Kompleksitas Algoritma Dalam Algoritma Pengurutan

Rio Cahya Dwiyanto
Jurusan Teknik Informatika ITB, Bandung, email: kasrut_desu@yahoo.co.id

Abstract *– Makalah ini membahas tetang beberapa algoritma, terutama algoritma pengurutan (sorting. .Dan pada setiap algoritma akan dibahas tentang kompleksitas algoritma tersebut dalam hal ini big O. Dengan mengetahui big O inilah kita semua diharapkan dapat menilai kekompleksitasan suatu algoritma dan dengan mengetahui kompleksitas algoritma tersebut kita diharap dalam masa mendatang bisa membuat sebuah atau mungkin beberapa buah algoritma, tidak hanya algoritma pengurutan tapi juga algoritma yang lain yang merupakan algoritma yang mangkus. Beberapa algoritma yang disebutkan disini hanya sebuah teori tapi saya yakin algoritma ini dapat dibuat dan dijalankan dengan benar. Algoritma pengurutan yang dibahas kali ini adalah pengurutan data dengan menggunakan metode* Bubble Sort, Insertion Sort, Selection Sort, Merge Sort *dan* Quick Sort.

Kata Kunci *: Kompleksitas, Algoritma, Pengurutan (Sorting).*

1. PENDAHULUAN

Algoritma memegang peranan penting dalam bidang pemrograman. Sebegitu pentingnya suatu algoritma, sehingga perlu dipahami konsep dasar algoritma. Apalagi untuk seorang programer, tentu diperlukan suatu algoritma sehingga dapat membuat program yang lebih efektif dan efisien. Bagi kebanyakan orang, algoritma sangat membantu dalam memahami konsep logika pemrograman.

Algoritma adalah kumpulan instruksi yang dibuat secara jelas untuk menunjukan langkah-langkah penyelesaian suatu masalah. Pada umumnya algoritma kurang lebih sama dengan suatu prosedur yang sering dilakukan setiap hari, misalnya prosedur untuk mengganti ban bocor/pecah, prosedur pemakaian telepon umum, prosedur membuat kue dan lain-lain.

Dalam bidang komputer, misalnya EDP (*Elektronik Data Processing*) atau MIS (*Management Information System*), algoritma sering dimanfaatkan untuk menyelesaikan suatu masalah atau untuk proses pengambilan keputusan. Seorang sistem analisis (*analisist system*) tentunya menggunakan algoritma untuk merancang suatu sistem. Bagi seorang programer, algoritma digunakan untuk membuat modul-modul program.

Guna memahami suatu algoritma, harus dimiliki pengetahuan dasar matematika karena pada dasarnya algoritma lahir dari konsep logika matematika. Disini yang perlu dilatih adalah kemampuan logikanya agar benar-benar bisa menyusun langkah-langkah penyelesaian masalah dengan baik.

Dalam makalah ini, disajikan konsep dasar dan analisis algoritma, terutama algoritma pengurutan. Untuk mendapatkan algoritma yang efisien serta mendapatkan rumusan matematika sebagai ukuran kerumitan (kompleksitas) maka dibahas analisis algoritma dengan menggunakan notasi O (big O).

1.1 Analisis Algoritma dan Kompleksitas Algoritma

Seorang programer atau sistem analisis paling tidak harus memiliki dasar untuk menganalisis algoritma. Analisis algoritma sangat membantu di dalam meningkatkan efisiensi program. Kecanggihan suatu program bukan dilihat dari tampilan program, tetapi berdasarkan efisiensi algoritma yang terdapat didalam program tersebut.

Pembuatan program komputer tidak terlepas dari algoritma, apalagi program yang dibuat sangat kompleks. Program dapat dibuat dengan mengabaikan algoritma, tetapi jangan heran bila seandainya ada orang lain yang membuat program seperti program anda tersebut memiliki akses yang lebih cepat dan memakai memori yang sangat sedikit.

Analisis algoritma adalah bahasan utama dalam ilmu komputer. Dalam menguji suatu algoritma, dibutuhkan beberapa kriteria untuk mengukur efisiensi algoritma. Terdapat dua tipe analisis algoritma, yaitu :
1. Memeriksa kebenaran algoritma
Dapat dilakukan dengan cara perurutan, memeriksa bentuk logika, implementasi algoritma, pengujian dengan data dan menggunakan cara matematika untuk membuktikan kebenaran.
2. Penyederhanaan Algoritma
Membagi algoritma menjadi bentuk yang sederhana.

1.2 Notasi O (Big O)

Misalkan 4 program yang mensorting n bilangan dengan fungsi yang menyatakan sejumlah langkah yang dijumlahkan masing-masing program untuk sorting n bilangan :
f1(n) = n, f2(n) = n2, f3 (n) = 2n, f4(n) = n!
Bila n = 4 maka f1 (n) = 4, f2 (n) = f3 (n) = 16 dan f4 (n) = 24 sedangkan untuk n = 100, program ketiga akan memerlukan 21000 langkah.

Dalam analisis sebuah algoritma biasanya yang dijadikan ukuran adalah operasi aljabar seperti penjumlahan, pengurangan, perkalian dan pembagian, proses pengulangan (looping/Iterasi), proses pengurutan (sorting) dan proses pencarian (searching).

1.3 Algoritma Pengurutan

Sorting atau pengurutan adalah masalah klasik dalam dunia ilmu komputer. Untungnya kompleksitas dari sorting hanya sampai pada taraf P (Polynomial) dan bukan NP (Non-deterministic Polynomial). Mengapa saya membahas tentang algoritma sorting dan bukan algoritma yang lain? Karena menurut saya algoritma ini termasuk mudah dan sangat menarik untuk ditelaah dan dipelajari.

Dalam makalah inipun saya hanya membahas tentang beberapa buah sorting atau pengurutan. Yaitu *Bubble Sort, Insertion Sort, Selection Sort, Merge Sort* dan *Quick Sort*. *Bubble Sort* adalah algoritma pengurutan sederhana, yang bekerja dengan cara menunjuk tiap anggota list untuk diurutkan, membandingkan dua elemen pada suatu waktu dan menukarkan tempat mereka jika tidak sesuai keterurutan. Proses ini berlangsung hingga tidak ada lagi yang perlu dipertukarkan, yang berarti list sudah dalam kondisi terurut. Algoritma ini mendapat namanya dari cara elemen terkecil ”mengambang” (*Bubble*) pada atas list. Karena hanya menggunakan perbandingan pada elemen-elemennya, maka sering di sebut pengurutan perbandingan.

Sedangkan *Quick Sort* sendiri adalah sbuah algoritma terkenal yang dikembangkan oleh C. A. R. Hoare. Biasanya *Quicksort* lebih cepat dibanding algoritma pengurutan yang lain karena loop dalamnya secara mangkus dapat diterapkan pada sebagian besar arsitektur dan hampir setiap data memungkinkan untuk membuat pilihan desain yang memungkinkan meminimalisasi penggunaan waktu secara kuadratik. *Quicksort* adalah pengurutan perbandingan dan, dalam penerapan yang mangkus, bukan pengurutan stabil.

Definisi dari beberapa algoritma pengurutan tersebut akan dibahas pada bagian bab Pembahasan Algoritma Pengurutan.

Beberapa algoritma yang disebutkan disini hanya sebuah teori tapi saya yakin algoritma ini dapat dibuat dan dijalankan dengan benar.

2. PEMBAHASAN ALGORITMA PENGURUTAN (SORTING)

2.1 Pengurutan data dengan menggunakan metode Bubble Sort

Ide dari algoritma ini sangat menarik. Setiap pasangan data: x[j] dengan x[j-1], untuk semua i=1,...,n-1 harus memenuhi keterurutan, yaitu x[j] > x[j-1]. Apabila tidak memenuhi maka posisi kedua data harus ditukar. Untuk pemrograman konvensional maka pemeriksaan-pemeriksaan pasangan tersebut harus dilakukan satu demi satu, misalnya oleh bubble-sort dilakukan dari kanan ke kiri serta di dalam sejumlah iterasi. Pada iterasi ke-i, pemeriksaan tsb. dilakukan pula dalam Loop-for sbb:

```
for (j=n-1; j > i; j--) {
if (x[j] < x[j-1]) {
tmp = x[j];
x[j] = x[j-1];
x[j-1] = tmp;
}
}
```

Loop-for tersebut akan menggeser bilangan terkecil ke posisi i. Loop-for dilakukan hanya sampai ke i karena pada iterasi ke-i data dalam x[0], x[1], ..., x[i-1] merupakan yang paling kecil dan sudah terurut hasil pengeseran yang dilakukan setiap loop sebelumnya. Oleh sebab itu iterasi hanya dilakukan untuk harga i=0, 1, ..., n-2 atau sampai tidak terjadi penukaran dalam suatu iterasi.

```
void BubbleSort() {
ch = true;
for (i=0; i < n-2 && ch; i++) {
ch = false;
for (j=n-1; j > i; j--) {
if (x[j] < x[j-1]) {
tmp = x[j];
x[j] = x[j-1];
x[j-1] = tmp;
ch = true;
}
}
}
}
```

Contoh untuk mengurutkan data
34,43,65,90,48,82,93,86,26,76,49,23,56,37.
Pada iterasi i=0:
j=13: tukar 56-37 menjadi
34,43,65,90,49,82,93,86,26,76,49,23,37,56
j=12: tidak ada penukaran
j=11: tukar 49-23 menjadi
34,43,65,90,48,82,93,86,26,76,23,49,37,56
j=10: tukar 76-23 menjadi
34,43,65,90,48,82,93,86,26,23,76,49,37,56
j= 9: tukar 26-23 menjadi
34,43,65,90,48,82,93,86,23,26,76,49,37,56
j= 8: tukar 86-23 menjadi
34,43,65,90,48,82,93,23,86,26,76,49,37,56
j= 7: tukar 93-23 menjadi
34,43,65,90,48,82,23,93,86,26,76,49,37,56
j= 6: tukar 82-23 menjadi
34,43,65,90,48,23,82,93,86,26,76,49,37,56

j= 5: tukar 49-23 menjadi
34,43,65,90,23,48,82,93,86,26,76,49,37,56
j= 4: tukar 90-23 menjadi
34,43,65,23,90,48,82,93,86,26,76,49,37,56
j= 3: tukar 65-23 menjadi
34,43,23,65,90,48,82,93,86,26,76,49,37,56
j= 2: tukar 43-23 menjadi
34,23,43,65,90,48,82,93,86,26,76,49,37,56
j= 1: tukar 34-23 menjadi
23,34,43,65,90,48,82,93,86,26,76,49,37,56

Pada iterasi i=1:
j=13: tidak ada penukaran
j=12: tukar 49-37 menjadi
23,34,43,65,90,48,82,93,86,26,76,37,49,56
j=11: tukar 76-37 menjadi
23,34,43,65,90,48,82,93,86,26,37,76,49,56
j=10: tidak ada penukaran
j= 9: tukar 86-26 menjadi
23,34,43,65,90,48,82,93,26,86,37,76,49,56
j= 8: tukar 93-26 menjadi
23,34,43,65,90,48,82,26,93,86,37,76,49,56
j= 7: tukar 82-26 menjadi
23,34,43,65,90,48,26,82,93,86,37,76,49,56
...
j= 2: tukar 34-26 menjadi
23,26,34,43,65,90,48,82,93,86,37,76,49,56

Dan seterusnya. Hingga, pada setiap akhir iterasi berikutnya berturut-turut menjadi:
i=2: 23,26,34,37,43,65,90,48,82,93,86,49,76,56
i=3: 23,26,34,37,43,48,65,90,49,82,93,86,56,76
i=4: 23,26,34,37,43,48,49,65,90,56,82,93,86,76
i=5: 23,26,34,37,43,48,49,56,65,90,76,82,93,86
i=6: 23,26,34,37,43,48,49,56,65,76,90,82,86,93
i=7: 23,26,34,37,43,48,49,56,65,76,82,90,86,93
i=8: 23,26,34,37,43,48,49,56,65,76,82,86,90,93
i=9: 23,26,34,37,43,48,49,56,65,76,82,86,90,93

Dari kedua iterasi dengan increment linear demikian dapat mudah dilihat bahwa algoritma ini memiliki kompleksitas $O(n^2)$ dan proses akan menjadi amat cepat kalau data sudah sebagian besar terurut. Masalah yang dihadapi algoritma ini adalah banyaknya penukaran yang dilakukan selama proses dalam kondisi normal. Efisiensi dapat ditingkatkan dengan mengurangi proses penukaran dengan penggeseran dan penyisipan seperti halnya insertion sort. Untuk lingkungan pemrograman paralel dengan array processor pengurutan ini menjadi amat menarik untuk dikaji.

Jika Bubble Sort dalam setiap iterasi melakukan loop-for penukaran ke satu arah, Shaker Sort (suatu variant dari Bubble Sort) melakukan loop-for penukaran dengan arah bolak-balik dengan batas loop-for kiri dan kanan semakin menyempit.

Untuk bubble sort, best case dari Big-O (O) adalah O(n), dimana n adalah jumlah datanya. Best case terjadi bila data yang hendak disorting sudah terurut, misal jika ada 1500 data dan semuanya sudah terurut maka algoritma Bubble Sort tersebut hanya melewatinya satu kali yaitu O(1500).

Akan tetapi, worst case nya adalah $O(n^2)$, worst case terjadi apabila datanya betul-betul sangat tidak terurut, bayangkan bila ada 20 data, pada worst case komputasi yang dilakukan = $O(20^2)$ = O(4000) -> Lebih besar dibanding 1500 data pada kondisi best case!

2.2 Pengurutan data dengan menggunakan metode *Quick Sort*

Ide dari algoritma ini adalah secara rekursif membagi (atau mempartisi) data set ke dalam dua sub data set; kita sebut sub data set kiri dan sub data set kanan. Partisi ini dilakukan dengan kriteria:
- digunakan salah satu data sebagai pivot value
- sub data set kiri adalah data yang berharga <= pivot value
- sub data set kanan adalah data yang berharga > pivot value

Jika data set berada dalam array X berindeks dari l sampai dengan r maka pembagian ini mempertukarkan sejumlah isi array sehingga sub data set kiri berada dalam array yang sama berindeks l sampai dengan t-1 dan sub data set kanan berada dalam array berindeks t +1 sampai dengan r. X[t] sendiri ditempati oleh pivot. Setelah dilakukan pembagian tersebut maka algoritma Quick Sort diaplikasikan untuk masing-masing sub data set tsb dan seterusnya secara rekursif hingga terbentuk sub data set yang tidak dapat dibagi lagi yaitu jika hanya berisi satu sel (l = r). Bagian yang paling tricky adalah pada pembagian data set, kita sebut fungsi tersebut adalah Partition(l,r) yang menghasilkan t yaitu posisi pivotnya. Maka, algoritma Quick Sort adala sebagai berikut.

```
void QuickSort(int l,int r) {
if (l < r) {
t = Partition(l,r);
QuickSort(l,t-1);
QuickSort(t+1,r);
}
}
```

Proses Partisi diserahkan kepada anda untuk mempelajarinya sendiri. Dalam beberapa literatur terdapat variant-varuant yang pada dasarnya terjadi karena perbedaan cara menentukan harga pivot: bisa data pertama (X[l]), data terakhir (X[r]) atau data ditengah (X[(l+r)/2]) data set).
Kompleksitas algoritma Quick Sort adalah bergantung pada hasil partisi tersebut. Kondisi best case adalah jika dalam setiap partisi tidak dilakukan operasi pertukaran tempat dan pivot tepat berada ditengah

subset (O(n)). Kondisi average case adalah jika partisi menghasilkan sub data set yang balance (o(n log n)). Kondisi worse case adalah jika partisi menghasilkan sub data set kosong dan yang lain besar ($o(n^2)$). Walaupun demikian, algoritma ini cenderung untuk average case.

2.3 Pengurutan data dengan menggunakan metode *Insertion Sort*

Insertion Sort adalah sebuah algoritma pengurutan sederhana, dengan cara pengurutan perbandingan yang mengurutkan suatu array atau list dengan membuat suatu entry. Metode ini tidak begitu mangkus dalam mengolah data yang besar (banyak) daripada algoritma-algoritma yang lebih maju seperti *Quicksort*, tetapi mempunyai banyak keuntungan, diantaranya :

1. Sederhana dalam penerapan
2. Mangkus dalam pengolahan data yang kecil
3. Mangkus dalam data yang sudah sebagian terurut
4. Lebih mangkus dibanding *Bubble Sort* maupun *Selection Sort*
5. Stabil

Dalam Best case yang berarti list sudah dalam keadaan terurut, maka pengurutan ini hanya memakan waktu O(n): dalam setiap iterasi, elemen pertama list hanya dibandingkan dengan elemen terakhir dari list. Jika list sudah terurut atau sebagian terurut maka *Insertion Sort* akan lebih cepat dibandingkan dengan *Quicksort*.

Worst case dari algoritma ini adalah jika list terurut terbalik sehingga setiap eksekusi dari perintah harus memindai dan mengganti seluruh bagian sebelum menyisipkan elemen berikutnya. *Insertion Sort* membutuhkan waktu $O(n^2)$ pada worst case ataupun average case yang membuat algoritma ini tidak cocok dalam pengurutan elemen dalam jumlah besar. Walaupun demikian, loop dalam pada *Inserion Sort* sangat cepat, yang membuatnya salah satu algoritma pengurutan tercepat pada jumlah elemen yang sedikit.

2.4 Pengurutan data dengan menggunakan metode *Selection Sort*

Selection Sort adalah suatu algoritma pengurutan, lebih tepatnya lagi pengurutan perbandingan statis. Karena mempunyai kompleksitas $O(n^2)$ maka tidak begitu mangkus dalam pengolahan data dengan jumlah besar, dan biasanya lebih buruk performanya dibanding dengan *Insertion Sort* yang mirip. *Selection Sort* diakui karena kesederhanaan algoritmanya dan performanya lebih bagus daripada algoritma lain yang lebih rumit dalam situasi tertentu.

Algoritma ini bekerja sebagiar berikut:

1. Mencari nilai minimum dalam sebuah list
2. Menukarkan nilai ini dengan elemen pertama list
3. Mengulangi langkah di atas untuk sisa list dengan dimulai pada posisi kedua

Secara mangkus kita membagi list menjadi dua bagian yaitu bagian yang sudah diurutkan, yang didapat dengan membangun dari kiri ke kanan dan dilakukan pada saat awal, dan bagian list yang elemennya akan diurutkan.

Selection Sort tidak begitu sulit untuk manganalisis pembanding dibanding algoritma yang lain karena tidak ada loop yang bergantung pada data pada list. Memilih elemen terkecil memerlukan pemindaian seluruh n elemen (hal ini membutuhkan n-1 perbandingan) lalu memindahkannya ke posisi pertama. Memilih elemen kedua terkecil memerlukan pemindaian dari n-1 elemen dan begitu seterusnya sehingga $(n-1) + (n-2) + ... + 2 + 1 = n(n-1)/2 = O(n^2)$ perbandingan. Tiap-tiap pemindaian itu memerlukan satu kali pertukaran untuk n-1 elemen (dengan elemen terakhir sudah pada posisinya). Karena itu perbandingan mendominasi waktu pemrosesan yaitu $O(n^2)$.

2.5 Pengurutan data dengan menggunakan metode *Merge Sort*

Merge Sort adalah suatu algoritma pengurutan berbasis perbandingan. Algoritma ini stabil yang berarti menjaga keteraturan input sehingga sama dengan elemen keluaran yang sudah diurutkan. Algiritma ini merupakan salah satu contoh algoritma pembagian. Algoritma ini ditemukan oleh John von Neumann pada tahun 1945.

Algoritma ini bekerja secara:

1. Membagi list menjadi dua bagian sama besar
2. Membagi lagi list-list tersebut sehingga didapat list dengan ukuran satu elemen, dalam kasus ini list tersebut dikembalikan
3. Menggabungkan list-list tersebut menjadi satu list kembali

Misalkan kita mempunya list A yang berkisar dari A' pertama hingga A' terakhir. Kita terapkan *Mergesort* sehingga didapat 2 buah sublist (A' pertama ... A' tengah) dan (A' tengah+1 … A' terakhir) dimana tengah adalah bagian integer dari (A' pertama + A' terakhir)/2. Setelah itu direkursifkan hingga elemen terkecil dan digabungkan kembali menjadi 1 list.

Dalam pengurutan n data, *Mergesort* mempunyai performa average dan worst case O(n log n). Jika waktu proses dari *Mergesort* pada list dengan panjang n adalah T(n), maka rekurensnya $T(n) = 2T(n/2) + n$ mengikuti definisi dari algoritma.

Pada worst case, *Mergesort* melakukan kira-kira ($n \log n - 2^{\log n} + 1$) perbandingan, yaitu antara ($n \log n - n + 1$) dan ($n \log n + n + O(\log n)$).

2.6 Speedup

Pengukuran speedup sebuah algoritma adalah salah satu cara untuk mengevaluasi kinerja algoritma tersebut.

Speedup adalah perbandingan antara waktu yang diperlukan algoritma sekuensial yang paling efisien untuk melakukan komputasi dengan waktu yang dibutuhkan untuk melakukan komputasi yang sama pada sebuah mesin pipeline atau paralel.

Speedup = Worstcase running time dari algoritma sekuensial terefisien dibagi Worstcase running time dari algoritma paralel.

3. HASIL DAN PEMBAHASAN

Jadi, dari hasil analisis tersebut dapat diketahui masing-masing kompleksitas algoritma (dalam hal ini big O) dari masing-masing algoritma pengurutan sebagai berikut:

	Best case	Worst case
Bubble Sort	O(n)	$O(n^2)$
Quick Sort	O(n)	O(n log n)
Insertion Sort	O(n)	$O(n^2)$
Selection Sort	$O(n^2)$	$O(n^2)$
Merge Sort	-	O(n log n)

4. KESIMPULAN

1. Suatu algoritma tidak bisa dikatakan lebih mangkus daripada algoritma yang lain. Kemangkusan algoritma didasarkan pada banyak atau sedikitnya data dan penerapannya.
2. Pemilihan algoritma yang tepat adalah salah satu syarat programer yang baik. Karenanya programer harus memiliki banyak ”koleksi” dari berbagai macam algoritma dan mampu menerapkan suatu algoritma dengan baik dan benar.
3. Dengan mengukur kompleksitas algoritma (Big O) kita dapat menentukan secara kasar waktu proses yang diperlukan algoritma tersebut untuk menyelesaikan masalah dan juga kemangkusannya.
4. Suatu algoritma mempunyai kalabihan atau kekurangannya masing-masing. Untuk itulah programer yang baik harus dapat memilih suatu algoritma yang tepat pada suatu program.

DAFTAR REFERENSI

1[1] en.wikipedia.org/wiki/

2[2] www.balihack.or.id - Nubier Article

[3] Buku Ajar Metode Numerik, didanai oleh Proyek HEDS tahun 2002

[4] Analisis Kinerja Cluster Linux dengan Pustaka MPICH Terhadap Perkalian Matrix; Fani Fatullah, A.Beny Mutiara MQN, Chandra Yulianto