**Algoritma dan Komplekstas Algoritma**

****

**Oleh**

**Fadhila Dwi Kurniawan**

**E41182219**

**GOLONGAN E**

**PROGRAM STUDI TEKNIK INFORMATIKA**

**JURUSAN TEKNOLOGI INFORMASI**

**POLITEKNIK NEGERI JEMBER**

**2019**

Algoritma dan Kompleksitas Algoritma

1. Algoritma

Algoritma adalah urutan logis langkah-langkah penyelesaian masalah yang ditinjau secara sistematis. Penulisan algoritma dapat menggunakan kalimat deskriptif ataupun bahasa pemrograman. Penulisan algoritma yang bagus adalah algoritma yang pendek.

1. Efisiensi Algoritma

Efisiensi (kemangkusan) algoritma diukur dari berapa waktu dan jumlah ruang (space) memori yang dibutuhkan untuk menjalankan. Algoritma yang mangkus adalah algoritma yang meminimumkan kebutuhan ruang dan waktu.

1. Model Abstrak

Model abstrak pengukuran waktu/ruang harus independen dari pertimbangan mesin dan compiler manapun. Besaran yng dipakai untuk menerangkan model abstrak pengukuran ruang/waktu ini adalah kompleksitas algoritma.

Ada dua macam kompleksitas algoritma, yaitu :

1. Kompleksitas waktu T(n), diukur dari jumlah tahapan komputasi yang dibutuhkan untuk menjalankan algoritma sebagai fungsi dari ukuran masukan n.

Kompleksitas waktu dibedakan atas tiga macm, yaitu :

1. Tmax(n), yaitu kompleksitas waktu untuk kasus terburuk (worst case). Kebutuhan waktu maksimum.
2. Tmin(n), yaitu kompleksitas waktu untuk kasus terbaik (best case). Kebutuhan waktu minimum.
3. Targ(n), yaitu kompleksitas waktu untuk kasus rata-rata (average case). Kebutuhan waktu secara rata-rata.
4. Kompleksitas ruang S(n), diukur dari memori yang digunakan oleh struktur data yang terdapat di dalam algoritma sebagai fungsi dari masukan n.

Ukuran masukan (n) yaitu jumlah data yang diproses dalam sebuah algoritma. Misalnya pengurutan 1000 elemen larik, maka n=1000.

Contoh kompleksitas waktu :

1. Kompleksitas Waktu

Dalam praktek, kompleksitas waktu dihitung berdasarkan jumlah operasi abstrak yang *mendasari* suatu algoritma, dan memisahkan analisisnya dari implementasi.

**Contoh 2.**

Tinjau algoritma menghitung ratarata pada Contoh 1. Operasi yang mendasar pada algoritma tersebut adalah operasi penjumlahan elemen-elemen *ak* (yaitu jumlah=jumlah+ak),

Kompleksitas waktu HitungRataRata adalah  *T*(*n*) = *n.*

**Contoh 3.**

 Algoritma untuk mencari elemen terbesar di dalam sebuah larik (*array*) yang berukuran *n* elemen.

|  |
| --- |
| procedure CariElemenTerbesar(input a1, a2, …, an : integer, output maks : integer)  *{ Mencari elemen terbesar dari sekumpulan elemen larik integer a1, a2, …, an.*  *Elemen terbesar akan disimpan di dalam maks.*  *Masukan: a1, a2, …, an*  *Keluaran: maks (nilai terbesar)*  *}*  **Deklarasi**  k : integer    **Algoritma**  maks=a1  k=2  while k <= n do  if ak > maks then  maks=ak  endif  i-i+1  endwhile  *{ k > n }* |

Kompleksitas waktu algoritma dihitung berdasarkan jumlah operasi perbandingan elemen larik (A[i] > maks).

Kompleksitas waktu CariElemenTerbesar :  *T*(*n*) = *n* – 1.

Kompleksitas waktu dibedakan atas tiga macam :

1. *Tmax(n)* : kompleksitas waktu untuk kasus terburuk *(worst case) –>*kebutuhan waktu maksimum.
2. *Tmin(n)* : kompleksitas waktu untuk kasus terbaik *(best case) –>*kebutuhan waktu minimum.
3. *Tavg(n)*: kompleksitas waktu untuk kasus rata-rata *(average case) –>*kebutuhan waktu secara rata-rata
4. Pengelompokan Algoritma

**Pengelompokan Algoritma Berdasarkan Notasi *O*-Besar**

|  |  |
| --- | --- |
| Kelompok Algoritma | Nama |
| *O*(1)  *O*(log n)  *O*(*n*)  *O*(*n* log *n*)  *O*(*n*2)  *O*(*n*3)  *O*(2*n*)  *O*(*n*!) | konstan  logaritmik  lanjar  *n* log *n*  kuadratik  kubik  eksponensial  faktorial |

Urutan spektrum kompleksitas waktu algoritma adalah :

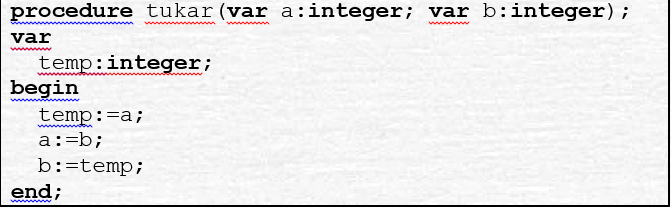
*O*(1)  *O*(log *n*)  *O*(*n*)  *O*(*n* log *n*)  *O*(*n* 2 )  *O*(*n* 3 )  ...  *O*(2 *n* )  *O*(*n*!)

 

algoritma polinomial algoritma eksponensial

Penjelasan :

*O*(1) Kompleksitas *O*(1) berarti waktu pelaksanaan algoritma adalah tetap, tidak bergantung pada ukuran masukan. Contohnya prosedur tukar di bawah ini:



Di sini jumlah operasi penugasan (*assignment*) ada tiga buah dan tiap operasi dilakukan satu kali. Jadi, *T*(*n*) = 3 = *O*(1).

*O*(log *n*) Kompleksitas waktu logaritmik berarti laju pertumbuhan waktunya berjalan lebih lambat daripada pertumbuhan *n*. Algoritma yang termasuk kelompok ini adalah algoritma yang memecahkan persoalan besar dengan mentransformasikannya menjadi beberapa persoalan yang lebih kecil yang berukuran sama (misalnya algoritma pencarian\_biner). Di sini basis algoritma tidak terlalu penting sebab bila *n* dinaikkan dua kali semula, misalnya, log *n* meningkat sebesar sejumlah tetapan.

*O*(*n*) Algoritma yang waktu pelaksanaannya lanjar umumnya terdapat pada kasus yang setiap elemen masukannya dikenai proses yang sama, misalnya algoritma pencarian\_beruntun. Bila *n* dijadikan dua kali semula, maka waktu pelaksanaan algoritma juga dua kali semula.

*O*(*n* log *n*) Waktu pelaksanaan yang *n* log *n* terdapat pada algoritma yang memecahkan persoalan menjadi beberapa persoalan yang lebih kecil, menyelesaikan tiap persoalan secara independen, dan menggabung solusi masing- masing persoalan. Algoritma yang diselesaikan dengan teknik bagi dan gabung mempunyai kompleksitas asimptotik jenis ini. Bila *n* = 1000, maka *n* log *n* mungkin 20.000. Bila *n* dijadikan dua kali semual, maka *n* log *n* menjadi dua kali semula (tetapi tidak terlalu banyak).

*O*(*n*2) Algoritma yang waktu pelaksanaannya kuadratik hanya praktis digunakan untuk persoalana yang berukuran kecil. Umumnya algoritma yang termasuk kelompok ini memproses setiap masukan dalam dua buah kalang bersarang, misalnya pada algoritma urut\_maks. Bila *n* =1000, maka waktu pelaksanaan algoritma adalah 1.000.000. Bila *n* dinaikkan menjadi dua kali semula, maka waktu pelaksanaan algoritma meningkat menjadi empat kali semula.

*O*(*n*3) Seperti halnya algoritma kuadratik, algoritma kubik memproses setiap masukan dalam tiga buah kalang bersarang, misalnya algoritma perkalian matriks. Bila *n* = 100, maka waktu pelaksanaan algoritma adalah 1.000.000. Bila *n* dinaikkan menjadi dua kali semula, waktu pelaksanan algoritma meningkat menjadi delapan kali semula.

*O*(2*n*) Algoritma yang tergolong kelompok ini mencari solusi persoalan secara "*brute force*", Bila *n* = 20, waktu pelaksanaan algoritma adalah 1.000.000. Bila *n* dijadikan dua kali semula, waktu pelaksanaan menjadi kuadrat kali semula!

*O*(*n*!) Seperti halnya pada algoritma eksponensial, algoritma jenis ini memproses setiap masukan dan menghubungkannya dengan *n* - 1 masukan lainnya, Bila *n* = 5, maka waktu pelaksanaan algoritma adalah 120. Bila *n* dijadikan dua kali semula, maka waktu pelaksanaan algoritma menjadi faktorial dari 2*n.*