Naufal Ariful Amri

140810180009

Studi Kasus 1: Pencarian Nilai Maksimal

Buatlah programnya dan hitunglah kompleksitas waktu dari algoritma berikut:

Algoritma Pencarian Nilai Maksimal

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

Deklarasi
    i : integer

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

Jawaban Studi Kasus 1:

Kompleksitas waktu algoritma dihitung berdasarkan jumlah operasi perbandingan elemen larik (A[i] > maks). Kompleksitas waktu CariElemenTerbesar :

Jumlah operasi perbandingan elemen tabel:

Line 4 -> n - 1 kali

Line 5 -> n - 1 kali

Line 7 -> n - 1 kali

Total

$$= (n-1) + (n-1) + (n-1)$$

$$= 3n - 3$$

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.

```
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

Jumlah operasi perbandingan elemen tabel:

1. *Kasus terbaik*: ini terjadi bila a_1 = x

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

2. *Kasus terburuk*: bila $a_n = x$ atau x tidak ditemukan.

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

3. *Kasus rata-rata*: Jika x ditemukan pada posisi ke-j, maka operasi perbandingan (a_k = x) akan dieksekusi sebanyak j kali.

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

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
                            i ← mid + 1 else
                            j ← mid – 1 endif
                    endif
         endwhile
         {found or i > j }
    If found then
            Idx ← mid
    else
            Idx ← 0
    endif
```

Jawaban Studi Kasus 3

1. *Kasus terbaik*: ini terjadi bila a_1 = x

$$T_{max}(n) = 1$$

2. *Kasus terburuk*: jika angka terakhir ketemu dengan cara setiap while akan membagi 2 (n , n/2 , n/4 , n/8, . . . 2logn)

$$T_{max}(n) = \frac{n}{2^k} = 1; n = 2^k ; k = \log_2 n$$

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 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-1
             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

Worst Case : Ketika j yang lebih kecil dari i ketemu di paling akhir. Maka worst case nya

$$T_{min}(n) = n^2$$

Best Case : Ketika j yang leih kecil dari I ketemu di paling awal. Maka best case nya

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

Operasi Insertion sort

Line 2 -> n - 1 kali
Line 4 -> n - 1 kali
Line 5 -> $\sum_{j=2}^{n} t_j \ kali$
Line 6 -> $\sum_{j=2}^{n} (t_j - 1) \ kali$
Line 7 -> $\sum_{j=2}^{n} (t_j - 1) \ kali$
Line 8 -> n - 1 kali

Studi Kasus 5: Selection Sort

Buatlah program selection sort dengan menggunakan bahasa C++
Hitunglah operasi perbandingan elemen larik dan operasi pertukaran pada algoritma selection sort.
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. Operasi Perbandingan

$i = 1 \rightarrow perbandingan = n - 1$

$i = 2 \rightarrow perbandingan = n - 2$

$i = 3 \rightarrow perbandingan = n - 3$

$i = k \rightarrow perbandingan = n - k$

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.

c. Kompleksitas waktu

Worst case dan Best case nya sama karena do-while terus dilakukan tanpa adanya break.

$$(n-1)+(n-2)+\ldots+1 = \sum_{i=1}^{n-1} i$$

$$\sum_{i=1}^{n-1} i = \frac{(n-1)+1}{2}(n-1) = \frac{n(n-1)}{2} = O(n^2)$$