TUGAS PRAKTIKUM 2

KOMPLEKSITAS WAKTU DARI ALGORITMA

MATA KULIAH

ANALISIS ALGORITMA

D10G.4205 & D10K.0400601

Disusun oleh :
Muhammad Reza Atthariq Kori
140810180060

Tugas 1

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

1. Operator Assignment

```
maks <- x1                      : 1 kali
i <- 2                          : 1 kali
while i ≤ n do
     if xi > maks then          : n-1 kali
          maks <- xi            : n-1 kali
     endif
     i <- i + 1                 : n-1 kali
endwhile
```

   t1 = 1+1+n-1+n-1+n-1 = 3n-1

2. Operator Pertambahan

```
maks <- x1
i <- 2
while i ≤ n do
     if xi > maks then
          maks <- xi
     endif
     i <- i + 1                 : n kali
endwhile
```

   t2 = n

Kompleksitas :

T(n) = t1 + t2 = 4n-1

Tugas 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 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

a. Program

b. Kompleksitas Waktu :

i <- 1	: 1 kali
found <- false	: 1 kali
while (i ≤ n) and (not found) do	
if xi = y then	: n kali

```
            found <- true              : n kali
      else
            i <- i + 1                 : n kali
      endif
endwhile
{i < n or found}                       : 1 kali

If found then {y ditemukan}            : n kali
      idx <- I                         : n kali
else
      idx <- 0 {y tidak ditemukan}     : n kali
endif
```

- Kompleksitas waktu terbaik (best case) : $T_{min}(n) = 1$ atau ini terjadi apabila $a_1 = x$.
- Kompleksitas waktu terburuk (worst case) : $T_{max}(n) = n$ atau ini terjadi apabila $a_n = x$ atau nilai x tidak ditemukan
- Kompleksitas waktu rata2 adalah Jika x ditemukan pada posisi ke-*j*, maka operasi perbandingan
  ($a_k = x$) akan dieksekusi sebanyak *j* kali.

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

Tugas 3

Studi Kasus 3: *Binary Search*

Diberikan larik bilangan bulan $x_1, x_2, \ldots\ x_n$ yang <u>telah terurut</u> 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 waktu terbaik (Best case)
  $T_{min}(n) = 1$
- Kasus waktu rata2 : jika terdapat pada index pada awal atau akhir elemen.
- Kasus waktu terburuk (Worst case):
  $T_{max}(n) = {}^2\log n$

Tugas 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 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
```

Jawaban Studi Kasus 4 :

Jawaban Studi Kasus 4

- Kasus waktu terbaik (Best case) : jika array yang ada sudah terurut dengan benar, jadi looping tidak akan dilakukan lagi.
- Kasus waktu rata2 : jika array elemen yang ada sudah terurut setengahnya / sebagian dari seluruh elemen

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).

Tugas 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, ... x_n : integer)
{   Mengurutkan elemen-elemen x_1, x_2, ... x_n dengan metode selection sort.
    Input: x_1, x_2, ... x_n
    OutputL x_1, x_2, ... x_n (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 x_j > x_imaks then
                imaks ← j
              endif
            endfor
            {pertukarkan x_imaks dengan x_i}
            temp ← x_i
            x_i ← x_imaks
            x_imaks ← 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) + … + 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.