**TUGAS PRAKTIKUM ANALISIS ALGORITMA**



Disusun Oleh :

Haris Putratama 140810170059

**PROGRAM STUDI TEKNIK INFORMATIKA**

**FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM**

**UNIVERSITAS PADJADJARAN**

**2019**

# 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

{i > n}

**Kompleksitas Waktu**

Kompleksitas waktu CariMaks :  T(n) = n – 1

# Studi Kasus 2: *Sequential Search*

Diberikan larik bilangan bulan 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 : integer, y : integer, output idx : integer)

{ Mencari di dalam elemen . Lokasi (indeks elemen) tempat ditemukan diisi ke dalam idx. Jika tidak ditemukan, makai idx diisi dengan 0.

Input:

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

**Kompleksitas Algoritma**

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

*T*min(*n*) = 1

1. *Kasus terburuk*: bila *an* = *x* atau   *x* tidak ditemukan.

*T*max(*n*)  =*n*

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

*T*avg(*n*) = (1+2+3+..+n)/n = (1/2n(1+n))/n = (n+1)/2

# Studi Kasus 3: *Binary Search*

Diberikan larik bilangan bulan 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 : integer, x : integer, output : idx : integer)

{ Mencari y di dalam elemen . Lokasi (indeks elemen) tempat y ditemukan diisi ke dalam idx. Jika y tidak ditemukan makai dx diisi dengan 0.

**Input:**

**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

**Kompleksitas Algoritma**

1. *Kasus terbaik :  T*min(*n*) = 1
2. *Kasus terburuk*:*T*max (*n*) = 2log *n*
3. *Kasus rerata :* *Tmin(n)/Tmax(n)= 1/*2log *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 : integer)

{ Mengurutkan elemen-elemen dengan metode insertion sort.

Input:

OutputL (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

**Kompleksitas Algoritma**

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).

# 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 : integer)

{ Mengurutkan elemen-elemen dengan metode selection sort.

Input:

OutputL (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

**Kompleksitas Algoritma**

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.