**PRAKTIKUM ANALISIS ALGORITMA**

**MODUL II (KOMPLEKSITAS WAKTU)**



**Disusun oleh :**

Muhammad Fahmi Alwan

140810170052

**PROGRAM STUDI S1 TEKNIK INFORMATIKA**

**FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM**

**UNIVERSITAS PADJADJARAN**

**2019**

Studi Kasus 1: Pencarian Nilai Maksimal

* **Algoritma**

procedure CariMaks(input x1, x2, …, xn: integer, output maks: integer){   
 Mencari elemen terbesar dari sekumpulan elemen larik integer x1, x2, …, xn.   
 Elemen terbesar akandisimpan 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

* **Program**

#include <iostream>

using namespace std;

main(){

int angka[5] = {6,2,3,4,5};

int max = angka[0];

int i=1;

while(i<5){

if (angka[i] > max){

max = angka[i];

}

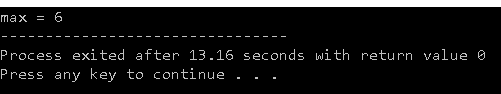
i++;

}

cout << "max = " << max;

}

Hasil :



* **Kompleksitas Algoritma**

1. Operasi pengisian nilai

Angka[5] 🡨 {6,2,3,4,5} 1 kali

Max 🡨 angka[0] 1 kali

I 🡨 1 1 kali

Max 🡨 angka[i] n-1 kali

I++ n-1 kali

Total = 1 + 1 + 1 + n – 1 + n – 1 = 1 + 2n

2. Operasi penjumlahan

I++ n-1 kali

Total = 1 + 2n + n – 1 = 3n

3. Operasi keluaran

Cout 🡨 max 1 kali

Total = 3n + 1 = 3n + 1

Studi Kasus 2 : Sequential Search

* **Algoritma**

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 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

* **Program**

#include<iostream>

#include <chrono>

using namespace std;

typedef int larik [];

void linearSearch(larik a, int n, int kunci, int& found, int& lokasi){

found = lokasi = 0;

while (!found && lokasi < n) {

if (a[lokasi] == kunci){

found = 1;

}else {

lokasi=lokasi+1;

}

}

}

main() {

larik x = {0,1,2,3,4,5,6,7,8,9};

int n,kunci,found,lokasi;

cout << "Kunci Pencarian data : " ;

cin >> kunci;

auto start = chrono::steady\_clock::now();

linearSearch(x, 10, kunci, found, lokasi);

if (found)

cout << "Ditemukan di posisi : " << lokasi+1 <<endl ;

else

cout << "Tidak ditemukan";

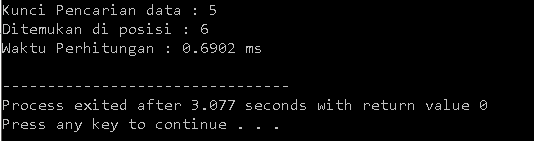
auto end = chrono::steady\_clock::now();

auto diff = end - start;

cout << "Waktu Perhitungan : " <<chrono::duration <double, milli> (diff).count() << " ms" << endl;

}

Hasil :



* **Analisis**

1. *Kasus terbaik* : kasus ini terjadi ketika *y1 = x* maka,

*Tmin(n) = n*

1. *Kasus terburuk* : kasus ini terjadi ketika *yn = x* atau *x* tidak di temukan maka,

*Tmax(n) = n*

1. Kasus rata-rata : kasus ini terjadi ketika x ditemukan namun pada posisi ke-/, maka operasi perbandingan (yk = x) akan dieksekusi sebanyak ke-/ kali.

*Tavg(n)* =

Studi Kasus 3 : Binary Search

* Algoritma

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

* Program

#include <iostream>

#include <chrono>

using namespace std;

main () {

int n, i, search, first, last, middle;

int arr[] = {0,1,2,3,4,5,6,7,8,9};

cout<<endl<<"Masukkan angka yang akan dicari :";

cin>>search;

int posisi;

for (int i=0; i<10-1; i++) {

posisi=i;

for (int j=i+1;j<10;j++) {

if (arr[posisi]>arr[j]) {

posisi=j;

}

}

swap(arr[i], arr[posisi]);

}

auto start = chrono::steady\_clock::now();

first = 0;

last = 10-1;

middle = (first+last)/2;

while (first <= last)

{

if(arr[middle] < search)

{

first = middle + 1;

}

else if(arr[middle] == search)

{

cout<<search<<" ditemukan di indeks ke "<<middle+1<<endl;

break;

}

else

{

last = middle - 1;

}

middle = (first + last)/2;

}

if(first > last)

{

cout<<"Error! "<<search<<" tidak ditemukan dalam list";

}

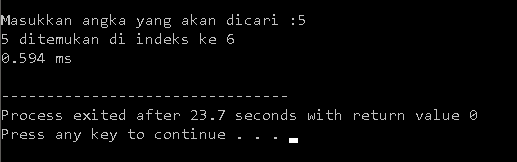
auto end = chrono::steady\_clock::now();

auto diff = end - start;

cout << chrono::duration <double, milli> (diff).count() << " ms" << endl;

}

Hasil :



* Analisis

1. *Kasus terbaik* : kasus ini terjadi ketika *y1 = x* maka,

*Tmin(n) = n*

1. *Kasus terburuk* : kasus ini terjadi ketika *yn = x* atau *x* tidak di temukan maka,

*Tmax(n) =* 2log *n*

Studi kasus 4 : Insertion Sort

* Algoritma

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

* Program

#include<iostream>

#include<conio.h>

using namespace std;

int main()

{

system("cls");

int size, arr[50], i, j, temp;

cout<<"Ukuran Array : ";

cin>>size;

cout<<"Element Array : ";

for(i=0; i<size; i++)

{

cin>>arr[i];

}

cout<<"Sorting array\n";

for(i=1; i<size; i++)

{

temp=arr[i];

j=i-1;

while((temp<arr[j]) && (j>=0))

{

arr[j+1]=arr[j];

j=j-1;

}

arr[j+1]=temp;

}

cout<<"Array setelah sorting : \n";

for(i=0; i<size; i++)

{

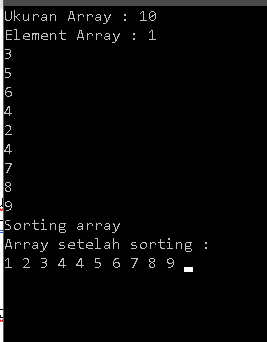
cout<<arr[i]<<" ";

}

getch();

}

Hasil :



* Analisis

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

* Algoritma

procedure SelectionSort(input/outputx1, 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

* Program

#include<iostream>

#include<conio.h>

using namespace std;

int main()

{

system("cls");

int size, arr[50], i, j, temp;

cout<<"Ukuran Array : ";

cin>>size;

cout<<"Element Array : ";

for(i=0; i<size; i++)

{

cin>>arr[i];

}

cout<<"Sorting array \n";

for(i=0; i<size; i++)

{

for(j=i+1; j<size; j++)

{

if(arr[i]>arr[j])

{

temp=arr[i];

arr[i]=arr[j];

arr[j]=temp;

}

}

}

cout<<"Array setelah sorting :\n";

for(i=0; i<size; i++)

{

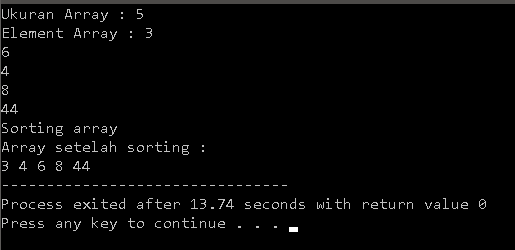
cout<<arr[i]<<" ";

}

getch();

}

Hasil :



* Analisis

1. Jumlah operasi perbandingan elemen

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.

1. 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.