

LAPORAN PRAKTIKUM 2
ANALISIS ALGORITMA



Disusun oleh :

Alvin

140810180013

PROGRAM STUDI S1 TEKNIK INFORMATIKA
FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM
UNIVERSITAS PADJADJARAN
2020

PENDAHULUAN

Dalam memecahkan suatu masalah dengan komputer seringkali kita dihadapkan pada pilihan berikut:

1. Menggunakan algoritma yang waktu eksekusinya cepat dengan komputer standar
2. Menggunakan algoritma yang waktu eksekusinya tidak terlalu cepat dengan komputer yang cepat

Dikarenakan keterbatasan sumber daya, pola pemecahan masalah beralih ke pertimbangan menggunakan algoritma. Oleh karena itu diperlukan algoritma yang efektif dan efisien atau lebih tepatnya Algoritma yang mangkus.

Algoritma yang mangkus diukur dari berapa **jumlah waktu dan ruang (space) memori** yang dibutuhkan untuk menjalankannya. Algoritma yang mangkus ialah algoritma yang meminimumkan kebutuhan waktu dan ruang. Penentuan kemangkusan algoritma adakah dengan melakukan pengukuran kompleksitas algoritma.

Kompleksitas algoritma terdiri dari kompleksitas waktu dan ruang. Terminologi yang diperlukan dalam membahas kompleksitas waktu dan ruang adalah:

1. Ukuran input data untuk suatu algoritma, n .
Contoh algoritma pengurutan elemen-elemen larik, n adalah jumlah elemen larik. Sedangkan dalam algoritma perkalian matriks n adalah ukuran matriks $n \times n$.
2. Kompleksitas waktu, $T(n)$, adalah jumlah operasi yang dilakukan untuk melaksanakan algoritma sebagai fungsi dari input n .
3. Kompleksitas ruang, $S(n)$, adalah ruang memori yang dibutuhkan algoritma sebagai fungsi dari input n .

KOMPLEKSITAS WAKTU

Kompleksitas waktu sebuah algoritma dapat dihitung dengan langkah-langkah sebagai berikut:

1. Menetapkan ukuran input
2. Menghitung banyaknya operasi yang dilakukan oleh algoritma.
Dalam sebuah algoritma terdapat banyak jenis operasi seperti operasi penjumlahan, pengurangan, perbandingan, pembagian, pemanggilan prosedur, dsb.

CONTOH

Algoritma Menghitung Nilai Rata-rata

```
procedure HitungRerata (input  $x_1, x_2, \dots, x_n$ : integer, output  $r$ : real)
{ Menghitung nilai rata-rata dari sekumpulan elemen larik integer  $x_1, x_2, \dots, x_n$ .
  Nilai rata-rata akan disimpan di dalam variable  $r$ .
  Input:  $x_1, x_2, \dots, x_n$ 
  Output:  $r$  (nilai rata-rata)
}
```

Deklarasi

```
i : integer
jumlah : real
```

Algoritma

```
Jumlah  $\square$  0
i  $\square$  1
while i  $\leq$  n do
  jumlah  $\square$  jumlah +  $a_i$ 
  i  $\square$  i + 1
endwhile
{i > n}
r  $\square$  jumlah/n {nilai rata-rata}
```

Menghitung Kompleksitas Waktu dari Algoritma Menghitung Nilai Rata-rata

Jenis-jenis operasi yang terdapat di dalam Algoritma HitungRerata adalah:

- Operasi pengisian nilai/*assignment* (dengan operator “ $=$ ”)
- Operasi penjumlahan (dengan operator “ $+$ ”)
- Operasi pembagian (dengan operator “ $/$ ”)

Cara menghitung kompleksitas waktu dari algoritma tersebut adalah dengan cara menghitung masing-masing jumlah operasi. Jika operasi tersebut berada di sebuah loop, maka jumlah operasinya bergantung berapa kali loop tersebut diulangi.

(i) Operasi pengisian nilai (*assignment*)

| | |
|------------------------------------|--------|
| jumlah $\leftarrow 0$, | 1 kali |
| k $\leftarrow 1$, | 1 kali |
| jumlah \leftarrow jumlah + a_k | n kali |
| k $\leftarrow k+1$, | n kali |
| r \leftarrow jumlah/n, | 1 kali |

Jumlah seluruh operasi pengisian nilai (*assignment*) adalah

$$t_1 = 1 + 1 + n + n + 1 = 3 + 2n$$

(ii) Operasi penjumlahan

| | |
|------------------|--------|
| Jumlah + a_k , | n kali |
| k+1, | n kali |

Jumlah seluruh operasi penjumlahan adalah

$$t_2 = n + n = 2n$$

(iii) Operasi pembagian

Jumlah seluruh operasi pembagian adalah

| | |
|----------|--------|
| Jumlah/n | 1 kali |
|----------|--------|

Dengan demikian, kompleksitas waktu algoritma dihitung berdasarkan jumlah operasi aritmatika dan operasi pengisian nilai adalah:

$$T(n) = t_1 + t_2 + t_3 = 3 + 2n + 2n + 1 = 4n + 4$$

Studi Kasus 1: Pencarian Nilai Maksimal

Buatlah programnya dan hitunglah kompleksitas waktu dari algoritma berikut:

Algoritma Pencarian Nilai Maksimal

```
procedure CariMaks(input  $x_1, x_2, \dots, x_n$ : integer, output maks: integer)
{ Mencari elemen terbesar dari sekumpulan elemen larik integer  $x_1, x_2, \dots, x_n$ . Elemen terbesar akan
  disimpan di dalam maks
  Input:  $x_1, x_2, \dots, x_n$ 
  Output: maks (nilai terbesar)
}

Deklarasi
  i: integer

Algoritma
  maks  $\leftarrow x_1$ 
  i  $\leftarrow 2$ 
  while i  $\leq n$  do
    if  $x_i > \text{maks}$  then
      maks  $\leftarrow x_i$ 
    endif
    i  $\leftarrow i + 1$ 
  endwhile
```

Jawab:

```
/*
Nama      : Alvin
NPM       : 140810180013
Kelas    : A
Program   : Pencarian nilai maksimum
*/

#include<iostream>
using namespace std;

main() {
    int x[5]={21,22,99,24,25};
    int n= sizeof(x)/sizeof(x[0]);

    //deklarasi
    int maks = x[0];
    int i= 2;

    //algoritma
    while (i<= n){
        if(x[i] > maks){
            maks = x[i];
        }
        i=i+1;
    }

    cout<<"Nilai maks dari array adalah : "<<maks;

}
```

Kompleksitas waktu:

$$\begin{aligned} T(n) &= 2(n-2) + (n-2) + 2 \\ &= 3n - 4 \end{aligned}$$

Studi Kasus 2: Sequential Search

Diberikan larik bilangan bulat x_1, x_2, \dots, x_n yang telah terurut menaik dan tidak ada elemen ganda. Buatlah programnya dengan C++ dan hitunglah kompleksitas waktu terbaik, terburuk, dan rata-rata dari algoritma pencarian beruntun (*sequential search*). Algoritma *sequential search* berikut menghasilkan indeks elemen yang bernilai sama dengan y . Jika y tidak ditemukan, indeks 0 akan dihasilkan.

```
procedure SequentialSearch(input  $x_1, x_2, \dots, x_n$  : integer,  $y$  : integer, output idx : integer)
{ Mencari  $y$  di dalam elemen  $x_1, x_2, \dots, x_n$ . Lokasi (indeks elemen) tempat  $y$  ditemukan diisi ke dalam idx.
  Jika  $y$  tidak ditemukan, maka idx diisi dengan 0.
  Input:  $x_1, x_2, \dots, x_n$ 
  Output: idx
}
```

Deklarasi

i : integer

found : boolean { bernilai true jika y ditemukan atau false jika y tidak ditemukan }

Algoritma

$i \leftarrow 1$

found \leftarrow false

while ($i \leq n$) and (not found) do

 if $x_i = y$ then

 found \leftarrow true

else

$i \leftarrow i + 1$

endif

endwhile

{ $i < n$ or found }

If found then { y ditemukan }

 idx $\leftarrow i$

else

 idx \leftarrow 0 { y tidak ditemukan }

endif

Jawab:

```
/*
Nama           : Alvin
NPM            : 140810180013
Kelas         : A
Prigram        : Sequential Search
*/

#include<iostream>
using namespace std;

main() {
    int x[5] = {1,2,99,10,1}; //daftar list yang ada
    int y = 99; //yang dicari
    int n = sizeof(x)/sizeof(x[0]);

    //deklrasi
    int i = 1;
    int idx; //output
    bool found = false;

    //algoritma
    while(i<=n && !found){
```

```

        if(x[i] == y){
            found = true;
        }else
            i = i+1;
    }
    if(found == true){
        idx = i;
    }else
        idx = 0; //tidak ditemukan
    cout<<"Hasil cari index elemen : "<<idx;
}

```

Kompleksitas waktu:

- Kasus terbaik: ini terjadi bila $x_1 = y$
 $T_{\min}(n) = 1$
- Kasus terburuk: bila $x_n = y$ atau y tidak ditemukan.
 $T_{\max}(n) = n$
- Kasus rata-rata: Jika y ditemukan pada posisi ke- j , maka operasi perbandingan ($a_k = y$) akan dieksekusi sebanyak j kali.

$$T_{\text{avg}}(n) = (1+2+3+\dots+n)/n = (1/2n(1+n))/n = (n+1)/2$$

Studi Kasus 3: Binary Search

Diberikan larik bilangan bulat x_1, x_2, \dots, x_n yang telah terurut menaik dan tidak ada elemen ganda. Buatlah programnya dengan C++ dan hitunglah kompleksitas waktu terbaik, terburuk, dan rata-rata dari algoritma pencarian bagi dua (*binary search*). Algoritma *binary search* berikut menghasilkan indeks elemen yang bernilai sama dengan y . Jika y tidak ditemukan, indeks 0 akan dihasilkan.

```

procedure BinarySearch(input  $x_1, x_2, \dots, x_n$  : integer,  $x$  : integer, output : idx : integer)
{ Mencari  $y$  di dalam elemen  $x_1, x_2, \dots, x_n$ . Lokasi (indeks elemen) tempat  $y$  ditemukan diisi ke dalam idx.
  Jika  $y$  tidak ditemukan maka idx diisi dengan 0.
  Input:  $x_1, x_2, \dots, x_n$ 
  Output: idx
}
Deklarasi
    i, j, mid : integer
    found : Boolean
Algoritma
    i  $\leftarrow$  1
    j  $\leftarrow$  n
    found  $\leftarrow$  false
    while (not found) and (i  $\leq$  j) do
        mid  $\leftarrow$  (i + j) div 2
        if  $x_{\text{mid}} = y$  then
            found  $\leftarrow$  true
        else

```

```

            if  $x_{\text{mid}} < y$  then {mencari di bagian kanan}
                i  $\leftarrow$  mid + 1
            else {mencari di bagian kiri}
                j  $\leftarrow$  mid - 1
            endif
        endwhile
    {found or i > j}

    If found then
        idx  $\leftarrow$  mid
    else
        idx  $\leftarrow$  0
    endif

```

Jawab:

```
/*
Nama      : Alvin
NPM       : 140810180013
Kelas    : A
Program   : Binary Search
*/

#include<iostream>
using namespace std;

main() {
    int x[5]={1,3,7,9,11}; //input
    int idx; //output
    int y = 7; //angka yang dicari
    int n = sizeof(x)/sizeof(x[0]);

    //deklarasi
    int i, j, mid;
    bool found;

    //algoritma
    i = 1;
    j = n;
    found = false;
    while(!found && i<= j){
        mid = (i + j)/2;
        if (x[mid] == y){
            found = true;
        }
        else if(x[mid] < y){ //menacri bagian kanan
            i = mid+1;
        }
        else{
            j = mid - 1; //mencari bagian kiri
        }
    }
    if(found == true){
        idx=mid;
    }else
    idx= 0;

    cout<<"Hasil cari indeks elemen : "<<idx;

}
```

Kompleksitas waktu:

Best case: Jika ditemukan pada arr[mid] atau indeks di tengah yaitu $T_{\min}(n) = 1$

Average case: Jika ditemukan pada indeks di awal atau di akhir

Worst case: Jika tidak ditemukan sama sekali yaitu $T_{\max}(n) = {}^2\log n$

Studi Kasus 4: Insertion Sort

1. Buatlah program insertion sort dengan menggunakan bahasa C++
2. Hitunglah operasi perbandingan elemen larik dan operasi pertukaran pada algoritma insertion sort.
3. Tentukan kompleksitas waktu terbaik, terburuk, dan rata-rata untuk algoritma insertion sort.

```
procedure InsertionSort(input/output  $x_1, x_2, \dots, x_n$  : integer)
{  Mengurutkan elemen-elemen  $x_1, x_2, \dots, x_n$  dengan metode insertion sort.
  Input:  $x_1, x_2, \dots, x_n$ 
  Output:  $x_1, x_2, \dots, x_n$  (sudah terurut menaik)
}
Deklarasi
  i, j, insert : integer
Algoritma
  for i  $\leftarrow$  2 to n do
    insert  $\leftarrow$   $x_i$ 
    j  $\leftarrow$  i
    while (j < i) and ( $x[j-i] >$  insert) do
       $x[j] \leftarrow x[j-1]$ 
      j  $\leftarrow$  j-1
    endwhile
     $x[j] =$  insert
  endfor
```

Jawab:

```
/*
Nama      : Alvin
NPM       : 140810180013
Kelas    : A
Program   : Insertion Sort
*/

#include<iostream>
using namespace std;

main() {
    int x[5]={1,7,11,31,2};
    int n = sizeof(x)/sizeof(x[0]);

    //deklarasi
    int i , j, insert;

    //Algoritma
    for(i=1; i<n; i++){
        insert= x[i];
        j = i - 1;

        while(j >= 0 && x[j] > insert){
            x[j+1] = x[j];
            j = j - 1;
        }
        x[j+1] = insert;
    }
    for(j = 0; j < n ; j++ ){
        cout<<x[j]<<" ";
    }
}
```


Kompleksitas waktu:

Best case: Jika array sudah terurut sehingga loop while tidak dijalankan

Average case: Jika sebagian elemen array sudah terurut

Worst case: Jika array harus diurutkan sebanyak n kali, Dalam kasus terburuk, bisa ada inversi $n * (n-1) / 2$. Kasus terburuk terjadi ketika array diurutkan dalam urutan terbalik. Jadi kompleksitas waktu kasus terburuk dari jenis penyisipan adalah $O(n^2)$.

| J | Perbandingan | Perpindahan | Total operasi |
|-----|--------------|-------------|----------------------------|
| 2 | 1 | 1 | 2 |
| 3 | 2 | 2 | 4 |
| 4 | 3 | 3 | 6 |
| n | $(n-1)$ | $(n-1)$ | $2(n-1)$ |

Studi Kasus 5: Selection Sort

1. Buatlah program selection sort dengan menggunakan bahasa C++
2. Hitunglah operasi perbandingan elemen larik dan operasi pertukaran pada algoritma selection sort.
3. Tentukan kompleksitas waktu terbaik, terburuk, dan rata-rata untuk algoritma insertion sort.

```
procedure SelectionSort(input/output  $x_1, x_2, \dots, x_n$  : integer)
{ Mengurutkan elemen-elemen  $x_1, x_2, \dots, x_n$  dengan metode selection sort.
  Input:  $x_1, x_2, \dots, x_n$ 
  Output:  $x_1, x_2, \dots, x_n$  (sudah terurut menaik)
}
Deklarasi
  i, j, imaks, temp : integer
Algoritma
  for i  $\leftarrow$  n downto 2 do {pass sebanyak n-1 kali}
    imaks  $\leftarrow$  1
    for j  $\leftarrow$  2 to i do
      if  $x_j > x_{\text{imaks}}$  then
        imaks  $\leftarrow$  j
      endif
    endfor
    {pertukarkan  $x_{\text{imaks}}$  dengan  $x_i$ }
    temp  $\leftarrow$   $x_i$ 
     $x_i \leftarrow x_{\text{imaks}}$ 
     $x_{\text{imaks}} \leftarrow$  temp
  endfor
```

Jawab:

```
/*
Nama       : Alvin
NPM        : 140810180013
Kelas     : A
Program    : Selection Sort
*/

#include<iostream>
using namespace std;

main() {
  int x[5] = { 1,4,2,5,3};
  int n = sizeof(x)/sizeof(x[0]);

  //deklarasi
  int i, j, imaks, temp;
```

```

//algoritma
for ( i=2 ; i<n; i++){
    imaks = 1;
    for( j=2; j<i; j++){
        if ( x[j] > x[imaks]){
            imaks = j;
        }
    }
    temp = x[i];
    x[i] = x[imaks];
    x[imaks] = temp;
}
for (int i=0; i<n; i++){
    cout<<x[i]<<" ";
}
}

```

Kompleksitas waktu:

- (i) Jumlah operasi perbandingan elemen

Untuk setiap loop ke-i,

i = 1 → jumlah perbandingan = n-1

i = 2 → jumlah perbandingan = n-2

i = k → jumlah perbandingan = n-k

i = n-1 → jumlah perbandingan = 1

sehingga $T(n) = (n-1) + (n-2) + \dots + 1 = n(n-1)/2$ dimana kompleksitas waktu ini berlaku menjadi yang terbaik, rata-rata maupun yang terburuk karena algoritma ini tidak melihat apakah arraynya sudah urut atau tidak terlebih dahulu.

- (ii) Jumlah operasi pertukaran

Untuk setiap loop ke-1 sampai n-1 terjadi satu kali pertukaran elemen sehingga $T(n) = n-1$.