**Tugas Praktikum Analisis Algoritma**

**Modul 2 (Kompleksitas Waktu Algoritma)**



Disusun oleh :

Baby Cattleya Gustina Permatagama

(140810160048)

S-1 Teknik Informatika   
Fakultas Matematika & Ilmu Pengetahuan Alam   
Universitas Padjadjaran  
Jalan Raya Bandung - Sumedang Km. 21 Jatinangor 45363

**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 :**

/\*

Nama Program : Mencari nilai terbesar dalam array

Oleh : Baby Cattleya G.P.

Tanggal Buat : 11 Maret 2019

\*/

#include<iostream>

usingnamespacestd;

intmain()

{

int i, n;

int x[100];

cout <<"Masukan jumlah element array (1-100): ";

cin >> n;

cout << endl;

// Input number

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

{

cout <<"Masukan angka ke- "<< i + 1<<" : ";

cin >> x[i];

}

int maks = x[0];

// Loop untuk mencari angka terbesar ke arr[0]

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

{

if(x[i] > x[0])

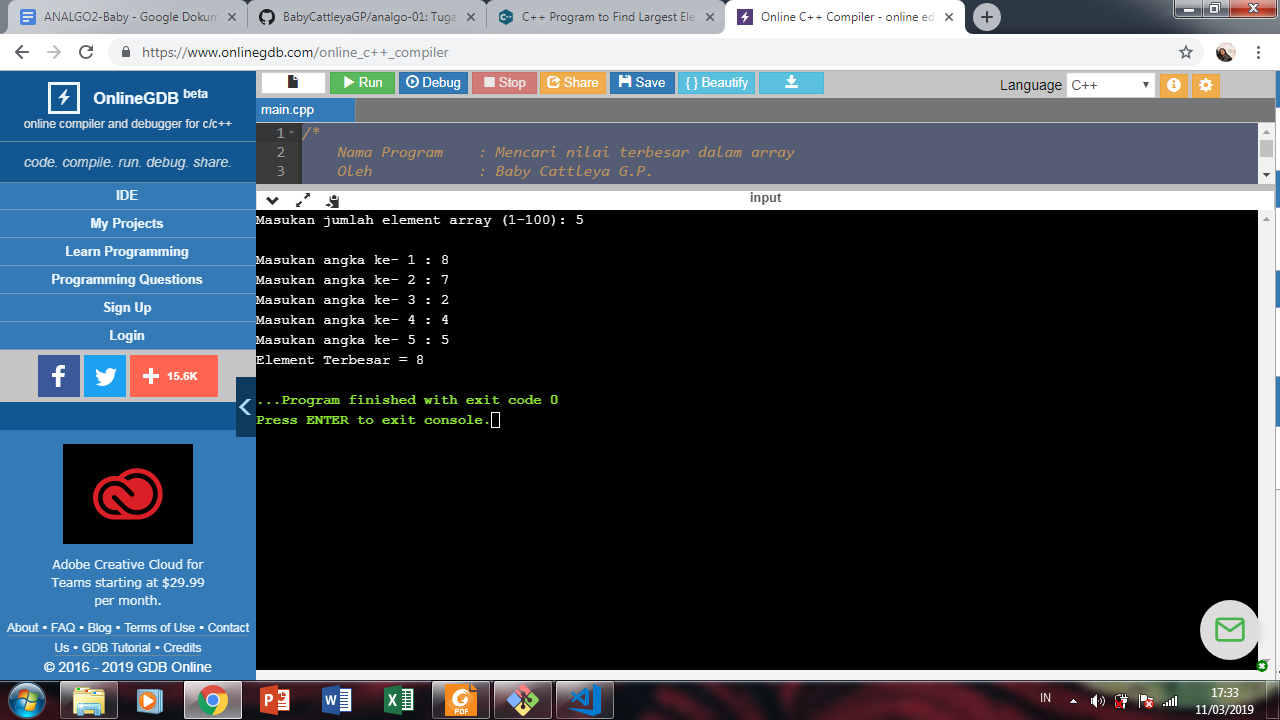
maks = x[i];

}

cout <<"Element Terbesar = "<< maks;

return 0;

}



**Perhitungan Kompleksitas :**

* Kompleksitas waktu algoritma dihitung berdasarkan jumlah operasi khasnya (perbandingan elemen larik) (A[i] > maks). Kompleksitas waktu CariElemenTerbesar : T(n) = n – 1.
* Kompleksitas waktu algoritma berdasarkan seluruh operasi :
  + Operasi Pengisian Nilai

maks <-- x1, 1 kali

i <-- 2, 1 kali

maks <-- x1, n kali

i <-- i+1, n kali

Komplekstas : T(n) = 2+2n

**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 :**

/\*

Nama Program : Sequential Search

Oleh : Baby Cattleya G.P.

Tanggal Buat : 11 Maret 2019

\*/

#include<iostream>

#include<chrono>

Using namespace std;

typedefint 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

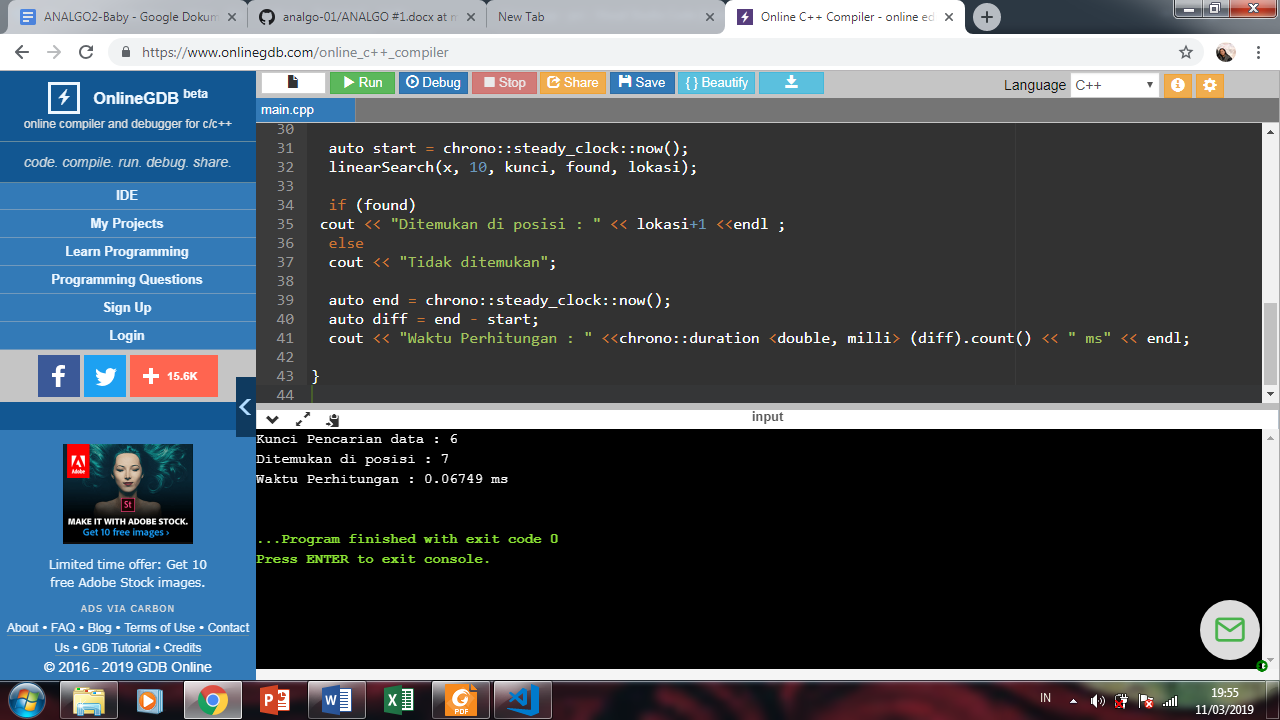
cout <<"Tidak ditemukan";

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

auto diff = end - start;

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

}

****

**Perhitungan Kompleksitas :**

Perhitungan Kompleksitas berdasarkan jumlah operasi perbandingan elemen tabel:

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

*T*min(*n*) = 1

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

*T*max(*n*) = *n*

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

*T*avg(*n*) = 

Cara lain: asumsikan bahwa *P*(*aj* = *x*) = 1/*n*. Jika *aj* = *x* maka *Tj* yang dibutuhkan adalah *Tj = j.* Jumlah perbandingan elemen larik rata-rata:

*T*avg(*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 :**

/\*

Nama Program : Binary Search

Oleh : Baby Cattleya G.P.

Tanggal Buat : 11 Maret 2019

\*/

#include<iostream>

#include<chrono>

using name space 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;

}

elseif(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";

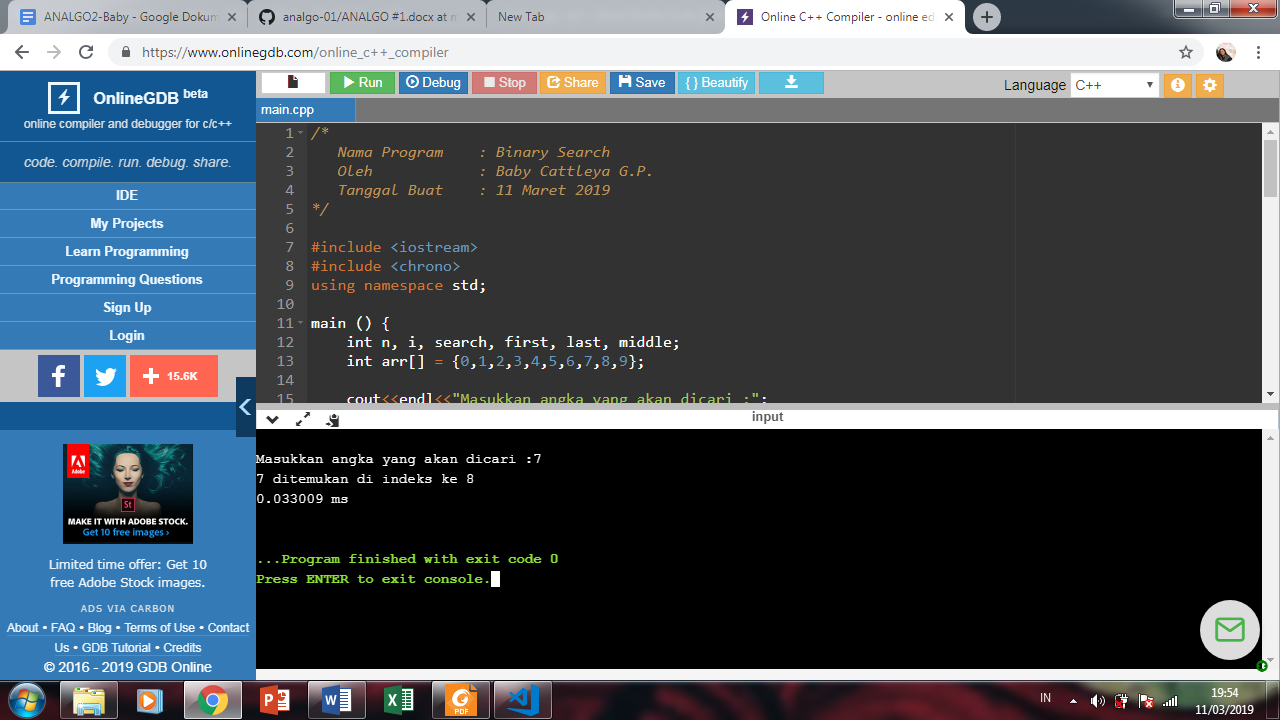
}

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

auto diff = end - start;

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

}



**Perhitungan Kompleksitas :**

1*. Kasus terbaik*

*T*min(*n*) = 1

2.*Kasus terburuk*

*T*max (*n*) = 2log *n*

Contoh ketika jumlah elemen dalam array 8:

Ketika n=8, Binary Search dijalankan dengan mereduksi ukuran menjadi 4

Ketika n=4, Binary Search dijalankan dengan mereduksi ukuran menjadi 2

Ketika n=2, Binary Search dijalankan dengan mereduksi ukuran menjadi 1

Dapat dilihat bahwa binary search dipanggil sebanyak tiga kali (3 elemen dalam array yang dieksekusi) untuk n = 8.

Sehingga didapat 8 = 23 atau secara general kita katakan n = 2k . ketika kita mengeksekusi x pencarian, “while loop” juga dieksekusi sebanyak x kali dan n di reduksi ukurannya menjadi 1.

Pada contoh penerapan di atas, dapat disimpulkan jumlah maksimum total operasi yang dilakukan adalah sebanyak 3. Nilai dari k dapat dinotasikan menjadi 2k = n sehingga k = log2 n. Jumlah operasi yang dilakukan untuk mencari k adalah sebanyak = 3 log n. pada kasus terburuk, yaitu kasus dimana k tidak terdapat dalam array adalah T(n) = log2 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 :**

/\*

Nama Program    : Insertion Search

Oleh            : Baby Cattleya G.P.

TanggalBuat    : 11 Maret 2019

\*/

#include<iostream>

#include<conio.h>

usingnamespacestd;

intmain()

{

   clrscr();

   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++)

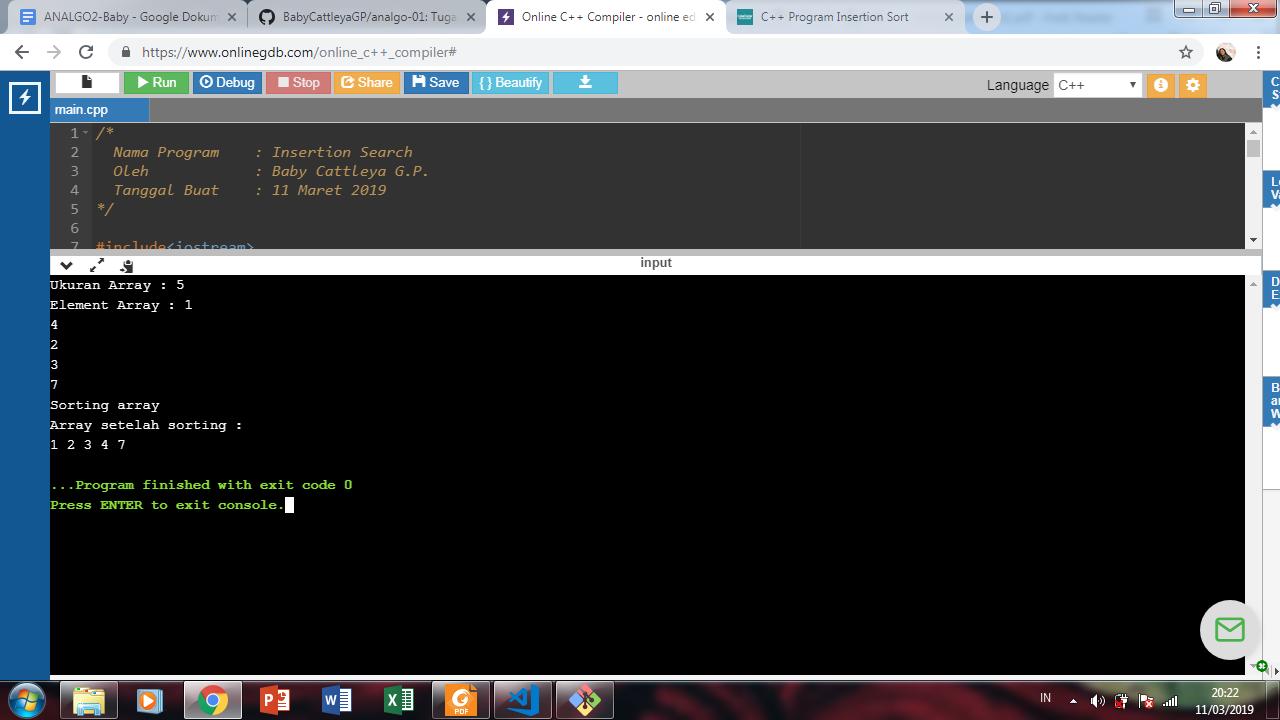
   {

       cout<<arr[i]<<" ";

   }

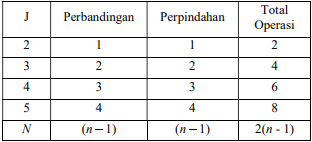
   getch();

}

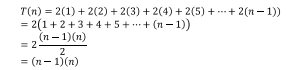
****

**Perhitungan Kompleksitas :**

**Hitunglah operasi perbandingan elemen larik dan operasi pertukaran pada algoritma.**

****

Sehingga total kompleksitas waktu T(n) untuk *worst case* yang dibutuhkan adalah:

****

* Best Case

Untuk kasus terbaik algoritma ini berjalan 1 kali, yaitu jika elemen dalam tabel telah terurut. Loop while tidak pernah dijalankan.

* Worst Case

Untuk kasus terburuk algoritma ini berjalan Nmax kali.

**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 :**

/\*

 Nama Program    : Selection Search

 Oleh            : Baby Cattleya G.P.

 TanggalBuat    : 11 Maret 2019

\*/

#include<iostream>

#include<conio.h>

using namespace std;

int main()

{

clrscr();

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++)

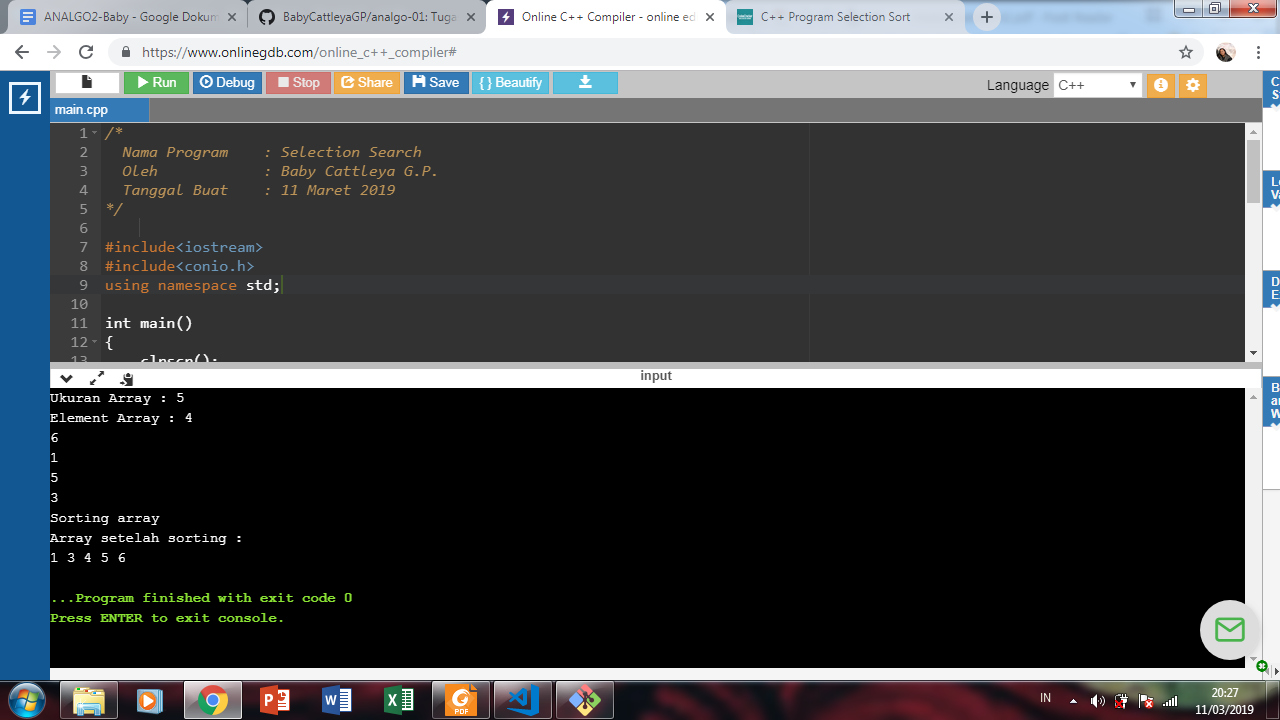
{

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

}

getch();

}

****

**Perhitungan Kompleksitas :**

(i) 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.

(ii) 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.