Perbandingan Kecepatan/Waktu Komputasi Beberapa Algoritma Pengurutan (*Sorting*)

Indrayana[1], Muhamad Ihsan Fauzi[2]

Laboratorium Ilmu dan Rekayasa Komputasi
Departemen Teknik Informatika, Institut Teknologi Bandung
Jl. Ganesha 10, Bandung

E-mail : if13034@students.if.itb.ac.id[1], if13071@students.if.itb.ac.id[2]

Abstrak

Setiap permasalahan yang dihadapi dalam bidang informatika dan komputasi memiliki metode tertentu untuk menyelesaikannya. Seiring berkembangnya kemajuan di bidang tersebut, tuntutan untuk menemukan metode pemecahan masalah secara lebih tepat, mangkus dan kuat menjadi sebuah kebutuhan, terutama untuk permasalahan klasik. Salah satu masalah klasik di bidang tersebut adalah pengurutan data (*sorting*). Pengurutan data (*sorting*) memegang peranan penting dalam banyak aplikasi dan persoalan yang mengacu pada banyak data (minimal lebih dari satu), dan seringkali menjadi upa-masalah yang banyak dipertimbangkan agar keseluruhan permasalahan dapat diselesaikan dengan lebih baik dan cepat. Algoritma pengurutan (*sorting*) yang banyak digunakan umumnya menggunakan metode pemecahan *brute force* atau *divide and conquer*. Secara *brute force* , algoritma pengurutan (*sorting)* yang dipakai adalah algoritma *bubble sort*. Secara *devide and conquer*, algoritma yang digunakan contohnya adalah *merge sort, insertion sort,selection sort*, dan *quick sort*. Pada makalah ini akan dipaparkan penjelasan singkat dan perbandingan kecepatan dan kemangkusan algoritma-algoritma pengurutan data dengan elemen *integer*.

Kata kunci: *algoritma pengurutan, sorting, brute force, divide and conquer, merge sort, insertion sort, selection sort, quick sort*

1. Pendahuluan

Pengaksesan data yang lebih baik, kuat, dan cepat memerlukan pengolahan data yang lebih baik pula. Salah satu jenis pengolahan data yang menjadi permasalahan klasik adalah pengurutan data *integer*. Pengurutan data (*sorting*) memegang peranan penting dan menjadi upa-masalah yang banyak dipertimbangkan agar keseluruhan permasalahan (terutama mengenai pengolahan data) menjadi lebih baik dan lebih cepat untuk diselesaikan.

2. Algoritma Pengurutan (*Sorting*)

Pengertian *algoritma* secara umum adalah langkah-langkah penyelesain suatu masalah dengan urutan dan metode tertentu, yang dipengaruhi oleh pola pikir terhadap suatu masalah. Algoritma pengurutan (*sorting*) pada dasarnya adalah membandingkan antar data atau elemen berdasarkan kriteria dan kondisi tertentu. Pada pengurutan (*sorting*) *integer* (bilangan bulat positif atau negatif, termasuk nol), kriteria yang umum digunakan adalah lebih besar (>) atau lebih kecil (<) terhadap elemen *integer* yang lain

3. Pengurutan Menggunakan Algoritma *Brute Force*

3.1. Pengertian Algoritma *Brute Force*

Brute Force adalah sebuah pendekatan yang lempang (s*traight forward*) untuk memecahkan suatu masalah, biasanya didasarkan pada pernyataan masalah (*problem statement*) dan definisi konsep yang dilibatkan. Algoritma *brute force* memecahkan masalah dengan sangat sederhana, langsung dan dengan cara yang jelas (*obvious way*) [1].

Kekuatan algoritma *brute force* terletak pada kemampuannya untuk menemukan semua pemecahan masalah yang mungkin. Akan tetapi algoritma *brute force* membutuhkan langkah yang sangat banyak karena menelusuri semua kemungkinan penyelesaian masalah, sehingga cenderung menjadi tidak mangkus jika digunakan untuk memecahkan masalah dengan masukan yang sangat besar.

3.2. Algoritma *Bubble Sort*

Di antara beberapa algoritma pengurutan yang ada, algoritma *bubble sort* merupakan teknik yang paling sederhana. Proses pencarian solusi dilakukan secara

brute force, langsung ke intinya, yaitu membandingkan elemen-elemen dalam tabel. Elemen yang lebih kecil ditukar posisinya dengan elemen yang lebih besar bila posisi elemen yang lebih kecil ada di bawah elemen yang lebih besar. Jika tabel belum terurut, proses diulang kembali sampai elemen paling kecil berada di posisi teratas dan elemen lainnya sudah terurut.

Pseudo-code algoritma ini sebagai berikut [1] :

```
procedure BubbleSort (input/output L :
TabelInt, input n : integer)
{ Mengurutkan tabel L[1..N] sehingga terurut
  menaik dengan metode pengurutan bubble
  sort.

  Masukan : Tabel L yang sudah terdefenisi
            nilai-nilainya.
  Keluaran: Tabel L yang terurut menaik
            sedemikian sehingga
            L[1] ≤ L[2] ≤ … ≤ L[N].
}

Deklarasi
    i    : integer { pencacah untuk jumlah
           langkah }
    k    : integer     { pencacah,untuk
           pengapungan pada setiap langkah}
    temp : integer { peubah bantu untuk
           pertukaran }

Algoritma:
   for i ← 1 to n - 1 do
     for k ← n downto i + 1 do
        if L[k] < L[k-1] then
          {pertukarkan L[k] dengan L[k-1]}
           temp ← L[k]
           L[k] ← L[k-1]
           L[k-1] ← temp
        endif
     endfor
   endfor
```

Kompleksitas algoritma ini adalah $O(n^2)$.

4. Pengurutan Menggunakan *Divide and Conquer*

4.1 Pengertian *Divide and Conquer*

Divide and Conquer adalah metode pemecahan masalah yang bekerja dengan membagi masalah (*problem*) menjadi beberapa upa-masalah (*sub-problem*) yang lebih kecil, kemudian menyelesaikan masing-masing upa-masalah secara independen, dan akhirnya menggabung solusi masing-masing upa-masalah sehingga menjadi solusi masalah semula [1].

Algoritma *divide and conquer* menawarkan penyederhanaan masalah, dengan pendekatan tiga langkah sederhana : pembagian masalah menjadi sekecil mungkin, penyelesaian masalah-masalah yang telah dikecilkan, kemudian digabungkan kembali untuk mendapat solusi optimal secara keseluruhan.

4.2. Algoritma Pengurutan Mudah Dibagi/Sulit Digabung

Algoritma tipe ini mengimplementasikan algoritma *divide and conquer* dengan cara membagi tabel menjadi bagian kiri dan bagian kanan secara rekursif. Kemudian menggabungkan hasil pengurutan masing-masing bagian menjadi tabel semula yang sudah terurut. Proses pembagiannya mudah karena hanya memerlukan proses pembagian biasa. Akan tetapi proses penggabungannya menjadi sulit karena elemen-elemen pada tabel kecil belum tentu terurut sehingga pada setiap tahap penggabungan, harus terus dilakukan pembandingan dan pengurutan kembali. Algoritma pengurutan jenis ini antara lain :

1. *Merge Sort*
2. *Insertion Sort*

4.2.1. *Merge Sort*

Pseudo-code algoritma ini sebagai berikut [1] :

```
procedure   MergeSort(input/output   A   :
TabelInt, input i, j : integer)

{ Mengurutkan tabel A[i..j] dengan algoritma
  Merge Sort
  Masukan: Tabel A dengan n elemen
  Keluaran: Tabel A yang terurut
}

Deklarasi:
   k : integer

Algoritma:
   if i < j then       { Ukuran(A)> 1}
     k←(i+j) div 2
     MergeSort(A, i, k)
     MergeSort(A, k+1, j)
     Merge(A, i, k, j)
   endif


procedure Merge(input/output A : TabelInt,
input kiri,tengah,kanan : integer)

{ Menggabung tabel A[kiri..tengah] dan tabel
    A[tengah+1..kanan]
  menjadi tabel A[kiri..kanan] yang terurut
    menaik.
  Masukan:   A[kiri..tengah]   dan   tabel
    A[tengah+1..kanan]   yang   sudah   terurut
    menaik.
  Keluaran:   A[kiri..kanan]   yang   terurut
    menaik.
}

Deklarasi
    B : TabelInt
    i, kidal1, kidal2 : integer

Algoritma:
```

```
    kidal1←kiri       { A[kiri .. tengah] }
    kidal2←tengah + 1 {A[tengah+1..kanan] }
    i←kiri
    while (kidal1 ≤ tengah) and (kidal2 ≤
    kanan) do
       if A_kidal1 ≤ A_kidal2 then
           B_i←A_kidal1
           kidal1←kidal1 + 1
       else
           B_i←A_kidal2
           kidal2←kidal2 + 1
       endif
       i←i + 1
    endwhile
    { kidal1 > tengah or kidal2 > kanan }

    { salin sisa A bagian kiri ke B, jika
    ada }
    while (kidal1 ≤ tengah) do
       B_i←A_kidal1
       kidal1←kidal1 + 1
       i←i + 1
    endwhile
   { kidal1 > tengah }

   { salin sisa A bagian kanan ke B, jika
    ada }
    while (kidal2 ≤ kanan) do
       B_i←A_kidal2
       kidal2←kidal2 + 1
       i←i + 1
    endwhile
    { kidal2 > kanan }

   { salin kembali elemen-elemen tabel B ke
    A }
    for i←kiri to kanan do
       A_i←B_i
    endfor
   { diperoleh tabel A yang terurut membesar
}
```

Kompleksitas algoritma ini adalah $O(n^{2}\log n)$

4.2.2. *Insertion Sort*

Pseudo-code algoritma ini sebagai berikut [1] :

```
procedure InsertionSort(input/output A :
TabelInt, input i, j : integer)

{ Mengurutkan tabel A[i..j] dengan algoritma
Insertion Sort.
  Masukan: Tabel A dengan n elemen
  Keluaran: Tabel A yang terurut
}

Deklarasi:
   k : integer

Algoritma:
   if i < j then       { Ukuran(A)> 1}
     k←i
     Insertion(A, k+1, j)
     Merge(A, i, k, j)
   endif
```

Kompleksitas algoritma ini adalah $O(n^2)$

4.3. Algoritma pengurutan Sulit dibagi/ mudah digabung

Algoritma tipe ini mengimplementasikan algoritma *divide and conquer* dengan cara membagi tabel menjadi bagian kiri dan bagian kanan secara rekursif. Proses pembagiannya sulit karena pada setiap tahap pembagian harus selalu dilakukan pembandingan sehingga tabel bagian kiri elemen-elemennya lebih kecil daripada tabel bagian kanan. Setelah proses pembagian selesai, tabel-tabel kecil dapat dengan mudah digabungkan dengan metode konkatenasi biasa. Hal ini dikarenakan semua tabel sudah dalam keadaan terurut. Algoritma pengurutan jenis ini antara lain :

1. *Selection Sort*
2. *Quick Sort*

4.3.1. *Selection Sort*

Pseudo-code algoritma ini sebagai berikut [1] :

```
procedure SelectionSort(input/output A :
TabelInt, input i,j: integer)

{ Mengurutkan tabel A[i..j] dengan algoritma
Selection Sort.
  Masukan: Tabel A[i..j] yang sudah
terdefinisi elemen-elemennya.
  Keluaran: Tabel A[i..j] yang terurut
menaik.
}


Algoritma:
   if  i < j then        { Ukuran(A) > 1 }
     Bagi(A, i, j)
     SelectionSort(A, i+1, j)
   endif


procedure Bagi(input/output A : TabInt,
input i,j: integer)

{ Mencari elemen terkecil di dalam tabel
A[i..j], dan menempatkan elemen terkecil
sebagai elemen pertama tabel.

  Masukan: A[i..j]
  Keluaran: A[i..j] dengan A_i adalah elemen
terkecil.
}

Deklarasi
  idxmin, k, temp : integer

Algoritma:
  idxmin←i
  for k←i+1 to jdo
    if A_k < A_idxmin then
      idxmin←k
    endif
  endfor

 { pertukarkan A_i dengan A_idxmin }
 temp←A_i
 A_i←A_idxmin
 A_idxmin←temp
```

Kompleksitas algoritma ini adalah $O(n^2)$

4.3.2. *Quick Sort*

Pseudo-code algoritma ini sebagai berikut [1] :

```
procedure QuickSort(input/output A : TabelInt, input i,j: integer)

{ Mengurutkan tabel A[i..j] dengan algoritma Quick Sort.
  Masukan: Tabel A[i..j] yang sudah terdefinisi elemen-elemennya.
  Keluaran: Tabel A[i..j] yang terurut menaik.
}

Deklarasi
   k : integer

Algoritma:
    if i < j then     { Ukuran(A) > 1 }
       Partisi(A, i, j, k)   { Dipartisi pada indeks k }
         QuickSort(A, i, k)      { Urut A[i..k] dengan Quick Sort }
         QuickSort(A, k+1, j)    { Urut A[k+1..j] dengan Quick Sort }
    endif


procedure Partisi(input/output A : TabelInt, input i, j : integer, output q : integer)

{ Membagi tabel A[i..j] menjadi upatabel A[i..q] dan A[q+1..j]
  Masukan: Tabel A[i..j]yang sudah terdefinisi harganya.
  Keluaran upatabel A[i..q] dan upatabel A[q+1..j] sedemikian sehingga
           elemen tabel A[i..q] lebih kecil dari elemen tabel A[q+1..j]
}

Deklarasi
   pivot, temp : integer

Algoritma:
   pivot←A(i+j) div 2 { pivot = elemen tengah}
   p←i
   q←j
   repeat
      while Ap < pivot do
         p←p + 1
      endwhile
      { Ap >= pivot}

      while Aq > pivot do
         q←q – 1
      endwhile
      { Aq <= pivot}

      if p ≤ q then
        {pertukarkan Ap dengan Aq }
        temp←Ap
        Ap←Aq
        Aq←temp

     {tentukan awal pemindaian berikutnya }
        p←p + 1
        q←q - 1
      endif
   until p > q
```

Kompleksitas algoritma ini adalah $O(n^2 \log n)$ untuk kasus terbaik dan kasus rata-rata, $O(n^2)$ untuk kasus terburuk.

6. Perbandingan Algoritma Sorting

6.1. Pengujian dengan Sorting Timer

Tiap *algoritma sorting integer* diatas memiliki kelebihan dan kekurangan satu sama lain.

Bubble Sort :

1. Algoritma singkat
2. Metode paling sederhana dan kuat
3. Waktu kompleksitas yang sama untuk semua kasus

Merge Sort dan *Insertion Sort :*

1. Kompleksitas *merge sort* relatif lebih kecil
2. Mudah membagi masalah, tetapi sulit menggabungkannya kembali
3. Membutuhkan *method* tambahan (Merge)

Selection Sort dan *Quick Sort :*

1. Kompleksitas *selection sort* relatif lebih kecil
2. Mudah menggabungkannya kembali, tetapi sulit membagi masalah
3. Membutuhkan method tambahan

Walaupun tiap *algoritma sorting* menawarkan perbedaan metode dan sudut pandang penyelesaian masalah yang berbeda, kompleksitas waktu yang dibutuhkan tetap menjadi masalah utama yang dipertimbangkan untuk menentukan algoritma mana yang lebih baik dan tepat untuk digunakan.

Tabel 1 menunjukkan perbandingan kompleksitas algoritma-algoritma tersebut untuk berbagai kasus :

Tabel 1 Kompleksitas Algoritma

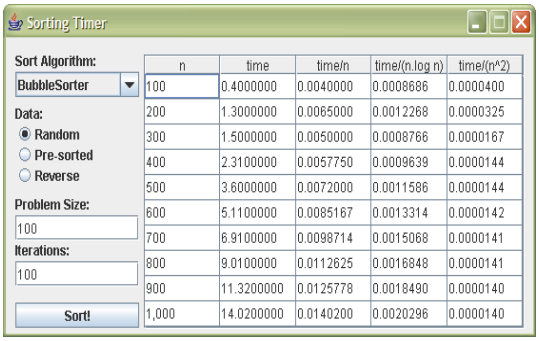
Algoritma	Kompleksitas (O)		
	Kasus terbaik	Kasus rata-rata	Kasus Terburuk
Bubble Sort	n^2	n^2	n^2
Merge Sort	$n^2 \log n$	$n^2 \log n$	$n^2 \log n$
Insertion Sort	n^2	n^2	n^2
Selection Sort	n^2	n^2	n^2
Quick Sort	$n^2 \log n$	$n^2 \log n$	n^2

Untuk mengetahui kecepatan atau waktu tiap algoritma, kami menggunakan sebuah perangkat lunak penghitung kecepatan algoritma dalam bahasa Java bernama *Sorting Timer* dan *Sorting Comparation* Kode sumber program ini bisa didapat dan dipelajari dari :
http://fajran.net/kuliah/web/sda/tugas2 [2].

Kami mencoba menguji algoritma-algoritma sorting yang sesuai dengan pseudocode diatas, dengan

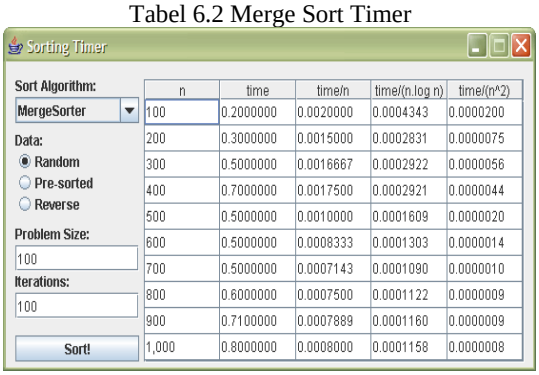
mengatur perangkat lunak agar banyaknya jumlah n menjadi 100 buah (secara acak), dan mengatur *iterasi* menjadi 100. Penguian dilakukan disebuah laptop dengan spesifikasi : platform Windows XP™ Home Edition, processor Intel Centrino™ 1.3 GHz, dan 512 DDR RAM. Hasil pengujian dapat dilihat pada tabel2\-tabel berikut :

Tabel 6.1 Bubble Sort Timer



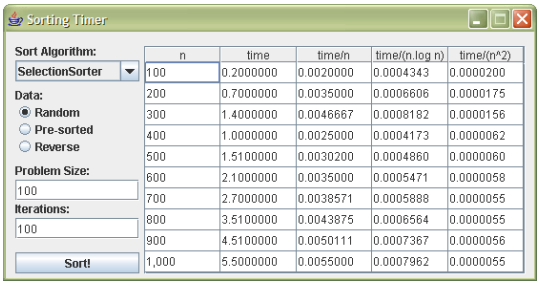
n	time	time/n	time/(n.log n)	time/(n^2)
100	0.4000000	0.0040000	0.0008686	0.0000400
200	1.3000000	0.0065000	0.0012268	0.0000325
300	1.5000000	0.0050000	0.0008766	0.0000167
400	2.3100000	0.0057750	0.0009639	0.0000144
500	3.6000000	0.0072000	0.0011586	0.0000144
600	5.1100000	0.0085167	0.0013314	0.0000142
700	6.9100000	0.0098714	0.0015068	0.0000141
800	9.0100000	0.0112625	0.0016848	0.0000141
900	11.3200000	0.0125778	0.0018490	0.0000140
1,000	14.0200000	0.0140200	0.0020296	0.0000140

Tabel 6.2 Merge Sort Timer



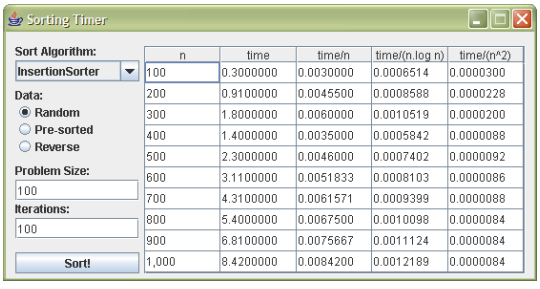
n	time	time/n	time/(n.log n)	time/(n^2)
100	0.2000000	0.0020000	0.0004343	0.0000200
200	0.3000000	0.0015000	0.0002831	0.0000075
300	0.5000000	0.0016667	0.0002922	0.0000056
400	0.7000000	0.0017500	0.0002921	0.0000044
500	0.5000000	0.0010000	0.0001609	0.0000020
600	0.5000000	0.0008333	0.0001303	0.0000014
700	0.5000000	0.0007143	0.0001090	0.0000010
800	0.6000000	0.0007500	0.0001122	0.0000009
900	0.7100000	0.0007889	0.0001160	0.0000009
1,000	0.8000000	0.0008000	0.0001158	0.0000008

Tabel 6.3 Selection Sort Timer



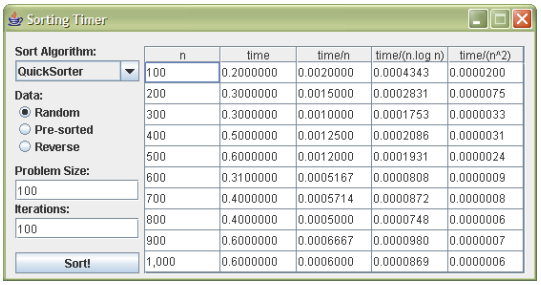
n	time	time/n	time/(n.log n)	time/(n^2)
100	0.2000000	0.0020000	0.0004343	0.0000200
200	0.7000000	0.0035000	0.0006606	0.0000175
300	1.4000000	0.0046667	0.0008182	0.0000156
400	1.0000000	0.0025000	0.0004173	0.0000062
500	1.5100000	0.0030200	0.0004860	0.0000060
600	2.1000000	0.0035000	0.0005471	0.0000058
700	2.7000000	0.0038571	0.0005888	0.0000055
800	3.5100000	0.0043875	0.0006564	0.0000055
900	4.5100000	0.0050111	0.0007367	0.0000056
1,000	5.5000000	0.0055000	0.0007962	0.0000055

Tabel 6.4 Insertion Sort Timer



n	time	time/n	time/(n.log n)	time/(n^2)
100	0.3000000	0.0030000	0.0006514	0.0000300
200	0.9100000	0.0045500	0.0008588	0.0000228
300	1.8000000	0.0060000	0.0010519	0.0000200
400	1.4000000	0.0035000	0.0005842	0.0000088
500	2.3000000	0.0046000	0.0007402	0.0000092
600	3.1100000	0.0051833	0.0008103	0.0000086
700	4.3100000	0.0061571	0.0009399	0.0000088
800	5.4000000	0.0067500	0.0010098	0.0000084
900	6.8100000	0.0075667	0.0011124	0.0000084
1,000	8.4200000	0.0084200	0.0012189	0.0000084

Tabel 6.5 Quick Sort Timer



n	time	time/n	time/(n.log n)	time/(n^2)
100	0.2000000	0.0020000	0.0004343	0.0000200
200	0.3000000	0.0015000	0.0002831	0.0000075
300	0.3000000	0.0010000	0.0001753	0.0000033
400	0.5000000	0.0012500	0.0002086	0.0000031
500	0.6000000	0.0012000	0.0001931	0.0000024
600	0.3100000	0.0005167	0.0000808	0.0000009
700	0.4000000	0.0005714	0.0000872	0.0000008
800	0.4000000	0.0005000	0.0000748	0.0000006
900	0.6000000	0.0006667	0.0000980	0.0000007
1,000	0.6000000	0.0006000	0.0000869	0.0000006

Tabel 6.6 Perbandingan Waktu Algoritma

N	Waktu Algoritma				
	BS	MS	IS	SS	QS
100	0.40	0.20	0.30	0.20	0.20
200	1.30	0.30	0.91	0.70	0.30
300	1.50	0.50	1.80	1.40	0.30
400	2.31	0.70	1.40	1.00	0.50
500	3.60	0.50	2.30	1.51	0.60
600	5.11	0.50	3.11	2.10	0.31
700	6.91	0.50	4.31	2.70	0.40
800	9.01	0.60	5.40	3.51	0.40
900	11.32	0.71	6.81	4.51	0.60
1000	14.02	0.80	8.42	5.50	0.60

Ket : BS : *Bubble Sort*
MS : *Merge Sort*
IS : *Insertion Sort*
SS : *Selection Sort*
QS : *Quick Sort*

Dari Tabel 6.6 diatas bisa kita lihat perbedaan waktu yang cukup jelas di antara algoritma-algoritma tersebut. *Bubble Sort* (kompleksitas $O(n^2)$) membutuhkan waktu paling lama untuk mengurutkan data. Sedangkan waktu tercepat dihasilkan oleh algoritma *quick sort* dan *merge sort*. Hal ini sesuai dengan kompleksitas algoritma kedua algoritma tersebut yang lebih mangkus $O(n^2 \log n)$. Walaupun *insertion sort* dan *selection sort* memiliki kompleksitas algoritma yang sama dengan *bubble sort*($O(n^2)$), tetapi waktu yang dihasilkan lebih cepat. Perbedaan ini sebanding dengan jumlah n yang semakin meningkat Hal ini disebabkan *bubble sort* memeriksa semua kemungkinan solusi secara *brute force* walaupun data sudah terurut (kasus terbaik). Hal ini menunjukkan metode *divide and conquer* relatif lebih baik daripada metode *brute force* (terutama untuk n yang relatif besar).

7. Kesimpulan

Salah satu masalah klasik di bidang informatika dan komputasi adalah pengurutan data (*sorting*). Beberapa algoritma pengurutan yang digunakan antara lain *bubble sort* yang mengimplementasikan algoritma *brute force*, serta *merge sort*, *insertion sort*, *selection sort*, dan *quick sort* yang mengimplementasikan algoritma *divide and conquer*.

Hasil pengujian menunjukkan algoritma *bubble sort* membutuhkan waktu komputasi yang paling lama. Sedangkan algoritma *quick sort* dan *merge sort* yang paling cepat. Walaupun *insertion sort* dan *selection sort* memiliki kompleksitas algoritma yang sama dengan *bubble sort*($O(n^2)$), tetapi waktu yang dihasilkan lebih cepat. Hal ini menunjukkan metode *divide and conquer* relatif lebih baik daripada metode *brute force* (terutama untuk n yang relatif besar).

Daftar Pustaka

[1] Munir, Rinaldi, Ir., M.T., 2005, *Diktat Kuliah IF2251 Strategi Algoritmik*, Bandung.

[2] Nikmati Saja! : Stuktur Data dan Algoritma. http://fajran.net/kuliah/web/sda/, Diakses tanggal 19 Mei 2005, pukul 14.30 WIB