LAPORAN PRAKTIKUM 2

ANALISIS ALGORITMA

DISUSUN OLEH

AHMAD IRFAN FADHOLI

140810180034

PROGRAM STUDI TEKNIK INFORMATIKA

FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM

UNIVERSITAS PADJADJARAN

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
```

PEMBAGIAN KOMPLEKSITAS WAKTU

Hal lain yang harus diperhatikan 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) $T_{min}(n)$: 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) $T_{max}(n)$: kompleksitas waktu untuk kasus terburuk (***worst case***)
merupakan kebutuhan waktu maksimum yang diperlukan algoritma sebagai fungsi dari n.

```cpp
/*
Nama : Ahmad Irfan Fadholi
NPM : 14081080034
Tgl : 10 Maret 2020
Deskripsi : Studi Kasus No 1*/

#include <iostream>
using namespace std;

int main(){
    int x[5] = {500, 333, 231, 5, 1};
    int n = sizeof(x)/sizeof(x[0]);
    int maks = x[0];
    int i = 2;

    while (i <= n){
        if (x[i] > maks)
            maks = x[i];
        i = i + 1;
    }
    cout << "Maximum = " << maks;

    return 0;
}
```

Operasi Assignment = 1 + 1 + (n-1) + (n-1) = 2n
Operasi Perbandingan = n-1
Operasi Penjumlahan = n-1

Maka $T_{max}(n)$ = 4n-2

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 x_1, x_2, ... x_n : integer, y : integer, output idx : integer)
{ Mencari y di dalam elemen x_1, x_2, ... x_n. Lokasi (indeks elemen) tempat y ditemukan diisi ke dalam idx.
  Jika y tidak ditemukan, makai idx diisi dengan 0.
  Input: x_1, x_2, ... x_n
  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 ← o {y tidak ditemukan}
endif
```

```cpp
/*
Nama : Ahmad Irfan Fadholi
NPM : 14081080034
Tgl : 10 Maret 2020
Deskripsi : Studi Kasus No 2*/

#include <iostream>
using namespace std;

int main()
{
    int x[5] = {739, 323, 69, 1, 33};
    int find = 739;
    int n = sizeof(x) / sizeof(x[0]);

    int index;
    int i = 1;
    bool found = false;

    while (i <= n && not found){
        if (x[i] == find)
            found = true;
        else
            i = i + 1;
    }

    if (found == true){
        index = i;
        cout << "Ditemukan pada Index " << index;
    }

    else{
        index = 0;
        cout << "Not Found";
```

```
35.     }
36.
37.     return 0;
38. }
```

Jawab :

1.Operasi Assignment = 4

2.Operasi Perbandingan = 2

$T_{min}(n)$ = 4 + 2 = 6

1.Operasi Assignment = 1 + 1 + n + 1 = 3 + n

2.Operasi Perbandingan = n + 1

3.Operasi Penjumlahan = n

$T_{max}(n)$ = 3+n+n+1+n=3n+4

($T_{min}(n)$ + $T_{max}(n)$)/ 2 = (6+4+3n) / 2 = (10+3n) / 2

$T_{avg}(n)$ = (10+3n) / 2

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 xmid < 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
```

```
1.  /*
2.  Nama : Ahmad Irfan Fadholi
3.  NPM : 14081080034
4.  Tgl : 10 Maret 2020
5.  Deskripsi : Studi Kasus No 3 - Binary Search*/
6.
7.  #include <iostream>
8.  using namespace std;
9.
10. int main(){
```

```
11.     int x[5] = {14, 31, 59, 72, 98};
12.     int find = 59;
13.     int mid;
14.     int n = sizeof(x) / sizeof(x[0]);
15.     int index;
16.     int i = 1;
17.     int j = n;
18.     bool found = false;
19.
20.     while (not found && i <= j){
21.         mid = (i + j) / 2;
22.         if (x[mid] == find)
23.             found = true;
24.         else if (x[mid] < find)
25.             i = mid + 1;
26.         else
27.             j = mid - 1;
28.     }
29.
30.     if (found){
31.         index = mid;
32.         cout << "Index ditemukan pada :  " << index;
33.     }
34.     else{
35.         index = 0;
36.         cout << "Not Found";
37.     }
38.
39.     return 0;
40. }
```

Jawab :

1.Operasi Assignment = 6

2.Operasi Perbandingan = 2

$T_{min}(n)$ = 6 + 2 = 8

Panjang array akan berubah pada setiap iterasi:

•Iterasi 1 = n

•Iterasi 2 = n/2

•Iterasi 3 = n/22

•Iterasi x = n/2k-1 ~ n/2k (-1 diabaikan karena kecil dibanding n/2k)

Panjang array menjadi 1.

Maka,

n/2k= 1

n= 2k

log 2(n) = log 2(2k) = k log 2(2)

k= log 2(n)

$T_{max}(n)$ = (log 2(n))

($T_{min}(n)$ + $T_{max}(n)$)/ 2 = (1 + log 2(n)) / 2

$T_{avg}(n)$ = (1 + log 2(n)) / 2

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 x1, x2, ... xn : integer)
{   Mengurutkan elemen-elemen x1, x2, ... xn dengan metode insertion sort.
    Input: x1, x2, ... xn
    OutputL x1, x2, ... xn (sudah terurut menaik)
}
Deklarasi
        i, j, insert : integer
Algoritma
        for i ← 2 to n do
            insert ← xi
            j ← i
            while (j < i) and (x[j-i] > insert) do
                x[j]← x[j-1]
                j←j-1
            endwhile
            x[j] = insert
        endfor
```

```
1.  /*
2.  Nama : Ahmad Irfan Fadholi
3.  NPM : 14081080034
4.  Tgl : 10 Maret 2020
5.  Deskripsi : Studi Kasus No 4 - Insertion Sort*/
6.
7.  #include <iostream>
8.  using namespace std;
9.
10. int x[100], y[100], n;
11.
12. void InsertionSort(){
13.     int temp, i, j;
```

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

int main(){
    cout << "Insertion Sort\nMasukkan Banyak data : ";
    cin >> n;
    for (int i = 1; i <= n; i++){
        cout << "Masukkan data ke-" << i << " : ";
        cin >> x[i];
        y[i] = x[i];
    }
    InsertionSort();
    cout << "\nHasil Output : ";

    for (int i = 1; i <= n; i++){
        cout << x[i] << " ";
    }
}
```

Jawab :

1.Operasi Assignment: 2(n-1) + (n-1) = 3n-3

2.Operasi Perbandingan: 2*((n-1) + (n-1)) = 2*(2n-2) = 4n-4

3.Operasi Pertukaran: (n-1) * n = n2-n

$T_{min}(n)$ = 3n-3 + 4n-4 + 1 = 7n – 6

$T_{max}(n)$ = 3n-3 + 4n-4 + n2-n = n2+6n-6

($T_{min}(n)$+ $T_{max}(n)$)/ 2 = (7n–6 + n2+6n-6) / 2

$T_{avg}(n)$ = (n2 + 13n - 12) / 2

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
```

```
1.  /*
2.  Nama : Ahmad Irfan Fadholi
3.  NPM : 14081080034
4.  Tgl : 10 Maret 2020
5.  Deskripsi : Studi Kasus No 5 - Selection Sort*/
6.
7.  #include <iostream>
8.  using namespace std;
9.
10. int x[100], y[100],n;
11.
12. void swap(int a, int b){
13.     int temp;
14.     temp = x[b];
15.     x[b] = x[a];
16.     x[a] = temp;
17. }
18.
19. void SelectionSort(){
20.     int pos, i, j;
21.     for (i = 1; i <= n - 1; i++){
22.         pos = i;
23.         for (j = i + 1; j <= n; j++){
24.             if (x[j] < x[pos])
25.                 pos = j;
26.         }
27.         if (pos != i)
28.             swap(pos, i);
29.     }
30. }
31.
```

```
32. int main(){
33.     cout << "Selection Sort \nMasukkan banyak data : ";
34.     cin >> n;
35.
36.     for (int i = 1; i <= n; i++){
37.         cout << "Masukkan data ke-" << i << " : ";
38.         cin >> x[i];
39.         y[i] = x[i];
40.     }
41.     SelectionSort();
42.     cout << "\nHasil Output : ";
43.     for (int i = 1; i <= n; i++){
44.         cout << " " << x[i];
45.     }
46.     return 0;
47. }
```

Jawab :

1.Operasi Perbandingan = ½(n^2-n)

2.Operasi Pertukaran = n-1

$T_{min}(n)$ = (4n-4) + ½(n2-n) +1 ~ n2

$T_{max}(n)$ = ½(n2-n) + (n-1) ~ n2

($T_{min}(n)$ + $T_{max}(n)$)/ 2 = (n2+ n2) / 2

$T_{avg}(n)$ = n2