Komparasi Algoritma Sorting pada Bahasa Pemrograman Java

Mohammad Reza Maulana
STMIK Widya Pratama Pekalongan
E-mail: reza.stmikwp@gmail.com

Abstrak

Algoritma *Sorting* merupakan salah satu tugas mendasar yang dilakukan oleh komputer. Pengurutan data yang dilakukan Algoritma Sorting memegang fungsi penting agar sebuah masalah dapat diselesaikan lebih cepat dan tepat. Proses pengurutan data banyak digunakan pada proses-proses umum seperti mengurutkan data tanggal dari yang terbaru ke yang lama ataupun sebaliknya. Terdapat banyak Algoritma *Sorting* yang dapat digunakan, diantaranya *Insertion Sort*, *Selection Sort*, *Quick Sort* dan *Merge Sort*. Pada penelitian ini akan dilakukan komparasi waktu eksekusi dalam proses pengurutan data dari masing-masing Algoritma *Sorting* tersebut. Data yang digunakan pada penelitian ini adalah data dengan tipe integer (angka bulat) dan didapatkan dengan cara mengacak nilai. Jumlah dari data juga bermacam-macam untuk melihat konsistensi kinerja dari masing-masing Algoritma *Sorting*. Bahasa pemrograman yang digunakan untuk mengimplementasikan Algoritma *Sorting* adalah Java. Setelah proses implementasi Agoritma *Sorting* ke dalam bahasa pemrograman Java dilakukan proses pengujian dan evaluasi untuk melihat hasil. Dari hasil pengujian dan evaluasi secara umum Algoritma Sorting yang paling cepat dalam mengurutkan data adalah *Quick Sort*, sedangkan untuk eksekusi waktu paling lama adalah *Insertion Sort.* Pengaturan parameter dan variabel mungkin akan berpengaruh terhadap hasil pengujian dan evaluasi.

Kata Kunci : Algoritma Sorting, Komparasi Algoritma, Bahasa Pemrograman Java

1. PENDAHULUAN

1.1 Latar Belakang

Salah satu tugas komputer yang paling umum dilakukan adalah melakukan proses pengurutan(*sorting*) satu set data (Ladner 1999). Algoritma *Sorting* merupakan tugas di dalam komputer yang sangat fundamental dan saat ini terdapat banyak jenis Algoritma *Sorting* yang telah dikembangkan (Ladner 1999). Pengurutan data memegang peranan penting yang banyak dipertimbangkan agar keseluruhan permasalahan (terutama mengenai pengolahan data) menjadi lebih baik dan lebih cepat untuk diselesaikan (Anisya Sonita 2015).

Algorima *Sorting* banyak digunakan untuk menyelesaiakan kasus komputasi yang sederhana yang terdapat pada contoh sehari-hari. Sebagai contoh Algoritma *Sorting* digunakan untuk mengurutkan data mahasiswa berdasarkan urutan Nomor Induk Mahasiswa(NIM), pengurutan tanggal dari yang paling lama ke tanggal yang paling baru, dan lain sebagainya.

Terdapat banyak jenis Algoritma *Sorting* yang populer untuk melakukan proses pengurutan data. Diantaranya adalah *Insertion Sort*, *Selection Sort*, *Quick Sort* dan *Merge Sort*. Setiap algoritma pengurutan memiliki pendekatan dan metode yang berbeda-beda dalam menjalankan fungsinya. Secara garis besar, algoritma pengurutan dapat dikelompokkan menjadi dua kategori, yaitu algoritma pengurutan berbasis perbandingan (*comparison based*) dan tidak berbasis perbandingan (*non-comparison based*) (Audy 2015).

Algoritma sorting dapat diterapkan pada semua bahasa pemrograman. Bahasa pemrograman yang akan digunakan pada penelitian ini adalah menggunakan bahasa pemrograman java. Berdasarkan rangking yang dirilis oleh TIOBE index bahasa pemrograman Java menempati urutan paling atas (BV 2017).

Pada penelitian sebelumnya, membandingkan dua Algoritma Sorting yaitu *Insertion Sort* dan *Selection Sort* menggunakan bahasa Pemrograman C++, didapatkan hasil bahwa

dalam hal kecepatan waktu eksekusi *Insertion Sort* lebih cepat dibandingakan *Selection Sort* (Maulana 2016). Selain itu pada penelitian lain yang membandingkan Algoritma *Insertion Sort* dengan *Merge Sort* menggunakan bahasa pemrograman C++ didapatkan hasil bahwa *Merge Sort* lebih cepat waktu eksekusinya dibandingkan dengan *Insertion Sort* (Saptadi dan Sari 2012).

Pada penelitian ini akan dilakukan komparasi waktu eksekusi dari masing-masing algoritma yang telah disebutkan sebelumnya yaitu, *Insertion Sort*, *Selection Sort*, *Quick Sort* dan *Merge Sort.* Selain keempat Algoritma Sorting tersebut akan dibandingkan juga algoritma sorting dari Class yang telah disediakan oleh Java, yaitu *Array.sort()*. Hal ini juga dilakukan untuk melihat perbandingan eksekusi waktu dari Algoritma Sorting bawaan dari Java.

1.2 Tujuan Penelitian

Dari penjelasan yang telah dijabarkan sebelumnya, tujuan yang ingin dicapai adalah :

1. Menerapkan Algoritma Sorting *Insertion Sort*, *Selection Sort*, *Quick Sort* dan *Merge Sort* dalam bahasa pemrograman Java.
2. Menguji dan membandingkan kecepatan waktu eksekusi *Insertion Sort*, *Selection Sort*, *Quick Sort* dan *Merge Sort* serta *Sorting Arrays.sort()* Java dalam proses pengurutan.

2. METODE PENELITIAN

Metode penelitian yang digunakan dalam penelitian ini adalah metode eksperimen dengan tahapan pengumpulan data, eksperimen, dan evaluasi.

2.1 Metode Pengumpulan Data

Pada tahap pengumpulan data ini menggunakan jenis data sekunder. Data sekunder dikumpulkan dari beberapa buku, jurnal ilmiah dan website.

2.1.1 Algortima Sorting

1. *Insertion Sort*

Insertion Sort adalah algoritma pengurutan sederhana yang relatif efisien untuk list data kecil dan sering digunakan sebagai bagian dari algoritma yang lebih mutakhir. Algoritma ini bekerja dengan cara mengambil elemen dari list secara satu per satu dan menyisipkan elemen tersebut ke dalam posisi yang benar di list baru yang telah diurutkan (N. Wirth 1985).

Pseudocode dari *Insertion Sort* adalah sebagai berikut (Bentley 2000):

```
i ← 1
while i < length(A)
    j ← i
    while j > 0 and A[j-1] > A[j]
        swap A[j] and A[j-1]
        j ← j - 1
    end while
    i ← i + 1
end while
```

2. *Selection Sort*

Algoritma *Selection Sort* memilih elemen maksimum/minimum array, lalu menempatkan elemen maksimum/minimum itu pada awal atau akhir array (tergantung pada urutannya *ascending/descending*). Selanjutnya elemen tersebut tidak disertakan pada proses selanjutnya. Karena setiap kali *Selection Sort* harus membandingkan elemen-elemen data, algoritma ini termasuk dalam *Comparison-Based Sorting* (Ananda, et al. 2009).

Terdapat dua pendekatan dalam metode pengurutan dengan *Selection Sort*:

a. Algoritma pengurutan maksimum (*Maximum Selection Sort*), yaitu memilih elemen maksimum sebagai basis pengurutan.
b. Algoritma pengurutan minimum (*Minimum Selection Sort*), yaitu memilih elemen minimum sebagai basis pengurutan.

3. *Quick Sort*

Quick Sort adalah salah satu jenis algoritma pengurutan dengan konsep *divide and conquer* dengan mengandalkan operasi partisi, untuk mempartisi sebuah elemen array yang dipilih yang disebut sebagai pivot (Cormen, et al. 2009).

Pseudocode dari *Quick Sort* adalah sebagai berikut (Cormen, et al. 2009):

```
algorithm quicksort(A, lo, hi) is
```

```
    if lo < hi then
        p := partition(A, lo, hi)
        quicksort(A, lo, p - 1 )
        quicksort(A, p + 1, hi)

algorithm partition(A, lo, hi) is
    pivot := A[hi]
    i := lo - 1
    for j := lo to hi - 1 do
        if A[j] < pivot then
            i := i + 1
            swap A[i] with A[j]
    if A[hi] < A[i + 1] then
        swap A[i + 1] with A[hi]
    return i + 1
```

4. *Merge Sort*

Algoritma *Merge Sort* mengambil keuntungan dari kemudahan menggabungkan daftar yang sudah diurutkan ke dalam daftar urutan yang baru. Dimulai dengan proses membandingkan setiap dua elemen (misalnya, 1 dengan 2, 3 dengan 4, dan seterusnya) dan menukarnya jika nilai pertama lebih besar. Kemudian menggabungkan masing-masing dua elemen tersebut menjadi empat elemen dan seterusnya (N. Wirth 1985).

Pseudocode dari *Quick Sort* adalah sebagai berikut:

```
function merge_sort(list m)
    // Base case. A list of zero or
one elements is sorted, by
definition.
    if length of m ≤ 1 then
        return m

    // Recursive case. First, divide
the list into equal-sized sublists
    // consisting of the first half
and second half of the list.
    // This assumes lists start at
index 0.
    var left := empty list
    var right := empty list
    for each x with index i in m do
        if i < (length of m)/2 then
            add x to left
        else
            add x to right

    // Recursively sort both
sublists.
    left := merge_sort(left)
    right := merge_sort(right)

    // Then merge the now-sorted
sublists.
    return merge(left, right)

function merge(left, right)
    var result := empty list

    while left is not empty and
right is not empty do
        if first(left) ≤
first(right) then
            append first(left) to
result
            left := rest(left)
        else
            append first(right) to
result
            right := rest(right)

    // Either left or right may have
elements left; consume them.
    // (Only one of the following
loops will actually be entered.)
    while left is not empty do
        append first(left) to result
        left := rest(left)
    while right is not empty do
        append first(right) to
result
        right := rest(right)
    return result
```

5. Array.sort() Java

Di dalam class library Java terdapat sebuah method yang digunakan untuk mengurutkan data. Method tersebut berada di dalam class Array.

2.1.2 Sampel Data Penelitian

Pada penelitian ini akan digunakan data dengan tipe integer (angka) secara random seperti yang digunakan penelitian sebelumnya (Maulana 2016). Jumlah data akan disesuaikan masing-masing proses pengujian, yaitu dengan jumlah data 1.000, 10.000, 100.000 dan 1.000.000 agar mengetahui perbedaan performa kecepatan waktu eksekusi dari masing-masing Algoritma *Sorting*. Selain itu keragaman data yang dirandom disamakan dengan jumlah data, misal untuk jumlah data 1.000 ada kemungkinan data random dari 1 sampai 1.000.

2.2 Eksperimen

Pada tahap eksperimen dibuat dengan menggunakan model pembuatan method yang nantinya akan menjalankan dari masing-masing algoritma sorting. Masing-masing method nantinya akan dijalankan berdasarkan algoritma sorting yang akan diuji.

Untuk mendapatkan konsistensi dari kinerja algoritma, pengujian akan dilakukan sebanyak 10 kali pada tiap Algoritma *Sorting* dari tiap jumlah data. Nantinya akan dilakukan perhitungan rata-rata dari masing-masing percobaan berdasarkan Algoritma *Sorting* yang diuji.

2.3 Evaluasi

Setelah masing-masing Algoritma *Sorting* dilakukan pengujian, akan dilakukan evaluasi dari masing-masing Algoritma *Sorting*. Pada tahap ini akan dilakukan komparasi kecepatan proses pengurutan data dari masing-masing Algoritma *Sorting*.

3. HASIL DAN PEMBAHASAN

3.1 Hasil Implementasi Algoritma Sorting

Algoritma *Sorting* dari masing-masing metode telah selesai dibuat menggunakan bahasa pemrograman Java. Semua algoritma dibuat dalam satu Class dengan beberapa method yang dibuat sendiri-sendiri untuk masing-masing Algoritma *Sorting*.

Selain method untuk Algoritma Sorting, terdapat beberapa method untuk mengatur parameter dan variabel yang digunakan. Terdapat method untuk melakukan *generate* data dan method untuk mengeksekusi Algoritma *Sorting* yang dipilih serta menghitung waktu eksekusi. Waktu eksekusi dihitung menggunakan satuan seconds.

Berikut merupakan contoh beberapa method yang telah dibuat:

1. Method untuk generate data

```
public static int[]
      generateInput(int jum){
    int[] arr= new int[jum];
    Random ran=new Random();
    for (int i=0; i<jum;i++){
       arr[i]=ran.nextInt(jum);
    }
    return arr;
}
```

2. Method untuk menjalankan Algoritma Sorting dan Pengecekan waktu ekseskusi, sebagai contoh pemanggilan method Insertion Sort

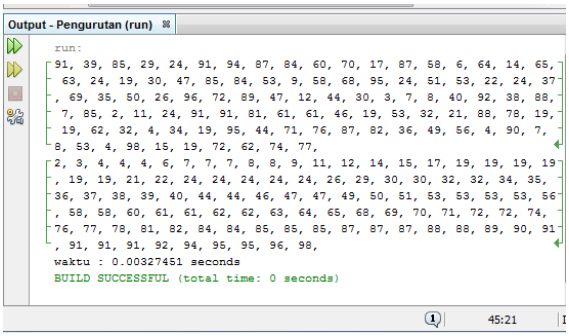
```
public static void main(String[]
args) {
    double startTime =
      System.nanoTime();
    int dataArray[] =
      generateInput(1000000);

    for(int i=0;
      i<dataArray.length;i++){
      System.out.print(dataArray[i
]     +", ");
    }

    System.out.println("");

    doInsertionSort(dataArray);

    for(int i=0;
      i<dataArray.length;i++){
      System.out.print(dataArray[i
]     +", ");
    }

    double endTime =
      System.nanoTime();
    double duration = (endTime -
      startTime)/1000000000;
    System.out.println("");
    System.out.println("waktu :
      "+duration+" seconds");
}
```

3. Method untuk Algortima Sorting Insertion Sort

```
public static int[]
doInsertionSort(int[] input){
    int temp;
    for (int i = 1; i <
      input.length; i++) {
        for(int j = i;j>0;j--){
          if(input[j]<input[j-1]){
             temp = input[j];
             input[j] = input[j-
1];
             input[j-1] = temp;
          }
        }
    }
    return input;
}
```

Setelah setting parameter dan variabel dan proses eksekusi Algoritma Sorting tersebut di dapatkan hasil pengurutan data yang telah

terurut dan waktu eksekusi. Hal tersebut dapat dilihat pada Gambar 1.



Gambar 1 Hasil Pengurutan data untuk Algoritma Sorting Insertion Sort

3.2 Hasil Pengujian dan Evaluasi Algoritma Sorting

Setelah selesai mengimplemmentasikan Algoritma *Sorting*, tahap selanjutnya melakukan pengujian dari masing-masing Algoritma *Sorting*. Untuk masing-masing Algoritma *Sorting*, pengujian dilakukan sebanyak 10 kali. Hal ini dilakukan untuk melihat konsistensi eksekusi waktu yang dibutuhkan untuk mengurutkan data dari masing-masing Algoritma.

Pengujian dilakukan untuk beberapa data list, diantaranya data yang berjumlah 1.000, 10.000, 100.000 dan 1.000.000.

Tabel 1 menunjukkan rata-rata dari hasil pengujian terhadap masing-masing Algoritma *Sorting*.

Tabel 1 Hasil Pengujian Algoritma Sorting

No	Algoritma Sorting	Waktu Eksekusi (seconds)			
		1.000	10.000	100.000	1.000.000
1	Selection Sort	0.022241181	0.173380708	8.829595726	784.8587412
2	Insertion Sort	0.019472058	0.187911421	9.575672376	874.2829735
3	Quick Sort	0.020194002	**0.090888385**	**1.032607549**	**8.755687510**
4	Merge Sort	**0.018374294**	0.124213673	1.032916429	11.28962665
5	Array Sort Java	0.021880292	0.093373723	1.050077253	10.66835099

Dari Tabel 1 dapat dilihat untuk jumlah data 1.000 bahwa untuk eksekusi waktu paling cepat adalah *Merge Sort* yaitu dengan waktu 0.018374294 seconds dan waktu paling lama adalah *Selection Sort* dengan waktu 0.022241181 seconds. Kemudian dengan secara konsisten untuk data 10.000, 100.000 dan 1.000.000 Algoritma *Sorting Quick Sort* melakukan proses pengurutan paling cepat dibandingkan dengan Algoritma *Sorting* lainnya dengan masing-masing waktu 0.090888385 seconds, 1.032607549 seconds dan 8.755687510 seconds. Untuk data 10.000 Algoritma *Sorting* yang memerlukan waktu paling lama adalah *Insertion Sort* dengan waktu 0.187911421 seconds. Begitupun untuk data 100.000 dan 1.000.000 Algoritma *Insertion Sort* membutuhkan waktu eksekusi paling lama yaitu 9.575672376 seconds dan 874.2829735 seconds.

4. KESIMPULAN

Dari hasil pengumpulan data, implementasi dan pengujian maka ada beberapa hal yang dapat disimpulkan, yaitu:

1. Semua Algoritma Sorting telah berhasil diimplementasikan pada bahasa pemrograman Java
2. Untuk hasil pengujian yang telah dilakukan, kecepatan waktu rata-rata paling cepat adalah Algoritma Sorting *Quick Sort.* Sedangkan untuk waktu eksekusi terlama untuk

mengurutkan data adalah Algoritma *Insertion Sort*.

Pengaturan parameter dan variabel dapat berpengaruh pada hasil pengujian. Untuk penelian berikutnya mungkin dapat melakukan pengujian parameter dan variabel untuk melihat hasil perbandingan dari masin-masing Algoritma *Sorting*.

5. REFERENSI

Ananda, Dahliar, Ahmad Suryan, Paramita Mayadewi, Lutce Rasiana, dan Hendra Kusmayadi. 2009. *Algoritma Pemrograman.* Bandung: Politeknik Telkom.

Anisya Sonita, Febrian Nurtaneo. 2015. "Analisis Perbandingan Algoritma Bubble Sort, Merge Sort, Dan Quick Sort Dalam Proses Pengurutan Kombinasi Angka Dan Huruf." *Jurnal Pseudocode* 75-80.

Audy. 2015. "Komparasi Algoritma Quicksort dan Bucket Sort pada Pengurutan Data Integer." *Ultimatics* 7-13.

Bentley, Jon. 2000. *Programming Pearls.* ACM Press/Addison–Wesley.

BV, TIOBE software. 2017. *TIOBE.* 30 November. Diakses December 5, 2017. https://www.tiobe.com/tiobe-index/.

Cormen, Thomas H., Charles E. Leiserson, Ronald L. Rivest, dan Clifford Stein. 2009. *Introduction to Algorithms (3rd ed.).* Cambridge, MA: The MIT Press.

Ladner, Richard E. 1999. "The Influence of Caches on the Performance of Sorting." *Journal of Algorithms.*

Maulana, Reza. 2016. "Analisa Perbandingan Kompleksitas Algoritma Selectionsort dan Insertionsort." *Informatika* 208-218.

Saptadi, Arief Hendra, dan Desi Windi Sari. 2012. "Analisis Algoritma Insertion Sort, Merge Sort Dan Implementasinya Dalam Bahasa Pemrograman C++." *Jurnal Infotel* 10-17.

Wirth, Niklaus. 1985. *Algorithms & Data Structures.* Upper Saddle River: Prentice-Hall.