BAB II

MATERI PENUNJANG

2.1 Relasi Berulang (Rekurensi)

Definisi 2.1.1

Sebuah relasi rekurensi (recurrence relation) untuk deret $a_0, a_1, \ldots$ merupakan sebuah persamaan yang mengkaitkan a_n dengan pendahulunya $a_0, a_1, \ldots, a_{n-1}$.

Contoh

Diberikan instruksi untuk membentuk sebuah deret berikut

1. Awali dengan 5.
2. Jika diberikan sebarang suku, tambahkan 3 untuk mendapatkan suku berikutnya.

Sehingga jika dibuat daftar suku - suku deret tersebut, maka didapatkan

5, 8, 11, 14, 17, ... (2.1)

Jika deret (2.1) dinyatakan sebagai $a_1, a_2, \ldots$ maka instruksi ke-1 dapat dinyatakan ulang sebagai

$a_1 = 5$ (2.2)

dan instruksi ke-2 dinyatakan sebagai

$a_n = a_{n-1} + 3 \quad , n \geq 2$ (2.3)

Dengan mengambil $n = 2$ dalam persamaan (2.3), maka didapatkan

$a_2 = a_1 + 3$

$= 5 + 3 = 8$

Dengan mengambil n = 3 dalam persamaan (2.3), maka didapatkan

$a_3 = a_2 + 3$

$= 8 + 3$

$= 11$

Jadi dengan menggunakan persamaan (2.2) dan (2.3) dapat dihitung sebarang suku dalam deret tersebut.

Relasi rekurensi seperti pada persamaan (2.3), nilai - nilai awal seperti pada (2.2) harus diberikan. Nilai - nilai awal ini disebut syarat awal (initial conditions).

Menyelesaikan suatu relasi rekurensi yang melibatkan a_0, a_1, ... sama halnya dengan mencari rumus eksplisit untuk bentuk umum a_n. Terdapat beberapa metode penyelesaian relasi rekurensi, salah satu diantaranya adalah dengan cara iterasi (iteration).

Contoh

Carilah bentuk umum pada contoh relasi berulang di atas.

Penyelesaian

Pada contoh di atas, terdapat

$a_1 = 5$

$a_n = a_{n-1} + 3 \quad , n \geq 2$ (2.4)

Dengan mengganti n dengan n -1, maka persamaan (2.4) dapat dituliskan

$a_{n-1} = a_{n-2} + 3 \quad , n \geq 3$ (2.5)

Jika persamaan (2.5) disubstitusikan ke dalam persamaan (2.4), maka didapatkan

$a_n = a_{n-2} + 3 + 3 \quad , n \geq 3$

$= a_{n-2} + 2.3 \quad , n \geq 3 \qquad (2.6)$

Dengan mengganti n dengan n - 2, maka persamaan (2.4) dapat dituliskan

$a_{n-2} = a_{n-3} + 3 \quad , n \geq 4 \qquad (2.7)$

Jika persamaan (2.7) disubstitusikan ke dalam persamaan (2.6), maka didapatkan

$a_n = a_{n-3} + 3 + 2.3 \quad , n \geq 4$

$= a_{n-3} + 3.3 \quad , n \geq 4$

Jadi secara umum didapatkan

$a_n = a_{n-k} + k.3 \quad , k = 0, 1, 2, 3, \ldots \qquad (2.8)$

misalkan k = n - 1, maka persamaan (2.8) dapat dinyatakan

$a_n = a_{n-(n-1)} + (n - 1).3$

$= a_{n-n+1} + (n - 1).3$

$= a_1 + 3.(n - 1)$

karena $a_1 = 5$, maka didapatkan bentuk umumnya yaitu

$a_n = 5 + 3.(n - 1) \qquad (2.9)$

2.2 Bahasa Pemrograman Pascal

2.2.1 Rekursif

Rekursif berarti suatu proses yang dapat memanggil dirinya sendiri. Rekursif merupakan teknik pemrograman yang berdayaguna yang digunakan pada pemrograman

dengan mengekspresikannya dalam suku - suku dari program lain dengan menambahkan langkah - langkah sejenis.

Contoh

```
function fakt ( n : integer ) : integer ;
begin
    if  n = 0 then
       fakt := 1 ;
    else
       fakt := n * fakt (n - 1) ;
end ;
```

2.2.2 Grafik

Terdapat syarat - syarat yang harus dipenuhi sebelum membuat program grafik dengan Bahasa Pascal, yaitu :

1. Adanya perangkat keras yang sesuai artinya suatu perangkat keras yang ditunjang Turbo Pascal. Hal ini tidak akan banyak mengalami kesulitan karena telah banyaknya perangkat keras yang ditunjang Turbo Pascal.
2. Adanya berkas BGI

   Berkas - berkas BGI merupakan contoh graphics driver yang disediakan Borland supaya program yang dibuat dengan Turbo Pascal dapat mengendalikan berbagai macam adapter tampilan. Tidak semua berkas BGI digunakan, cukup yang menunjang perangkat lunak saja. Berikut ini adalah tabel berkas BGI beserta perangkat keras yang ditunjangnya.

Tabel 1

NAMA BERKAS	PERANGKAT KERAS
CGA . BGI	IBM CGA , MCGA
EGA VGA . BGI	IBM EGA , VGA
HERC . BGI	Hercules Monokrom
ATT . BGI	AT & T
PC3270 . BGI	IBM 3270 PC

3. Berkas Graph .TPU

Berkas Graph . TPU merupakan berkas yang diperlukan pada saat kompilasi, sehingga harus dapat dijangkau oleh compiler.

4. Berkas - berkas huruf (font)

Berkas - berkas *.CHR berisikan vektor - vektor model huruf yang diperlukan jika akan menggunakan fasilitas gambar teks pada program grafik.

Setiap program Turbo Pascal yang menggunakan kemampuan grafik memiliki kerangka sebagai berikut :

```
Program (nama program) ;
Uses graph ;
Var GraphDrv, GraphMode, SandiSalah : integer ;
Begin
   GraphDrv := detect ;
   Initgraph ( GraphDrv, GraphMode, ' ' ) ;
   SandiSalah := graph_result ;
   if SandiSalah < > grok then
                Writeln (' kesalahan graphics') ;
   Grapherrormsg (SandiSalah) ;
   Writeln (' program dihentikan ') ;
   Halt (1) ;
End .
```

Penjelasan

Uses graph menyatakan bahwa program tersebut akan menggunakan unit graph. GraphDrv, GraphMode, SandiSalah merupakan tiga variabel global. GraphDrv menyatakan nomor graphics driver yang akan digunakan. Nomor ini dapat dinyatakan dengan angka ataupun dengan nama konstanta yang sudah dideklarasikan dalam unit graph. Berikut ini adalah tabel nilai dan nama konstanta graphics driver.

Tabel 2

NAMA KONSTANTA	NILAI
Current Driver	- 128
Detect	0
CGA	1
MCGA	2
EGA	3
EGA 64	4
EGA Mono	5
IBM 8514	6
Herc Mono	7
ATT 400	8
VGA	9
PC 3270	10

Pada kerangka program di atas, GraphDrv diisi dengan detect (0), yang artinya program menentukan sendiri jenis perangkat keras yang terpasang. Hal ini dimungkinkan karena biasanya terdapat daftar perlengkapan yang tersimpan dalam memori utama. Keuntungan cara ini adalah program tidak akan mengalami masalah, walaupun dijalankan pada berbagai komputer dengan perangkat keras grafiknya. Dengan detect, mode yang otomatis dipilih adalah mode tertinggi yaitu CGA Hi

atau 4. Jika tidak menggunakan detect, maka harus menetapkan nilai GraphMode. Contohnya jika diisikan CGA kedalam Graph Drv, maka harus mengubah pernyataan pada bagian atas program utama, yaitu :

```
...
GraphDrv := CGA ;
GraphMode := CGA Co
InitGraph ( GraphDrv, GraphMode, ' ' ) ;
...
```

Pada saat prosedur Initgraph (GraphDrv, GraphMode, ' ') dieksekusi, adapter tampilan dideteksi, selanjutnya graphics driver yang sesuai akan berpindah dari berkas ke memori, dan komputer pindah ke mode grafik. String kosong ' ' menunjukkan tempat grafik driver ada di direktori yang aktif saat itu. Jika berkas BGI yang digunakan terdapat di direktori lain, maka string kosong dapat diganti dengan nama jalur (path name) lengkap, contohnya 'C : \ Driver'.

Fungsi graph result menghasilkan suatu nilai yang merupakan sandi kesalahan yang dimasukkan ke dalam variabel SandiSalah. Unit graph telah menetapkan apa saja sandi - sandi tersebut. Langkah selanjutnya adalah memeriksa nilai SandiSalah, dengan membandingkan grok (0). Jika tidak sama berarti pada operasi Initgraph terjadi kesalahan. Jika SandiSalah dijadikan argumen untuk prosedur Grapherrormsg, maka dapat langsung diketahui kesalahan apa yang terjadi di layar. Selanjutnya program akan dihentikan dengan Halt (1). Jika operasi Initgraph

berhasil, maka SandiSalah akan sama dengan grok (0) dan proses berlanjut ke program grafik.

Titik (pixel) dapat ditampilkan dengan prosedur put pixel yang memerlukan tiga argumen yaitu koordinat x (baris), koordinat y (lajur), dan warna pixel yang bersangkutan. Pembuatan bilangan acak dapat dilakukan dengan fungsi random, yang memerlukan satu argumen untuk menentukan batas atas bilangan acak yang dihasilkannya, sedangkan batas bawahnya adalah nol.

Contoh

Put pixel (random (x), random (y), red) ;

Sebelum menghasilkan bilangan acak agar disertakan randomize supaya urutan bilangan acak yang dihasilkan tidak selalu sama.

Terdapat tiga prosedur menggambar garis yang disediakan oleh Turbo Pascal yaitu line, line to, dan line rel. Prosedur line membutuhkan 4 argumen yang semuanya integer menyatakan koordinat dua titik yang akan dihubungkan dengan garis. Prosedur line to hanya membutuhkan dua argumen yaitu koordinat x dan y yang merupakan titik tujuannya, sedangkan line rel memerlukan dua argumen yang menyatakan banyaknya pergeseran yang harus dilakukan untuk mencapai ujung garis.

Contoh

Line (10, 20, 30, 40) ;

line to (10, 20);

Line rel (50, - 50) ;

2.3 Algoritma

Algoritma merupakan suatu urutan dari barisan langkah - langkah atau instruksi yang digunakan untuk menyelesaikan suatu masalah. Istilah algoritma pertama kali diperkenalkan oleh seorang ahli matematika yaitu Abu Ja'far Muhammad Ibnu Musa Alkhawarizmi. Jika terdapat suatu masalah, maka kemungkinan lebih dari satu algoritma yang dapat digunakan untuk menyelesaikan masalah tersebut. Suatu algoritma harus memenuhi beberapa kriteria berikut :

1. Adanya output

   Suatu algoritma harus mempunyai output. Output tersebut tentunya merupakan solusi dari masalah yang ada.

2. Efektif dan efisien

   Suatu algoritma dikatakan efektif jika algoritma tersebut menghasilkan solusi yang sesuai dengan masalah yang diselesaikan (tepat guna), sedangkan suatu algoritma dikatakan efisien jika waktu proses algoritma tersebut relatif singkat dan penggunaan memorinya relatif kecil.

3. Jumlah langkah berhingga

   Banyaknya langkah atau instruksi dalam suatu algoritma harus berhingga, karena jika tidak demikian, proses yang dilakukan memerlukan waktu yang lama dan diperoleh hasil yang tidak sesuai dengan masalah yang ada.

4. Berakhir

   Proses mencari solusi suatu masalah harus berakhir.

5. Terstruktur

Urutan dari barisan langkah - langkah yang digunakan harus disusun sedemikian rupa agar proses penyelesaiannya tidak berbelit - belit, sehingga waktu prosesnya akan relatif singkat. Hal ini akan memperlihatkan bahwa bagian - bagian dari proses tersebut dapat dibedakan secara jelas (input, proses, output) yang memudahkan dalam melakukan proses pemeriksaan ulang.

Hal - hal yang menyangkut studi tentang algoritma yaitu :

1. Merencanakan suatu algoritma

Sebelum menyatakan suatu algoritma, terlebih dahulu menentukan model atau desain penyelesaiannya. Suatu masalah dapat diselesaikan dengan berbagai macam model atau desain. Tentunya ada satu model yang terbaik dari sekian algoritma yang ada, sehingga teknik variasi desain harus dikuasai sebaik - baiknya.

2. Menyatakan suatu algoritma

Menyatakan suatu algoritma adalah membuat barisan langkah - langkah atau konstruksi secara terurut yang digunakan untuk menyelesaikan suatu masalah. Pernyataan tersebut hendaknya dibuat secara singkat dan terstruktur. Menyatakan suatu algoritma dapat dilakukan dengan dua cara yaitu diagram alur (flow chart) dan bahasa semu.

3. Validitas suatu algoritma

Indikasi suatu algoritma valid (absah) adalah jika penyelesaian yang diperoleh dapat memecahkan permasalahan yang ada, selain itu solusi, perhitungan, dan prosedurnya selalu benar untuk semua kemungkinan input yang ada.

4. Menganalisa suatu algoritma

Studi tentang analisa algoritma menyangkut 2 hal yaitu running time dan banyaknya memori (storage) yang digunakan.

5. Menguji suatu algoritma

Algoritma yang akan digunakan untuk menyelesaikan suatu masalah dengan cepat dan tepat diperoleh solusinya dengan alat bantu komputer. Algoritma tersebut disajikan terlebih dahulu ke dalam bahasa pemrograman. Pengujian terhadap program tersebut dilakukan dengan dua fase yang dilaksanakan secara simultan yaitu :

a. Fase debugging

Suatu fase dari proses program eksekusi yang melakukan koreksi terhadap kesalahan program yaitu kesalahan dalam bahasa pemrograman baik secara logika maupun sintaks.

b. Fase profiling

Suatu fase yang akan bekerja jika program tersebut sudah benar. Fase ini juga digunakan untuk mengukur waktu tempuh (running time) dan penggunaan memori (storage).

2.4 Kompleksitas Algoritma

Analisa algoritma mengacu pada proses memperoleh perkiraan dari waktu dan ruang yang diperlukan untuk mengeksekusi algoritma. Kompleksitas algoritma mengacu pada jumlah waktu dan ruang yang dibutuhkan untuk mengeksekusi algoritma. Waktu yang

diperlukan untuk mengeksekusi sebuah algoritma merupakan sebuah fungsi dari masukan. Misalkan untuk sebarang masukan berukuran n, algoritma A membutuhkan tepat c_1n satuan waktu dan algoritma B membutuhkan c_2n^2 satuan waktu. Untuk ukuran masukan tertentu algoritma B mungkin menjadi superior. Sebagai contoh andaikan untuk sebarang masukan berukuran n, algoritma A membutuhkan 300n satuan waktu dan algoritma B membutuhkan $5n^2$ satuan waktu. Untuk masukan berukuran n = 5, algoritma A membutuhkan 1500 satuan waktu dan algoritma B membutuhkan 125 satuan waktu. Dengan demikian B lebih cepat. Tentu saja untuk n > 60 algoritma A lebih cepat dari pada B.

Misalkan terdapat sebuah komputer yang dapat mengeksekusi satu langkah dalam 1 mikrodetik (10^{-6} detik). Tabel 3 menunjukkan waktu eksekusi berdasarkan asumsi tersebut, untuk berbagai ukuran masukan (log yang ada merupakan log dengan basis dua).

Tabel 3

Banyaknya langkah penyelesaian untuk masukan berukuran n	waktu untuk mengeksekusi jika n =			
	3	6	9	12
1	10^{-6} detik	10^{-6} detik	10^{-6} detik	10^{-6} detik
log log n	10^{-6} detik	10^{-6} detik	2×10^{-6} detik	2×10^{-6} detik
log n	2×10^{-6} detik	3×10^{-6} detik	3×10^{-6} detik	4×10^{-6} detik
n	3×10^{-6} detik	6×10^{-6} detik	9×10^{-6} detik	10^{-5} detik
n log n	5×10^{-6} detik	2×10^{-5} detik	3×10^{-5} detik	4×10^{-5} detik
n^2	9×10^{-6} detik	4×10^{-5} detik	8×10^{-5} detik	10^{-4} detik
n^3	3×10^{-5} detik	2×10^{-4} detik	7×10^{-4} detik	2×10^{-3} detik
2^n	8×10^{-6} detik	6×10^{-5} detik	5×10^{-4} detik	4×10^{-3} detik

Perhatikan bahwa eksekusi algoritma yang memerlukan 2^n langkah untuk sebuah masukan berukuran n hanya layak untuk ukuran masukan yang sangat kecil. Algoritma yang membutuhkan n^2 dan n^3 langkah juga menjadi tidak layak untuk ukuran masukan yang relatif besar.

2.5 Running Time Program

Running time suatu program adalah banyaknya waktu yang digunakan oleh suatu program dalam menyelesaikan suatu masalah. Running time suatu program biasanya dinotasikan T(n). Hal - hal yang mempengaruhi running time suatu program adalah sebagai berikut :

1. Besar dan jenis input data

   Adanya kenyataan bahwa running time program tergantung pada input menunjukkan bahwa running time suatu program harus ditetapkan sebagai suatu fungsi dari input. Running time program tidak tergantung pada nilai input, akan tetapi tergantung pada ukuran input. Misalnya dalam suatu pengurutan (sorting) terdapat sebuah daftar item sebagai input. Item - item tersebut harus diurutkan, sehingga dihasilkan item - item yang sama sebagai output baik secara ascending atau descending. Ukuran input pada sebuah sorting program adalah jumlah item - item yang diurutkan atau panjang dari daftar item. Di samping ukuran input, running time program juga tergantung pada jenis input data. Jika jenis data yang diinginkan dengan tingkat ketelitian tunggal (single precision) maka running timenya relatif cepat dari pada menggunakan data dengan tingkat

ketelitian ganda (double precision). Demikian pula jika dibandingkan dengan data - data yang triple precision atau quadruple precision.

2. Banyaknya langkah

Makin banyak langkah atau instruksi yang digunakan maka makin lama running time yang diperlukan dalam proses tersebut.

3. Jenis operasi

Jenis operasi tersebut meliputi operasi aritmatika, operasi nalar atau logika, dan sebagainya.

4. Komputer dan kompilator

Meskipun suatu program yang dibuat telah mencapai running time yang sangat efisien (meliputi 3 hal di atas), namun jika digunakan komputer yang berkemampuan lambat maka running timenya akan menjadi lambat. Kompilator yang digunakan juga berpengaruh terhadap running time program.

Adanya kenyataan bahwa running time program tergantung pada komputer dan kompilator yang digunakan, hal ini menunjukkan bahwa running time tidak dapat dinyatakan sebagai satuan waktu standar (detik), akan tetapi hanya dapat dinyatakan running time suatu program adalah sebanding dengan n atau n^2 atau yang lainnya. Oleh karena itu dalam menentukan running time suatu program diasumsikan bahwa running time yang diperoleh merupakan jumlahan operasi yang dilaksanakan oleh komputer yang ideal. Salah satu cara untuk menentukan jumlahan operasi yang dilaksanakan adalah menggunakan notasi Big-O

Dalam melakukan analisa algoritma terdapat tiga hal yang sering diperhitungkan yaitu :

1. Best case running time yaitu suatu keadaan yang merupakan nilai minimum dari fungsi T(n) untuk setiap input yang mungkin atau keadaan terbaik dimana algoritma memerlukan waktu yang singkat.
2. Worst case running time yaitu suatu keadaan yang merupakan nilai maksimum dari fungsi T(n) untuk setiap input yang mungkin atau keadaan terburuk dimana algoritma memerlukan waktu yang paling lama.
3. Average case running time yaitu keadaan dari waktu tempuh yang ekuivalen dengan nilai ekspektasi dari fungsi T(n) untuk setiap input data yang mungkin. Nilai ekspektasi dari fungsi T(n) = E, yang didefinisikan sebagai :

   $E = n_1p_1 + n_2p_2 + ... + n_kp_k$

   dengan

   $n_1, n_2, ..., n_k$ merupakan nilai - nilai yang muncul.

   $p_1, p_2 ..., p_k$ merupakan probabilitas dari setiap nilai - nilai yang muncul.

Dalam menentukan running time suatu algoritma dipergunakan worst case running time, hal ini berdasarkan alasan - alasan sebagai berikut :

1. Worst case running time merupakan waktu maksimal yang diperlukan suatu algoritma dalam menyelesaikan suatu masalah.

2. Worst case running time dari suatu algoritma merupakan suatu batas atas dari suatu running time. Dengan demikian akan memberikan jaminan bahwa algoritma yang digunakan akan mempunyai akhir proses.
3. Untuk beberapa algoritma, worst case running time merupakan kejadian yang sering terjadi. Sebagai contoh dalam proses pencarian data base untuk mendapatkan suatu informasi, worst case running time selalu terjadi ketika informasi yang dibutuhkan tidak berada dalam data base.

2.6 Menghitung Running Time

2.6.1 Notasi Big - O

Definisi 2.6.1.1

T(n) adalah O(f(n)) jika terdapat 2 buah konstanta bulat positif c dan n_0 sedemikian hingga $T(n) \leq cf(n)$, untuk setiap $n \geq n_0$.

Contoh

Tunjukkan bahwa $T(n) = (n + 1)^2$ adalah $O(n^2)$.

Penyelesaian

Misalkan $T(0) = n_0 = 1$ dan $T(1) = c = 4$, maka berdasarkan definisi 2.6.1.1 berlaku $(n + 1)^2 \leq 4n^2$, untuk setiap $n \geq 1$

Teorema 2.6.1.1

Jika T(n) adalah fungsi polinomial dalam n dengan derajad m, yang ditulis dengan notasi $T(n) = a_m n^m + a_{m-1} n^{m-1} + ... + a_1 n + a_0$, maka T(n) adalah $O(n^m)$

Bukti

$$T(n) \leq \sum_{i=0}^{m} |a_i| n^i$$

$$\leq n^m \sum_{i=0}^{m} |a_i| n^{i-m}$$

$$\leq n^m \sum_{i=0}^{m} |a_i|, \text{ untuk } n \geq 1$$

sehingga T(n) adalah $O(n^m)$

■

Contoh

Tunjukkan $T(n) = 3n^3 + 2n^2$ adalah $O(n^3)$

Penyelesaian

Berdasarkan teorema 2.6.1.1 karena $T(n) = 3n^3 + 2n^2$ merupakan fungsi polinomial dalam n dengan derajad 3, maka T(n) adalah $O(n^3)$.

Contoh

Tunjukkan $T(n) = 3^n$ bukan $O(2^n)$

Penyelesaian

Andaikan $T(n) = 3^n$ adalah $O(2^n)$ sehingga berdasarkan definisi 2.6.1.1 terdapat dua buah konstanta bulat positif c dan n_0 sedemikian sehingga $3^n \leq c2^n$ untuk $n \geq 1$ Dengan demikian $c \geq (3/2)^n$, untuk $n \geq 1$. Akan tetapi $(3/2)^n$ besarnya berubah - ubah sesuai dengan besarnya n. Jadi tidak ada konstanta c yang dapat melebihi $(3/2)^n$ untuk semua n. Jadi pengandaian diingkar, sehingga didapatkan $T(n) = 3^n$ bukan $O(2^n)$.

Definisi 2.6.1.2

$T(n) = \Omega(g(n))$ jika terdapat 2 buah konstanta c dan n_0 sedemikian hingga $T(n) \geq cg(n)$, untuk setiap $n \geq n_0$.

Definisi 2.6.1.3

$T(n) = \theta(h(n))$ jika dan hanya jika $T(n) = O(h(n))$ dan $T(n) = \Omega(h(n))$.

2.6.2 Kombinasi Fungsi Pertumbuhan

Banyak algoritma yang dibuat dalam dua atau lebih sub prosedur. Jumlah langkah yang digunakan oleh komputer untuk menyelesaikan sebuah permasalahan dengan input tertentu pada sebuah algoritma adalah jumlahan setiap langkah yang digunakan oleh sub prosedur tersebut. Untuk mendapatkan estimasi Big-O pada jumlah langkah yang dibutuhkan, sangat perlu untuk menentukan estimasi Big-O pada jumlah langkah yang digunakan pada setiap sub prosedur dan selanjutnya estimasi tersebut dikombinasikan.

Teorema 2.6.2.1

Jika $T_1(n)$ adalah $O(f(n))$ dan $T_2(n)$ adalah $O(g(n))$, maka $T_1(n) + T_2(n)$ adalah $O(\max (f(n), g(n)))$.

Bukti

Berdasarkan definisi 2.6.1.1 terdapat 4 buah konstanta c_1, c_2, n_1, dan n_2 sedemikian sehingga

$T_1(n) \leq c_1 f(n)$, untuk $n \geq n_1$

$T_2(n) \leq c_2 g(n)$, untuk $n \geq n_2$.

Misalkan $n_0 = \max(n_1, n_2)$, maka berlaku

$T_1(n) \leq c_1 f(n)$, untuk $n \geq n_0$

$T_2(n) \leq c_2 g(n)$, untuk $n \geq n_0$, sedemikian hingga berlaku

$T_1(n) + T_2(n) \leq c_1 f(n) + c_2 g(n)$.

Misalkan $c_3 = \max(c_1, c_2)$, maka

$T_1(n) + T_2(n) \leq c_3 f(n) + c_3 g(n)$

$\leq c_3(f(n) + g(n))$

$\leq c_3 [\max(f(n), g(n)) + \max(f(n), g(n))]$

$\leq 2c_3 \max(f(n), g(n)) \leq c \max(f(n), g(n))$

untuk $c = 2c_3$ dan $n \geq n_0$

■

Teorema 2.6.2.1 di atas disebut dengan aturan penjumlahan (rule of sums). Aturan penjumlahan tersebut dapat digunakan untuk menghitung running time barisan atau urutan dari langkah - langkah suatu program.

Contoh

Tunjukkan running time 3 buah langkah yang berurutan berikut $O(n^2)$, $O(n^3)$, dan $O(n\log n)$ adalah $O(n^3)$.

Penyelesaian

Running time dua langkah yang pertama dieksekusi secara berurutan adalah $O(\max(n^2, n^3))$ yaitu $O(n^3)$, sehingga untuk running time seluruh langkah adalah $O(\max n^3, n \log n))$ yaitu $O(n^3)$.

Jadi secara umum running time suatu barisan langkah - langkah adalah running time terbesar dari barisan langkah - langkah tersebut. Pada suatu keadaan tertentu terdapat dua langkah atau lebih yang running timenya tidak seimbang (yang satu lebih besar dari pada yang lain).

Contoh

Tunjukkan running time untuk dua langkah berikut :

$$f(n) = \begin{cases} n^4 & \text{, jika n genap.} \\ n^2 & \text{, jika n ganjil.} \end{cases}$$

$$g(n) = \begin{cases} n^2 & \text{, jika n genap.} \\ n^3 & \text{, jika n ganjil.} \end{cases}$$

adalah $O(n^4)$ jika n genap dan $O(n^3)$ jika n ganjil.

Penyelesaian

Pada kasus tersebut aturan penjumlahan harus diaplikasikan secara langsung, yaitu $O(\max(f(n), g(n))$ adalah $O(n^4)$ jika n genap dan $O(n^3)$ jika n ganjil.

Teorema 2.6.2.2

Jika $T_1(n)$ adalah $O(f(n))$ dan $T_2(n)$ adalah $O(g(n))$, maka $T_1(n).T_2(n)$ adalah $O(f(n).g(n))$.

Bukti

Berdasarkan definisi 2.6.1.1 terdapat 4 buah konstanta c_1, c_2, n_1, dan n_2 sedemikian sehingga

$T_1(n) \leq c_1 f(n)$, untuk $n \geq n_1$

$T_2(n) \leq c_2 g(n)$, untuk $n \geq n_2$

Misalkan $n_0 = \max(n_1, n_2)$ maka

$T_1(n) \leq c_1 f(n)$, untuk $n \geq n_0$

$T_2(n) \leq c_2 g(n)$, untuk $n \geq n_0$

sehingga berlaku

$$T_1(n) \cdot T_2(n) \leq c_1 f(n) \cdot c_2 g(n)$$

$$\leq c_1 . c_2 (f(n).g(n))$$

$$\leq c\,(f(n).g(n))$$

untuk $c = c_1 . c_2$ dan $n \geq n_0$

■

Teorema 2.6.2.2 di atas disebut dengan aturan perkalian (rule of product).

2.6.3 Aturan Umum Analisa Algoritma

Jika ditemukan beberapa algoritma yang berbeda dalam menyelesaikan permasalahan yang sama, maka harus dipilih salah satu yang sesuai dengan kebutuhan. Cara yang digunakan yaitu dengan analisa algoritma. Analisa algoritma dapat digunakan untuk

menentukan efisiensi dari beberapa algoritma, dengan demikian dapat diambil keputusan terbaik untuk menggunakan suatu algoritma.

Secara umum running time pada sebuah statemen atau kelompok statemen mungkin terparameterisasi oleh ukuran input atau beberapa variabel. Parameter untuk running time keseluruhan algoritma (program) adalah n (ukuran input). Adapun aturan umum untuk menganalisa algoritma yaitu :

1. Running time setiap assigment (tugas), baca (read), dan statement write dapat dinyatakan dalam O(1).
2. Running time pada barisan statemen ditentukan dengan aturan penjumlahan (rule of sums) yaitu running time terbesar dari beberapa barisan statemen.
3. Running time pada statemen if adalah harga pada kondisi statemen - statemen yang dieksekusi ditambah waktu untuk mengevaluasi kondisi. Waktu untuk mengevaluasi kondisi tersebut biasanya adalah O(1). Adapun running time untuk susunan if then else adalah waktu untuk mengevaluasi kondisi ditambah waktu terbesar yang dibutuhkan untuk statemen -statemen yang dieksekusi ketika kondisi benar dan salah.
4. Running time pada sebuah loop adalah jumlah seluruh waktu sekitar loop meliputi waktu mengeksekusi sekumpulan statemen dan waktu mengevaluasi kondisi untuk penghentian (biasanya yang terakhir adalah O(1)). Total waktu dari statemen - statemen suatu loop bersarang adalah waktu statemen - statemen yang merupakan hasil kali dari ukuran - ukuran seluruh loop.

2.6.4 Analisa Struktur Kontrol

Analisa algoritma biasanya proses berjalannya terbalik. Pertama menetapkan waktu yang diperlukan oleh instruksi tunggal (sering kali dibatasi dengan konstanta) kemudian menggabungkan waktu tersebut berdasarkan struktur kontrol, selanjutnya menggabungkan instruksi - instruksi pada program.

2.6.4.1 Loop for, while, dan repeat.

Loop for adalah loop termudah untuk dianalisa dibandingkan dengan loop - loop yang lain.

for i := 1 to m do P(i).

Misalkan loop tersebut bagian dari suatu algoritma yang panjang. Terdapat P(i) yang merupakan statemen atau blok statemen yang berbeda di dalam loop. Diberikan t yang merupakan waktu yang dibutuhkan untuk mengeksekusi P(i). Pada kasus ini jika loop digunakan m kali, setiap iterasi membutuhkan waktu t, maka total waktu yang dibutuhkan oleh loop adalah m.t.

Loop while dan repeat biasanya lebih sulit untuk dianalisa dari pada loop for, karena tidak adanya langkah pasti untuk mengetahui berapa besar waktu yang dibutuhkan untuk melalui loop tersebut.

Contoh

Hitung running time Prosedur Bublesort berikut ini

```
Procedure Buble ( var A : array [1..n] of integer ) ;
Var i, j, temp : integer ;
Begin
1.  For i := 1 to n - 1 do
2.      For j := n down to i + 1 do
3.          if A[j - 1] > A[j] then
            Begin
4.              temp := A[j - 1] ;
5.              A[j - 1] := A[j] ;
6.              A[j] := temp ;
            End;
End.
```

Penyelesaian

Misalkan T(n) adalah running time prosedur Buble. Pertama akan dianalisa setiap pernyataan penugasan yang memerlukan waktu konstan, yaitu pada baris 4, 5, dan 6 yang masing - masing memerlukan waktu O(1). Dengan aturan penjumlahan (rule of sums) running time pernyataan pada baris 4, 5, dan 6 adalah O(max(1,1,1)) = O(1). Selanjutnya akan dianalisa pernyataan - pernyataan kondisi dan looping. Untuk pernyataan if, pengujian kondisi memerlukan O(1). Dengan demikian worst case running time untuk group if dari pernyataan 3 - 6 memerlukan waktu O(1).

Selanjutnya untuk baris 2 - 6 tubuh loop memerlukan waktu O(1) bagi setiap iterasi. Jumlah iterasi loop adalah n - i , sehingga dengan aturan perkalian (rule of product) waktu yang diperlukan loop pada baris 2 - 6 adalah O((n - i) x 1) yaitu O(n - i).

Pada baris 1 dieksekusi n - 1 kali, sehingga running time prosedur tersebut adalah :

$$T(n) = \sum_{i=1}^{n-1} (n - i) = n(n - 1) / 2 = n^2/2 - n/2$$

Dengan demikian T(n) adalah $O(n^2)$.

2.6.4.3 Prosedur Rekursif

Rekursif berarti suatu proses yang dapat memanggil dirinya sendiri. Rekursif merupakan teknik pemrograman berdayaguna. Analisa prosedur rekursif meliputi solusi persamaan rekurensi.

Contoh

Hitung running time fungsi faktorial berikut ini.

```
Function Fact ( n : integer ) : integer ;
Begin
1.      if n ≤ 1 then
2.          fact := 1 ;
        Else
3.          Fact := n * fact (n - 1) ;
End ;
```

Penyelesaian

Misalkan T(n) adalah running time untuk fungsi fact(n). Running time untuk baris 1 dan 2 adalah O(1) dan untuk baris 3 running timenya adalah O(1) + T(n -1). Dengan

demikian untuk beberapa konstanta c dan d didapatkan persamaan rekurensi sebagai berikut :

$$T(n) = \begin{cases} d & , \text{jika } n \leq 1 \\ c + T(n-1) & , \text{jika } n > 1 \end{cases} \qquad (2.10)$$

Misalkan $n > 2$, dengan mensubstitusikan $n - 1$ pada n dalam persamaan (2.10) maka didapatkan $T(n-1) = c + T(n-2)$ sedemikian hingga persamaan (2.10) dapat ditulis $T(n) = c + c + T(n-2)$

$= 2c + T(n-2)$, jika $n > 2$

Untuk $n > 3$ dengan menggunakan persamaan (2.10) dapat memperluas $T(n-2)$ sehingga dihasilkan $T(n) = 3c + T(n-3)$, dan seterusnya sehingga secara umum dapat dituliskan :

$T(n) = ic + T(n-i)$, jika $n > i$

jika $i = n - 1$, maka didapatkan

$T(n) = c(n-1) + T(1)$

$= c(n-1) + d \qquad (2.11)$

Berdasarkan persamaan(2.11) dapat disimpulkan running time fungsi fact(n) adalah O(n).

2.7 Pengurutan (Sorting)

Pengurutan diartikan sebagai proses pengurutan kembali sekumpulan obyek ke dalam suatu urutan tertentu. Tujuan pengurutan adalah untuk mendapatkan kemudahan dalam pencarian anggota dari suatu himpunan.

Dalam suatu pengurutan larik (array) terdapat aspek ekonomis yang perlu diperhatikan, meliputi kapasitas pengingat yang tersedia dan waktu yang diperlukan untuk melakukan permutasi sehingga semua elemen akhirnya terurutkan. Ukuran efisiensi yang baik dapat diperoleh dari banyaknya perbandingan dan pemindahan yang dilakukan.

2.7.1 Penggabungan (Merging)

Merging yaitu penggabungan dua kumpulan data yang keduanya dalam keadaan urut menjadi satu kumpulan yang juga dalam keadaan urut. Misalkan terdapat dua buah larik (array) yang akan dimerge sudah dalam keadaan urut naik sebagai berikut :

larik 1

11	23	45	67	68

larik 2

9	12	21	42	56

Ambil elemen pertama dari larik 1 (l_1) dan larik 2 (l_2). Bandingkan nilai kedua elemen ini, jika $l_1 > l_2$, maka l_2 dikopikan ke larik ke-3, jika tidak l_1 dikopikan ke larik ke-3. elemen berikutnya yang dibandingkan adalah elemen yang terletak pada subskrip berikutnya. Dalam contoh di atas ($l_1 = 11$ dan $l_2 = 9$) yang terjadi adalah $l_1 > l_2$ sehingga l_2 dikopikan ke larik ke - 3. Untuk perbandingan berikutnya digunakan $l_1 = 11$ dan $l_2 = 12$. Proses ini

diulang terus sampai salah satu larik habis dikopikan terlebih dahulu. Kemudian larik satunya dikopikan ke larik ke-3. Hasil merging kedua larik di atas adalah sebagai berikut :

9	11	12	21	23	42	45	56	67	68

2.7.2 Mergesort

Metode Mergesort dapat dibedakan menjadi dua macam yaitu

1. Iteratif Mergesort

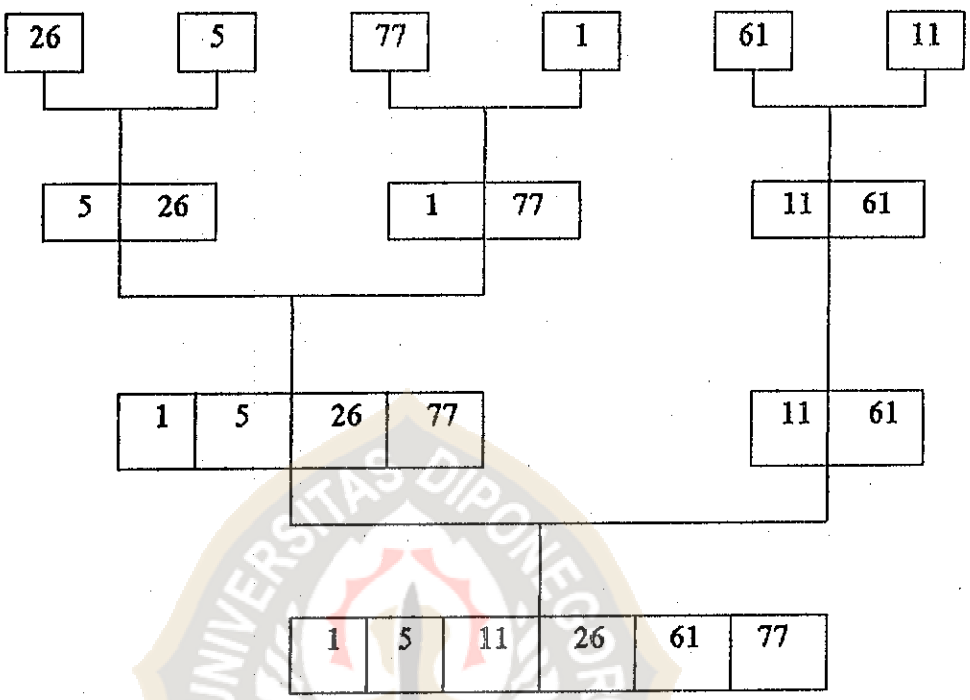
Larik yang mempunyai n elemen dianggap sebagai n buah larik. Untuk setiap pasang larik dilakukan proses merging sehingga akan diperoleh n/2 larik yang masing - masing terdiri dari dua elemen (jika n ganjil, maka terdapat sebuah larik dengan elemen satu). Selanjutnya dilakukan merging kembali untuk setiap pasang larik sehingga diperoleh n/4 buah larik. Langkah ini diteruskan sampai diperoleh sebuah larik yang sudah dalam keadaan urut.

Contoh Larik yang akan diurutkan sebagai berikut

{ 26, 5, 77, 1, 61, 11 }

Penyelesaian

Dengan menggunakan Iteratif Mergesort didapatkan



2. Rekursif Mergesort

Terdapat barisan dengan n elemen yang ditempatkan dalam suatu array A. Dari array A(1), A(2), ..., A(n) pisahkan menjadi dua bagian sehingga didapatkan A(1), A(2), ..., A(n/2) dan A(n/2 + 1), A(n/2 + 2), ..., A(n). Jika pembagian array A tersebut hasilnya masih terlalu besar, maka setiap bagian tadi dibagi lagi menjadi dua bagian yang lebih kecil sehingga dari setiap bagian akan diurutkan secara rekursif, selanjutnya hasil sortir dari setiap bagian tersebut digabungkan dan diurutkan lagi sedemikian sehingga akan dihasilkan urutan yang tunggal dari n elemen semula.

Contoh : Array yang akan diurutkan adalah sebagai berikut

{ 8, 10, 2, 5, 11, 1}

Penyelesaian

Dengan menggunakan Rekursif Mergesort didapatkan :