Semester Ganjil - 2023/2024

Algoritma Pemrograman dan Struktur Data

Materi 5: RUNNING TIME dan BIG O NOTATION

Dosen pengampu:
Suamanda Ika Novichasari, M.Kom.
Imam Adi Nata, M.Kom

PROGRAM STUDI S1 TEKNOLOGI INFORMASI
FAKULTAS TEKNIK
UNIVERSITAS TIDAR

Jl. Kapten Suparman No.39, Tuguran, Potrobangsan, Kec. Magelang Utara, Kota Magelang, Jawa Tengah 56116

Learning Objective

Mahasiswa mampu menjelaskan Running Time & Big O Notation pada algoritma tertentu

Mahasiswa mampu menghitung Running Time & Big O Notation pada algoritma tertentu

Course Material

Running Time

Big O Notation

Pre Test
10 Menit

- Hitunglah kompleksitas algoritma di samping ini !

```
//PROGRAM Soal nomor 6
// Mengimplementasikan kombinasi struktur dasar
algoritma

//DECLARATION
      DECLARE integer A, b, x

//IMPLEMENTATION / DEFINITION
      A ← 1
      b ← 0
      x ← 1
      Do {
            If (x mod 2)=0 then {
                  A ← A + 2
            }else{
                  b ← b + 1
            }
            x ← x + 1
      }while (x>5)
      Print (A,b)
```

BAB MATERI

Running Time

Subbab ini mempelajari tentang deskripsi dari running time

Apa itu *Running Time ?*

- **Running Time** = Waktu Komputasi
- Ketika menyusun algoritma sebaiknya menggunakan urutan langkah yang paling **efektif** dan **efisien**.
- Keefektifan algoritma dapat diukur dengan penggunaan **waktu (time)** dan ukuran memori **(space)**.
- Algoritma yang efisien adalah algoritma yang **meminimumkan** kebutuhan **waktu** dan **ukuran**.

Apa itu *Running Time ?*

- Kebutuhan waktu dan ukuran memori suatu algoritma **bergantung** pada **ukuran masukan (n)**, yang menyatakan jumlah data yang diproses.
- Waktu yang diperlukan oleh sebuah algoritma dapat diukur dengan menghitung banyaknya **operasi/instruksi** yang dieksekusi.
- Jika diketahui besaran waktu (dalam satuan detik) untuk melaksanakan sebuah operasi tertentu, maka dapat dihitung berapa waktu sesungguhnya untuk melaksanakan algoritma tersebut.

Apa itu *Kompleksitas Algoritma?*

- Besaran yang digunakan untuk menerangkan model pengukuran waktu/ruang dari sebuah algoritma disebut **kompleksitas algoritma**.
- Dengan menggunakan besaran kompleksitas algoritma, dapat ditentukan **laju peningkatan waktu (ruang)** yang diperlukan algoritma dengan meningkatnya ukuran masukan n.

Jenis *Kompleksitas Algoritma*

Kompleksitas waktu (T(n))

- mengukur jumlah tahapan komputasi yang dibutuhkan untuk menjalankan algoritma untuk ukuran sebanyak n masukan.

Kompleksitas ruang (S(n))

- mengukur memori yang digunakan oleh struktur data yang terdapat di dalam algoritma untuk ukuran sebanyak n masukan.

Jenis *Kompleksitas waktu*

Tmax(n) :

- untuk kasus terburuk (worst case) atau kebutuhan waktu maksimum.

Tmin(n) :

- untuk kasus terbaik (best case) atau kebutuhan waktu minimum.

Tavg(n):

- untuk kasus rata-rata (average case), atau kebutuhan waktu secara ratarata

02

Simple Search

- Algoritma **simple search**
  - jika terdapat 100 data maka kemungkinan terburuk akan dilakukan pengecekan sebanyak 100 kali.
  - Jika ada 1000 data maka akan dilakukan pengecekan sebanyak 1000 kali.
  - Jadi algoritma simple search memiliki **kemungkinan terburuk** terjadi pengecekan sebanyak jumlah data, waktu komputasi tersebut sering disebut dengan **linier time**

02

Simple Search

- Kasus terbaik:

$$T_{min}(n) = 1$$

- Kasus terburuk:

$$T_{max}(n) = n$$

- Kasus rata-rata: Jika nilai yang dicari ditemukan pada posisi ke-j, maka operasi perbandingan nilai yang dicari akan dieksekusi sebanyak j kali.

$$T_{avg}(n) = \frac{(1+2+3+\ldots+n)}{n} = \frac{\frac{1}{2}n(1+n)}{n} = \frac{(n+1)}{2}$$

02

Binary Search

- Berbeda dengan ***binary search***,
  - jika terdapat 100 data maka kemungkinan terburuk akan dilakukan pengecekan sebanyak 7 kali.
  - Jika terdapat 240.000 data maka kemungkinan terburuk terjadi pengecekan sebanyak 18 kali.
  - Waktu komputasi binary search tersebut disebut **logarithmic time** atau **log time.**

02

Binary Search

- Kasus terbaik:

$$T_{min}(n) = 1$$

- Kasus terburuk:

$$T_{max}(n) = \log_2 n$$

- Kasus rata-rata: Jika nilai yang dicari ditemukan pada posisi ke-j, maka operasi perbandingan nilai yang dicari akan dieksekusi sebanyak j kali.

BAB MATERI

Big O Notation

Subbab ini mempelajari konsep big O notation

Apa itu notasi *Big O*?

- **Notation Big O** adalah notasi khusus yang dapat digunakan untuk mendeskripsikan kecepatan sebuah algoritma.
- Notasi Big O **mencerminkan** kompleksitas algoritma.
- Sebagai contoh jika terdapat n buah data, algoritma simple search harus melakukan pengecekan pada setiap elemen sehingga akan terjadi operasi sebanyak n kali.
- Waktu komputasi algoritma simple search jika dinotasikan menjadi O(n).

Apa itu notasi *Big O*?

- Notasi Big O **tidak mencerminkan** kecepatan algoritma dalam hitungan detik, namun melalui notasi Big O dapat **diketahui** seberapa cepat algoritma tersebut **berkembang**.
- Notasi Big O merepresentasikan **kemungkinan terburuk (worst case).**
- Jika kompleksitas algoritma = O(n) waktu, maka dapat dipastikan bahwa algoritma tersebut tidak akan pernah lebih lambat dari O(n) waktu.

Urutan notasi *Big O*?

- Berikut merupakan 5 notasi Big O yang sering ditemui, diurutkan dari yang paling cepat ke yang paling lambat :
  - O(log n), juga dikenal dengan log time. Contoh: Binary search.
  - O(n), juga dikenal dengan linear time. Contoh: Simple search.
  - O(n * log n). Contoh : algoritma pengurutan yang cepat seperti quicksort
  - O(n2). Contoh : algoritma pengurutan yang lambat seperti selection sort
  - O(n!). Merupakan algoritma yang sangat lambat seperti traveling sales person

Cara menghitung *Big O*?

Berikut cara untuk menghitung notasi Big O untuk setiap instruksi di dalam algoritma :

a. **Pengisian nilai (assignment), perbandingan, operasi aritmetik, read, print** membutuhkan waktu O(1).
b. **Pengaksesan elemen larik atau memilih field tertentu dari sebuah record** membutuhkan waktu O(1).

Cara menghitung *Big O*?

Contoh:

```
read(x);                O(1)
x ← x + a[k];           O(1) + O(1) + O(1) = O(1)
print(x);               O(1)
```

Kompleksitas waktu = O(1) + O(1) + O(1) = O(1)

Penjelasan: O(1) + O(1) + O(1) = O(max(1,1)) + O(1)

= O(1) + O(1) = O(max(1,1)) = O(1)

Cara menghitung *Big O*?

c. **if C then S1 else S2**; membutuhkan waktu TC + max(TS1,TS2)

Contoh:

```
read(x);                    O(1)
if (x mod 2 = 0){           O(1)
    x ← x + 1;              O(1)
    print(x);               O(1)
}
```

Kompleksitas waktu :
= O(1) + O(1) + max(O(1)+O(1), O(1))
= O(1) + max(O(1),O(1))
= O(1) + O(1)
= O(1)

Cara menghitung *Big O*?

d. **Kompleksitas waktu perulangan for** adalah jumlah pengulangan dikali dengan kompleksitas waktu badan (body looping).

Contoh :

```
for (i=1;i>n;i++){
        jumlah ← jumlah + a[i];     O(1)
}
```

Kompleksitas waktu :

= n . O(1)
= O(n .1)
= O(n)

Cara menghitung *Big O*?

Contoh:

perulangan bersarang

```
for (i=1;i>n;i++){
      for (j=1;j>n;j++){
            a[i,j] ← 0;     O(1)
      }
}
```

Kompleksitas waktu :

nO(n) = O(n.n) = O(n2)

Cara menghitung *Big O*?

Contoh: **perulangan bersarang dengan 2 buah instruksi**

Total waktu untuk badan perulangan
= O(1) + O(1) = O(1)
Perulangan terluar dieksekusi sebanyak
n kali
Perulangan terdalam dieksekusi sebanyak
i kali, i = 1, 2, …, n
Jumlah pengulangan seluruhnya
= 1 + 2 + … + n = n(n + 1)/2
Kompleksitas waktu :
= n(n + 1)/2 .O(1)
= O(n(n + 1)/2) = O(n2)

```
for (i=1;i>n;i++){
    for (j=1;j>i;j++){
        a ← a + 1;          O(1)
        b ← b - 2;          O(1)
    }
}
```

Cara menghitung *Big O*?

Contoh: **perulangan tunggal sebanyak n-1 putaran**

```
i:=2;                          O(1)
while (i <= n){                O(1)
    jumlah:=jumlah + a[i];     O(1)
    i:=i+1;                    O(1)
}
```

Kompleksitas waktu :

= O(1) + (n-1) { O(1) + O(1) + O(1) }
= O(1) + (n-1) O(1)
= O(1) + O(n-1)
= O(1) + O(n)
= O(n)

Cara menghitung *Big O*?

Contoh: **perulangan yang tidak dapat ditentukan panjangnya**

- Pengulangan akan **berhenti bila x yang dicari ditemukan** di dalam senarai.
- Jika jumlah elemen senarai adalah n, maka kompleksitas waktu terburuknya adalah **O(-n) yaitu kasus x tidak ditemukan**.

```
Ketemu ← false;
while (p <> Nil) and (not ketemu){
      if (p^.kunci = x){
            ketemu:=true
      }else{
            P ← p^.lalu
      }
{ p = Nil or ketemu }
```

Jawaban Pre test

Waktu untuk assignment =
O(1) + O(1) + O(1) = **O(1)**
Waktu untuk struktur if :
max(O(1), O(1)) = **O(1)**
Total waktu pada tubuh perualangan :
O(1) + O(1) = **O(1)**
Perulangan dieksekusi sebanyak 4 kali (5-1) :
4(**O(1)**) = **O(4)**
Kompleksitas waktu keseluruhan :
= **O(1)** + **O(4)** + O(1)
= max{**O(1)**, **O(4)**, O(1)}
= O(4)

```
/IMPLEMENTATION / DEFINITION
    A ← 1                          O(1)
    b ← 0                          O(1)
    x ← 1                          O(1)
    Do {
        If (x mod 2)=0 then {
            A ← A + 2              O(1)
        }else{
            b ← b + 1              O(1)
        }
        x ← x + 1                  O(1)
    }while (x<5)
    Print (A,b)                    O(1)
```

Rangkuman

Notation Big O

- adalah notasi khusus yang dapat digunakan untuk mendeskripsikan kecepatan sebuah algoritma.
- Notasi Big O mencerminkan kompleksitas algoritma.

Kompleksitas algoritma

- Besaran yang digunakan untuk menerangkan model pengukuran waktu/ruang dari sebuah algoritma
- binary search O(log n) lebih cepat dari simple search O(n), akan menjadi jauh lebih cepat setelah daftar item berkembang atau jumlah n berkembang.

Kecepatan algoritma tidak diukur dalam hitungan detik.

3 jenis Kompleksitas waktu

- Tmax(n) : kasus terburuk (worst case) atau kebutuhan waktu maksimum.
- Tmin(n) : kasus terbaik (best case) atau kebutuhan waktu minimum.
- Tavg(n): kasus rata-rata (average case), atau kebutuhan waktu secara rata-rata

5 notasi Big O yang sering ditemui :

- O(log n), juga dikenal dengan log time. Contoh: Binary search.
- O(n), juga dikenal dengan linear time. Contoh: Simple search.
- O(n * log n). Contoh : algoritma pengurutan yang cepat seperti quicksort
- O(n2). Contoh : algoritma pengurutan yang lambat seperti selection sort
- O(n!). Merupakan algoritma yang sangat lambat seperti traveling sales person

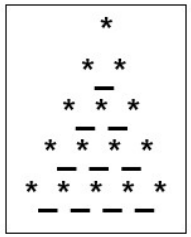
Tugas !

Hitunglah kompleksitas waktu dari jawaban tugas 2 yang sudah dikerjakan pada pertemuan 2 !

NGANTAR ALGORITMA PEMROGRAMAN : ALGORITMA PEMROGRAMAN DAN STRUKTUR DATA | SEMESTER GANJIL 2023/2024

Tugas Pertemuan 2!

Buatlah pseudocode untuk menampilkan segitiga bintang sama sisi seperti berikut :



PROGRAM STUDI S1 TEKNOLOGI INFORMASI | FAKULTAS TEKNIK | UNIVERSITAS TIDAR


Kampus Merdeka
INDONESIA JAYA
UNIVERSITAS TIDAR
TERIMA KASIH
#NEXT... Array & Linked List