Komparasi Algoritma Mergesort dengan Quicksort pada Pengurutan Data Integer

Atika Azzahra Akbar 13514077
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13514077@std.stei.itb.ac.id

***Abstrak*—Dalam makalah ini akan dikomparasikan dua buah algoritma pengurutan yaitu algoritma mergesort dan quicksort. Komparasi kedua algoritma dilakukan dengan pertama-tama mencari kompleksitas algoritma waktu dan kompleksitas algoritma asimpotik dari keduanya dan kemudian membandingkan hasilnya. Selain itu dalam makalah ini akan dijabarkan hasil percobaan pengurutan data integer dengan menggunakan kedua algoritma. Kemudian akan dikomparasikan hasil percobaan kedua algoritma dan akan dibuat kesimpulan berdasarkan kedua cara komparasi.**

***Kata Kunci*— Komparasi, Kompleksitas Algoritma, Mergesort, Quicksort.**

I. PENDAHULUAN

Kebutuhan akan algoritma pengurutan pada suatu program sangatlah tinggi. Contohnya pada program perangkat lunak utnuk pengolahan data seperti Microsoft Excel, Microsoft Access atau Oracle. Dimana pada kehidupan nyata, algoritma ini digunakan tidak hanya untuk mengurut sepuluh sampai seratus data, namun bisa sampai jutaan data. Jika orang yang memrogram program pengolah jutaan data tersebut menggunakan algortima yang tidak mangkus, bisa-bisa dibutuhkan waktu berhari-hari untuk mengurutkan data. Maka dari itu seorang pemrogram harus mengerti tentang kefektifan semua algoritma agar dapat dibuat program yang efektif sesuai besar data dan kebutuhan penggunannya.

Ada banyak sekali jenis algoritma pengurutan contohnya Algoritma *insertion sort, selection sort, buble sort, shell sort, mergesort, heapsort,* dan *quicksort*. Setiap jenis algoritma pengurutan tersebut memiliki kefektifan dalam masing-masing kondisi. Oleh karena itu digunakan sebuah besar pengukuran untuk mengukur kefektifan sebuah algoritma, yang dalam hal disebut sebagai kompleksitas algoritma. Terdapat dua macam besar pengukuran yaitu kompleksitas waktu yang diukur berdasarkan jumlah tahapan komputasi yang diperlukan suatu algoritma bila dimasukan sebnayak n data dan kompleksitas ruang yang dihitung melalui memori yang digunakan oleh suatu algoritma dari ukuran masukan sebanyak n.

Dewasa ini dari sekian banyak algoritma pengurutan, terdapat beberapa algoritma yang lebih menonjol dibandingkan algoritma lainnya dikarenakan kompleksitas algoritmanya yang kecil sehingga sangat efisien bila digunakan, yaitu Mergesort dan Quicksort. Dari kedua algoritma yang sudah terkenal mangkus ini akan dipilih lagi algoritma yang paling mangkus. Pada makalah ini akan dikomparasikan algoritma Mergesort dan Quicksort pada pengurutan data bilangan bulat dengan kompleksitas waktu, kompleksitas waktu Asimpotik, dan waktu hasil percobaan.

II. TEORI DASAR

A. Kompleksitas Algoritma

Seperti yang telah dijelaskan pada bagian satu pendahuluan, kompleksitas algortima merupakan sebuah besaran abstrak yang diciptakan untuk kebutuhan pengukuran sebuah algoritma. Terdapat dua macam tipe pengukuran yaitu kompleksitas ruang dan kompleksitas waktu. Untuk memudahkan dalam membahas kompleksitas sebuah algoritma dibutuhkan beberapa terminologi. Seperti yang dikutip dari referensi [1] terdapat tiga terminologi yaitu:

1. Ukuran besar masukan data untuk suatu algoritma n.
2. Kompleksita waktu, T(n), adalah waktu yang dibutuhkan untuk melaksanakan algoritma (dari awal sampai akhir) sebagai fungsi dari ukuran masukan n.
3. Kompleksitas ruang, S(n), adalah ruang memori yang dibutuhkan algoritma sebagai fungsi dari ukuran masukan n.

B. Kompleksitas Waktu

Pada makalah ini hanya akan dilakukan komparasi menggunakan kompleksitas waktu, maka dari itu pada subab ini hanya akan dijelaskan tentang kompleksitas waktu.

Kompleksitas waktu diukur dari jumlah tahapan komputasi yang dibutuhkan untuk menjalankan algoritma sebagai fungsi dari ukuran masukan n. Untuk itu, pertama-tama kita harus memahami algoritma yang dihadapi. Idealnya, kita harus menghitung semua opertasi yang ada di dalam suatu algoritma, namun untuk alas an

praktis, biasanya yang dihitung adalah jumlah operasi abstrak yang mendasari suaut algoritma, dan memisahkan analisisnya dari implementasi [1]. Terdapat tiga jenis kompleksitas waktu yaitu:

1. $T_{max}(n)$: kompleksitas waktu untuk kasus terburuk.
2. $T_{min}(n)$: kompleksitas waktu untuk kasus terbaik.
3. $T_{avg}(n)$: kompleksitas waktu untuk kasus rata-rata dengan terlebih dahulu membuat asumsi bahwa tiap data memiliki peluang yang sama untuk terletak dalam larik.

C. Kompleksitas Waktu Asimpotik

Untuk mempermudah perbandingan antara sebuah algoritma dengan algoritma lainnya kita dapat melihat bagaimana waktu terbaik dan waktu terburuk berkembang bersama seiring meningkatnya besar data masukan. Oleh karena itu, digunakan notasi kompleksitas waktu asimpotik yang disebut notasi-O Besar dengan definisi formal dibawah ini seperti dikutip dari referensi [1].

T(n) = O(f(n)) (dibaca "T(n) adalah O(f(n))" yang artinya T(n) berorde paling besar f(n)) bila terdapat konstanta C dan n_o sedemikian sehingga

$$T(n) \leq C(f(n))$$

untuk $n \geq n_o$

Dengan adanya O(f(n)) ini, jika algoritma dimasukan n semakin besar, waktu yang diperlukan algoritma tersebut tidak akan pernah melebihi suatu konstanta C dikali dengan f(n). Dapat kita simpulkan bahwa f(n) adalah batas atas (upper bound).

Kompleksitas algoritma asimpotik memiliki beberapa teorema, seperti dikutip dari [1]:

Teorema 1

Bila $T(n) = a_m n^m + a_{m-1} n^{m-1} + \ldots + a_1 n + a_o$ adalah polinom berderajat m maka $T(n) = O(n^m)$

Teorema 2

Misalkan $T_1(n) = O(f(n))$ dan $T_2(n) = O(g(n))$ maka

(a) $T_1(n)+T_2(n) = O(f(n))+O(g(n)) = O(\max(f(n), g(n))$
(b) $T_1(n)\, T_2(n) = O(f(n))O(g(n)) = O(f(n), g(n))$
(c) $O(c(f(n)) = O(f(n))$ sebab c konstanta
(d) $f(n) = O(f(n))$

Selain notasi-O Besar, terdapat notasi lain yang digunakan untuk menyatakan kompleksitas waktu asimpotik yaitu notasi omega-besar dan notasi tetha-besar.

Definisi notasi omega-besar [1]:

T(n) = Ω(g(n)) (dibaca "T(n) adalah Ω (g(n))" yang artinya T(n) berorde paling kecil g(n)) bila terdapat konstanta C dan n_o sedemikian sehingga:

$$T(n) \geq C(g(n))$$

untuk $n \geq n_o$

Definisi notasi tetha-besar [1]:

T(n) = Θ(h(n)) (dibaca "T(n) adalah Ω (h(n))" yang artinya T(n) berorde sama dengan h(n) jika T(n)=O(f(n)) dan T(n) = Ω(g(n))

D. Mergesort

Mergesort merupakan algoritma pengurutan yang didasari oleh paradigm divide-and-conquer. Algoritma ini memiliki kompleksitas waktu untuk kasus terburuk yang pertumbuhannya lebih rendah dibandingkan insertion sort [2].

Paradigma, divide-and-conquer, membagi masalah kedalam sub-masalah, dan kemudian menggambungkan lagi solusi dari semua sub-masalah tersebut. Dikarenakan divide-and-conquer menyelesaikan masalah secara rekursif, setiap subproblem harus lebih kecil dari masalah originalnya dan harus terdapat basis untuk setiap sub-masalah [3]. Algoritma divide-and-conquer dapat dibagi menjadi tiga:

1. Divide: membagi masalah ke dalam sub-masalah.
2. Conquer: menyelesaikan sub-masalah dengan metode rekursif.
3. Combine: menggambungkan hasil solusi dari setiap sub-masalah dengan prosedur bernama merge(A, p, q, r)

Algoritma dari mergesort kurang lebih dalam

```
Mergesort(tab[], l,  r)
if r > 1
    1. Temukan point tengah untuk membagi larik
       menjadi dua bagian
       middle = (l+r)/2
    2. Panggil Mergesort untuk bagian setengah
       pertama
       Mergesort(tab, l, m)
    3. Panggil Mergesort untuk bagian setengah
       kedua
       Mergesort(tab, m+1, r)
    4. Gabungkan kedua larik yang telah diurut pada
       poin 2 dan 3
       Merge(Tab, l, m, r)
```

implementasi bahasa C seperti dibawah ini [4]:

E. Quicksort

Mirip seperti Mergesort, quicksort menggunakan paradigm algoritma divide-and-conquer juga. Namun penerapan algoritma divide-and-conquer pada quicksort agak berbeda dengan mergesort. Pada mergesort semua pekerjaan dilimpahkan di fungsi penggambungan sedangkan di quicksort fungsi penggabunggan hampir tidak melakukan apa-apa. Algoritma divide-and-conquer pada quicksort dapat dibagi menjadi tiga [5]:

1. Divide: memilih salah satu elemen pada sublarik untuk menjadi pivot. Kemudian elemen-elemen lainnya diatur ulang sehingga elemen yang bernilai lebih kecil atau sama dengan pivot berada di kiri dan sisanya berada di kanan elemen pivot.

2. Conquer: secara rekursif mengatur sublarik.
3. Combine: tidak dilakukan pekerjaan yang spesifik karena semua sublarik telah teratur.

Algoritma dari quicksort kurang lebih dalam implementasi bahasa C seperti dibawah ini [6]:

```
Quicksort
if First > Last
        1. Mempartisi elemen di sublarik
        2. Mengaplikasikan Quicksort lagi secara
           rekursif kepada sublarik pertama
        3. Mengaplikasikan Quicksort lagi secara
           rekursif kepada sublarik pertama
End

Kasus berhenti jika:
    1. (First = Last) hanya ada satu nilai pada sublarik
    2. (First > Last) tidak ada elemen yang harus
       diurutkan
```

IV. PEMBAHASAN DAN PENGOLAHAN DATA

A. Kompleksitas Algoritma Mergesort dan Quicksort

Seperti yang telah dijelaskan di bagian pendahuluan bahwa makalah ini hanya akan melakukan komparasi menggunakan kompleksitas waktu, maka pada subab ini hanya akan dicari besar dari kompleksitas waktu algoritma mergesort dan quicksort.

Untuk mendapatkan kompleksitas waktu dari algoritma mergesort, pertama-tama kita harus menganalisi dari algoritma divide-and-conquer:

1. Divide: membagi masalah ke a submasalah dengan ukuran tiap submasalahnya sebesar n/b, maka waktu yang dibutuhkan adalah D(n)
2. Conquer: menyelesaikan submasalah aT(n/b)
3. Combine: menggabungkan solusi membutuhkan waktu sebesar C(n).

Maka didapat:

$$T(n)\begin{cases}\Theta(1) & \text{jika } n \le c \\ aT(n/b) + D(n) + C(n) & \text{selain itu}\end{cases}$$

Pada algoritma mergesort dapat disimpulkan:

1. Divide: hanya terjadi satu kali pembagian dari satu masalah menjadi dua submasalah makan dinali D(n) = 1.
2. Conquer: secara rekursif menyelesaikan dua submasalah dengan tiap sub memiliki ukuran sebesar n/2 maka diperlukan waktu sebesar 2T(n/2).
3. Combine: menggambungkan solusi yang memiliki n sublarik membutuhkan waktu sebesar C(n) = n.

Maka didapat:

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

$T_{min}(n)$ dan $T_{max}(n)$ sama untuk algoritma mergesort karena pada kondisi terbaik walau sudah terurut masalah tetap dibagi menjadi dua begitu juga pada kondisi terburuk.

Kompleksitas waktu asimpotik untuk algoritma mergesort didapat dengan menyelesaikan kerekursifan pada di kompleksitas waktunya.

$T(n) = 2T(n/2) + n$

$T(2)= 2(2T(n/4) + n/2) + n$
$= 2^2T(n/4) + 2n$

$T(3)= 2^2(2T(n/8) + n/4) + 2n$
$= 2^3T(n/8) + 3n$

.
.

$T(n) = 2^kT(n/2^k) + kn$

Kemudian dengan mensubtitusi k dengan nilai dibawah ini:

$n/2^k = 1$
$n = 2^k$
$\log n = k$

Maka persamaan kompleksitas algoritma mergesort menjadi:

$T(n) = nT(1) + n \log n$
$= n + n \log n$

Dapat diambil kesimpulan bahwa kompleksitas algoritma asimpotik untuk algoritma mergesort adalah [7]:

$T(n) = n + n \log n = O(n \log n) = \Omega(n \log n) = \Theta(n \log n)$

Sedangkan untuk algoritma pengurutan quicksort nilai kompleksitas waktunya tergantung pada posisi pivot yang dipilih untuk membagi masalah menjadi submasalah yang dalam algoritma ini pembagian dua submasalah didasari dari besar nilai elemen dibandingkan dengan nilai elemen pivot. Selain itu total waktu untuk menggabungkan solusi dari submasalah-submasalah selalu αn dimana α adalah konstan.

Misalkan partisi pada quicksort menghasilkan dua submalasah dengan ukuran k dan (n-k). Maka nilai kompleksitas waktunya dapat kita tulis seperti dibawah ini [8]:

$$T(n)\begin{cases}1 & \text{jika } n = 1 \\ T(n) = T(k) + T(n-k) + \alpha n & \text{jika } n > 1\end{cases}$$

Karena nilai dari k tergantung pada kondisi yang didapat, maka kondisi akan dibagi menjadi dua yaitu kondisi terburuk dan kondisi terbaik.

Pertama akan dicari T(n) pada kondisi terburuk. Kondisi terburuk pada algoritma pengurutan quicksort didapat ketika pivot elemen yang dipilih adalah elemen terakhir yang harus diurutkan dari larik sehingga kita akan mendapatkan dua submasalah dengan ukuran elemen sebesar k=1 dan (n-1). Dengan begitu kita memiliki kompleksitas waktu seperti dibawah ini:

$$T(n) = T(1) + T(n-1) + \alpha n$$

Kemudian nilai kompleksitas waktu diatas kita selesaikan rekursif-nya menjadi:

$T(n) = T(n - 1) + T(1) + \alpha n$
$= [T(n – 2) + T(1) + \alpha(n – 1)] + T(1) + \alpha n$
$= T(n – 2) + 2T(1) + \alpha(n – 1 + n)$
$= [T(n – 3) + T(1) + \alpha(n – 2)] + 2T(1) + \alpha(n – 1 + n)$
$= T(n – 3) + 3T(1) + \alpha(n – 2 + n – 1 + n)$

.
.
$= T(n-i) + iT(1) + \alpha(\sum_{j=1}^{i}(n \quad j))$

Namun karena kompleksitas waktu diatas hanya dapat disubstitusi oleh i = n – 1. Hal ini dikarenakan n – i dapat menghasilkan nilai negatif), maka nilai i harus disubtitusi, menjadi:

$T(n) = T(1) + (n-1)T(1) + \alpha(\sum_{j=1}^{i}(n \quad j))$

$= nT(1) + \alpha(n(n-2) - (n-2)(n-1)/2)$

Dari nilai kompleksitas waktu algoritma quicksort kondisi terburuk, kemudian akan dicari nilai kompleksitas algoritma asimpotiknya dengan menyelesaikan kerekursifan pada kompleksitas waktunya.

$T(n) = n + \alpha((n^2 - 2n) - ((n^2 - 3n + 2)/2)) = O(n^2)$

Kedua akan dicari nilai kompleksitas waktu algoritma quicksort untuk kondisi terbaik. Kondisi terbaik algoritma quicksort didapatkan ketika pivot elemen yang diambil adalah sebuah nilai yang membagi dua masalah menjadi dua submasalah dengan ukuran yang sama yaitu k = n/2 dan (n – k) = n/2. Maka nilai kompleksitas waktu dari algoritma quicksort menjadi:

$T(n) = 2T(n/2) + \alpha n$

Kemudian nilai kompleksitas waktu diatas kita selesaikan rekursif-nya menjadi [8]:

$T(n) = 2T(n/2) + \alpha n$

$T(2) = 2(2T(n/4) + \alpha n/2) + \alpha n$

$= 2^2T(n/4) + 2\alpha n$

$T(3) = 2^2(2T(n/8) + \alpha n/4) + 2\alpha n$

$= 2^3T(n/8) + 3\alpha n$

.
.

$T(n) = 2^kT(n/2^k) + k\alpha n$

Dari nilai kompleksitas waktu algoritma quicksort kondisi terbaik, kemudian akan dicari nilai kompleksitas algoritma asimpotiknya dengan menyelesaikan kerekursifan pada kompleksitas waktunya dengan mensubtitusikan nilai k dengan:

$n/2^k = 1$

$n = 2^k$

$\log n = k$

$T(n) = nT(1) + \alpha n \log n = O(\alpha n \log n) = O(n \log n)$

B. Pengaplikasian Algoritma Mergesort dan Quicksort pada Pengurutan Data Integer dengan Bahasa C

Pada subab ini akan dilakukan percobaan pengurutan data bilangan bulat dengan dua algoritma yaitu mergesort dan quicksort. Akan dihitung waktu yang diperlukan untuk tiap algoritma mengurutkan data tersebut. Bahasa yang digunakan untuk percobaan ini adalah bahasa C.

Kode algoritma Mergesort untuk pengurutan data integer adalah sebagai berikut:

```
#include<stdlib.h>
#include<stdio.h>
#include "time.h"

void merge(int arr[], int l, int m, int r)
{
    int i, j, k;
    int n1 = m - l + 1;
    int n2 =  r - m;

    int L[n1], R[n2];

    for(i = 0; i < n1; i++)
        L[i] = arr[l + i];
    for(j = 0; j < n2; j++)
        R[j] = arr[m + 1+ j];

    i = 0;
    j = 0;
    k = l;
    while (i < n1 && j < n2)
    {
        if (L[i] <= R[j])
        {
            arr[k] = L[i];
            i++;
        }
        else
        {
            arr[k] = R[j];
            j++;
        }
        k++;
    }

    while (i < n1)
    {
        arr[k] = L[i];
        i++;
        k++;
    }

    while (j < n2)
    {
        arr[k] = R[j];
        j++;
        k++;
    }
}
void mergeSort(int arr[], int l, int r)
{
    if (l < r)
    {
        int m = l+(r-l)/2;
        mergeSort(arr, l, m);
        mergeSort(arr, m+1, r);
        merge(arr, l, m, r);
    }
}
void printTab(int Tab[], int size)
{
    int i;
    for (i=0; i < size; i++)
        printf("%d ", Tab[i]);
    printf("\n");
}

int main()
{
    clock_t start, finish;
    int arr[10] = {5, 3, 1, 2, 4, 8, 6, 7, 9,
10};
    int arr_size = sizeof(arr)/sizeof(arr[0]);
    start = clock();
    mergeSort(arr, 0, arr_size - 1);
    finish = clock();
    printf("Sorted array by MergeSort is \n");
    printTab(arr, arr_size);
    printf("The time taken is %lu millisecond",
1000000*(finish - start)/CLOCKS_PER_SEC);
    return 0;
 }
```

Kode algoritma quicksort untuk pengurutan data integer adalah sebagai berikut:

```c
#include<stdio.h>
#include "time.h"

void swap(int *a, int *b)
{
    int t = *a;
    *a = *b;
    *b = t;
}
int divide(int tab[], int l, int h)
{
    int x = tab[h];
    int i = (l-1);

    for (int j = l; j<= h - 1; j++)
    {
        if (tab[j] <= x)
        {
            i++;
            swap(&tab[i], &tab[j]);
        }
    }
    swap(&tab[i+1], &tab[h]);
    return (i+1);
}
void Quicksort(int tab[], int l, int h)
{
    if (l < h)
    {
        int d = divide(tab, l, h);
        Quicksort(tab, l, d - 1);
        Quicksort(tab, d + 1, h);
    }
}
void printTab(int tab[], int size)
{
    int i;
    for (i=0; i<size; i++)
    {
        printf("%d ", tab[i]);
    }
    printf("\n");

}
int main (){
    clock_t start, finish;
    int arr[10] = {5, 3, 1, 2, 4, 8, 6, 7, 9,
10};
    int arr_size = sizeof(arr)/sizeof(arr[0]);
    start = clock();
    Quicksort(arr, 0, arr_size-1);
    finish = clock();
    printf("Sorted array by QuickSort: \n");
    printTab(arr, arr_size);
    printf("The time taken is %lu millisecond",
1000000*(finish - start)/CLOCKS_PER_SEC);
    return 0;
}
```

Nilai larik pada kedua kode diatas akan diubah sesuai dengan data percobaan yang telah disiapkan oleh penulis makalah. Selain itu perhitungan waktu yang digunakan oleh setiap algoritma diukur dengan menggunakan library yang telah terdapat pada c yaitu time.h. Dimana pada program ini diimplementasikan pada variable start dan finish yang kemudian diselisihkan untuk mendapatkan nilai waktu dalam satuan millisekon.

Berikut dibawah ini adalah data-data yang penulis siapan untuk percobaan:

Data 1
int arr[60] =
{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60}

Data 2
int arr[10] = {5, 3, 1, 2, 4, 8, 6, 7, 9, 10}

Data 3
int arr[100] =
{21, 4, 3, 45, 5, 78, 7, 8, 9, 10, 11, 12, 53, 34, 15, 16, 29, 18, 19, 21, 20, 22, 35, 33, 25, 26, 27, 8, 29, 30, 31, 2, 33, 34, 35, 100, 7, 38, 9, 90, 41, 42, 43, 4, 45, 6, 77, 88, 49, 70, 51, 52, 53, 54, 55, 56, 7, 5, 59, 60, 5, 6, 8, 20, 21, 22, 34, 45, 56, 30, 23, 12, 23, 34, 24, 5, 6, 75, 4, 5, 5, 6, 8, 20, 21, 22, 34, 45, 56, 30, 23, 12, 23, 34, 24, 5, 6, 75, 4, 5}

Dengan memasukan data-data diatas menjadi input untuk program pengurut menggunakan algoritma mergesort dan quicksort didapatkan waktu untuk tiap algoritma seperti tabel dibawah ini:

Data	*Mergesort (millisecond)*	*Quicksort (millisecond)*
1	10	24
2	4	3
3	21	24

Ketiga data tersebut telah penulis buat untuk membuktikan setiap kondisi yang mungkin terjadi pada pengurutan mergesort dan quicksort.

Data pertama merupakan data yang akan menyebabkan terjadinya kondisi terburuk untuk quicksort. Sedangkan data kedua merupakan data yang akan menyebabkan kondisi terbaik untuk quicksort dan data ketiga merupakan data acak. Penulis tidak membuat data untuk kondisi terbaik dan terburuk algoritma mergesort karena nilai kompleksitas algoritma terburuk dan terbaik dari mergesort adalah sama. Sehingga semua data sudah mencakup semua kondisi untuk mergesort.

C. Komparasi Algortima Mergesort dan Quicksort Menggunakan Kompleksitas Algoritma dan Hasil Percobaan

Berdasarkan subbab sebelumnya telah dihitung kompleksitas algoritma untuk algoritma mergesort dan quicksort. Telah ditemukan bahwa untuk algoritma mergesort nilai kompleksitas algoritma kondisi teburuk dan terbaiknya adalah sama yaitu:

$$T(n) = 2T(n/2) + n$$

Dengan nilai kompleksitas algoritma simpotiknya:

$T(n) = n + n \log n = O(n \log n) = \Omega(n \log n) = \Theta(n \log n)$

Kemudian didapatkan pula nilai kompleksitas algoritma untuk algoritma quicksort pada kondisi terburuk dan

kondisi terbaik berikut dengan kompleksitas algoritma asimpotik yaitu:

$$T(n) = T(1) + T(n-1) + \alpha n$$

$$T(n) = n + \alpha((n^2 - 2n) - ((n^2 - 3n + 2)/2)) = O(n^2)$$

(Kondisi terburuk)

$$T(n) = 2T(n/2) + \alpha n$$

$$T(n) = nT(1) + \alpha n \log n = O(\alpha n \log n) = O(n \log n)$$

(Kondisi terbaik)

Dapat dilihat bahwa kompleksitas algoritma mergesort dan quicksort untuk kondisi terbaik sama besarnya sehingga tidak akan terdapat perbedaan begitu besar untuk pengurutan data pada kondisi terbaik. Sedangkan untuk kondisi terburuk, kompleksitas algoritma quicksort memiliki kompleksitas lebih besar sehingga akan terdapat perbedaan yang signifikan jika diperbandingkan antara keduanya.

Kemudian berdasarkan hasil percobaan menggunakan ketiga data didapatkan untuk data pertama bahwa algoritma mergesort memang lebih baik untuk mengurutkan sebuah data yang hampir kebanyakan datanya telah terurut. Sedangkan untuk data kedua, quicksort memperlihat performa yang lebih baik yaitu 3 ms namun tidak terlalu signifikan perbedaannya dengan pengurutan menggunakan algoritma mergesort yang menggunakan waktu sebesar 4 ms untuk mengurutkan data. Setelah itu data terakhir menghasilkan perbedaan yang tidak begitu signifikan antara algoritma mergesort dan quicksort.

V. KESIMPULAN

Maka dapat diambil kesimpulan bahwa untuk data yang sebagian besar dari datanya telah terurut, programer lebih baik memilih algoritma mergesort untuk mengurutkan. Namun secara garis besar, untuk jutaan data acak tidak akan begitu signifikan perbedaannya bila seorang programer menggunakan algoritma mergesort ataupun quicksort.

REFERENCES

[1] Munir, Rinaldi, Matematika Diskrit. Bandung : Penerbit Informatika, Palasari

[2] http://www.personal.kent.edu/~rmuhamma/Algorithms/MyAlgorithms/Sorting/mergeSort.htm, diakses pada tanggal 4 Desember 2015 Pukul 18.00 WIB

[3] https://www.khanacademy.org/computing/computer-science/algorithms/merge-sort/a/divide-and-conquer-algorithms, diakses pada tanggal 4 Desember 2015 Pukul 18.30 WIB

[4] http://geeksquiz.com/merge-sort/, diakses pada tanggal 4 Desember 2015 Pukul 18.45 WIB

[5] https://www.khanacademy.org/computing/computer-science/algorithms/quick-sort/a/overview-of-quicksort, diakses pada tanggal 5 Desember 2015 Pukul 9.00 WIB

[6] http://www.cise.ufl.edu/~ddd/cis3020/summer-97/lectures/lec17/sld001.htm, diakses pada tanggal 5 Desember 2015 Pukul 9.15 WIB

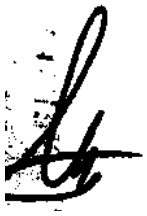
[7] https://courses.cs.washington.edu/courses/cse373/09wi/lectures/cse373-lect22-wanswers.pdf, diakses pada tanggal 5 Desember 2015 Pukul 9.34 WIB

[8] http://www.cise.ufl.edu/class/cot3100fa07/quicksort_analysis.pdf, diakses pada tanggal 5 Desember 2015 Pukul 10.00 WIB

PERNYATAAN

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

Bandung, 8 Desember 2015



ttd

Atika Azzahra Akbar 13514077