LAPORAN PRAKTIKUM
ANALISIS ALGORITMA

DISUSUN OLEH

SALMA ALIFIA SHAFIRA 140810180058

UNIVERSITAS PADJADJARAN

FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM

TEKNIK INFORMATIKA

2020

Studi Kasus 1: Pencarian Nilai Maksimal

Buatlah programnya dan hitunglah kompleksitas waktu dari algoritma berikut:

Algoritma Pencarian Nilai Maksimal

```
procedure CariMaks(input x1, x2, …, xn: integer, output maks: integer)
{   Mencari elemen terbesar dari sekumpulan elemen larik integer x1, x2, …, xn. Elemen terbesar akan
    disimpan di dalam maks
    Input: x1, x2, …, xn
    Output: maks (nilai terbesar)
}

Deklarasi
        i : integer

Algoritma
        maks ← x1
        i ← 2
        while i ≤ n do
           if xi > maks then
                maks ← xi
           endif
           i ← i + 1
        endwhile
```

Jawaban Studi Kasus 1

- Operator Assignment:

Baris 1) 1 kali
Baris 2) 1 kali
Baris 5) n-1 kali
Baris 7) n-1 kali
t_1 = 1 + 1 + (n-1) + (n-1) = **2n**

- Operator Perbandingan:

Baris 3) n-1 kali
Baris 4) n-1 kali
t_2 = (n -1) + (n-1) = **2n - 2**

- Operator Penjumlahan:

Baris 7) n-1 kali
t_3 = **n-1**

```
1   /*
2   Nama    : Salma Alifia Shafira
3   Kelas   : B
4   NPM     : 140810180058
5   */
6
7   #include<iostream>
8   using namespace std;
9
10  int main(){
11      int i, n, maks;
12
13      cout << "Jumlah data : "; cin >> n;
14      int x[n];
15      for(i=1; i<=n; i++){
16          cout << "x [ " << i <<" ] : "; cin >> x[i];
17      }
18      maks = x[1];
19      i = 2;
20      while(i<n){
21          if(x[i] > maks){
22              maks = x[i];
23          }
24          i = i+1;
25      }
26      cout << "Maks = " << maks;
27  }
28
```

```
C:\Users\salma\Desktop\CoolYeah!\SEMESTER4\Praktikum\Analgo\Latihan2\Case1.exe
Jumlah data : 3
x [ 1 ] : 2
x [ 2 ] : 1
x [ 3 ] : 4
Maks = 2
--------------------------------
Process exited after 16.81 seconds with return value 0
Press any key to continue . . .
```

PEMBAGIAN KOMPLEKSITAS WAKTU

Hal lain yang harus diperhatik an dalam menghitung kompleksitas waktu suatu algoritma adalah parameter yang mencirikan ukuran input. Contoh pada algoritma pencarian, waktu yang dibutuhkan untuk melakukan pencarian tidak hanya bergantung pada ukuran larik (n) saja, tetapi juga bergantung pada nilai elemen (x) yang dicari.

Misalkan:

- Terdapat sebuah larik dengan panjang elemen 130 dimulai dari $y_1, y_2, \ldots y_n$
- Asumsikan elemen-elemen larik sudah terurut. Jika $y_1 = x$, maka waktu pencariannya lebih cepat 130 kali dari pada $y_{130} = x$ atau x tidak ada di dalam larik.
- Demikian pula, jika $y_{65} = x$, maka waktu pencariannya ½ kali lebih cepat daripada $y_{130} = x$

Oleh karena itu, kompleksitas waktu dibedakan menjadi 3 macam:

(1) $T_{NIN}(n)$: kompleksitas waktu untuk kasus terbaik (***best case***)
merupakan kebutuhan waktu minimum yang diperlukan algoritma sebagai fungsi dari n.

(2) $T_{avg}(n)$: kompleksitas waktu untuk kasus rata-rata (***average case***)
merupakan kebutuhan waktu rata-rata yang diperlukan algoritma sebagai fungsi dari n. Biasanya pada kasus ini dibuat asumsi bahwa semua barisan input bersifat sama. Contoh pada kasus *searching* diandaikan data yang dicari mempunyai peluang yang sama untuk tertarik dari larik.

(3) $T_{NAS}(n)$: kompleksitas waktu untuk kasus terburuk (***worst case***)
merupakan kebutuhan waktu maksimum yang diperlukan algoritma sebagai fungsi dari n.

Studi Kasus 2: *Sequential Search*

Diberikan larik bilangan bulan $x_1, x_2, \ldots x_n$ 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 x1, x2, ... xn : integer, y : integer, output idx : integer)
{   Mencari y di dalam elemen x1, x2, ... xn. Lokasi (indeks elemen) tempat y ditemukan diisi ke dalam idx.
    Jika y tidak ditemukan, makai idx diisi dengan 0.
    Input: x1, x2, ... xn
    Output: idx
}
```

Deklarasi

i : integer

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

Algoritma

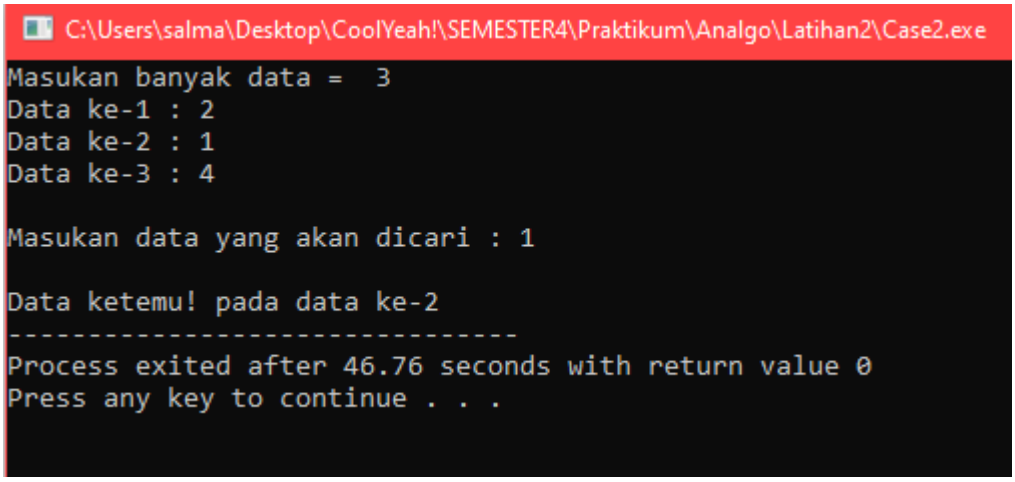
```
i ← 1
found ← false
while (i ≤ n) and (not found) do
    if xi = y then
        found ← true
    else
        i ← i + 1
    endif
endwhile
{i < n or found}

If found then {y ditemukan}
        idx ← i
else
        idx ← 0 {y tidak ditemukan}
endif
```

Jawaban Studi Kasus 2

Kasus terbaik: ini terjadi bila a1 = x.
Tmin(n) = 1
Kasus terburuk: bila an = x atau x tidak ditemukan.
Tmax(n) = n
Kasus rata-rata: Jika x ditemukan pada posisi ke-j, maka operasi perbandingan (ak = x) akan dieksekusi sebanyak jkali.
Tavg(n) = (1+2+3+..+n)/n = (1/2n(1+n))/n = (n+1)/2

```
4   NPM     : 140810180058
5   */
6
7   #include <iostream>
8   using namespace std;
9
10  main()
11  {
12      int n, cari, A[100], index, jwb;
13      bool ketemu = false;
14
15      cout << "Masukan banyak data =  "; cin >> n;
16
17      for(int i=0; i<n; i++)
18      {
19         cout << "Data ke-" << i+1 << " : ";
20         cin >> A[i];
21      }
22
23      cout << "\nMasukan data yang akan dicari : "; cin >> cari;
24
25      for(int i=0; i<n; i++){
26          if(A[i] == cari){
27              ketemu = true;
28              index = i;
29              i = n;
30          }
31      }
32
33      if(ketemu == true){
34         cout << "\nData ketemu! pada data ke-" << index+1;
35      }
36      else{
37         cout << "\nMaaf. Data tidak ditemukan!";
38      }
39
40      return 0;
41  }
```



Studi Kasus 3: *Binary Search*

Diberikan larik bilangan bulan $x_1, x_2, \ldots x_n$ 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 x1, x2, ... xn : integer, x : integer, output : idx : integer)
{   Mencari y di dalam elemen x1, x2, ... xn. Lokasi (indeks elemen) tempat y ditemukan diisi ke dalam idx.
    Jika y tidak ditemukan makai dx diisi dengan 0.
    Input: x1, x2, ... xn
    Output: idx
}
Deklarasi
        i, j, mid : integer
        found : Boolean
Algoritma
        i ← 1
        j ← n
        found ← false
        while (not found) and ( i ≤ j) do
                mid ← (i + j) div 2
                if xmid = y then
                    found ← true
                else
```

```
            if x_mid < y then     {mencari di bagian kanan}
                i ← mid + 1
            else                  {mencari di bagian kiri}
                j ← mid – 1
            endif
        endif
    endwhile
    {found or i > j }

    If found then
        Idx ← mid
    else
        Idx ← 0
    endif
```

Jawaban Studi Kasus 3

Kasus terbaik

Tmin(n) = 1

Kasus terburuk

Tmax (n) = 2log n

```
/*
Nama    : Salma Alifia Shafira
Kelas   : B
NPM     : 140810180058
*/

#include<iostream>
using namespace std;

int main()
{

    int n, i, arr[100], cari, awal, akhir, tengah;

    cout<<"Masukkan banyak data : ";cin>>n;

    for (i=0; i<n; i++)
    {
        cout<<"Data ke-"<<i+1<<" :";
        cin>>arr[i];
    }

    cout<<"\nMasukkan data yang akan di cari :"; cin>>cari;
    awal = 0;
    akhir = n-1;

    while (awal <= akhir)
    {
        tengah = (awal+akhir)/2;
        if(arr[tengah] < cari)
        {
            awal = tengah + 1;

        }
        else if(arr[tengah] == cari)
        {
            cout<<cari<<" ditemukan pada data ke-"<<tengah+1<<"\n";
            break;
```

```
39          }
40          else
41          {
42              akhir = tengah - 1;
43          }
44          tengah = (awal + akhir)/2;
45      }
46
47      if(awal > akhir)
48      {
49          cout<<"Data "<<cari<<" Tidak Ditemukan!";
50      }
51
52      return 0;
53  }
```

```
C:\Users\salma\Desktop\CoolYeah!\SEMESTER4\Praktikum\Analgo\Latihan2\Case3.exe
Masukkan banyak data : 3
Data ke-1 :2
Data ke-2 :1
Data ke-3 :4

Masukkan data yang akan di cari :4
4 ditemukan pada data ke-3

--------------------------------
Process exited after 7.147 seconds with return value 0
Press any key to continue . . .
```

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 $x_1, x_2, \ldots x_n$: integer)
{ Mengurutkan elemen-elemen $x_1, x_2, \ldots x_n$ dengan metode insertion sort.
Input: $x_1, x_2, \ldots x_n$
OutputL $x_1, x_2, \ldots x_n$ (sudah terurut menaik)
}
Deklarasi
i, j, insert : integer
Algoritma
for i ← 2 to n do
insert ← x_i
j ← i
while (j < i) and (x[j-i] > insert) do
x[j]← x[j-1]
j←j-1
endwhile
x[j] = insert
endfor

Jawaban Studi Kasus 4
Loop sementara dijalankan hanya jika i> j dan arr [i] <arr [j]. Jumlah total iterasi loop sementara (Untuk semua nilai i) sama dengan jumlah inversi. Kompleksitas waktu keseluruhan dari jenis penyisipan adalah O (n + f (n)) di mana f (n) adalah jumlah inversi.Jika jumlah inversi adalah O(n), maka kompleksitas waktu dari jenis penyisipan adalah O(n).

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 (n2).

```cpp
using namespace std;

int data[100],data2[100],n;

void insertion_sort()
{
    int temp,i,j;
    for(i=1;i<=n;i++){
        temp = data[i];
        j = i -1;
        while(data[j]>temp && j>=0){
            data[j+1] = data[j];
            j--;
        }
        data[j+1] = temp;
    }
}
int main()
{
    cout<<"Masukkan Jumlah Data : "; cin>>n;
    cout<<endl;

    for(int i=1;i<=n;i++)
    {
      cout<<"Masukkan data ke-"<<i<<" : ";
      cin>>data[i];
      data2[i]=data[i];
    }

    insertion_sort();
    cout<<"\nData Setelah di Sort : "<<endl;
    for(int i=1; i<=n; i++)
    {
      cout<<data[i]<<" ";
    }

    return 0;
}
```

```
C:\Users\salma\Desktop\CoolYeah!\SEMESTER4\Praktikum\Analgo\Latihan2\Case4.exe
Masukkan Jumlah Data : 3

Masukkan data ke-1 : 2
Masukkan data ke-2 : 1
Masukkan data ke-3 : 4

Data Setelah di Sort :
1 2 4
--------------------------------
Process exited after 3.193 seconds with return value 0
Press any key to continue . . .
```

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 x1, x2, … xn : integer)
{ Mengurutkan elemen-elemen x1, x2, … xn dengan metode selection sort.
    Input: x1, x2, … xn
    OutputL x1, x2, … xn (sudah terurut menaik)
}
Deklarasi
        i, j, imaks, temp : integer
Algoritma
        for i ← n downto 2 do {pass sebanyak n-1 kali}
            imaks ← 1
            for j ← 2 to i do
              if xj > ximaks then
                imaks ← j
              endif
            endfor
            {pertukarkan ximaks dengan xi}
            temp ← xi
            xi ← ximaks
            ximaks ← temp
        endfor
```

Jawaban Studi Kasus 5

a. Jumlah operasi perbandingan element. Untuk setiap *pass* ke-*i*,

i = 1 –> jumlah perbandingan = *n* – 1
i = 2 –> jumlah perbandingan = *n* – 2
i = 3 –> jumlah perbandingan = *n* – 3
: *i* = *k* –> jumlah perbandingan = *n* – *k*
: *i* = *n* – 1 –> jumlah perbandingan = 1

Jumlah seluruh operasi perbandingan elemen-elemen larik adalah $T(n) = (n-1) + (n-2) + \ldots + 1$

Ini adalah kompleksitas waktu untuk kasus terbaik dan terburuk, karena algoritma Urut tidak bergantung pada batasan apakah data masukannya sudah terurut atau acak.

b. Jumlah operasi pertukaran

Untuk setiap *i* dari 1 sampai *n* – 1, terjadi satu kali pertukaran elemen, sehingga jumlah operasi pertukaran seluruhnya adalah $T(n) = n - 1$.

Jadi, algoritma pengurutan maksimum membutuhkan $n(n-1)/2$ buah operasi perbandingan elemen dan *n* – 1 buah operasi pertukaran.

```cpp
/*
Nama    : Salma Alifia Shafira
Kelas   : B
NPM     : 140810180058
*/

#include <iostream>
using namespace std;

int data[100],data2[100];
int n;

void tukar(int a, int b)
{
    int t;
    t = data[b];
    data[b] = data[a];
    data[a] = t;
}
void selection_sort()
{
    int pos,i,j;
    for(i=1;i<=n-1;i++)
    {
        pos = i;
        for(j = i+1;j<=n;j++)
        {
           if(data[j] < data[pos]) pos = j;
        }
        if(pos != i) tukar(pos,i);
    }
}

int main()
{
    cout<<"Masukkan Jumlah Data : ";cin>>n;

    for(int i=1;i<=n;i++)
    {
        cout<<"Masukkan data ke-"<<i<<" : ";
        cin>>data[i];
        data2[i]=data[i];
    }

    selection_sort();

    cout<<"Data Setelah di Sort : "<<endl;
    for(int i=1; i<=n; i++)
    {
        cout<<" "<<data[i];
    }

    return 0;
}

```

```
C:\Users\salma\Desktop\CoolYeah!\SEMESTER4\Praktikum\Analgo\Latihan2\Case5.exe
Masukkan Jumlah Data : 3
Masukkan data ke-1 : 2
Masukkan data ke-2 : 1
Masukkan data ke-3 : 4
Data Setelah di Sort :
 1 2 4
--------------------------------
Process exited after 7.985 seconds with return value 0
Press any key to continue . . .
```