

TUGAS 2 PRAKTIKUM
KOMPLEKSITAS WAKTU DARI ALGORITMA



Muhammad Luthfiansyah
140810170023

PROGRAM STUDI S-1 TEKNIK INFORMATIKA
FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM
UNIVERSITAS PADJADJARAN
2020

Nomor 1

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:

- Source code

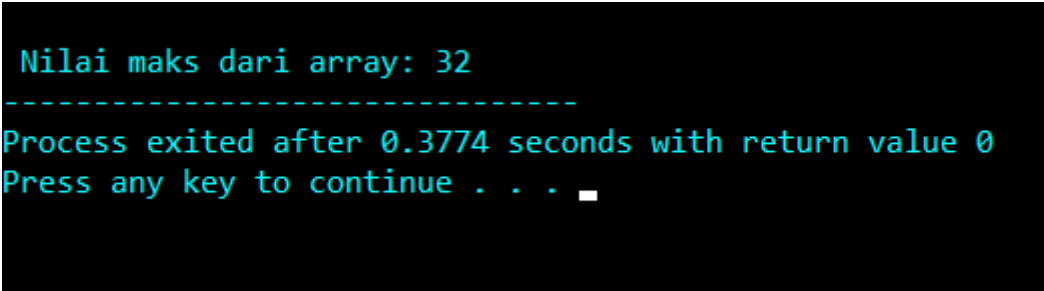
```
#include <iostream>
using namespace std;

int main(){
    int arr[5] = {2,4,8,16,32};
    int panjangArr = sizeof(arr)/sizeof(arr[0]);
    int maks = arr[0];
    int i = 2;

    while(i<panjangArr){
        if(arr[i] > maks){
            maks = arr[i];
        }
        i++;
    }

    cout<<"Nilai maks dari array: "<<maks;
}
```

- Hasil screenshot



```
Nilai maks dari array: 32
-----
Process exited after 0.3774 seconds with return value 0
Press any key to continue . . .
```

- Kompleksitas waktu:

Best case: Jika nilai maks berada pada `arr[0]` atau index paling awal

Average case: Jika nilai maks berada pada `arr[(n-1)/2]` atau index di tengah

Worst case: Jika nilai maks berada pada `arr[n-1]` atau index paling akhir

Nomor 2

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:

- Source code

```
#include <iostream>
using namespace std;

int main(){
    int arr[5] = {1,3,5,7,9};
    int panjangArr = sizeof(arr)/sizeof(arr[0]);
    int i = 1;
    int y = 3;
    int index;
    bool found = false;
```

```

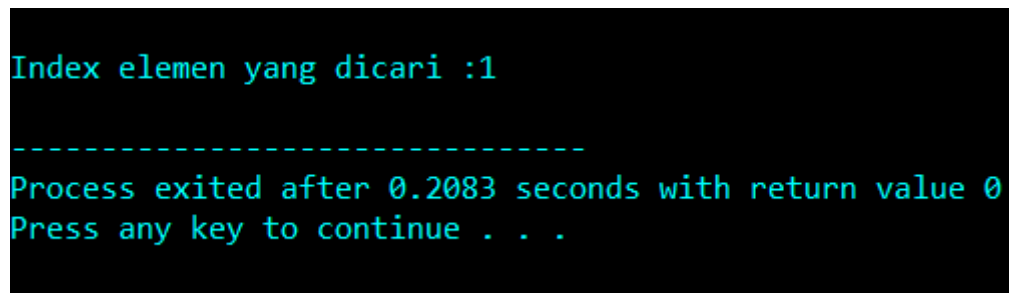
while(i <= panjangArr && !found){
    if(arr[i] == y){
        found = true;
    }
    else i++;
}

if(found == true){
    index = i;
}
else index = 0; //jika tidak ditemukan index = 0

cout<<"\nIndex elemen yang dicari :"<<index<<endl;
}

```

- Hasil screenshoot



```

Index elemen yang dicari :1
-----
Process exited after 0.2083 seconds with return value 0
Press any key to continue . . .

```

- Kompleksitas waktu:
 Best case: Jika ditemukan pada arr[0] atau indeks paling awal
 Average case: Jika ditemukan pada arr[(n-1)/2] atau indeks di tengah
 Worst case: Jika ditemukan pada arr[n-1] atau indeks paling akhir atau tidak ditemukan sama sekali

Nomor 3

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
    endif
  endwhile
  {found or i > j}

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

Jawab:

- Source code

```
#include <iostream>
using namespace std;

int main(){
    int arr[5] = {1,2,3,4,5};
    int i = 2;
    int j = sizeof(arr)/sizeof(arr[0]);
    int y = 4;
```

```

int index, mid;
bool found = false;

while(!found && i <= j){
    mid = (i + j)/2;
    if(arr[mid] == y){
        found = true;
    }
    else if(arr[mid] < y){
        i = mid + 1;
    }
    else {
        j = mid - 1;
    }
}

if(found == true){
    index = mid;
}
else index = 0;

cout<<"\nIndex elemen yang dicari: "<<index<<endl;
}

```

- Hasil screenshoot

```

Index elemen yang dicari: 3
-----
Process exited after 0.02829 seconds with return value 0
Press any key to continue . . .

```

- Kompleksitas waktu:

Best case: Jika ditemukan pada arr[mid] atau indeks di tengah

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

Worst case: Jika tidak ditemukan sama sekali

Nomor 4

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:

- Source code

```
#include <iostream>
using namespace std;

int main(){
    int arr[5] = {4,5,1,2,3};
    int panjangArr = sizeof(arr)/sizeof(*arr);
    int i, j, insert;

    for (i = 1; i < panjangArr; i++){
        insert = arr[i];
        j = i - 1;

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


- Hasil screenshot

```
12345
-----
Process exited after 0.2678 seconds with return value 0
Press any key to continue . . .
```

- 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

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

Nomor 5

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:

- Source code

```
#include <iostream>
using namespace std;

int main(){
    int arr[5] = {5,2,4,3,1};
    int panjangArr = sizeof(arr)/sizeof(*arr);

    for(int i = 0; i < panjangArr - 1; i++){
        int minIdx = i;
        for(int j = i + 1; j < panjangArr; j++){
            if(arr[j] < arr[minIdx]) minIdx = j;
        }
        int temp = arr[minIdx];
        arr[minIdx] = arr[i];
        arr[i] = temp;
    }

    for(int i = 0; i < panjangArr; i++){
        cout<<arr[i];
    }
}
```

- Hasil screenshot

```
12345
-----
Process exited after 0.2749 seconds with return value 0
Press any key to continue . . .
```

- Kompleksitas waktu:

- (i) Jumlah operasi perbandingan elemen

Untuk setiap loop ke-i,

$i = 1 \rightarrow \text{jumlah perbandingan} = n-1$

$i = 2 \rightarrow \text{jumlah perbandingan} = n-2$

$i = k \rightarrow \text{jumlah perbandingan} = n-k$

$i = n-1 \rightarrow \text{jumlah perbandingan} = 1$

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 sudahurut atau tidak terlebih dahulu.

- (ii) Jumlah operasi pertukaran

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