TELAAH WAKTU EKSEKUSI PROGRAM TERHADAP KOMPLEKSITAS WAKTU ALGORITMA *BRUTE FORCE* DAN *DIVIDE AND CONQUER* DALAM PENYELESAIAN OPERASI LIST

Andhika Hendra Estrada S.

Sekolah Teknik Elektro dan Informatika
INSTITUT TEKNOLOGI BANDUNG

if15118@students.if.itb.ac.id
andhika_estrada@students.itb.ac.id
yuki_nebula@yahoo.com

ABSTRAK

List atau senarai, sering disebut pula array, adalah suatu tipe data yang mempunyai banyak sekali kegunaan, bisa dijadikan dasar berbagai macam tipe data yang lebih rumit, simplikasi (penyederhanaan) suatu masalah, bahkan bisa merepresentasikan suatu database sederhana. Dalam pemakaiannya tentu saja banyak sekali operasi yang perlu kita lakukan terhadap senarai ini. Diantaranya adalah sorting (pengurutan).

Dalam pengurutan senarai, ternyata mempunyai banyak sekali metode penyelesaian. Bebrapa di antaranya adalah "*bruteforce*" dan "*divide and conquer*". Bruteforce dikenal sebagai algoritma yang sederhana dan langsung pada tujuan, sedangkan *divide and conquer* dikenal sebagai algoritma yang sangat efektif dan hemat waktu. Salah satu contoh algoritma sorting dengan metode *bruteforce* adalah *Bubble Sort,* sedangkan contoh metode *divide and conquer* adalah *MergeSort* dan *QuickSort*.

Dengan menghitung teori kompleksitas waktu, didapatkan bahwa metode *divide and conquer* lebih unggul dalam hal konsumsi waktu. Namun tentu saja teori tidak selamanya betul di segala kondisi dan mungkin bisa di telaah sehingga nantinya programmer bisa memilih metode mana yang paling baik untuk permasalahan dan problem mereka.

Kata kunci: Algoritma, Brute Force, Bubble Sort, Merge Sort, Divide and Conquer, List, Senarai, Kompleksitas waktu.

1. PENDAHULUAN

Bentuk list(array, senarai) merupakan suatu bentuk yang sangat umum di dunia programmer. Bentuk ini nantinya bisa dijadikan suatu bentuk tipe data yang kegunaannya sangatlah luas. Bahkan ada beberapa programmer yang menyatakan bahwa tipe data serumit apapun hanya terbentuk dari dua hal yaitu komposisi atom dan list[1].

Sebanding dengan kegunaanya tentu saja programmer juga harus memiliki kode ,metode dan algoritma yang tepat untuk mengolahnya. Beberapa metode tersebut adalah *bruteforce* dan *divide and conquer (D&Q)*. Dalam pengoperasian list banyak sekali algoritma terkenal yang merepresentasikan kedua metode tersebut. Antara lain yaitu, *BubbleSort* (*bruteforce*), *MergeSort (D&Q),* dan *QuickSort (D&Q)*.

Secara teoritis *D&Q* lebih efisien daripada *bruteforce*, namun dilihat dari potongan kode tampak bahwa bruteforce sangatlah sederhana, cepat dalam proses development, dan mudah melakukan koreksi dan perbaikan jika terjadi error.

Tentunya jika dalam proses runtime terjadi perbedaan yang signifikan, maka programmer pasti cenderung memilih *D&Q*, namun jika tidak pastilah mereka memilih metode *bruteforce.*

2. METODE

Adapun metode yang digunakan dalam telaah waktu ini adalah dengan menganalisis nilai kompleksitas algoritma dari tiap-tiap metode operasi sorting list, lalu membuat kode program yang disisipi timer di awal dan di akhir program. Lalu membandingkan dan menganalisa hasil runtime tersebut.

2.1 Analisis kompleksitas waktu

2.1.1 Algoritma Bubble Sort

Algoritma BubbleSort adalah algoritma yang cukup sederhana dalam melakukan pengurutan suatu senarai (list). Kodenya hanya beberapa baris saja, namun sudah bisa menghasilkan list yang terurut dengan benar.

Gagasan dasar dari algoritma BubbleSort adalah membandingkan sepasang elemen yang berurutan di dalam larik dan mempertukarkan keduanya jika perlu. Dengan melakukan hal ini secara berulang-ulang (yang dalam hal ini pasangan elemen yang dibandingkan naik satu posisi tiap kali pengulangan), akan mengapungkan elemen yang paling kecil sebagai elemen teratas di dalam list. Satu proses ini disebut satu pass[2]. Pada pass kedua, dilakukan hal yang sama yaitu mengapungkan elemen terkecil di dalam list.

Begitu seterusnya untuk pass yang lain, dan hingga akhir program nantinya akan ada n-1 pass.

Pseudo code algoritmanya ditunjukkan sebagai berikut

```
prosedure BubleSort (array int a)
Deklarasi
      i,k,temp : integer
Algoritma
      for (i=0;i<(n-1);i++)
            for (k=n;k<=i;k--)
             if a[k] < a [k-1] do
                  temp = a[k]
                  a[k] = a [k-1]
                  a[k-1] =  temp
            endif
```

Namun di balik kesederhanaannya, operasi yang dilakukan oleh algoritma ini sangatlah banyak. Dengan jelas kita melihat dua loop berkalang (*nested loop*), dan dengan menghitung kompleksitas waktunya kita dapatkan kompleksitas waktu kuadratik $T(n) = O(n^2)$.

2.1.2 Algoritma Merge Sort

Algoritma Merge Sort dilakukan dengan memenuhi kondisi sebagai berikut :

1. Untuk kasus n = 1, maka table sudah terurut dengan sendirinya.
2. Untuk kasus n > 1, maka :
   a. DIVIDE : bagi tabel a menjadi 2 bagian yang hampir sama besar.
   b. CONQUER: secara rekursif, terapkan algoritma D-and-C pada masing-masing bagian.
   c. MERGE : gabung hasil pengurutan kedua bagian sehingga diperoleh tabel a yang terurut.

Dengan menerapkan kedua aturan diatas secara terus menerus maka nantinya akan didapatkan list yang terurut. Pseudo Code untuk algoritma MergeSort adalah sebagai berikut

```
Prosedur MergeSort (array int a,
              int i, int j)
Deklarasi :
       k : integer
Algoritma :
       If  i<j then
             k ←(i+j) div 2
             MergeSort(a,i,k)
             MergeSort(a,k+1,j)
             Merge(a,i,k,j)
```

Untuk menyederhanakan perhitungan kompleksitas waktu MergeSort, kita membuat asumsi ukuran tabel adalah perpangkatan dari 2, yaitu $n=2^k$ dengan k adalah bilangan bulat positif[2]. Kompleksitas waktu dihitung dari jumlah perbandingan dengan elemen-elemen tabel.

T(n) = jumlah perbandingan pada pengurutan dua buah subtabel + jumlah perbandingan pada prosedur Merge.

Kompleksitas prosedur Merge adalah t(n) = cn = O(n), sehingga kompleksitas algoritma Merge Sort menjadi (dalam bentuk relasi rekurens):

$$T(n) \begin{cases} a & ,n = 1 \\ 2T\left(\frac{n}{2}\right) + cn & ,n > 1 \end{cases}$$

dalam hal ini,a dan c adalah konstanta.

Penyelesaian persamaan rekurens:

$T(n) = T(n/2)+cn$
$= 2(2T(n/4) + cn/2) + cn$
$= 4T(n/4) + 2cn$
$= 4(2T(n/8) + cn/4) + 2\,cn$
$= \ldots$
$= 2^k\, T(n/2^k)+kcn$

Persamaan terakhir dapat diselesaikan karena basis rekursif adalah ketika ukuran tabel adalah 1.
sehingga

$T(n) = nT(1) + cn\ ^2\log n$

sehingga

$T(n) = nT(1) + cn\ ^2\log n$
$= na + cn\ ^2\log n$
$= O(n(^2\log n))$

Jadi,MergeSort mempunyai kompleksitas waktu asimptotik O ($n\ ^2\log n$) (lebih baik daripada bruteforce). Kelemahan MergeSort terletak pada penggunaan tabel temporer untuk menyimpan hasil penggabungan dari dua bagian tabel.

2.1.3 Algoritma QuickSort

Di dalam algoritma qst, proses pembagian tabel disebut partisi (partition)[2]. Penggabungan tabel tidak dinyatakan secara eksplisit dalam sebuah prosedur, sebab pada prooses partisi sekaligus mengurutkan dan menggabungkan.

Aturan – aturan yang dijalankan ketika kita melakukan algoritma qst adalah sebagai berikut :

1. pilih elemen x € { a_1, $a_{2,..,}$ a_n}
2. scan tabel darikiri sampai ditemukan elemen a_p >= x
3. scan tabel dari kanan sampai ditemukan elemen a_q <=x
4. pertukarkan a_p <=> a_q
5. ulangi langkah 2, dari posisi p+1, dan langkah 3 dari posisi q-1, sampai kedua pemindaian bertemu di tengah tabel.

Pseudo-code Quick Sort :

```
prosedur QuickSort (array int a, int i, int j )
Deklarasi
        k : integer

Algoritma
        if i<j then
                Partisi(a,i,j,k)
                QuickSort(a,i,k)
                QuickSort (a, k+1,j)
        endif
```

1. Kasus Terbaik

Kasus terbaik terjadi apabila pivot adalah elemen median sehingga kedua subtabel berukuran relatif sama setiap kali terjadi partisi. Menentukan median tabel adalah persoalan tersendiri, sebab kita harus menentukan median dari tabel yang belum terurut. Kompleksitas waktu pengurutan dihitung dari jumlah perbandingan elemen-elemen tabel:

$T_{min}(n)$ = waktu partisi + waktu pemanggilan rekurens QuickSort unutk dua bagian tabel partisi

Kompleksitas prosedur partisi adalah t(n) = cn = O(n).

$$T(n) \begin{cases} a & ,n = 1 \\ 2T\left(\frac{n}{2}\right) + cn & ,n > 1 \end{cases}$$

Persamaan pada bagian rekurensi bila diselesaikan (lihat penyelesaian persamaan yang serupa pada MergeSort), menghasilkan T(n) = na + cn ^{2}log n = O (n ^{2}log n).

2. Kasus terburuk

Kasus terburuk terjadi bila pada setiap partisi pivot selalu elemen maksimum (atau elemen minimum) tabel. Hal ini menyebabkan pembagian menghasilkan subtabel kiri (atau kanan) berukuran satu elemen dan subtabel kanan (atau kiri) berukuran n-1 elemen. Kompleksitas waktu pengurutan dalam keadaan ini :

$$T(n) \begin{cases} a & ,n = 1 \\ T(n-1) + cn & ,n > 1 \end{cases}$$

Persamaan pada bagian rekurensi jika diselesaikan akan menghasilkan T(n) = O (n^2).

3. Kasus rata-rata

Kasus ini terjadi jika pivot dipilih seara acak dari elemen tabel, dan peluang tiap elemen dipilih menjadi pivot adalah sama. Kompleksitas waktunya adalah T(n) = O (n ^{2}log n).

Meskipun kasus terbaik QuickSort mempunyai kompleksitas algoritma yang sama dengan kompleksitas MergeaSort, yaitu O (n ^{2}log n) tetapi QuickSort tidak membutuhkan tabel temporer sebagaimana MergeSort.

2.2 Pengukuran waktu eksekusi

Bahasa yang digunakan dalam eksekusi ini adalah bahasa java agar merepresentasikan semua jenis mesin. Sebagaimana semboyan sun terhadap bahasa ini *write once, run anywhere.*[1]

1. Template kode pengukur waktu eksekusi

Nantinya akan dibandingkan waktu eksekusi program untuk tiap-tiap algoritma, dalam menyelesaikan suatu list yang berisi nilai random.Awalnya kitamempunyai sebuah class yang menggenerate list sebagai berikut :

```
class listmaker {
    static int[] makelist (int n) {
        int [] apa = new int [n];
        Random r = new Random();
        for (int i=0;i<n;i++){
            apa[i] = r.nextInt(1000);
        }
        return apa;
    }
}
```

Class diatas akan menggenerate list yang isinya berupa nilai random dari 0 hingga 999. Lalu kita mempersiapkan suatu kelas yang nantinya dijadikan template pengukuran. Contoh kelasnya adalah sebagai berikut :

```
class contoh{
 public static void main (String args[]) {
  GregorianCalendar time1=new GregorianCalendar();
  int hh=time1.get(Calendar.HOUR);
  int ms=time1.get(Calendar.MILLISECOND );
  int ss=time1.get(Calendar.SECOND);
  int mm=time1.get(Calendar.MINUTE);

  // peletakan kode program
  // akhir peletakan kode

  GregorianCalendar time2=new GregorianCalendar();
  hh-=time2.get(Calendar.HOUR);
  ms-=time2.get(Calendar.MILLISECOND );
  ss-=time2.get(Calendar.SECOND);
  mm-=time2.get(Calendar.MINUTE);
  int total=-((ms)+(ss*1000)+(mm*60000)+(hh*3600000));
  System.out.println("Hasil tes ::"+total);
 }
}
```

2.2 Tabulasi Data

Setelah dilakukan eksekusi pada suatu komputer berspesifikasi :

- Microsoft Windows SP2
- Intel Pentium M 1.66 GHz
- Ram 448 MB

Didapatkan hasil pengukuran eksekusi program adalah sebagai berikut :

Tabel I Tabel Hasil Eksekusi Operasi List

N\Algo	BubbleSort (miliskn)	MergeSort (miliskn)	QuickSort (miliskn)
1.000	40	10	10
10.000	2493	30	20
100.000	121963	100	70

Tiap-tiap data diatas merupakan rata-rata hasil eksekusi 10 kali operasi list.

IV. KESIMPULAN

Dari tabel data didapatkan bahwa jika nilai n tidak terlalu besar (<1000), dan operasi hanya dilakukan sekali. Maka perbedaan waktu eksekusi dari ketiganya tidak terlalu signifikan, yaitu 40 milisecond untuk operasi bruteforce (BubbleSort) dan 10 milisecond untuk operasi dengan metode divide and conquer.

Namun untuk nilai n yang besar disarankan untuk membentuk program dengan struktur algoritma Divide and Qonquer karena waktu operasi untuk n yang cukup besar sangatlah berpengaruh.

Kesimpulan kedua yaitu, meskipun dalam teori kompleksitas waktu terburuk QuickSort mempunyai nilai lebih rendah daripada MergeSort, namun dalam proses runtime hal itu tidak menjadi suatu hambatan. Terlihat bahwa di dalam tabulasi data menunjukkan bahwa QuickSort memberikan hasil yang paling optimal dalam hal sorting list. Ini artinya proses pembentukan tabel baru di dalam memori untuk proses sorting, lebih memakan waktu serta tidak optimal dalam hal komsumsi memori.

IV. SARAN

Mungkin nantinya bisa ditemukan konstanta-konstanta baru yang mungkin diikutsertakan pada penghitungan kompleksitas waktu misalnya pembentukan data di memori atau pemanggilan prosedur baru, sehingga nilai kompleksitas waktu bisa lebih presisi.

REFERENSI

[1] Inggriani Liem, “Pemrograman Berorientasi Objek”, STEI , 2006.

[2] Rinaldi Munir, “Strategi Algoritmik”, STEI , 2006.

LAMPIRAN I

CODE PROGRAM

```
class mergesort{

    public static void merge(int[] a,
            int from, int mid, int to)
    {
        int n = to - from + 1;
        int[] b = new int[n];
        int i1 = from;
        int i2 = mid + 1;
        int j = 0;
        while (i1 <= mid && i2 <= to)
        {   if (a[i1] < a[i2])
            {   b[j] = a[i1];
                i1++;
            }
            else
            {   b[j] = a[i2];
                i2++;
            }
            j++;
        }
        while (i1 <= mid)
        {   b[j] = a[i1];
            i1++;
            j++;
        }
        while (i2 <= to)
          {   b[j] = a[i2];
              i2++;
              j++;
          }
          for (j = 0; j < n; j++)
              a[from + j] = b[j];
    }

    public static void mergeSort(int[] a,
                int from, int to)
    {   if (from == to) return;
        int mid = (from + to) / 2;
        mergeSort(a, from, mid);
        mergeSort(a, mid + 1, to);
        merge(a, from, mid, to);
    }

    public mergesort(int a[]) {
   GregorianCalendar time2=new GregorianCalendar()
   hh-=time2.get(Calendar.HOUR);
   ms-=time2.get(Calendar.MILLISECOND );
   ss-=time2.get(Calendar.SECOND);
   mm-=time2.get(Calendar.MINUTE);
   int total=-((ms)+(ss*1000)+(mm*60000)+(hh*3600000));
   System.out.println("Hasil tes ::"+total);
   // peletakan kode program
   mergeSort(a, 0, a.length - 1);
   // akhir peletakan kode program
   GregorianCalendar time1=new GregorianCalendar();
   int hh=time1.get(Calendar.HOUR);
   int ms=time1.get(Calendar.MILLISECOND );
   int ss=time1.get(Calendar.SECOND);
   int mm=time1.get(Calendar.MINUTE);

  }
 }
```

```
class Quicksort{
int number [];
public Quicksort(int a[]) {
  GregorianCalendar time1=new GregorianCalendar();
  int hh=time1.get(Calendar.HOUR);
  int ms=time1.get(Calendar.MILLISECOND );
  int ss=time1.get(Calendar.SECOND);
  int mm=time1.get(Calendar.MINUTE);

  // peletakan kode program
  number = a;
  int mulaisort = this.Sort(0,a.length-1);
  // akhir peletakan kode

  GregorianCalendar time2=new GregorianCalendar();
  hh-=time2.get(Calendar.HOUR);
  ms-=time2.get(Calendar.MILLISECOND );
  ss-=time2.get(Calendar.SECOND);
  mm-=time2.get(Calendar.MINUTE);
  int total=-((ms)+(ss*1000)+(mm*60000)+(hh*3600000));
  System.out.println("Hasil tes ::"+total);
 }

public int Sort(int left, int right){
    int v ;
    int i ;
    int j ;
    int nt;
    int t ;
    boolean cont01;
    boolean cont02;
    int aux03 ;
    t = 0 ;
    if (left < right){
        v = number[right] ;
        i = left - 1 ;
        j = right ;
        cont01 = true ;
        while (cont01){
        cont02 = true ;
        while (cont02){
            i = i + 1 ;
            aux03 = number[i] ;
            if (!(aux03<v)) cont02 = false ;
            else cont02 = true ;
        }
        cont02 = true ;
        while (cont02){
            j = j - 1 ;
            aux03 = number[j] ;
            if (!(v < aux03)) cont02 = false ;
            else cont02 = true ;
        }


        t = number[i] ;
        number[i] = number[j] ;
        number[j] = t ;
        //aux03 = i + 1;
        if ( j < (i+1)) cont01 = false ;
        else cont01 = true ;
        }
        number[j] = number[i] ;
        number[i] = number[right] ;
        number[right] = t ;
        nt = this.Sort(left,i-1);
        nt = this.Sort(i+1,right);
    }
    else nt = 0 ;
    return 0 ;
    }
}
```

```
class bubble{
 public bubble(int a[]) {
  GregorianCalendar time1=new GregorianCalendar();
  int hh=time1.get(Calendar.HOUR);
  int ms=time1.get(Calendar.MILLISECOND );
  int ss=time1.get(Calendar.SECOND);
  int mm=time1.get(Calendar.MINUTE);


    // peletakan kode program
  int out, in,temp;

  for(out= a.length-1; out>1; out--)
   for(in=0; in<out; in++)
    if( a[in] > a[in+1] ){
     temp = a[in];
     a[in] = a[in+1];
     a[in+1] = temp;
    }

  GregorianCalendar time2=new GregorianCalendar();
  hh-=time2.get(Calendar.HOUR);
  ms-=time2.get(Calendar.MILLISECOND );
  ss-=time2.get(Calendar.SECOND);
  mm-=time2.get(Calendar.MINUTE);
  int total=-((ms)+(ss*1000)+(mm*60000)+(hh*3600000));
  System.out.println("Hasil tes ::"+total);
  }
}
```

Main Program :

```
class maintester{
    public static void main (String rgs[]){
        int n=1000000;

        new bubble(listmaker.makelist(n));


        new mergesort(listmaker.makelist(n));
        new Quicksort(listmaker.makelist(n));

    }
}
```