PERBANDINGAN KECEPATAN ANTARA *SELECTION SORT*, *INSERTION SORT*, DAN *BUBBLE SORT*

Anggraini Kusumaningrum

Jurusan Manajemen Informatika AMIK "BSI Yogyakarta"

anggraini.kusumaningrum@gmail.com

ABSTRAK

Sorting atau pengurutan adalah salah satu proses yang sangat dibutuhkan di dalam pemrograman. Sorting atau pengurutan ini adalah proses mengatur sekumpulan objek menurut urutan atau susunan tertentu. Adanya kebutuhan akan pengurutan melahirkan beberapa macam pengurutan. Metode-metode pengurutan antara lain, yaitu Bubble Sort, Selection Sort (Maximum dan Minimum sort), Insertion Sort, Heap Sort, Shell Sort, Quick Sort, Merge Sort, Radix Sort, Tree Sort. Masing-masing dari metode pengurutan ini mempunyai kelebihan dan kelemahan. Pada suatu masalah pengurutan dapat dipakai berbagai macam metode. Namun, efisiensi suatu algoritma sorting tetap harus dipertimbangkan.

Efisiensi di dalam algoritma sangat dipertimbangkan, algoritma yang baik adalah algoritma yang efisien dimana algoritma tersebut dikatakan bagus karena dinilai dari aspek kebutuhan waktu yang singkat dan ruang yang kecil. Metode Selection Sort, Insertion Sort, dan Bubble Sort memiliki rumus kompleksi waktu yang sama, namum hasil waktu yang dihasilkan berbeda satu sama lainnya, hal ini dipengaruhi oleh algoritma dari masing-masing metode selain itu juga dipengaruhi oleh data input dan memory yang diperlukan. Jumlah perbandingan juga sangat pempengaruhi waktu yang diperlukan masing-masing metode.

Kata kunci: *selection sort*, *insertion sort*, *bubble sort*, kompleksitas, *pseudocode*

PENDAHULUAN

Di antara semua tugas perhitungan yang dipelajari oleh para ahli komputer selama lebih dari 40 tahun, *sorting* muncul sebagai yang paling mendapat perhatian. Operasi pengurutan (*sorting*) merupakan operasi yang sangat banyak dilakukan dalam 'Bussiness Data Processing', dalam hal ini pengurutan dilakukan secara naik (*ascending*).

TEORI

Algoritma

Menurut Antony Pranata (2005), definisi algoritma yang mengacu pada Microsoft Bookshelf adalah urutan langkah untuk memecahkan masalah logika atau matematika.

Dalam banyak kasus, algoritma yang dilakukan tidak selalu berurutan. Terkadang harus memilih dua atau beberapa pilihan atau mengulangi beberapa langkah.

Efisiensi dan Algoritma

Efisiensi di dalam algoritma sangat dipertimbangkan karena suatu masalah dapat diselesaikan dengan berbagai macam cara. Algoritma yang bagus adalah algoritma yang efisien dimana algoritma tersebut dikatakan bagus karena dinilai dari aspek kebutuhan waktu yang singkat dan ruang yang kecil.

Keefisienan algoritma dapat diukur dari orde yang terdapat dalam persamaan kompleksitas waktu. Kompleksitas waktu diukur dari jumlah tahapan komputasi yang dibutuhkan dalam menjalankan algoritma dimana kompleksitas waktu tersebut merupakan fungsi dari jumlah masukan N.

Pengurutan (*Sorting*)

Menurut Ir. P. Insap Santosa, M.Sc. (2001), pengurutan data (*sorting*) secara umum bisa didefinisikan sebagai satu proses untuk menyusun kembali himpunan objek menggunakan aturan tertentu. Secara umum ada dua jenis pengurutan data, yaitu pengurutan secara urut naik (*ascending*), yaitu dari data yang nilainya paling kecil sampai data yang nilainya paling besar, atau pengurutan data secara turun (*descending*) yaitu dari data yang nilainya paling besar sampai data yang nilainya paling kecil.

Data yang harus diurutkan tentunya sangat bervasiasi baik dalam hal banyak data maupun jenis data yang akan diurutkan. Sayangnya, tidak ada satu algoritma yang terbaik untuk menentukan algoritma mana yang paling baik untuk situasi tertentu karena ada beberapa faktor yang mempengaruhi efektifitas algoritma pengurutan data.

Tujuan pengurutan data adalah untuk lebih mempermudah pencarian data di kelak kemudian hari. Misalnya dalam buku telepon atau kamus bahasa, mudah untuk dibetulkan, disisipkan, atau digabungkan. Dalam keadaan terurut juga akan lebih mudah untuk mengecek apakah ada data yang hilang atau tidak.

Seperti dijelaskan sebelumnya pemilihan algoritma sangat ditentukan oleh struktur datanya. Jika seseorang diminta untuk membuat algoritma sendiri tentang *sorting*, kemungkinan besar akan membuat sebuah algoritma yang mirip dengan *selection sort* dan *insertion sort*. Oleh sebab itu yang dibahas dalam karya tulis ini adalah analisis tentang kecepatan algoritma *selection sort* dan *insertion sort*, karena dianggap mudah untuk diimplementasikan.

Hal-hal yang mempengaruhi kompleksitas waktu adalah sebagai berikut:

1. jumlah masukan data untuk suatu algoritma (n);
2. waktu yang dibutuhkan untuk menjalankan algoritma tersebut; dan

3. ruang memori yang dibutuhkan untuk menjalankan algoritma yang berkaitan dengan struktur data dari program.

Kompleksitas mempengaruhi performa atau kinerja dari suatu algoritma. Kompleksitas dibagi menjadi 3 jenis, yaitu *worst case*, *best case*, dan *average case*. Masing-masing jenis kompleksitas ini menunjukan kecepatan atau waktu yang dibutuhkan algoritma untuk mengeksekusi sejumlah kode.

PEMBAHASAN

Selection Sort

Selection sort merupakan algoritma *sorting* yang mudah diimplementasikan. Perulangan tidak tergantung dari data *array*. Metode pengurutan *selection sort* prosedur atau algoritmanya sebagai berikut:

1. Pengecekan dimulai dari data ke-1 hingga data ke-n;
2. Tentukan bilangan dengan indeks terkecil dari data bilangan tersebut;
3. Tukar bilangan dengan indeks terkecil tersebut dengan bilangan pertama ($i = 1$) dari data bilangan tersebut; dan
4. Lakukan langkah 2 dan 3 untuk bilangan berikut ($i = i + 1$), sampai didapatkan urutan yang benar.

Pseudocode 1: Algoritma *selection sort*

```
for i := 1 to Max - 1 do begin
   Pos := 1;
   for j := 1 to Max do
      if Data[j] < Data[Pos] then Pos := j;
   if i <> Pos then
      TukarData(Data[i], Data[Pos]);
end;
```

Dalam metode ini banyaknya pembandingan (untuk mencari elemen dengan nilai terkecil) adalah sebesar:

$$C = \frac{N^2 - N}{2} \quad \text{.....(1)}$$

Hasil pengurutan data pada metode ini tergantung terhadap data nilai ekstrim yang diinginkan, dengan demikian, total waktu yang dibutuhkan adalah:

$$C \times N \times N = CN^2 = \mathrm{O}(n^2) \quad \text{.....(2)}$$

Insertion Sort

Merupakan salah satu algoritma yang paling sederhana. Metode ini banyak digunakan oleh permainan kartu. Dalam metode ini elemen-elemen

(kartu) terbagi menjadi dua kelompok. Kelompok pertama, yaitu kelompok tujuan (kartu berada di tangan pemain dan sudah dalam keadaan urut). Dan kelompok kedua adalah kelompok sumber (kartu yang berada di atas meja dan belum dalam keadaan urut). Dalam setiap langkah, dimulai dari kartu kedua, kartu akan diambil dari kelompok sumber dan akan dipindah ke kelompok tujuan dengan cara menyisipkan langsung ke tempat yang sesuai secara urut.

Algoritma *insertion sort* pada dasarnya memilah data yang akan diurutkan menjadi dua bagian, yang belum diurutkan (kelompok sumber) dan yang sudah diurutkan (kelompok tujuan). Elemen pertama diambil dari bagian *array* yang belum urut dan kemudian diletakkan sesuai posisinya pada bagian lain *array* yang telah diurutkan. Langkah ini dilakukan terus menerus hingga tidak ada lagi elemen yang belum diurutkan.

Algoritmanya adalah sebagai berikut:

1. pengecekan dimulai dari data ke-1 sampai dengan data ke-n;
2. pengurutan dilakukan dengan cara membandingkan data ke-i (dimulai dari data ke-2 sampai dengan data terakhir);
3. bandingkan data ke-i tersebut dengan data sebelumnya ($i-1$). Jika lebih kecil maka data tersebut dapat disisipkan ke data awal (depan) sesuai dengan posisi yang seharusnya; dan
4. lakukan langkah ke-2 dan ke-3 untuk bilangan selanjutnya ($i = i+1$) sampai didapatkan urutan yang optimal.

Pseudocode 2: Algoritma *insertion sort*

```
for i := 2 to Max do begin
   Temp := Data[i];
   j := j - 1;
   while (Data[j] > Temp) and (j > 0) do begin
      Data[j + 1] := Data[j];
      Dec(j);
   end;
   Data[j + 1] := Temp;
end;
```

Dalam metode ini banyaknya perandingan (C) untuk nilai i tertentu adalah sebanyak $i-1$ kali, dengan paling sedikit satu kali, dan rata-rata sebesar $i/2$ kali. Banyaknya pemindahan atau penggeseran (M) untuk nilai i tertentu adalah $C_i + 2$ (termasuk sentinel). Dengan demikian banyaknya pembandingan dan pemindahan adalah:

- *Worst case*:

$$C = N - 1, M = 2(N - 1) \quad \text{.....(3)}$$

- *Average case*:

$$C = \frac{N^2 + N + 2}{4}, M = \frac{N^2 + 9N - 10}{4} \quad \text{.....(4)}$$

- *Best case*:

$$C = N^2 + N - 2, M = \frac{N^2 + 3N - 4}{2} \quad \text{.....(5)}$$

Insertion sort lebih sederhana dan lebih baik digunakan untuk jumlah data yang sangat kecil. Untuk jumlah data yang besar, metode ini tidak cocok untuk diterapkan kerena kompleksitas waktu dari algoritma tersebut adalah:

$$C \times N \times N = CN^2 = \mathrm{O}(n^2) \quad \text{.....(6)}$$

Bubble Sort

Metoda gelembung (*bubble sort*), sering juga disebut dengan metode pertukaran (*exchange sort*), merupakan metoda yang paling sederhana. Proses pencarian solusi dilakukan dengan cara *brute force*, langsung ke intinya, yaitu membandingkan elemen-elemen yang ada di dalam tabel. Metoda ini mendasar pada pertukaran dua buah elemen untuk mencapai keadaan urut yang diinginkan. Metoda ini cukup mudah untuk dipahami dan diprogram. Tetapi dari beberapa metoda yang telah dibahas sebelumnya, metoda ini merupakan metoda yang paling tidak efisien.

Untuk membawa vektor dalam keadaan urut dapat dilakukan dengan dua cara. Cara yang pertama adalah selalu meletakkan elemen dengan nilai paling besar pada posisi terakhir (posisi ke-N). Kemudian elemen dengan nilai paling besar kedua diletakkan pada posisi ke-($N-1$), dan seterusnya. Cara kedua adalah kebalikan cara pertama. Dengan kata lain, pada iterasi pertama akan diletakkan elemen dengan nilai terkecil pada posisi ke-1, kemudian elemen dengan nilai terkecil kedua diletakkan pada posisi ke-2, dan seterusnya.

Algoritmanya adalah sebagai berikut:

1. pengecekan dimulai dari data ke-1 sampai data ke-N;
2. bandingkan data ke-N dengan data sebelumnya ($N-1$) dan jika lebih kecil tukar bilangan tersebut dengan data yang ada di depannya (sebelumnya) satu per satu ($N-1$, $N-2$, $N-3$, ...); dan
3. lalukan perulangan pada langkah 2 sampai didapat urutan yang optimal.

Pseudocode 3: Algoritma *bubble sort*

```
for i := 1 to N - 1 do
   for j := 1 to N - i do
      if x[j] > x[j + 1] then
         Tukar(x[j], x[j+1])
```

Algoritma ini merupakan metode yang paling tidak efisien. Alasannya adalah bahwa apabila mengurutkan vektor sebanyak N elemen dan pada iterasi yang kurang dari $N-1$, maka iterasi tersebut harus tetap dilaksanakan sampai $N-1$. Dengan demikian, dalam metoda gelembung akan terjadi perbandingan dan pemindahan atau pertukaran dua elemen sebanyak:

- *Worst case*:

$$M = 0 \quad \text{..... (7)}$$

- *Average case*:

$$M = \frac{3(N^2 - N)}{4} \quad \text{..... (8)}$$

- *Best case*:

$$M = \frac{3(N^2 - 4)}{2} \quad \text{..... (9)}$$

Dan kompleksitas waktu yang diperlukan adalah:

$$C = \frac{N^2 - N}{2} = \mathrm{O}(n^2) \quad \text{..... (10)}$$

HASIL PERBANDINGAN ALGORITMA *SORTING*

Tiap algoritma *sorting* yang telah dibahas memiliki kelebihan dan kekurangan satu sama lain.

- *Bubble sort*:
  1) algoritma singkat;
  2) metode paling sederhana dan kuat; dan
  3) waktu kompleksitas yang sama untuk semua kasus.
- *Insertion sort*:
  1) kompleksitas relatif lebih kecil;
  2) mudah membagi masalah, tapi sulit menggabungkan kembali; dan
  3) membutuhkan metoda tambahan.
- *Selection sort*:
  1) kompleksitas relatif lebih kecil;

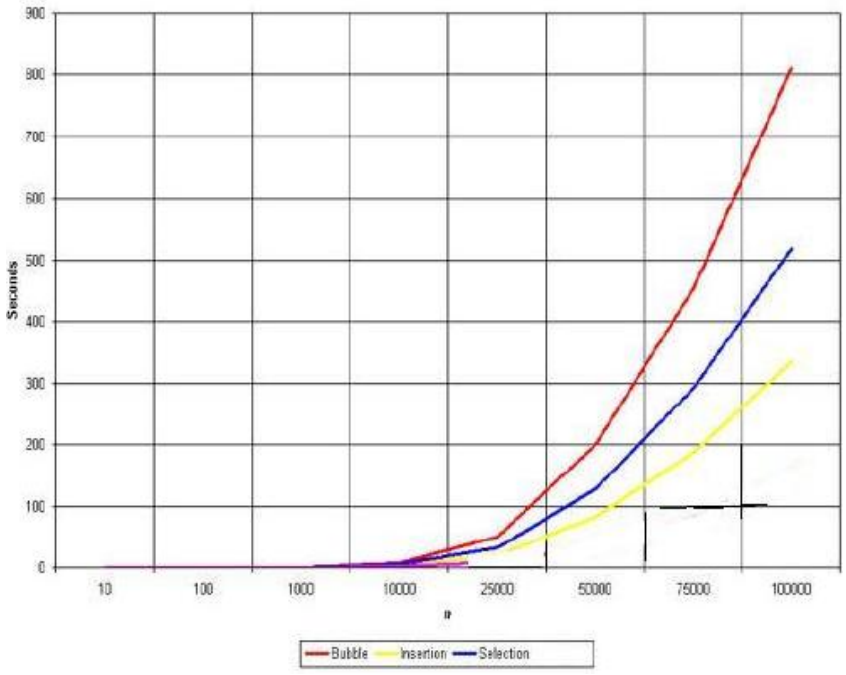
2) mudah menggabungkan kembali, tapi sulit membagi masalah; dan
3) membutuhkan metoda tambahan.

Walaupun tiap algoritma *sorting* menawarkan perbedaan metode dan sudut pandang penyelesaian masalah yang berbeda-beda, kompleksitas waktu yang diperlukan tetap menjadi masalah utama yang dipertimbangkan untuk menentukan algoritma mana yang lebih baik dan tepat untuk digunakan. Tabel 1 menunjukan perbandingan kompleksitas algorita-algoritma *sorting* yang telah dibahas dalam berbagai kasus.

Tabel 1: Tabel kompleksitas waktu *selection sort*, *insertion sort*, dan *bubble sort*

Nama Metode Sorting	Best Case	Average Case	Worst Case	Metode yang Digunakan
Selection Sort	N^2	N^2	N^2	Seleksi
Insertion Sort	N^2	N^2	N^2	Penyisipan
Bubble Sort	N^2	N^2	N^2	Pertukaran

Dari hasil perbandingan kompleksitas waktu yang diperlukan oleh *selection sort*, *insertion sort*, dan *bubble sort* ternyata memiliki kompleksitas waktu yang sama, namun pada uji coba yang dihasilkan ternyata waktu yang diperlukan oleh masing-masing metode *sort* tersebut menghasilkan waktu yang berbeda-beda. Grafik hasil perbandingan kompleksitas waktu dari *selection sort*, *insertion sort*, dan *bubble sort* dapat dilihat pada gambar 1.



Gambar 1: Grafik kompleksitas waktu *selection sort*, *insertion sort*, *bubble sort*

KESIMPULAN

Dari hasil perbandingan ketiga *sorting* yaitu *selection sort*, *insertion sort*, dan *bubble sort* yang memiliki rumus kompleksitas waktu yang sama, ternyata dihasilkan waktu proses *sorting* yang berbeda-beda. Beberapa faktor yang sangat mempengaruhi perbedaan waktu tersebut adalah sebagai berikut:

1. Data yang diinputkan. Jika data sudah dalam keadaan urut secara *ascending*, maka merupakan *best case*. Jika data dalam keadaan acak atau *random*, maka ada kemungkinan merupakan *average case* karena tidak harus membandingkan seluruh data. Namun jika data dalam keadaan urut secara *descending* maka akan terjadi *worst case*.
2. Selain data input, jumlah perbandingan juga sangat mempengaruhi. Seperti halnya *bubble sort* harus membandingkan tiap data secara keseluruhan dan akan diulang secara terus menerus sampai keadaan urut. Untuk *selection sort* tidak demikian data pertama yang sudah urut tidak akan dibandingkan lagi pada langkah berikutnya. Demikian juga dengan *insertion sort*.

DAFTAR PUSTAKA

Iradewa, Rokhmatun D, 2009, *Analisis Algoritma pada Masalah Sorting*, http://www.slideshare.net/iradewa/analisis-algoritma-pada-masalah-sorting.

Pranata, Antony, 2005, *Algoritma dan Pemrograman*, Graha Ilmu, Yogyakarta.

Rahmat, Antonius C, 2009, *Struktur Data*, http://flecture.ukdw.ac.id.

Ronny, 2009, *Studi Mengenai Perbandingan Sorting Algorithmics dalam Pemrograman dan Kompleksitasnya*, http://www.informatika.org.

Santoso, Insap, 2001, *Struktur Data*, Andi Offset, Yogyakarta.

_____, 2009, *Selection Sort*, http://en.wikipedia.org/wiki/selection_sort.