**B. M. S. College of Engineering**

**Bull Temple Road, Bangalore-560019**

(Affiliated to Visvesvaraya Technological

University, Belgaum)



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

**ANALYSIS AND DESIGN OF ALGORITHMS LABORATORY RECORD**

**NAME:** ANAGHA ACHARYA

**USN:** 1BM19CS224

**PROGRAM:** BACHELOR OF ENGINEERING

**SEMESTER:** IV

**SESSION:** APR-JUL 2021

**COURSE CODE:** 19CS4PCDBM

**COURSE TITLE:** ANALYSIS AND DESIGN OF ALGORITHMS

**CREDITS:** 4

PROGRAM 1: Write a recursive program to a. Solve Towers-of-Hanoi problem

b. To find GCD

1A.  Write a recursive program to solve Towers-of-Hanoi problem

#include<stdio.h>

#include<time.h>

void TowerOfHanoi(int n, char frompeg, char auxpeg, char topeg){

if(n==1){

printf("Move disc 1 from %c to %c", frompeg,topeg);

return;

}

TowerOfHanoi(n-1,frompeg,topeg,auxpeg);

printf("\nMove disc %d from %c to %c\n",n,frompeg,topeg);

TowerOfHanoi(n-1,auxpeg,frompeg,topeg);

}

void main(){

int n;

clock\_t start,end;

double time;

printf("Enter number of discs\n");

scanf("%d",&n);

start=clock();

TowerOfHanoi(n,'A','B','C');

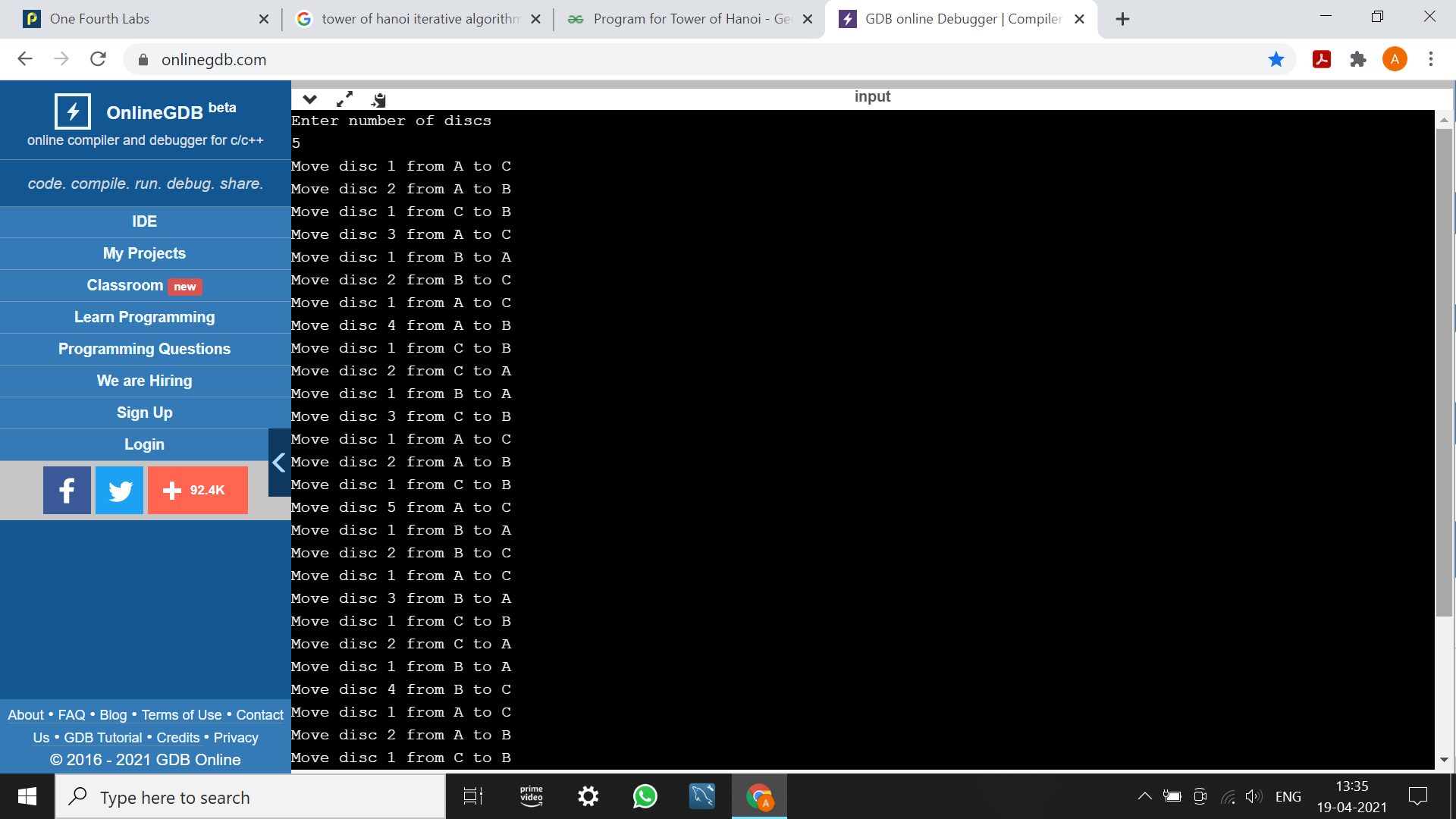
end=clock();

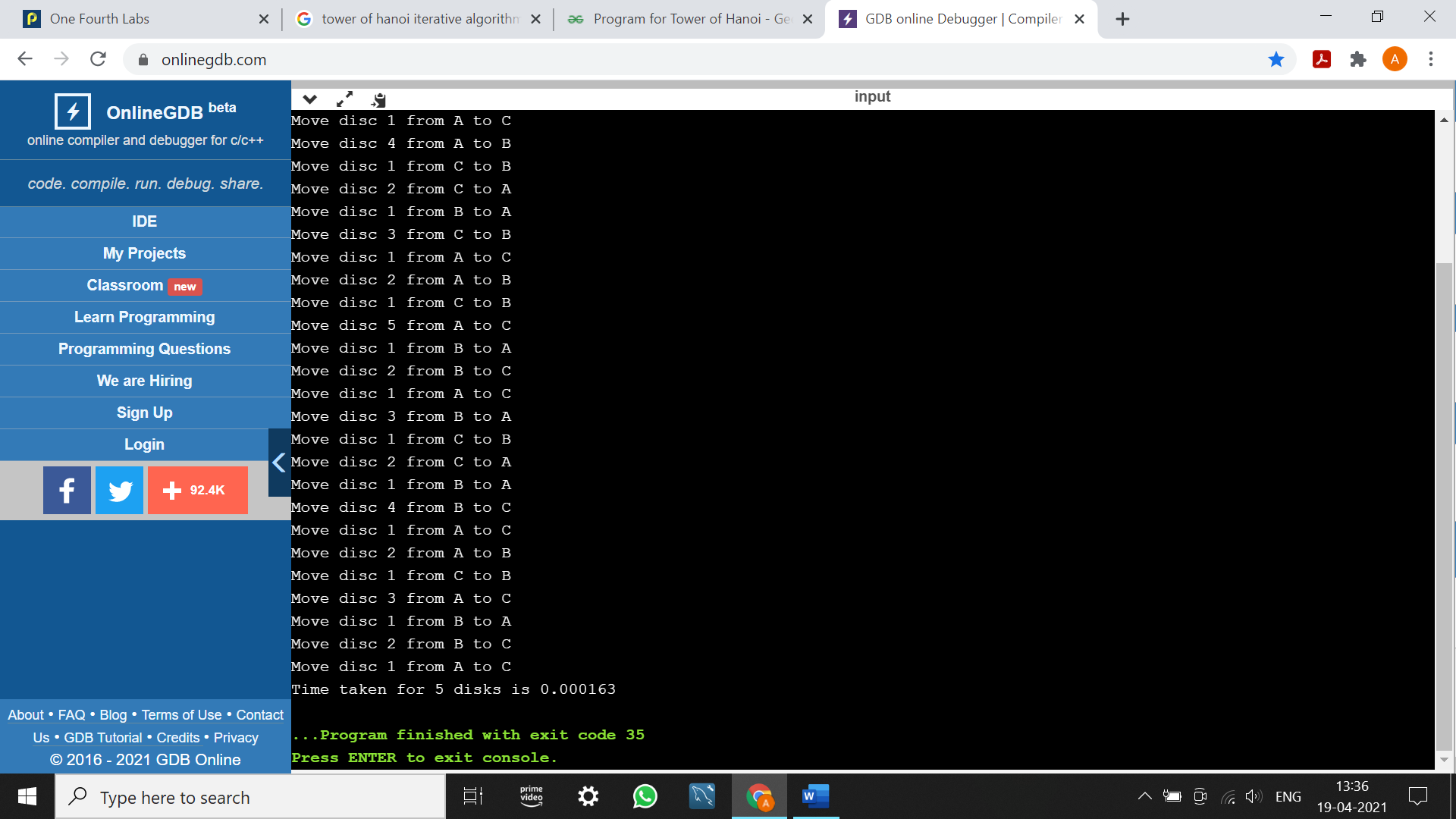
time=((float)(end-start))/CLOCKS\_PER\_SEC;

printf("\nTime taken for %d disks is %lf",n,time);

}

OUTPUT:





1B.  Write a recursive program to find GCD

#include<stdio.h>

#include<time.h>

int gcd(int m, int n){

if(n==0)

return m;

else

return gcd(n,m%n);

}

void main(){

int m,n,val;

clock\_t start,end;

double time;

printf("Enter the numbers\n");

scanf("%d %d",&m,&n);

start=clock();

val=gcd(m,n);

end=clock();

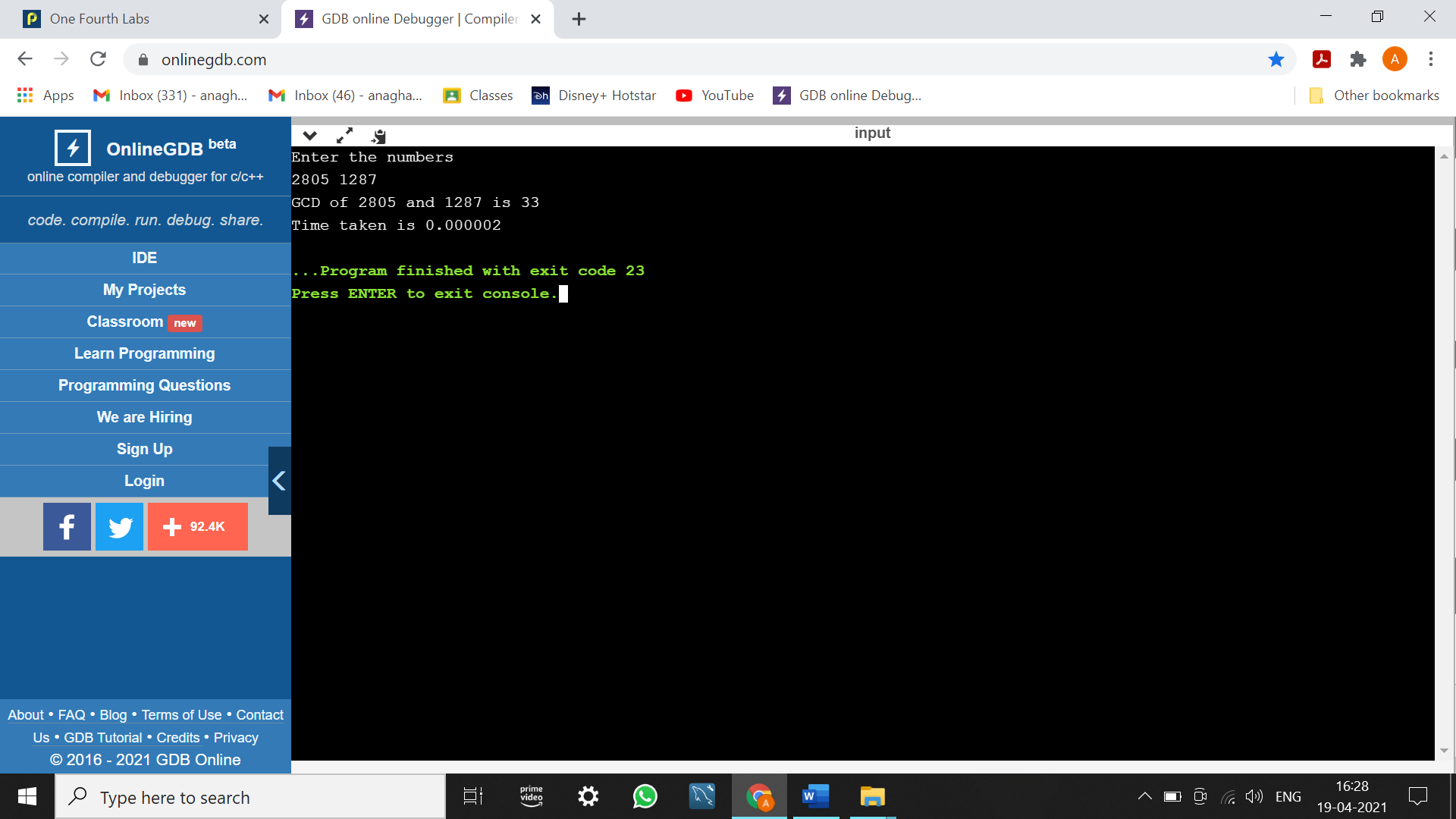
printf("GCD of %d and %d is %d", m,n,val);

time=((float)(end-start))/CLOCKS\_PER\_SEC;

printf("\nTime taken is %lf",time);

}

OUTPUT:



PROGRAM 2: Implement Recursive Binary search and Linear search and determine the time required to search an element. Repeat the experiment for different values of N and plot a graph of the time taken versus N.

#include<stdio.h>

#include<conio.h>

#include<stdlib.h>

#include<time.h>

int linear(int ele, int num, int a[], int index){

if(index>=num)

return 0;

if(ele==a[index])

return (index+1);

else

return linear(ele,num,a,index+1);

}

void sort(int num, int a[]){ // descending order //

int j,k,temp;

for(j=0;j<num-1;j++)

for(k=j+1;k<num;k++){

if(a[k]>a[j]){

temp=a[j];

a[j]=a[k];

a[k]=temp;

}

}

}

int binary(int ele, int a[],int low, int high){

int mid;

mid=(low+high)/2;

if (low>high)

return 0;

if (ele==a[mid])

return (mid+1);

else if(a[mid]>ele)

return binary(ele,a,mid+1,high);

else

return binary(ele,a,low,mid-1);

}

void main(){

int a[10000], num, pos, i, ch, ele;

clock\_t start,end;

double time;

printf("Enter number of elements: ");

scanf("%d", &num);

printf("Elements: ");

for(i=0;i<num;i++){

a[i]=(int)rand()%10000;

printf("%d ",a[i]);

}

for(;;){

printf("\nEnter choice\n1.Linear search\n2.Binary search\n3.Exit\n");

scanf("%d", &ch);

switch(ch){

case 1: printf("Enter element to be searched: ");

scanf("%d",&ele);

start=clock();

pos=linear(ele,num,a,0);

end=clock();

time=((double)(end-start))/CLOCKS\_PER\_SEC;

printf("Time taken=%1f\n",time);

break;

case 2: sort(num,a);

printf("sorted list:\n");

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

printf("%d ",a[i]);

printf("\nEnter element to be searched: ");

scanf("%d",&ele);

start=clock();

pos=binary(ele,a,0,num-1);

end=clock();

time=((double)(end-start))/CLOCKS\_PER\_SEC;

printf("Time taken=%1f\n",time);

break;

case 3: exit(1);

default: printf("Wrong choice!\n");

}

if(pos==0)

printf("Element not found\n");

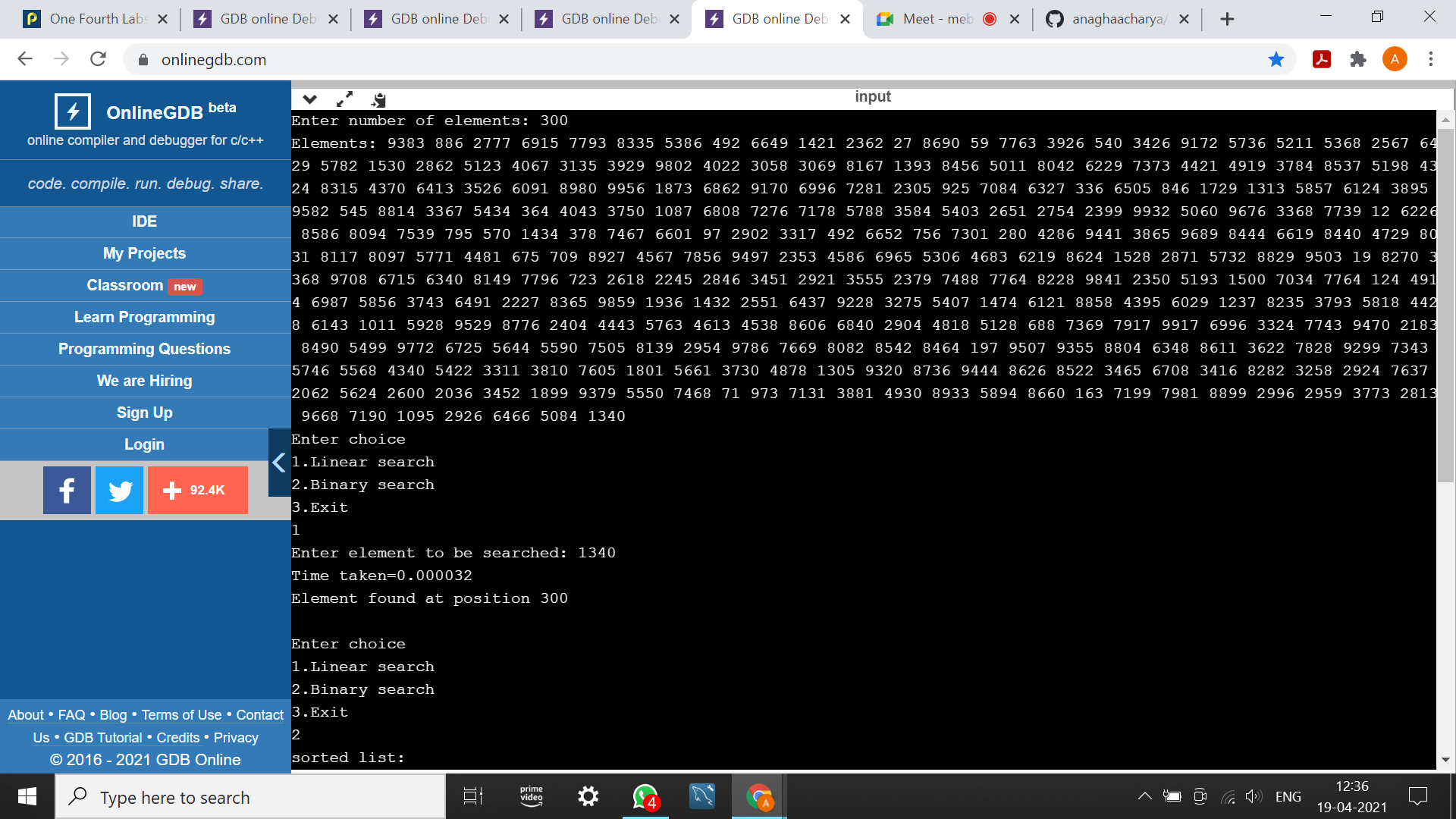
else

printf("Element found at position %d\n", pos);

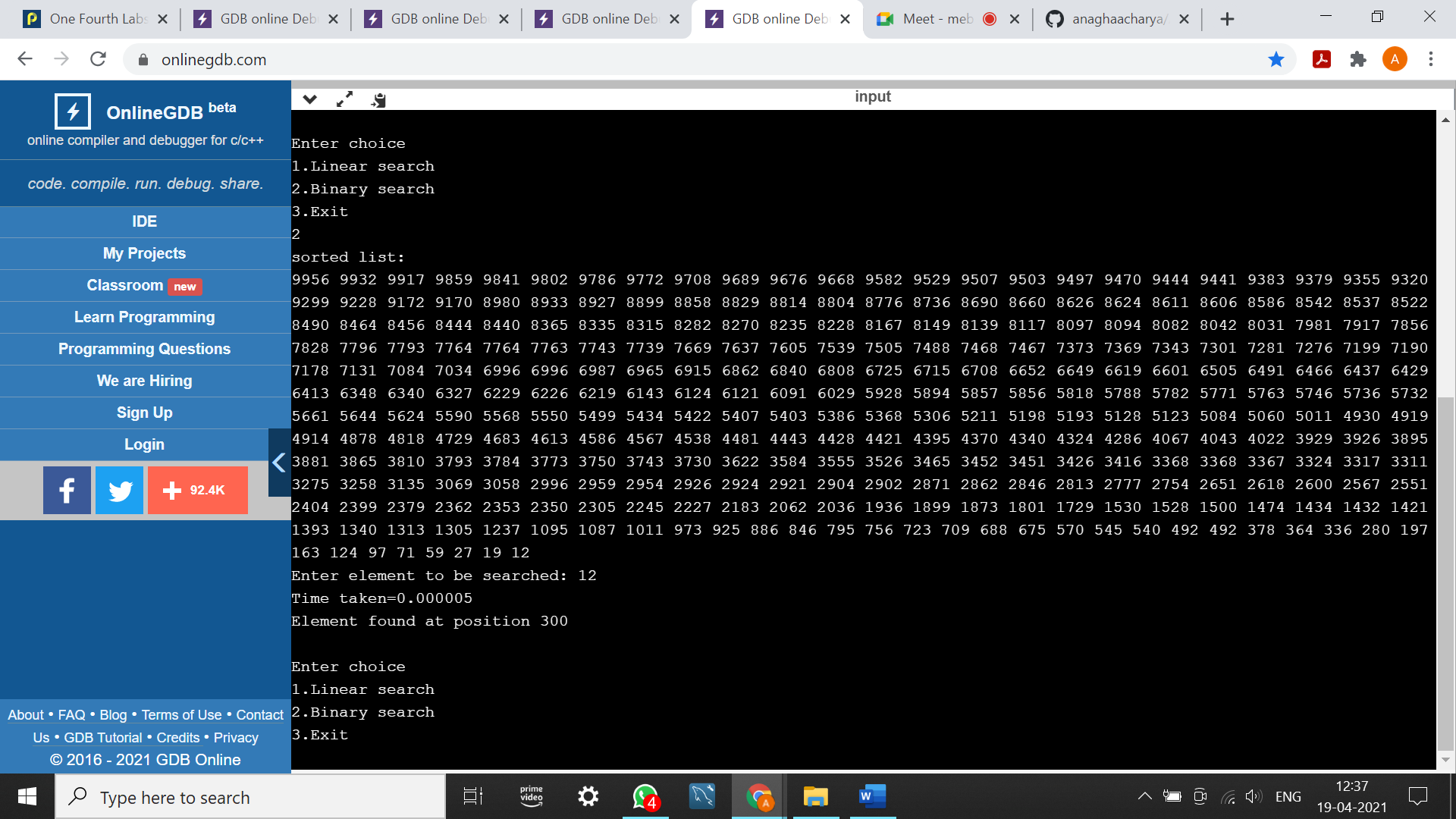
}

}

OUTPUT:



|  |  |
| --- | --- |
| N | Time Taken |
| 50 | 0.000003 |
| 100 | 0.000009 |
| 200 | 0.000019 |
| 500 | 0.00003 |
| 750 | 0.000043 |
| 1000 | 0.000053 |
|  |  |
|  |  |



|  |  |
| --- | --- |
| N | Time Taken |
| 50 | 0.000001 |
| 100 | 0.000002 |
| 200 | 0.000002 |
| 500 | 0.000002 |
| 750 | 0.000002 |
| 1000 | 0.000002 |
|  |  |

PROGRAM 3: Sort the given set of N integer elements using Selection Sort technique and compute its time taken. Run the program for different values of N and record the time taken to sort and plot.

#include<stdio.h>

#include<stdlib.h>

#include<time.h>

void swap(int a[],int min,int i){

int temp=a[min];

a[min]=a[i];

a[i]=temp;

}

void selsort(int a[],int i, int n){

int j,min;

min=i;

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

if(a[min]>a[j])

min=j;

}

swap(a,min,i);

if(i+1<n)

return selsort(a,i+1,n);

}

void main(){

int a[1000],i,n;

clock\_t start,end;

double time;

printf("Enter the number of elements\n");

scanf("%d",&n);

printf("The elements are:\n");

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

a[i]=(int)rand()%10000;

printf("%d ",a[i]);

}

start=clock();

selsort(a,0,n);

end=clock();

time=((double)(end-start))/CLOCKS\_PER\_SEC;

printf("\nSorted elements:\n");

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

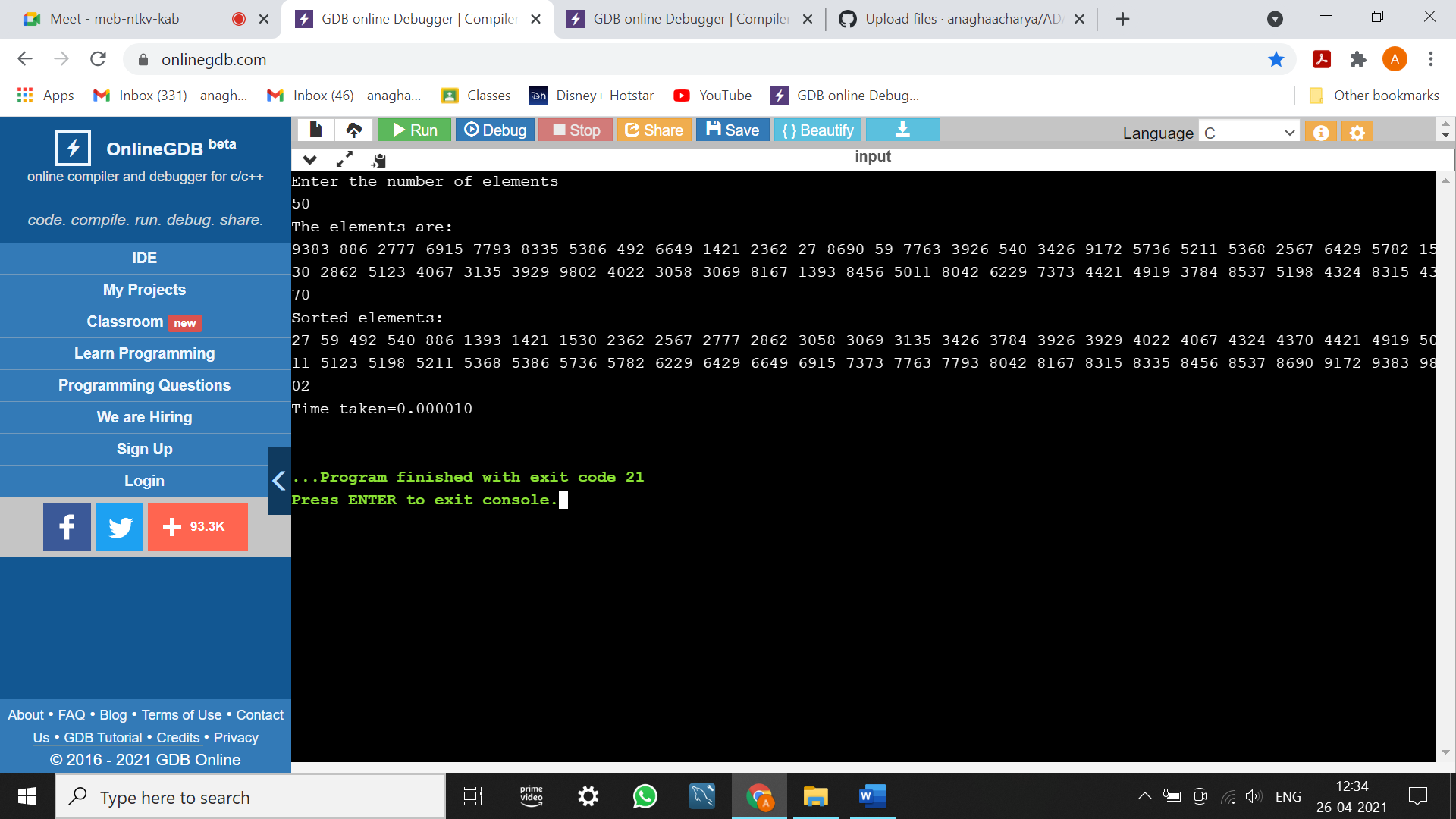
printf("%d ",a[i]);

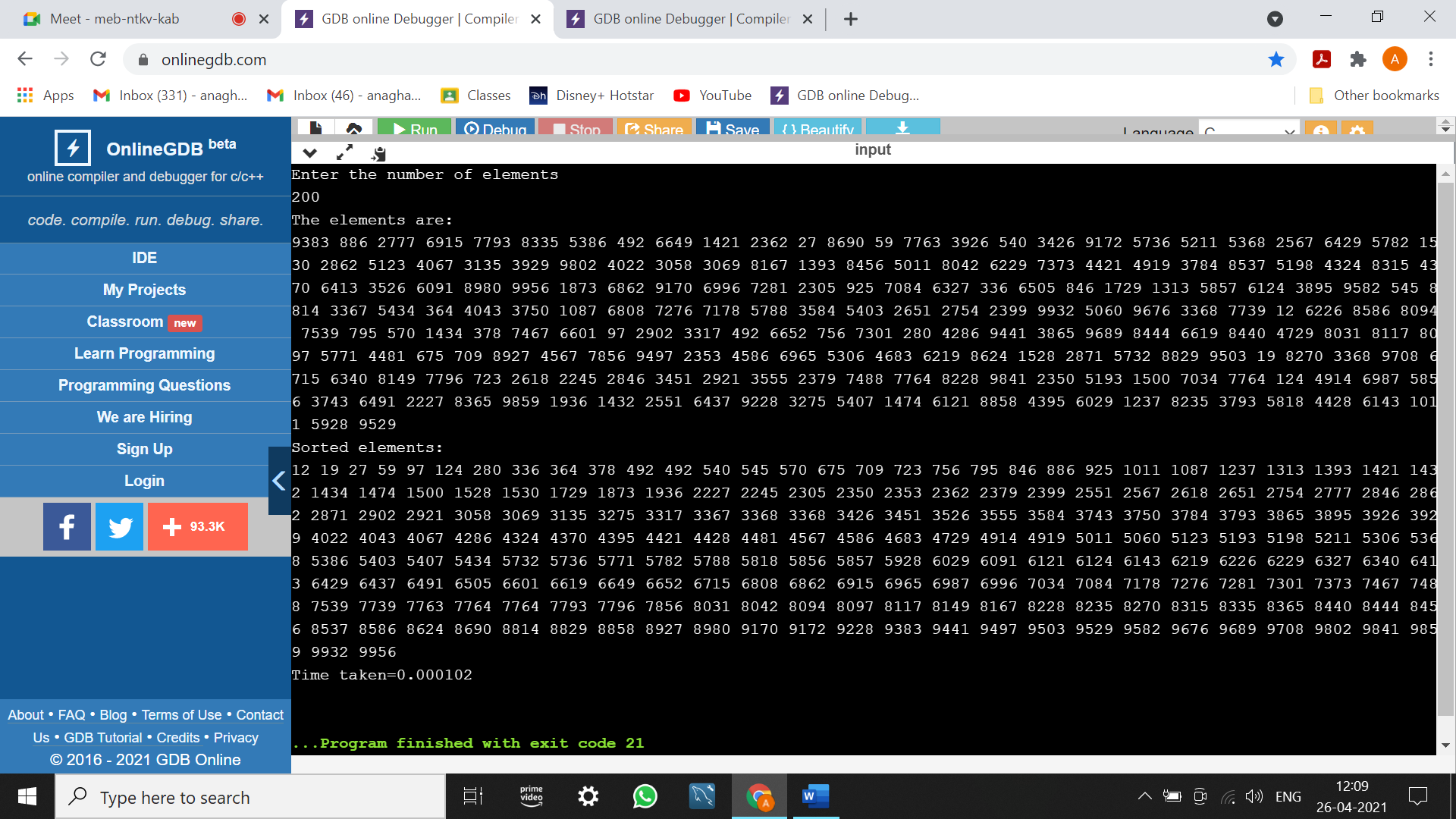
}

printf("\nTime taken=%1f\n",time);

}

OUTPUT:





|  |  |
| --- | --- |
| N | Time Taken |
| 50 | 0.00001 |
| 100 | 0.000031 |
| 200 | 0.000092 |
| 500 | 0.000475 |
| 750 | 0.000959 |
| 1000 | 0.002004 |

PROGRAM 4: Write program to do the following:

a. Print all the nodes reachable from a given starting node in a digraph using BFS method.  
b. Check whether a given graph is connected or not using DFS method.

a. #include<stdio.h>

#include<conio.h>

#include<time.h>

int a[20][20],q[20],visit[20],n,i,j,f=0,r=-1;

void bfs(int v)

{

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

if(a[v][i] && !visit[i])

q[++r]=i;

if(f<=r)

{

visit[q[f]]=1;

bfs(q[f++]);

}

}

void main()

{

int v;

clock\_t start,end;

double time;

printf("Enter the number of vertices:\n");

scanf("%d",&n);

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

{

q[i]=0;

visit[i]=0;

}

printf("Enter the graph data as matrix:\n");

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

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

scanf("%d",&a[i][j]);

printf("Enter the starting vertex:\n");

scanf("%d",&v);

start=clock();

bfs(v);

end=clock();

time=((double)(end-start))/CLOCKS\_PER\_SEC;

printf("The nodes reachable are:\n");

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

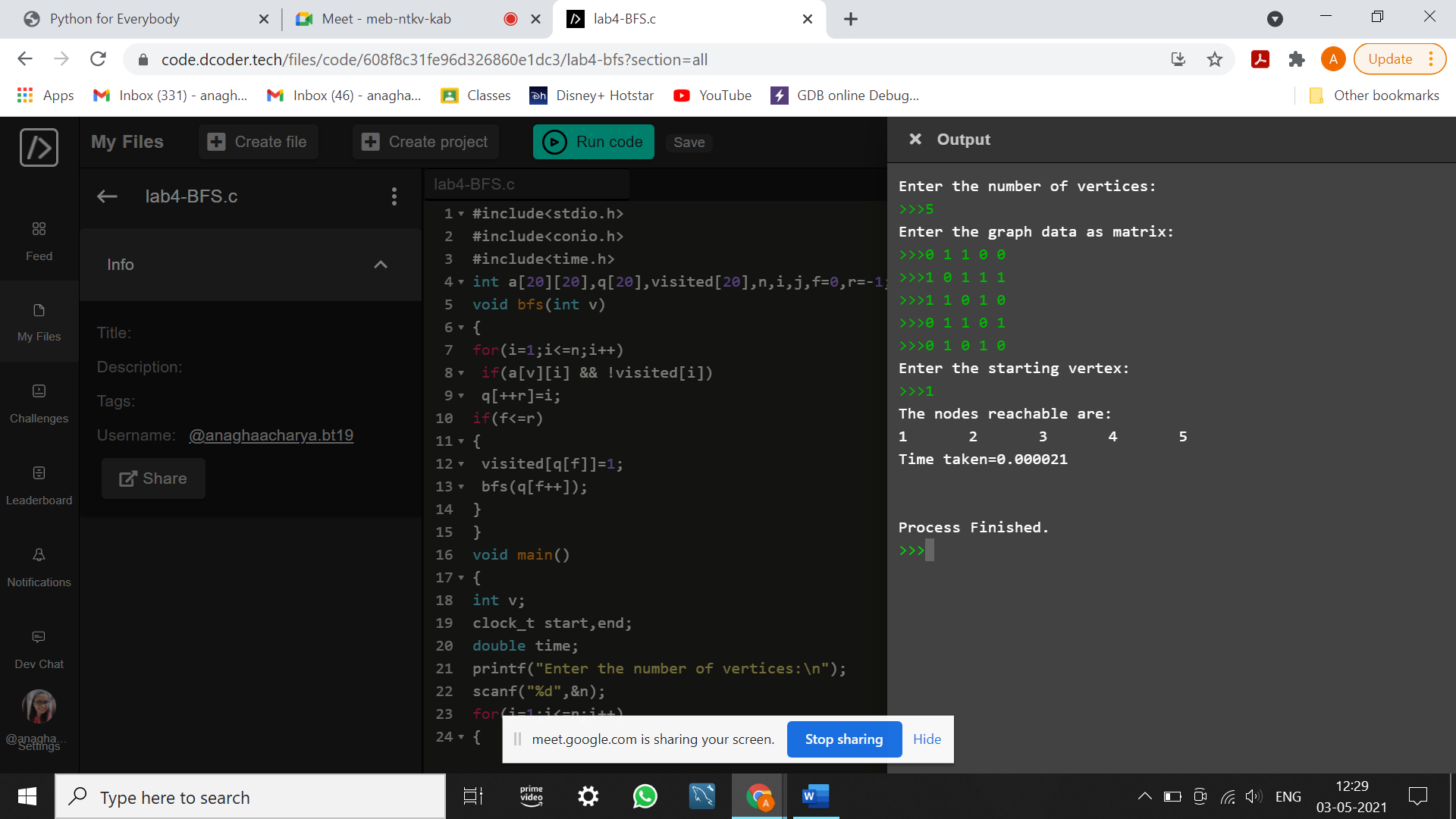
if(visit[i])

printf("%d\t",i);

printf("\nTime taken=%1f\n",time);

}

OUTPUT:



b. #include<stdio.h>

#include<conio.h>

#include<time.h>

int a[20][20],reach[20],n;

void dfs(int v){

int i;

reach[v]=1;

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

if(a[v][i] && !reach[i])

{

printf("%d->%d\n",v,i);

dfs(i);

}

}

void main()

{

int i,j,count=0;

clock\_t start,end;

float time;

printf("Enter number of vertices:\n");

scanf("%d",&n);

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

{

reach[i]=0;

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

a[i][j]=0;

}

printf("Enter the adjacency matrix:\n");

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

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

scanf("%d",&a[i][j]);

start=clock();

dfs(1);

end=clock();

time=((double)(end-start)/CLOCKS\_PER\_SEC);

printf("\n");

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

{

if(reach[i])

count++;

}

if(count==n)

printf("Graph is connected");

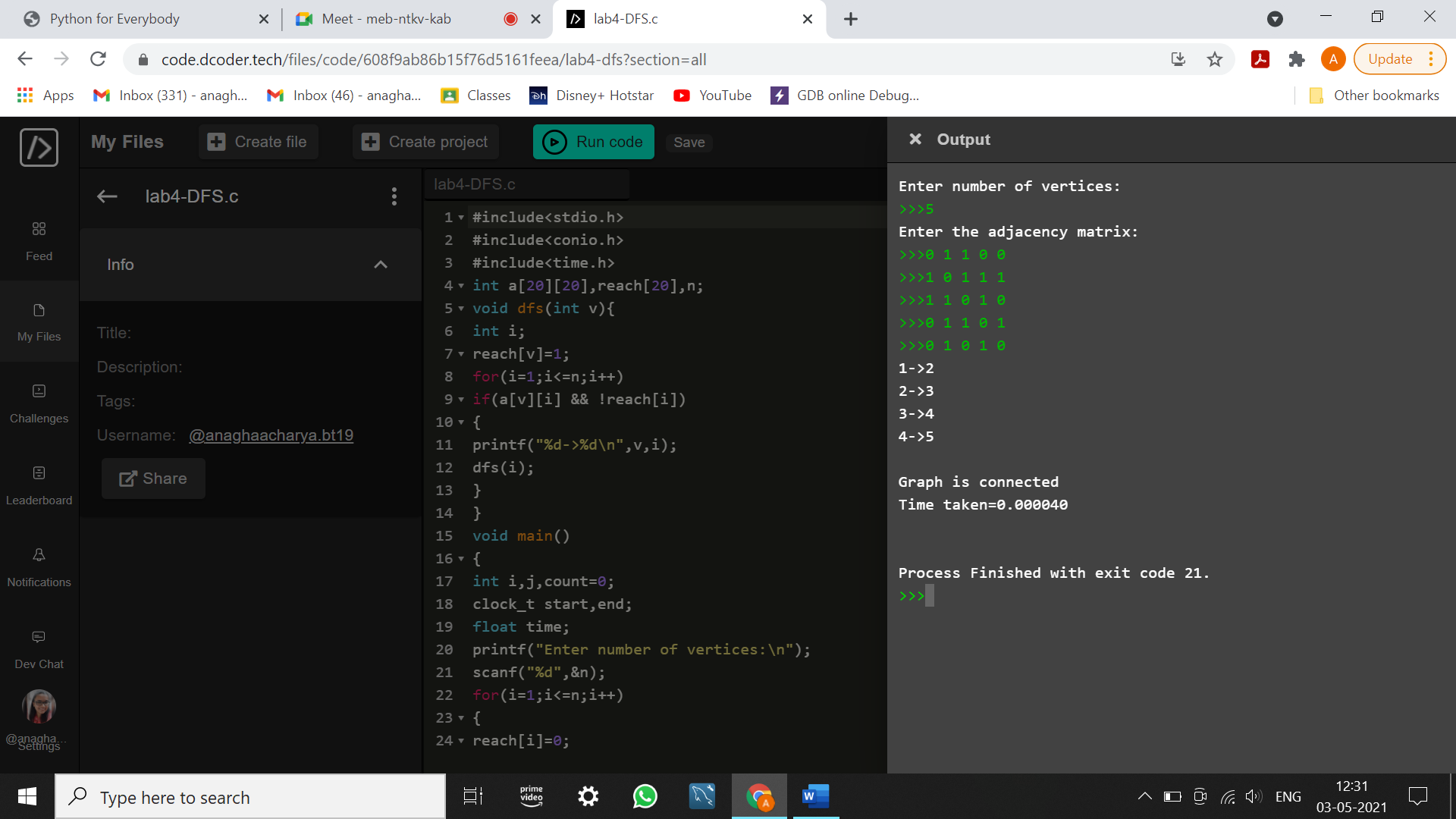
else

printf("Graph is not connected");

printf("\nTime taken=%1f\n",time);

}

OUTPUT:



PROGRAM 5: Sort a given set of N integer elements using Insertion Sort technique and compute its time taken.

#include<stdio.h>

#include<stdlib.h>

#include<time.h>

void insertionsort(int a[],int n)

{

int last,j;

if(n<=1)

return;

insertionsort(a,n-1);

last=a[n-1];

j=n-2;

while(j>=0 && a[j]>last){

a[j+1]=a[j];

j--;

}

a[j+1]=last;

}

void main(){

int a[1000],i,n;

clock\_t start,end;

double time;

printf("Enter the number of elements: ");

scanf("%d",&n);

printf("The numbers are:\n");

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

{

a[i]=(int)rand()%10000;

printf("%d ",a[i]);

}

start=clock();

insertionsort(a,n);

end=clock();

time=((double)(end-start))/CLOCKS\_PER\_SEC;

printf("\nSorted array:\n");

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

{

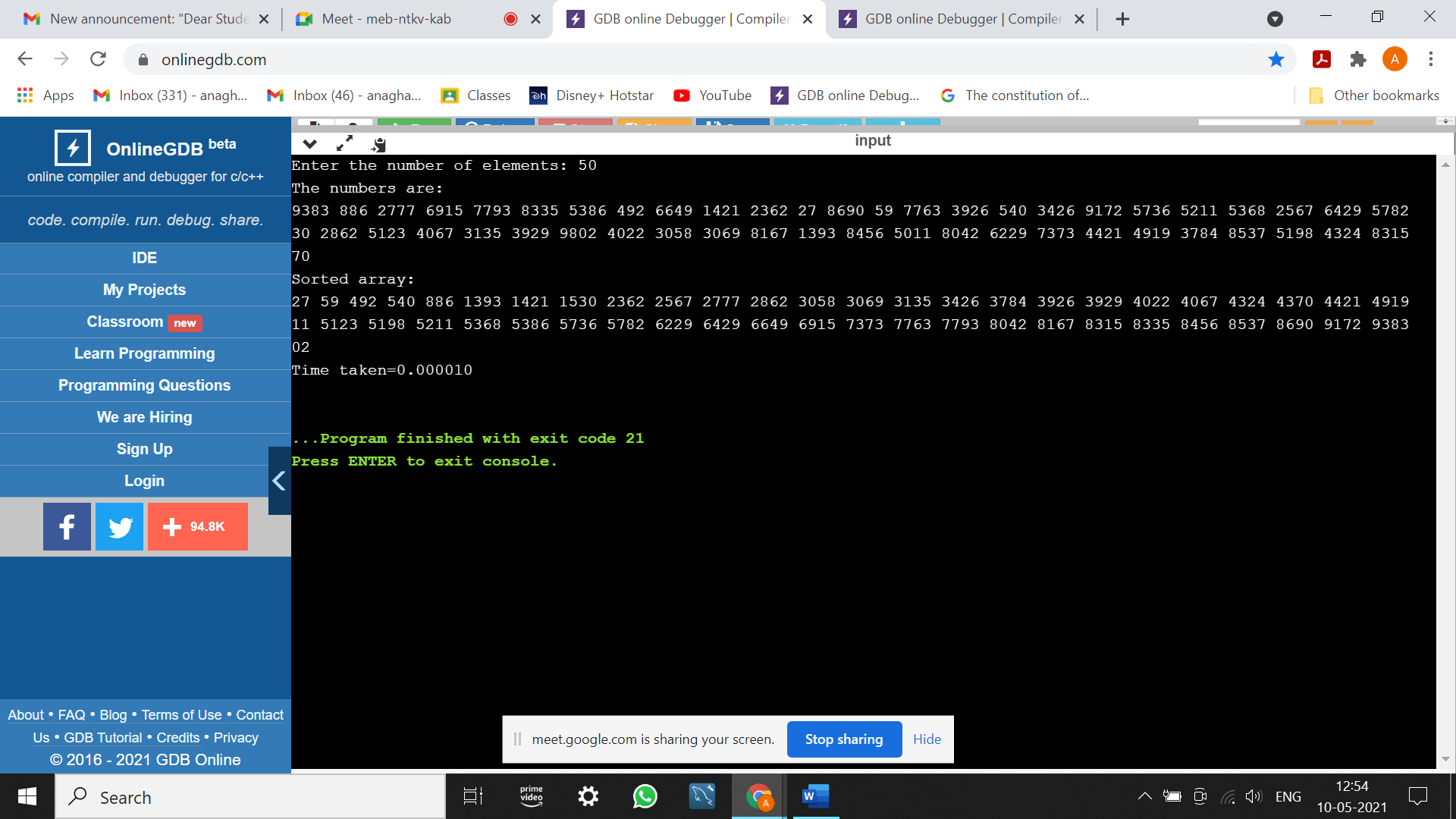
printf("%d ",a[i]);

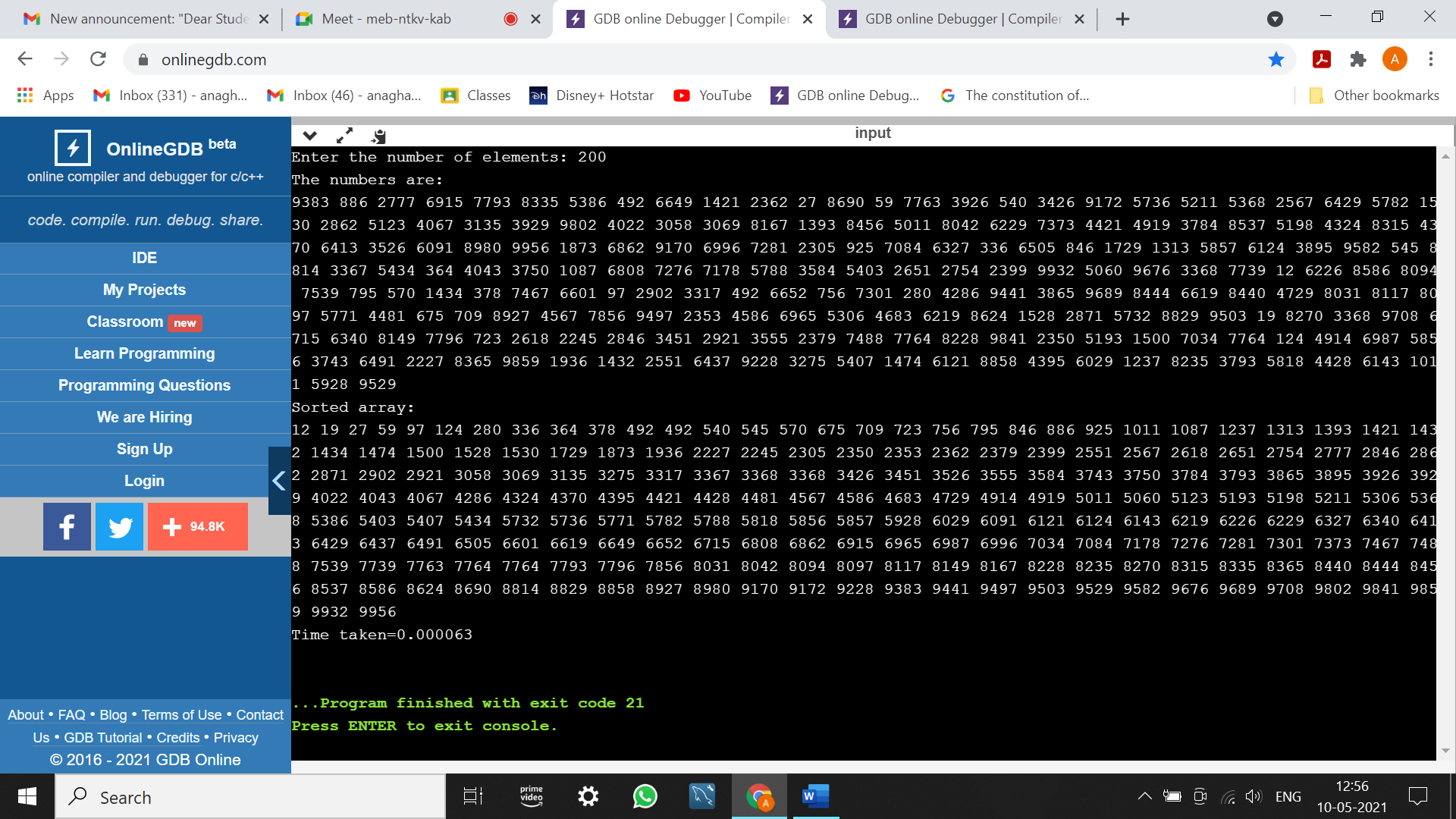
}

printf("\nTime taken=%1f\n",time);

}

OUTPUT:





PROGRAM 6: Write program to obtain the Topological ordering of vertices in a given digraph.

#include<stdio.h>

#include<time.h>

void findindegree(int a[10][10],int indegree[10],int n)

{

int i,j,sum;

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

{

sum=0;

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

sum=sum+a[i][j];

indegree[j]=sum;

}

}

void topological(int n,int a[10][10])

{

int i,k,u,v,top,t[100],stack[20],indegree[20];

k=1;

top=-1;

findindegree(a,indegree,n);

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

{

if(indegree[i]==0)

stack[++top]=i;

}

while(top!=-1){

u=stack[top--];

t[k++]=u;

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

{

if(a[u][v]==1)

{

indegree[v]--;

if(indegree[v]==0)

stack[++top]=v;

}

}

}

printf("Topological sequence is\n");

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

printf("%d ",t[i]);

}

void main()

{

int a[10][10],i,j,n;

clock\_t start,end;

double time;

printf("Enter the number of nodes:");

scanf("%d",&n);

printf("Enter the adjacency matrix:\n");

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

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

scanf("%d",&a[i][j]);

}

start=clock();

topological(n,a);

end=clock();

time=((double)(end-start))/CLOCKS\_PER\_SEC;

printf("\nTime taken=%1f",time);

}

|  |
| --- |
|  |
|  |
| OUTPUT: |
|  |
|  |
|  |
| |  | | --- | |  | |

PROGRAM 7: Implement Johnson Trotter algorithm to generate permutations.

#include <stdio.h>

#include<time.h>

int N, i;

int p[1000], q[1000];

int direct[1000];

void Move(int x, int d)

{

int z;

printf("\n");

z = p[q[x]+d];

p[q[x]] = z;

p[q[x]+d] = x;

q[z] = q[x];

q[x] = q[x]+d;

}

void Permutation(int n)

{

int i;

if (n > N){

int i;

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

printf("%d", p[i]);

}

else

{

Permutation( n+1 );

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

{

Move(n,direct[n]);

Permutation(n+1);

}

direct[n] = -direct[n];

}

}

void main ()

{

clock\_t start,end;

double time;

printf("Enter the value of N:");

scanf("%d", &N);

printf("\n");

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

{

direct[i] = -1; p[i] = i;

q[i] = i;

}

printf("The permutations generated are:\n");

start = clock();

Permutation(1);

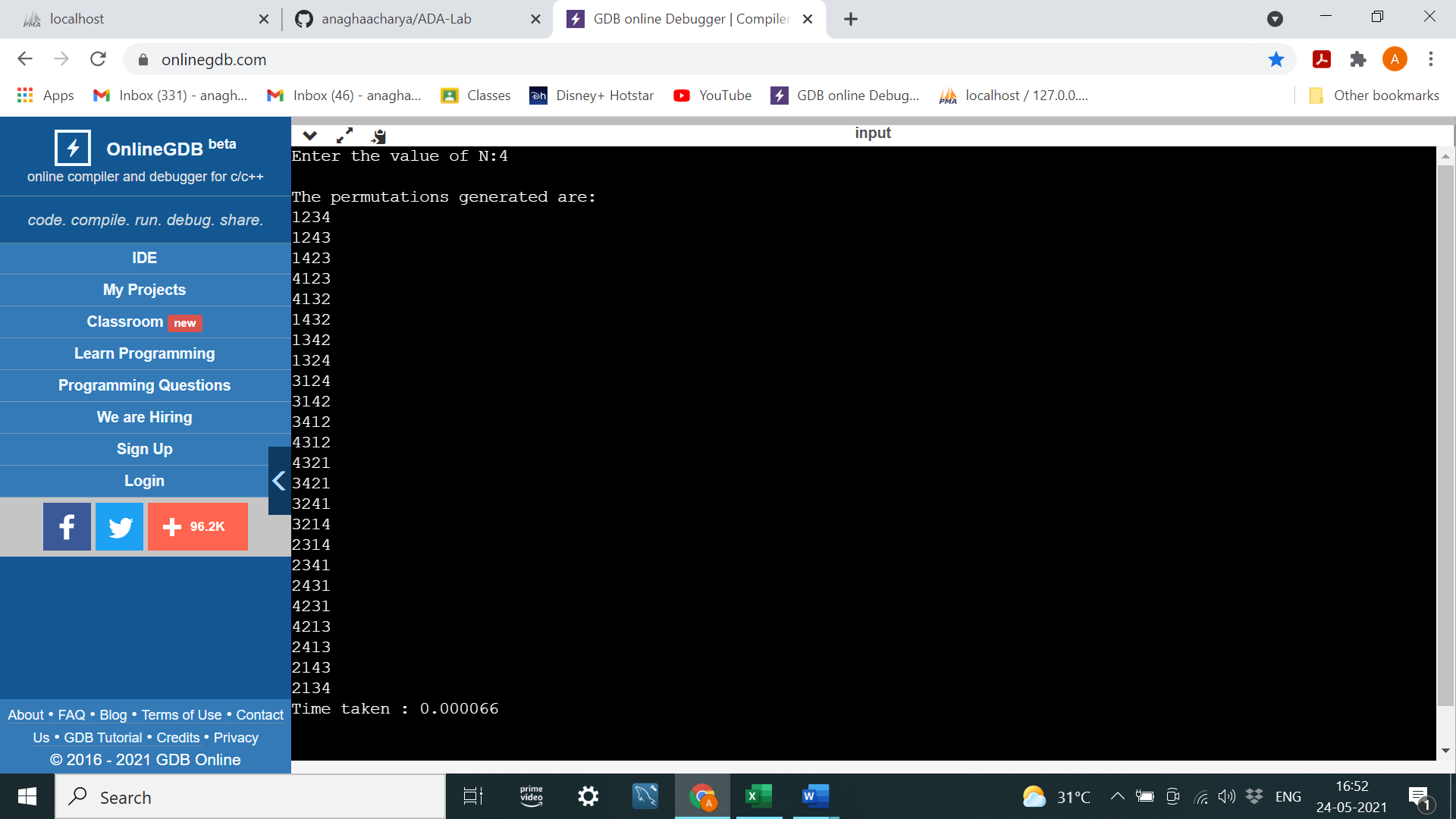
end = clock();

time = ((double)(end - start))/CLOCKS\_PER\_SEC;

printf("\nTime taken : %lf\n",time);

}

OUTPUT:



PROGRAM 8: Sort a given set of N integer elements using Merge Sort technique and compute its time taken. Run the program for different values of N and record the time taken to sort.

#include<stdio.h>

#include<stdlib.h>

#include<time.h>

void merge(int a[],int low,int mid,int high)

{

int i,k,l,m,temp[1000];

l=low;

i=low;

m=mid+1;

while((l<=mid)&&(m<=high)){

if(a[l]<=a[m]){

temp[i]=a[l];

l++;

}

else{

temp[i]=a[m];

m++;

}

i++;

}

if(l>mid)

{

for(k=m;k<=high;k++)

{

temp[i]=a[k];

i++;

}

}

else

{

for(k=l;k<=mid;k++)

{

temp[i]=a[k];

i++;

}

}

for(k=low;k<=high;k++)

{

a[k]=temp[k];

}

}

void mergesort(int a[],int low,int high)

{

int mid;

if(low<high){

mid=(low+high)/2;

mergesort(a,low,mid);

mergesort(a,mid+1,high);

merge(a,low,mid,high);

}

}

void main()

{

int a[1000],i,n;

clock\_t start,end;

double time;

printf("Enter the number of elements\n");

scanf("%d",&n);

printf("The elements are:\n");

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

a[i]=(int)rand()%10000;

printf("%d ",a[i]);

}

start=clock();

mergesort(a,0,n-1);

end=clock();

time=((double)(end-start))/CLOCKS\_PER\_SEC;

printf("\nSorted elements:\n");

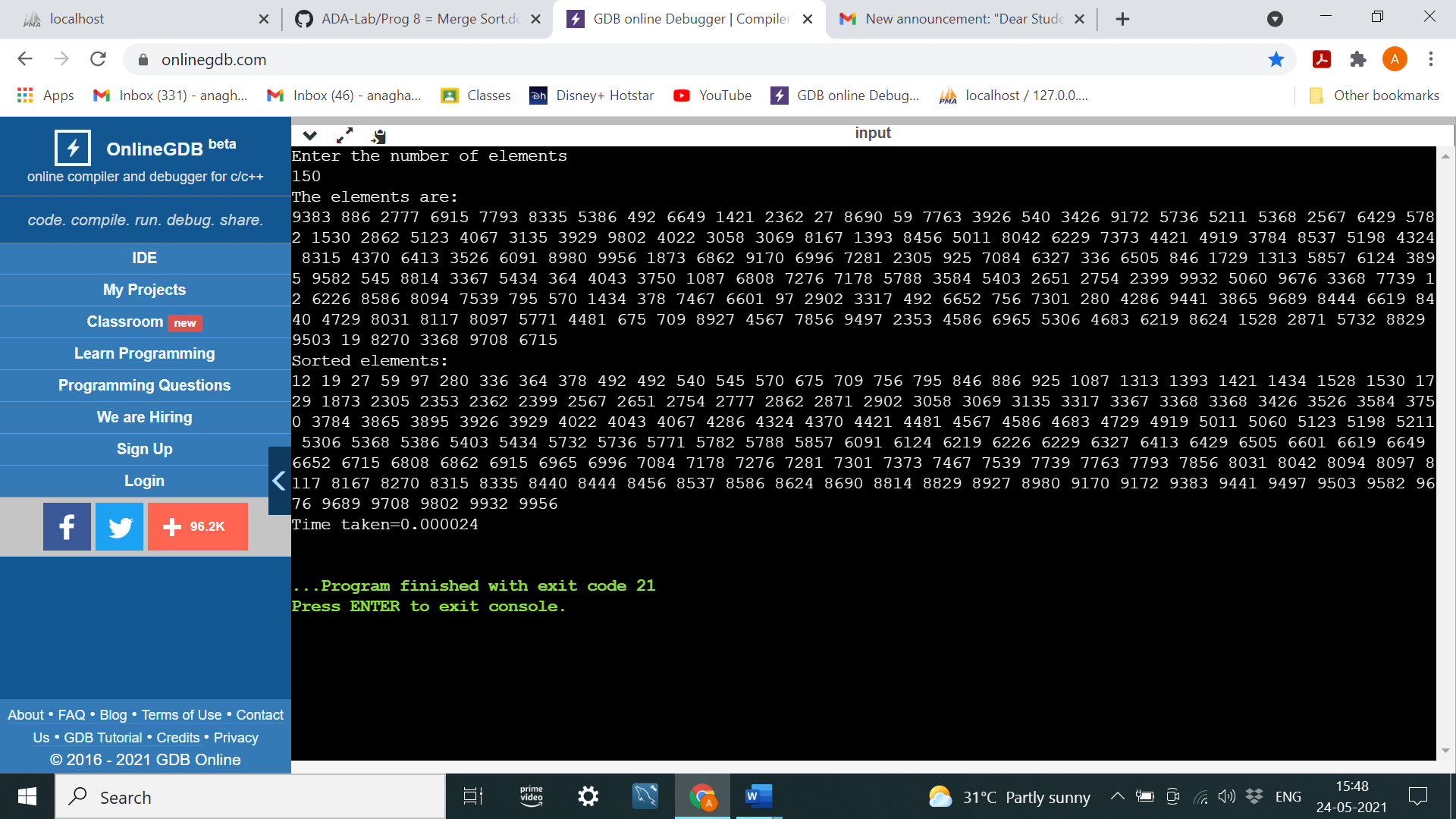
for(i=0;i<n;i++){

printf("%d ",a[i]);

}

printf("\nTime taken=%1f\n",time);

}



|  |  |
| --- | --- |
| N | Time Taken |
| 50 | 0.00001 |
| 100 | 0.000024 |
| 200 | 0.000041 |
| 500 | 0.000081 |
| 750 | 0.000118 |
| 1000 | 0.00017 |

PROGRAM 9: Sort a given set of N integer elements using Quick Sort technique and compute its time taken. Run the program for different values of N and record the time taken to sort.

#include<stdio.h>

#include<stdlib.h>

#include<time.h>

int partition(int a[],int low,int high){

int pivot,i,j,temp;

pivot=a[low];

i=low+1;

j=high;

while(1){

while(pivot>a[i]&&i<=high)

i++;

while(paivot<a[j])

j--;

if(i<j)

{

temp=a[i];

a[i]=a[j];

a[j]=temp;

}

else

{

temp=a[j];

a[j]=a[low];

a[low]=temp;

return j;

}

}

}

void quicksort(int a[],int low,int high){

int pivot\_pos;

if(low<high)

{

pivot\_pos=partition(a,low,high);

quicksort(a,low,pivot\_pos-1);

quicksort(a,pivot\_pos+1,high);

}

}

void main(){

int a[1000],i,n;

clock\_t start,end;

double time;

printf("Enter the number of elements\n");

scanf("%d",&n);

printf("The elements are:\n");

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

a[i]=(int)rand()%10000;

printf("%d ",a[i]);

}

start=clock();

quicksort(a,0,n-1);

end=clock();

time=((double)(end-start))/CLOCKS\_PER\_SEC;

printf("\nSorted elements:\n");

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

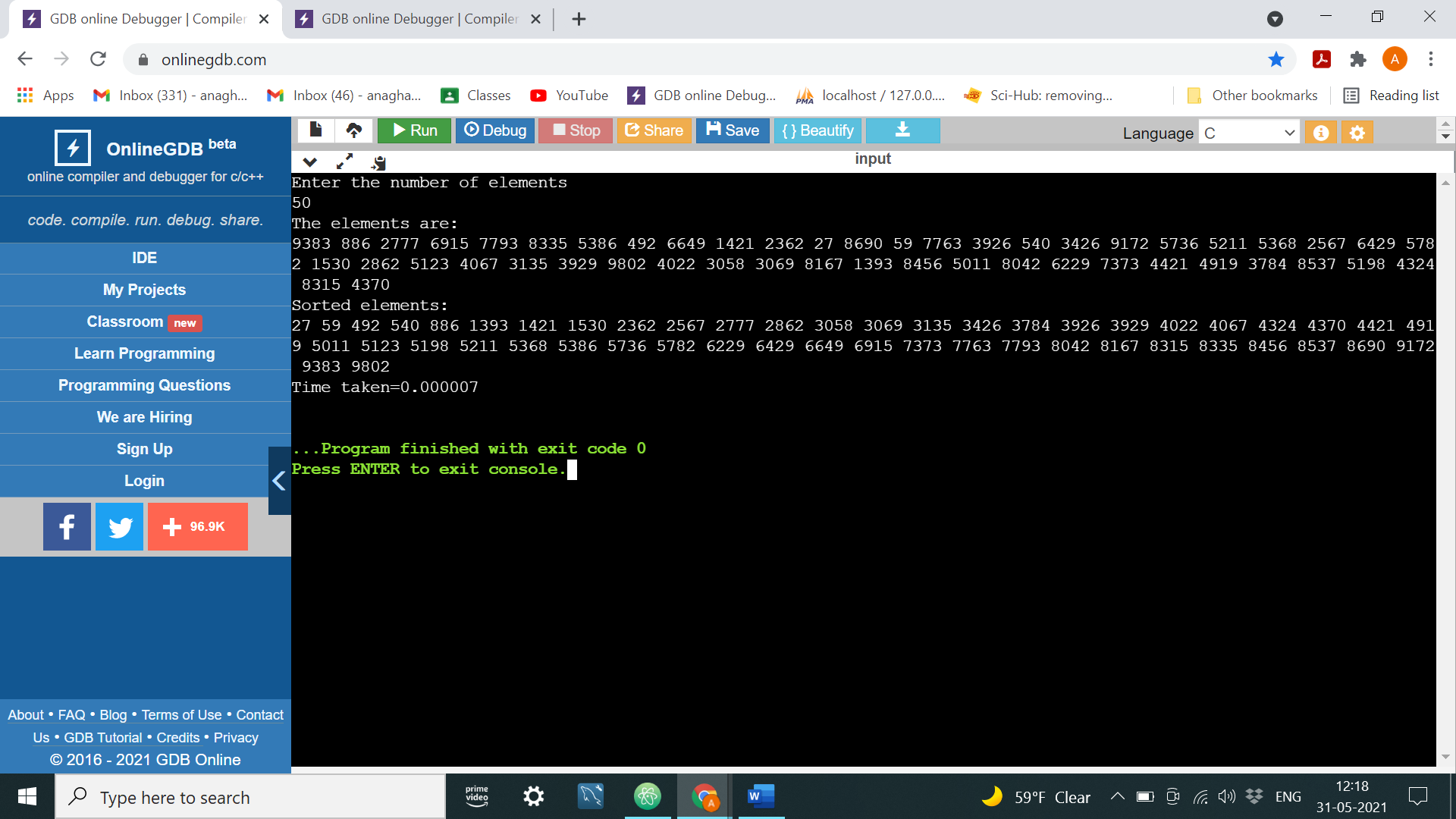
printf("%d ",a[i]);

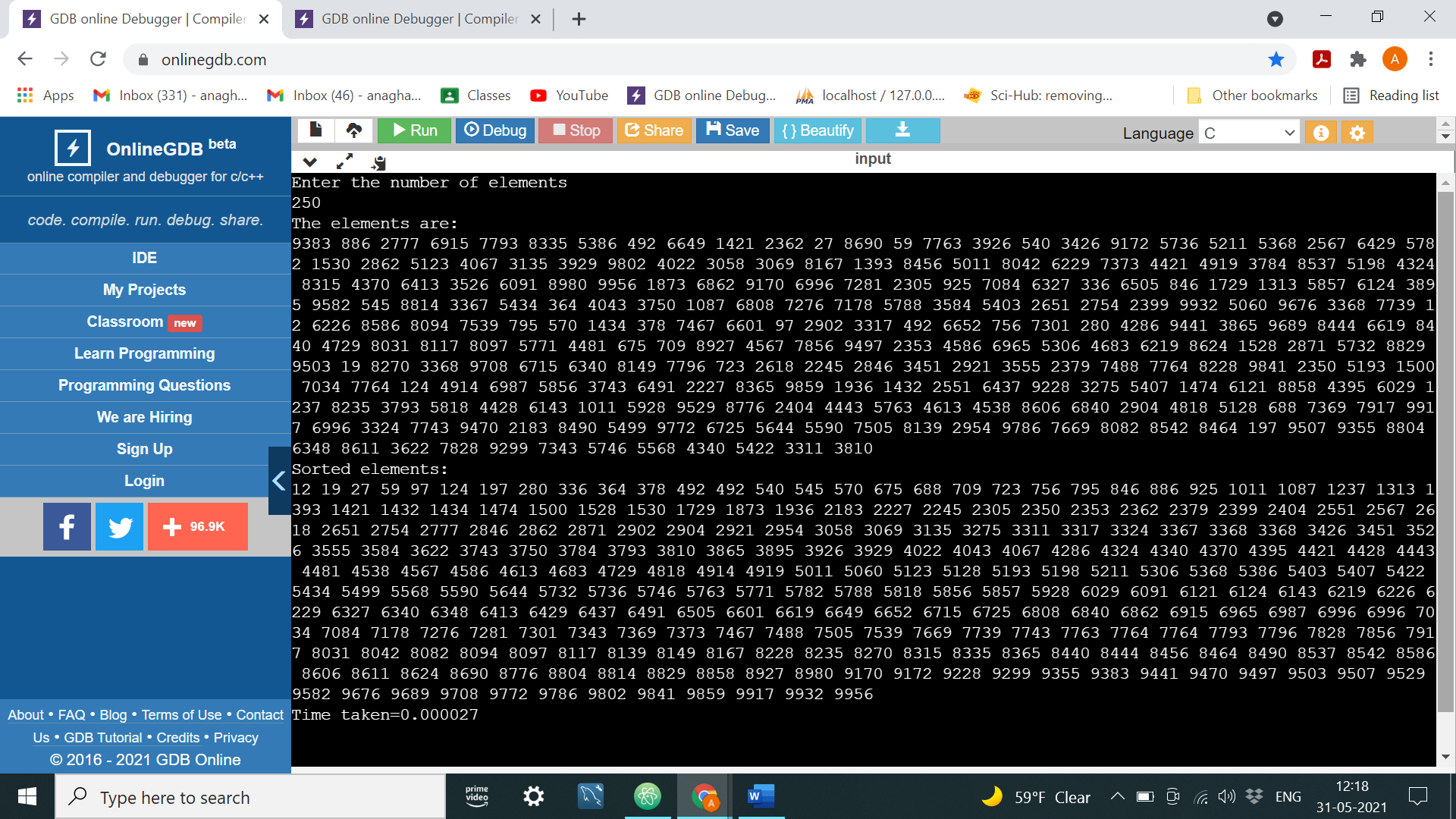
}

printf("\nTime taken=%1f\n",time);

}

OUTPUT:





|  |  |
| --- | --- |
| N | Time Taken |
| 50 | 0.000006 |
| 100 | 0.000011 |
| 200 | 0.000027 |
| 500 | 0.000058 |
| 750 | 0.000096 |
| 1000 | 0.000149 |