemen vektor yang akan diurutkan, misalnya vektor v, dan cacah elemen N.

Langkah 1 (menbentuk senarai berantai)
Untuk I=1 sampai N, tentukan :
Senarai[I].Info = v[I], dan
Senarai[I].kanan = I+1.
Kemudian, tentukan : senarai[N].kanan = 0.

Langkah 2 Tentukan : awal = 1 (awal senarai berantai).

Langkah 3 Kerjakan langkah sampai 16 untuk K = 1 sampai angka (angka adalah cacah digit terbanyak dari elemen vektor yang akan diurutkan).

Langkah 4 (Inisialisasi antrian).
Untuk I = 0 sampai 9, tentukan : Belakang [I] = 0.
Untuk I = 0 sampai 10, tentukan : Depan [I] = 0.

Langkah 5 (Memecahkan senarai berantai menjadi sejumlah antrian)
Kerjakan langkah 6 sampai 9 selama awal <> 0.

Langkah 6 Tentukan : P = awal,
Awal = senarai [Awal].Kanan, dan
Y = senarai [P].Info.

Langkah 7 (Menentukan digit satuan atau puluhan atau ratusan atau ribuan dan seterusnya)
Tentukan : Pangkat = 1.
Untuk I = 1 sampai K-1 :
Tentukan : Pangkat = Pangkat * 10.
Kemudian, tentukan : J = (Y div Pangkat) mod 10
(nilai digit satuan atau puluhan atau ratusan atau ribuan dan seterusnya).

Langkah 8 (Menempatkan dalam antrian)
Tentukan : Q = Belakang [J].
Test apakah : Q = 0 ?
Jika ya, tentukan : Depan [J] = P.
Jika tidak, tentukan : senarai [Q].Kanan = P.

Langkah 9 Tentukan : Belakang [J] = P.

Langkah 10 (Menyusun kembali antrian menjadi senarai berantai)
Tentukan : J = 0
Tentukan : J + 1 selama (J <= 9) dan (Depan [J]) = 0.

Langkah 11 Tentukan : Awal = Depan [J], (awal antrian).

Langkah 12 Kerjakan langkah 13 sampai 15 selama J <= 9.

Langkah 13 Tentukan : I = J + 1
Tentukan : I = I + 1 selama (I <= 9) dan (Depan [I] = 0).

Langkah 14 Test apakah I <= 9 ?
Jika ya, tentukan : P = I, dan
Senarai [Belakang [J]].Kanan = Depan [I].

Langkah 15 Tentukan : J = I.

Langkah 16 (Akhir senarai berantai)
Tentukan : senarai [Belakang [p]].Kanan = 0.

Langkah 17 (Mengambil bentuk senarai berantai menjadi vektor)
Untuk I = 1 sampai N, kerjakan :
V [I] = senarai [Awal].Info, dan
Awal = senarai [Awal].Kanan.

Langkah 18 selesai.

B. Kompleksitas Algoritma Merge Sort

Merge Sort merupakan algoritma pengurutan yang baik terutama untuk mengurutkan data yang jumlahnya sangat banyak. Untuk data yang sedikit, algoritma ini sebaiknya tidak digunakan karena ada beberapa algoritma lain yang bisa bekerja lebih cepat dari *Merge Sort*. Algoritma ini mengaplikasikan pembagian elemen array menjadi 2 buah *sub-array*. Lalu menggabungkan data yang ada pada kedua buah *sub-array* tersebut. Kompleksitas waktu untuk semua kasus dari algoritma *Merge Sort* adalah *O(n log n)*. Untuk menghitung kompleksitas dari algoritma *Merge Sort* bisa menerapkan pembagian *General* dan teorema *Conquer*.

Diketahui bahwa:

$$T(n) = aT\left(\frac{n}{b}\right) + O(n^k)$$

Dimana:

n : merupakan jumlah masalah utama
a : merupakan jumlah dari sub-masalah
n/b : merupakan jumlah pembagian data
$O(n^k)$: kompleksitas dari pembagian dan operasi *merging*

$$T(n) = O(n^{logba}) \quad \text{jika } a > b^k$$
$$T(n) = O(n^k \log n) \quad \text{jika } a = b^k$$
$$T(n) = O(n^k) \quad \text{jika } a < b^k$$

Pada *Merge Sort*, masalah utama dibagi kedalam 2 sub-masalah, jadi a = 2. Jumlah dari masalah utama dibagi dengan 2, jadi b = 2. Kompleksitas untuk pembagian adalah *O (1)* dan kompleksitas waktu untuk *merging* adalah *O (n)*. Total waktu untuk pembagian dan *merging* adalah *O (1) + O (n) = O (n) = O (n1)*. Jadi, k = 1. Karena $a = b^k$ maka kompleksitas dari *Merge Sort* adalah *O (nklog n) = O (n log n)*.

Kasus terbaik (*Best Case*) untuk algoritma *Merge Sort* ini yaitu pada saat data inputan berupa data yang terurut baik *ascending* maupun *descending*. Sedangkan kasus terburuknya (*Worst Case*) yaitu pada saat data inputan berupa data yang acak (*random*). Tetapi untuk kompleksitas waktu asimptotiknya sama, yaitu sebesar *O (n2log n)*.

V. ANALISA ALGORITMA

A. *Metode Quick Sort*

Metode *Quick Sort* memiliki Algoritma yang lebih rumit, tetapi hasilnya lebih cepat karena hanya memerlukan sejumlah langkah yang lebih sedikit untuk menghasilkan hal yang sama. Proses pengurutan data dengan menggunakan metode *Quick Sort* secara garis besar dapat dijelaskan sebagai berikut. Apabila diketahui sebuah vektor K dengan N buah elemen data yang akan diurutkan secara urut naik. Pertama-tama dipilih salah satu elemen data sembarang pada vektor K tersebut, biasanya adalah elemen apada urutan data pertama dan diberi nama X. Selanjutnya, semua elemen pada vektor K disusun dengan menempatkan elemen data X pada posisi data tertentu yaitu J sedemikian rupa sehingga elemen data pada urutan ke-1 samapai ke j-1 mempunyai harga lebih besar dari X. Sehinggga akan terdapat dua subvektor, yaitu subvektor bagian pertama yang memuat elemen-elemen data yang memiliki harga kurang dari X dan subvektor bagaian kedua yang memuat elemen-elemen data yang memiliki harga lebih dari X.

Prosedur di atas akan diulang pada subvektor pertama dan subvektor kedua, sehingga akan diperoleh emapat subvektor baru. Proses yang sama akan terus dilakukan secara berulang sehingga semua subvektor pada akhirnya hanya tinggal mempunyai satu elemen data. Dalam kondisi demikian, maka semua data dalam vektor K telah menjadi urut naik.

B. *Metode Merge Sort*

Metode *Merge Sort* (penggabungan) melakukan pengurutan dengan cara membagi data menjadi 2 (dua) kumpulan data dan masing-masing kumpulan diurutkan, kemudian setelah terurut dilakukan penggabungan dua kumpulan data tersebut dalam keadaan terurut.

Langkah-langkah pengurutan dengan *Merge Sort* secara garis besar dapat dijelaskan sebagai berikut ini. Sebuah vektor K dengan cacah elemen data sebanyak N pada kondisi tidak terurut. Untuk mengurutkan semua data dalam K, mula-mula setiap elemen dalam vektor K dianggap sebagai sebuah vektor yang masing-masing mempunyai sebuah elemen data. Dengan demikian akan terdapat N vektor dengan cacah elemen masing-masing adalah 1 buah. Selanjutnya, setiap pasang vektor yang berurutan digabungkan menjadi sebuah vektor yang baru. Jika menginginkan hasil secara urut naik, maka pada saat menggabungkan kedua vektor tersebut sekaligus dilakukan perbandingan dan pertukaran posisi antar elemen data sedemikian rupa sehingga data yang lebih kecil akan ditempatkan pada posisi yang mendahului data lain yang lebih besar. Pada akhir langkah pertama ini, akan memiliki vektor baru sebanyak N/2 yang masing-masing dalam kondisi urut naik. Cacah elemen pada masing-masing vektor adalah 2 buah, kecuali jika N bernilai ganjil maka akan terdapat N/2+1 vektor baru dimana salah satu vektor hanya memiliki sebuah elemen data saja.

Pada langkah selanjutnya, setiap pasang vektor yang berurutan dapat dilakukan penggabungan dan sekaligus dapat pertukaran posisi antar data sehingga akan terbentuk vektor-vektor baru. Demikian, proses penggabungan dan pertukaran posisi data secara terus menerus akan dilakukan sehinggga pada akhirnya akan diperoleh sebuah vektor baru yang memuat semua elemen data dalam vektor sumber dalam kondisi urut naik.

C. *Analisa Perbandingan Algoritma*

Pada algoritma *Quick Sort*, jarak dari kedua elemen yang ditukarkan dibuat cukup besar dengan tujuan untuk mempertinggi efektivitasnya. Sedangkan algoritma *Merge Sort* akan selalu membagi dua tiap sub-*array*nya hingga mencapai basis, sehingga kompleksitas dari algoritma *Merge Sort*, berlaku untuk semua kasus (*Worst Case = Best Case = Average Case*). Algoritma *Quick Sort* dan *Merge Sort* memiliki kompleksitas yang sama yaitu *O (n log n)* tetapi pada kasus *worst case* algoritma *Quick Sort* memiliki kompleksitas O (n^2). Metode *Quick Sort* dan metode *Merge Sort* dapat dilakukan dengan menggunakan rekursif atau tanpa rekursif.

Algoritma *Quick Sort* lebih dipilih dibanding algoritma *Merge Sort*, karena algoritma *Merge Sort* melakukan tiga kali lebih banyak *assignment records* dibanding algoritma *Quick Sort* secara rata-ratanya. Walaupun algoritma *Quick Sort* melakukan pembandingan elemen yang lebih banyak dalam kasus rata-ratanya.

Untuk mengetahui analisa perbandingan untuk O (n log n) algoritma pengurutan *Merge Sort* dan *Quick Sort*, dapat kita lihat seperti tabel 3.3 berikut ini :

Tabel 1. Analisa Rangkuman untuk O (n log n) Algoritma Pengurutan Quick Sort dan Merge Sort

Algoritma	**Pembandingan Elemen**
Merge Sort	$W(n) = n \log n$
	$A(n) = n \log n$
Quick Sort	$W(n) = n^2/2$
	$A(n) = 1.38n \log n$
Algoritma	***Assignment Records***
Merge Sort	$T(n) = 2n \log n$
Quick Sort	$A(n) = 0.69n \log n$

VI. PENUTUP

Dari hasil analisis pengurutan data (*Sorting*) menggunakan algoritma *Quick Sort dan Merge Sort* ini, dapat diambil kesimpulan sebagai berikut :

1. Algoritma *Merge Sort* memiliki waktu yang lebih cepat pada jumlah data yang relatif sedikit

(berkisar 1 s/d 10 data), baik pengurutan data pada data *Array* maupun data *Record*.

2. Algoritma *Quick Sort* selalu memiliki jumlah langkah yang sedikit dibandingkan algoritma *Merge Sort* yang memiliki jumlah langkah yang lebih banyak.
3. Dari hasil perbandingan pengurutan data ini, *Quick Sort* dipilih lebih baik dan lebih cepat karena *Assignment Record*snya yang paling sedikit diantara kedua algoritma pengurutan data tersebut, meskipun pembandingan elemen yang dilakukan *Quick Sort* lebih banyak dibandingkan *Merge Sort*.

DAFTAR PUSTAKA

1. Sumantri Slamet I. S, FX. Nursalmi, C. Hendrik, Wahyu, Wibowo, 1989, ***Pengantar Struktur Data***, Penerbit PT. Elex Media Kamputindo, Jakarta.
2. Kusumo Suryo Ario Drs, 2000, ***Microsoft Visual Basic 6***, Penerbit PT. Elex Media Komputindo, Jakarta.
3. Liem, Inggriani, 2007, ***Draft Diktat Kuliah Dasar Pemrograman (Bagian Pemrograman Prosedural)***, Program Studi Teknik Informatika, Institut Teknologi Bandung. hlm. 141-142
4. Munir, Rinaldi, 2006, ***Strategi Algoritmik,*** Jurnal Teknik Informatika Bandung.
5. Jogiyanto MH, Akt, MBA, Ph.D, 2009, ***Bahasa Pascal***, Penerbit Andi Offset, Yogyakarta.