**VISVESVARAYA TECHNOLOGICAL UNIVERSITY**

**“JnanaSangama”, Belgaum -590014, Karnataka.**

****

**LAB REPORT**

**on**

**Analysis and Design of Algorithms**

***Submitted by***

**Aravind Siddharth R(1BM20CS021)**

***in partial fulfillment for the award of the degree of***

**BACHELOR OF ENGINEERING**

***in***

**COMPUTER SCIENCE AND ENGINEERING**



**B.M.S. COLLEGE OF ENGINEERING**

**(Autonomous Institution under VTU)**

**BENGALURU-560019**

**May-2022 to July-2022**

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

**Bull Temple Road, Bangalore 560019**

(Affiliated To Visvesvaraya Technological University, Belgaum)

**Department of Computer Science and Engineering**



**CERTIFICATE**

This is to certify that the Lab work entitled “**Analysis and Design of Algorithms**” carried out by **Aravind Siddharth R(1BM20CS021),**who is bonafide student of **B. M. S. College of Engineering.** It is in partial fulfillment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum during the year 2022. The Lab report has been approved as it satisfies the academic requirements in respect of a **Analysis and Design of Algorithms - (19CS4PCADA)** work prescribed for the said degree.

Name of the Lab-In charge:               **Dr. Jyothi S Nayak**

Dr. Nagarathna

Professor Professor and Head

Department of CSE Department of CSE

BMSCE, Bengaluru BMSCE, Bengaluru

`

**Index Sheet**

| **Sl. No.** | **Experiment Title** | **Page No.** |
| --- | --- | --- |
| **1** | Write a recursive program to Solve  **a)** Towers-of-Hanoi problem **b)** To find GCD |  |
| **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. |  |
| **3** | Sort a 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. |  |
| **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. |  |
| **5** | Sort a given set of N integer elements using Insertion Sort technique and compute its time taken. |  |
| **6** | Write program to obtain the Topological ordering of vertices in a given digraph. |  |
| **7** | Implement Johnson Trotter algorithm to generate permutations. |  |
| **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. |  |
| **9** | Sort a given set of N integer elements using Quick Sort technique and compute its time taken. |  |
| **10** | Sort a given set of N integer elements using Heap Sort technique and compute its time taken. |  |
| **11** | Implement Warshall’s algorithm using dynamic programming |  |
| **12** | Implement 0/1 Knapsack problem using dynamic programming. |  |
| **13** | Implement All Pair Shortest paths problem using Floyd’s algorithm. |  |
| **14** | Find Minimum Cost Spanning Tree of a given undirected graph using Prim’s algorithm. |  |
| **15** | Find Minimum Cost Spanning Tree of a given undirected graph using Kruskals algorithm. |  |
| **16** | From a given vertex in a weighted connected graph, find shortest paths to other vertices using Dijkstra’s algorithm. |  |
| **17** | 1. Implement “Sum of Subsets” using Backtracking. “Sum of Subsets” problem: Find a subset of a given set S = {s1,s2,……,sn} of n positive integers whose sum is equal to a given positive integer d. For example, if S = {1,2,5,6,8} and d = 9 there are two solutions {1,2,6} and {1,8}. A suitable message is to be displayed if the given problem instance doesn’t have a solution. |  |
| **18** | Implement “N-Queens Problem” using Backtracking. |  |

**Course Outcome**

| **CO1** | Ability to **analyze** time complexity of Recursive and Non-Recursive algorithms using asymptotic notations. |
| --- | --- |
| **CO2** | Ability to **design** efficient algorithms using various design techniques. |
| **CO3** | Ability to **apply** the knowledge of complexity classes P, NP, and NP-Complete and prove certain problems are NP-Complete |
| **CO4** | Ability to **conduct** practical experiments to solve problems using an appropriate designing method and find time efficiency. |

**1a.Tower of Hanoi**

**#include<stdio.h>**

**void toh(char s,char m,char d,int n){**

**if(n == 0)return;**

**toh(s,d,m,n-1);**

**printf("from : %c to %c\n",s,d);**

**toh(m,s,d,n-1);**

**}**

**int main(){**

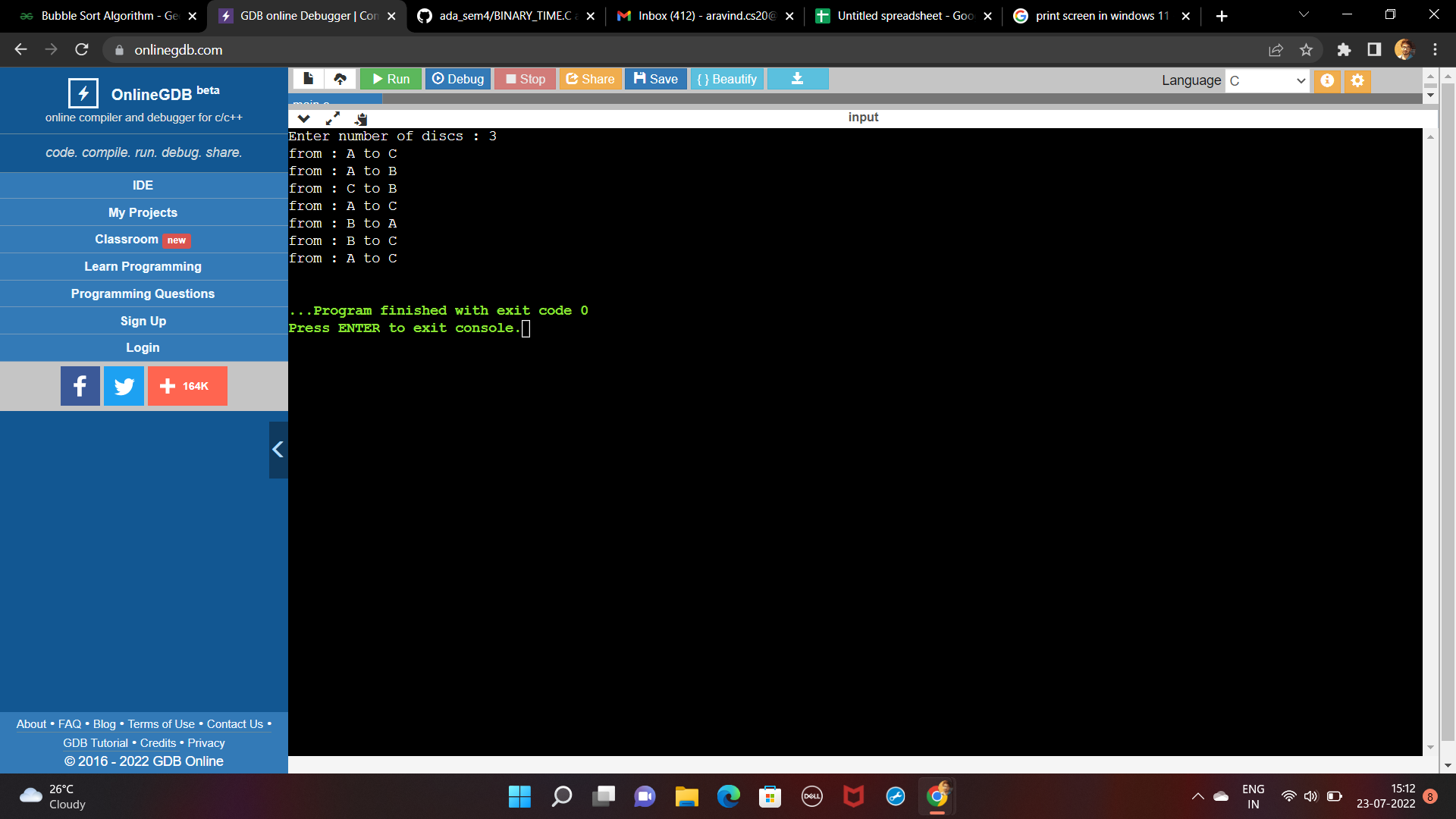
**int n;**

**printf("Enter number of discs : ");**

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

**toh('A','B','C',n);**

**}**

****

**1b.GCD**

**#include<stdio.h>**

**int gcd(int num1,int num2){**

**if(num2 == 0){**

**return num1;**

**}**

**gcd(num2,num1%num2);**

**}**

**int main(){**

**int num1,num2;**

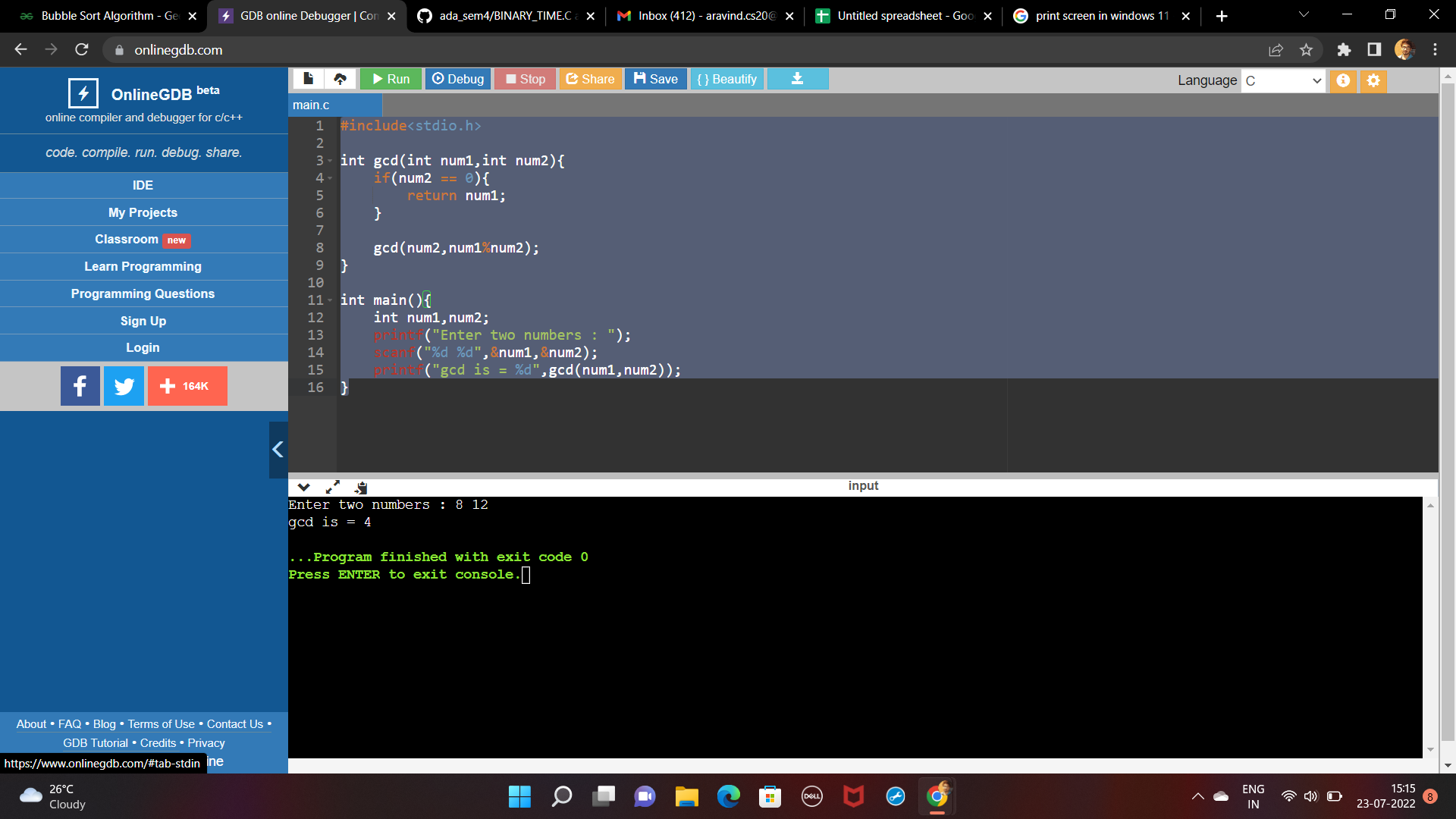
**printf("Enter two numbers : ");**

**scanf("%d %d",&num1,&num2);**

**printf("gcd is = %d",gcd(num1,num2));**

**return 0;**

**}**

****

**2a.Linear Search And Binary Search**

**#include<stdio.h>**

**#include<time.h>**

**int linearSearch(int arr[],int key,int index,int size){**

**if(index < size){**

**if(arr[index] == key)return index;**

**int res = linearSearch(arr,key,index+1,size);**

**}**

**else{**

**return -1;**

**}**

**}**

**int binarySearch(int array[],int key,int low,int high){**

**if(low > high) return -1;**

**int middle = (low+high)/2;**

**if(key == array[middle]) return middle;**

**else if(key > array[middle]) return binarySearch(array,key,middle+1,high);**

**else if(key < array[middle]) return binarySearch(array,key,low,middle-1);**

**}**

**int main(){**

**int n = 100;**

**clock\_t start,end;**

**while(n<=10000){**

**int arr[n];**

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

**arr[i] = i;**

**}**

**start = clock();**

**int ans = linearSearch(arr,n,0,n);**

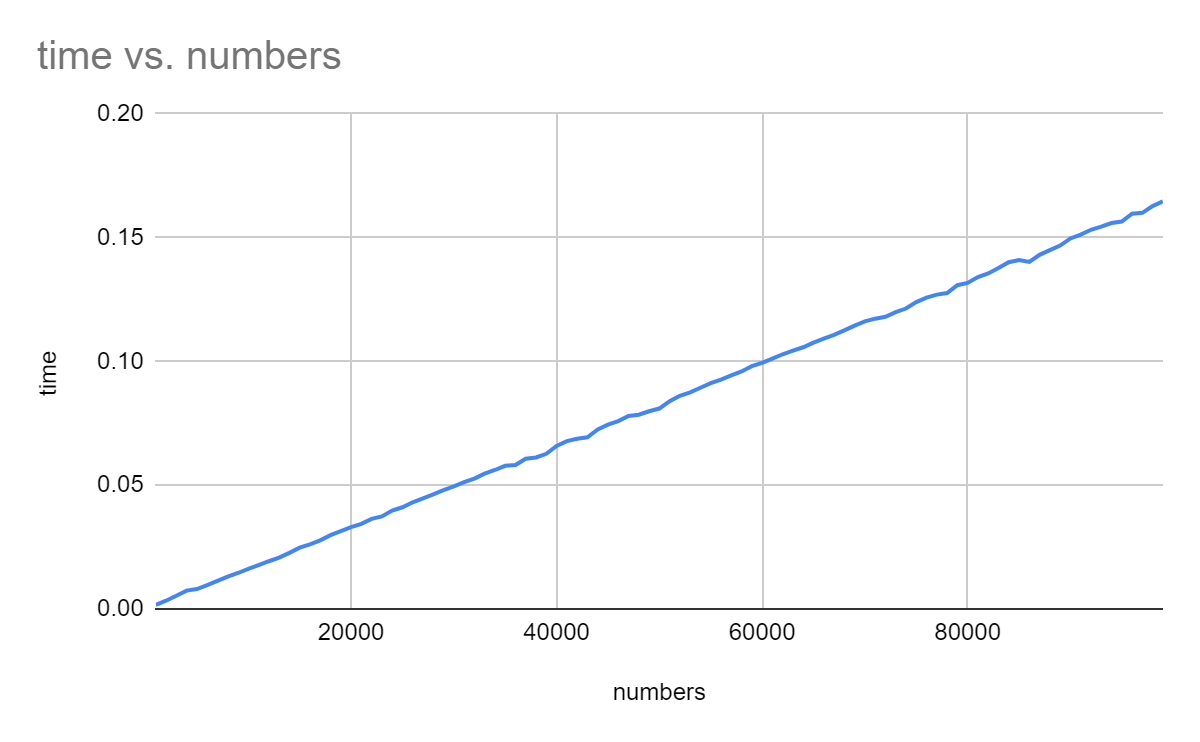
**end = clock();**

**printf("%f \n",(double)(end - start)/CLOCKS\_PER\_SEC);**

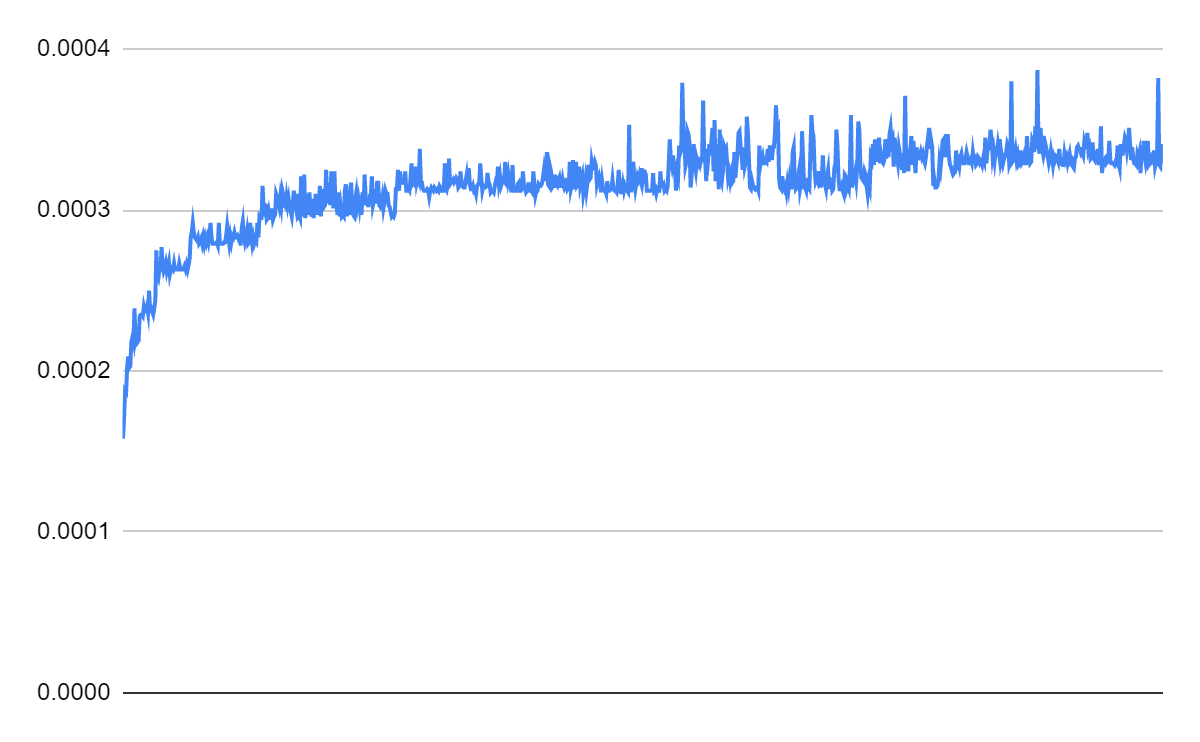
**n= n+100;**

**}**

**}**

****

**Linear Search**

****

**Binary Search**

**3.Selection Sort**

**#include<time.h>**

**#include<stdio.h>**

**void swap(int \*ptr1,int \*ptr2){**

**int temp = \*ptr1;**

**\*ptr1 = \*ptr2;**

**\*ptr2 = temp;**

**}**

**void selectionSort(int arr[], int n)**

**{**

**int i, j, min\_idx;**

**for (i = 0; i < n-1; i++)**

**{**

**min\_idx = i;**

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

**if (arr[j] < arr[min\_idx])**

**min\_idx = j;**

**swap(&arr[min\_idx], &arr[i]);**

**}**

**}**

**void printArray(int arr[],int n){**

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

**printf("%d ",arr[i]);**

**}**

**printf("\n");**

**}**

**int main(){**

**int n = 100;**

**clock\_t start,end;**

**while(n <= 10000){**

**int arr[n];**

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

**arr[i] = n-i;**

**}**

**start = clock();**

**selectionSort(arr,n);**

**end = clock();**

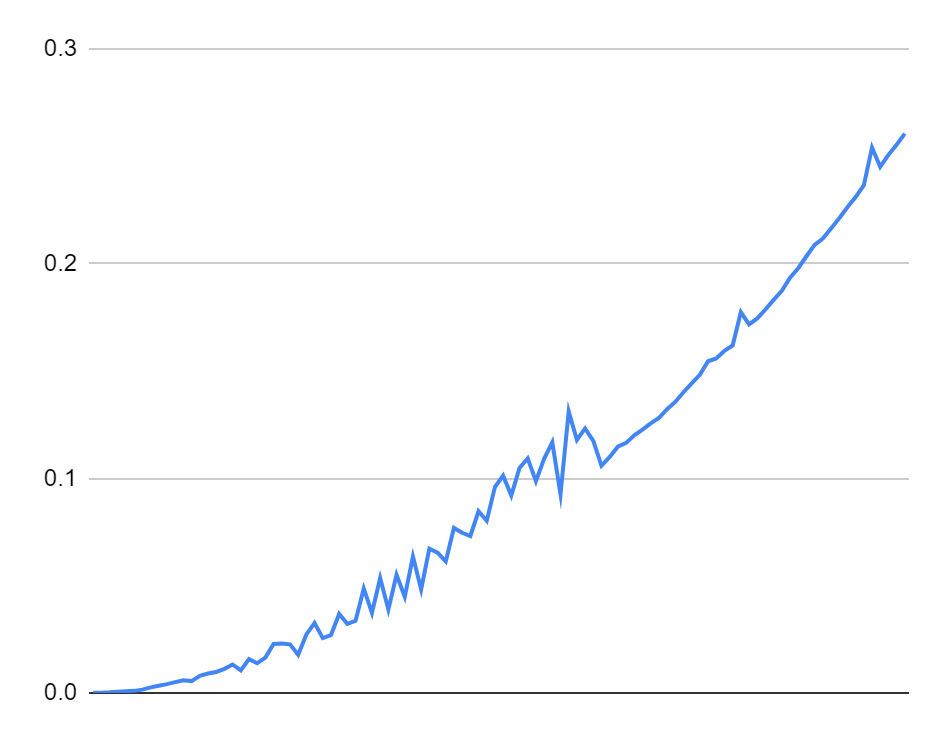
**//printArray(arr,n);**

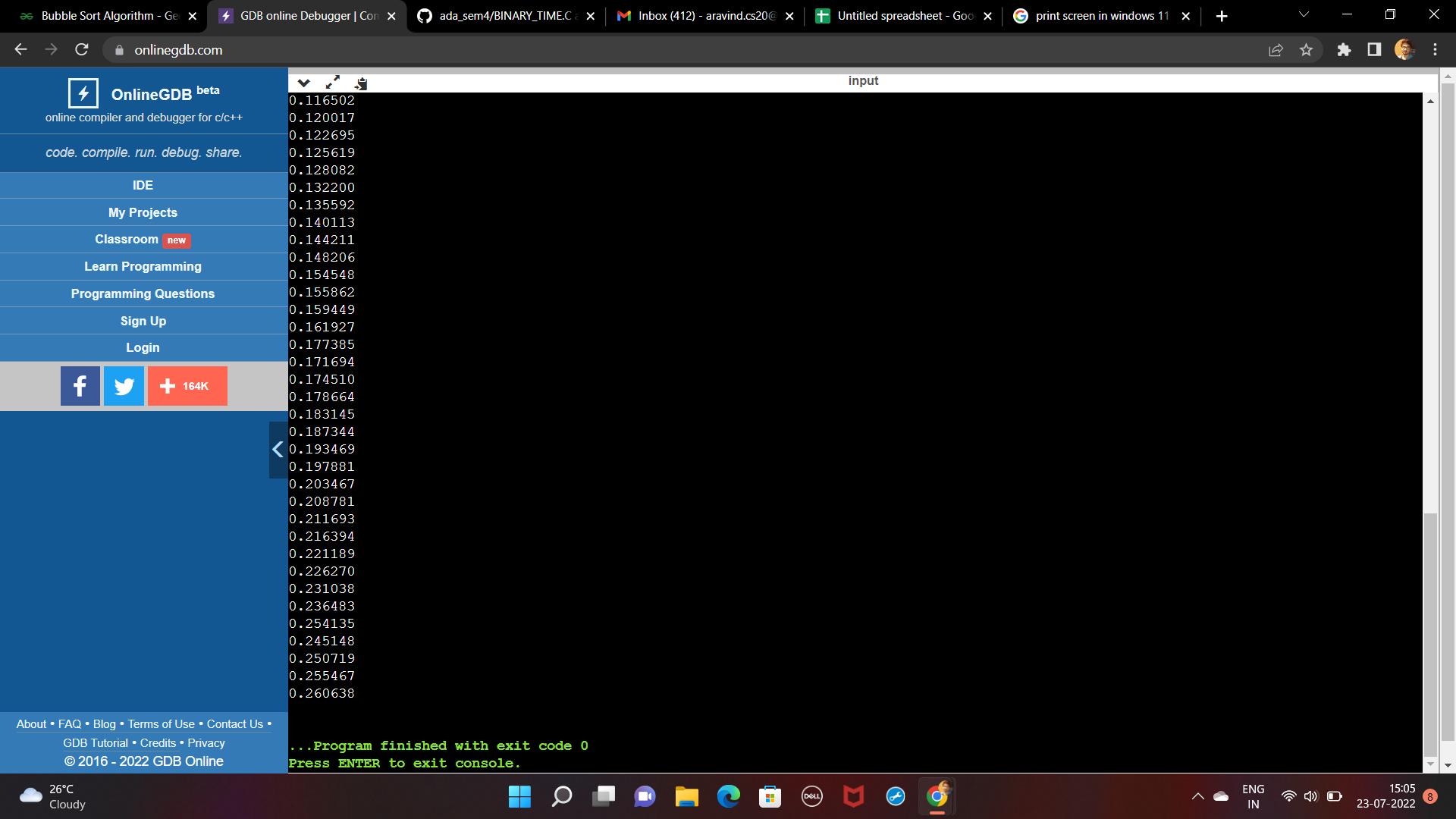
**n= n+100;**

**printf("%f \n",(double)(end-start)/CLOCKS\_PER\_SEC);**

**}**

**}**

****

****

**4a.BFS**

**#include<stdio.h>**

**#define v 5**

**void bfs(int adj[][v],int start,int n,int visited[]){**

**int queue[n];**

**int front = -1;**

**queue[++front] = start;**

**visited[start] = 1;**

**while(front != -1){**

**start = queue[front--];**

**printf("%d ",start);**

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

**if(adj[start][i]==1 && visited[i] == 0){**

**queue[++front] = i;**

**visited[i] = 1;**

**}**

**}**

**}**

**}**

**int main(){**

**int adj[v][v],start;**

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

**for(int i=0;i<v;i++){**

**for(int j=0;j<v;j++){**

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

**}**

**}**

**printf("Enter start state : ");**

**scanf("%d",&start);**

**int visited[v];**

**for(int i=0;i<v;i++){**

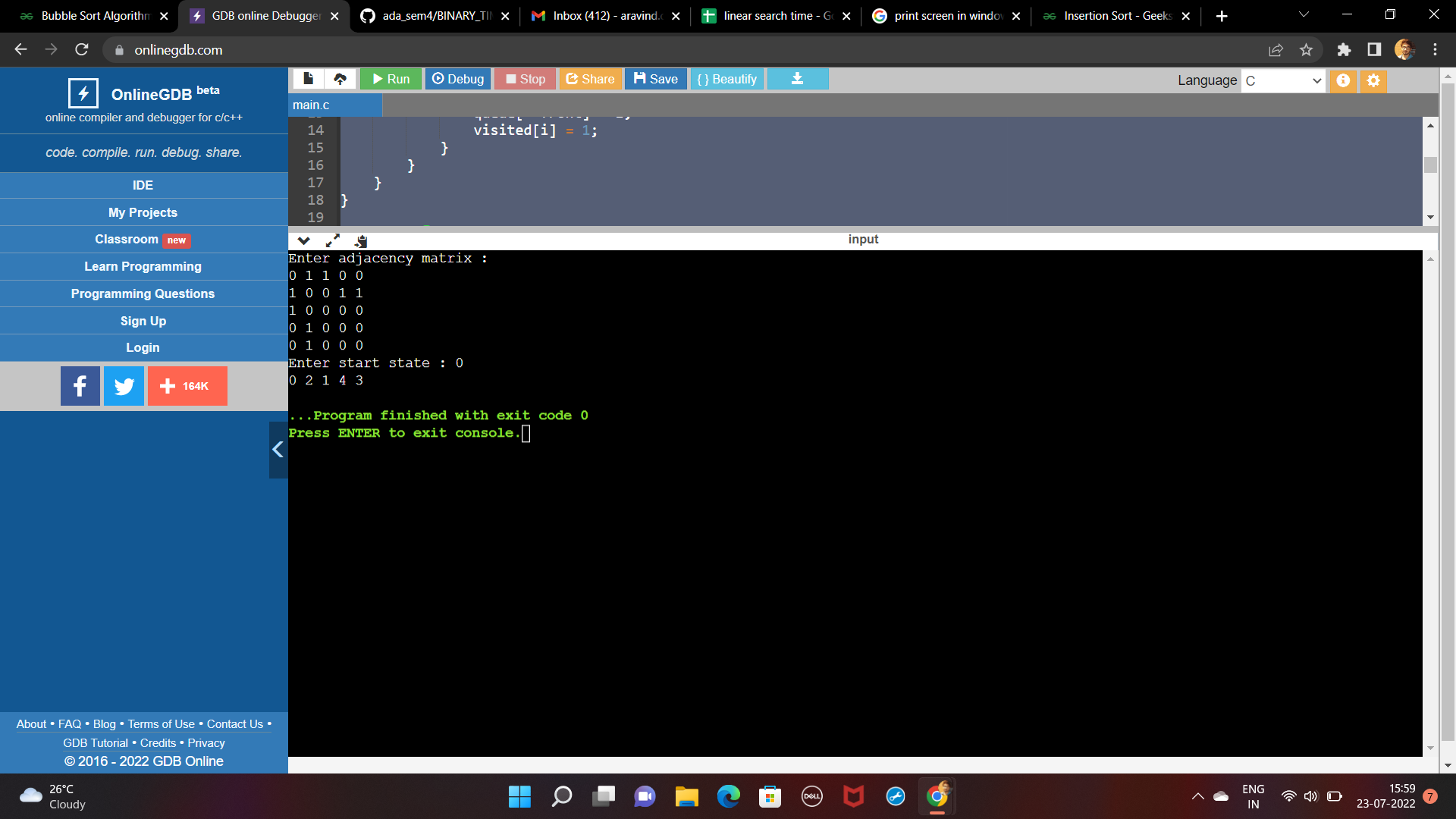
**visited[i] = 0;**

**}**

**bfs(adj,start,v,visited);**

**return 0;**

**}**

****

**4b.Connected Graph using DFS**

**#include<stdio.h>**

**#define v 5**

**void bfs(int adj[][5],int start,int n,int visited[]){**

**int queue[n];**

**int front = -1;**

**queue[++front] = start;**

**visited[start] = 1;**

**while(front != -1){**

**start = queue[front--];**

**printf("%d ",start);**

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

**if(adj[start][i]==1 && visited[i] == 0){**

**queue[++front] = i;**

**visited[i] = 1;**

**}**

**}**

**}**

**}**

**void dfs(int adj[][v],int start,int n,int visited[]){**

**printf("%d ",start);**

**visited[start] = 1;**

**for(int i=0;i<v;i++){**

**if(adj[start][i] == 1 && visited[i] == 0){**

**visited[i] = 1;**

**dfs(adj,i,n,visited);**

**}**

**}**

**}**

**int main(){**

**int adj[v][v],start,flag=0;**

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

**for(int i=0;i<v;i++){**

**for(int j=0;j<v;j++){**

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

**}**

**}**

**int visited[v];**

**for(int i=0;i<v;i++){**

**visited[i] = 0;**

**}**

**printf("dfs : ");**

**dfs(adj,0,v,visited);**

**printf("\n");**

**for(int i=0;i<v;i++){**

**if(visited[i] == 0){**

**flag = 1;**

**}**

**}**

**if(!flag){**

**printf("Connected Graph...");**

**}**

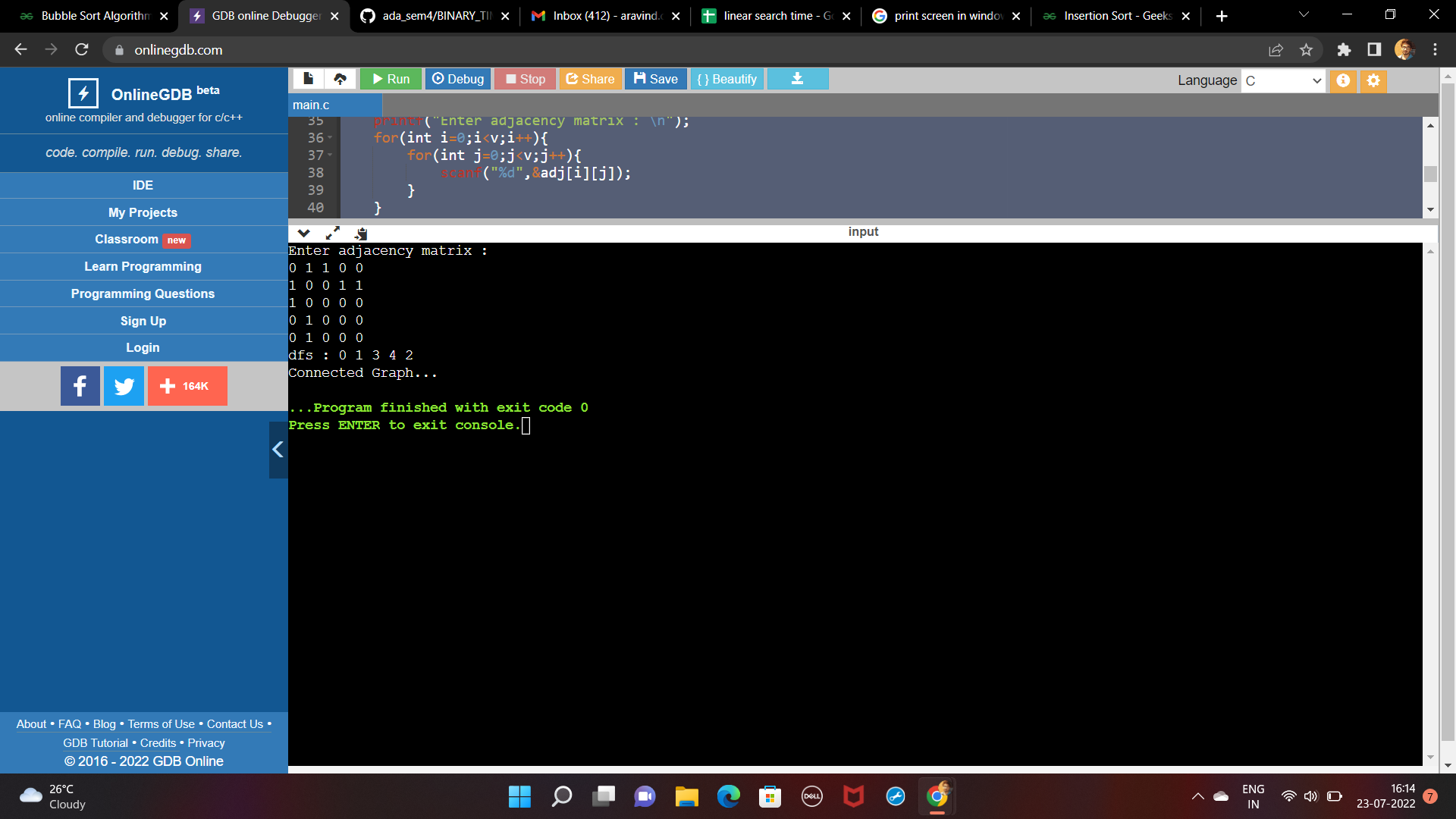
**else{**

**printf("Disconnected Graph...");**

**}**

**return 0;**

**}**

****

**5.Insertion Sort**

**#include<stdio.h>**

**#include<time.h>**

**void insertionSort(int arr[], int n)**

**{**

**int i, key, j;**

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

**key = arr[i];**

**j = i - 1;**

**while (j >= 0 && arr[j] > key) {**

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

**j = j - 1;**

**}**

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

**}**

**}**

**void printArray(int arr[],int n){**

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

**printf("%d ",arr[i]);**

**}**

**printf("\n");**

**}**

**int main(){**

**int n = 100;**

**clock\_t start,end;**

**while(n <= 10000){**

**int arr[n];**

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

**arr[i] = n-i;**

**}**

**start = clock();**

**insertionSort(arr,n);**

**end = clock();**

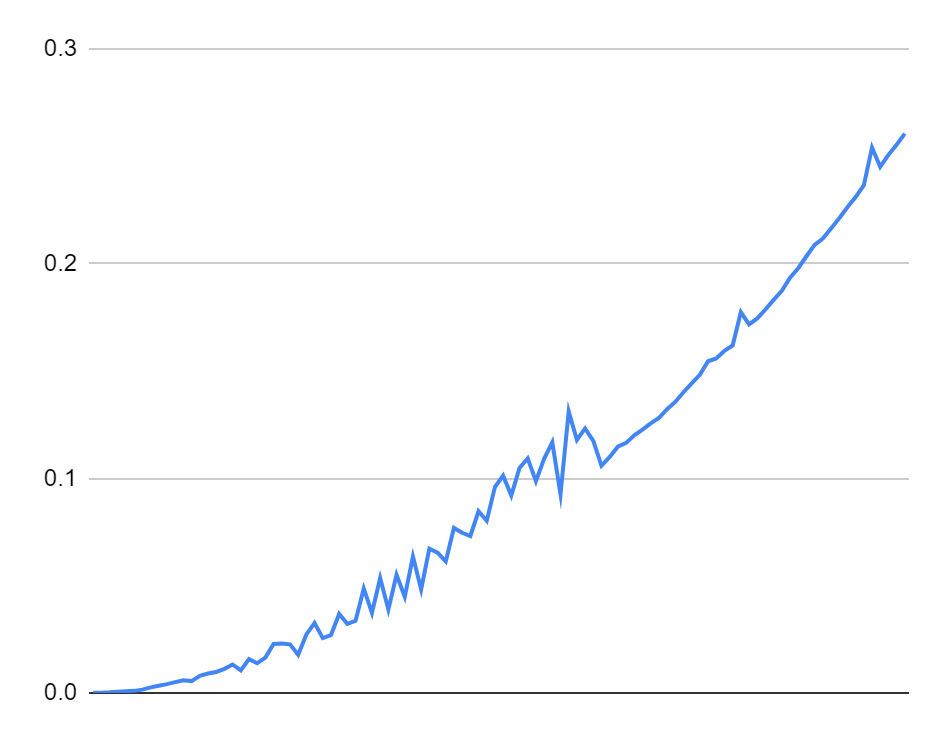
**//printArray(arr,n);**

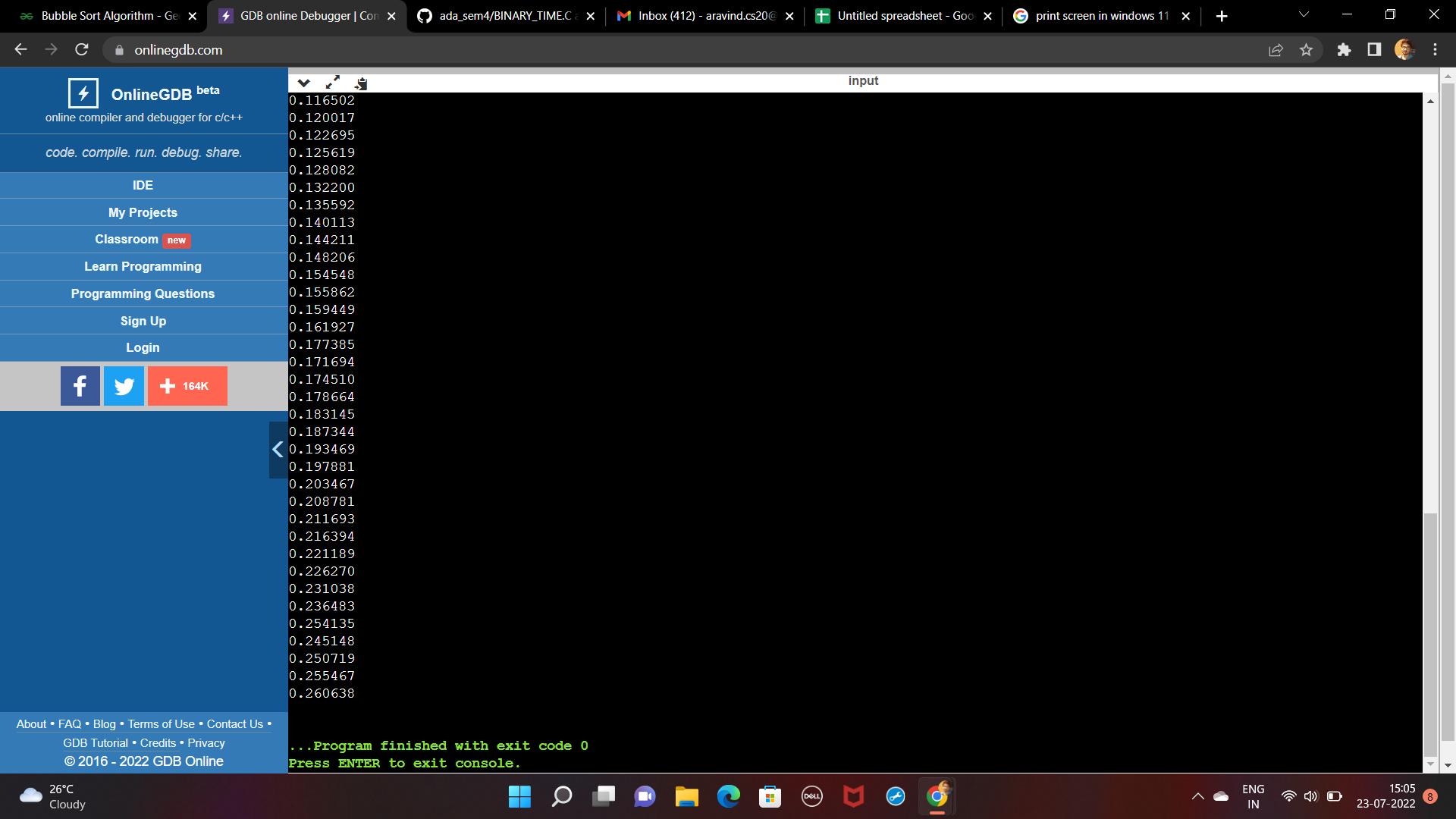
**n= n+100;**

**printf("%f \n",(double)(end-start)/CLOCKS\_PER\_SEC);**

**}**

**}**

****

****

**6.TopoSort**

**#include<bits/stdc++.h>**

**using namespace std;**

**void dfs(vector<int> adj[],vector<int>& visited,int v,stack<int> & s){**

**visited[v] = 1;**

**for(auto it: adj[v]){**

**if(!visited[it]){**

**dfs(adj,visited,it,s);**

**}**

**}**

**s.push(v);**

**}**

**void topo(vector<int> adj[],vector<int>& visited,int n){**

**stack<int> s;**

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

**if(!visited[i]){**

**dfs(adj,visited,i,s);**

**}**

**}**

**while(!s.empty()){**

**cout<<s.top()<<" ";**

**s.pop();**

**}**

**}**

**int main()**

**{**

**int n,m,u,v;**

**cout<<"Enter number of vertices and edges : ";**

**cin>>n>>m;**

**vector<int> adj[n];**

**for(int i=0;i<m;i++){**

**cin>>u>>v;**

**adj[u].push\_back(v);**

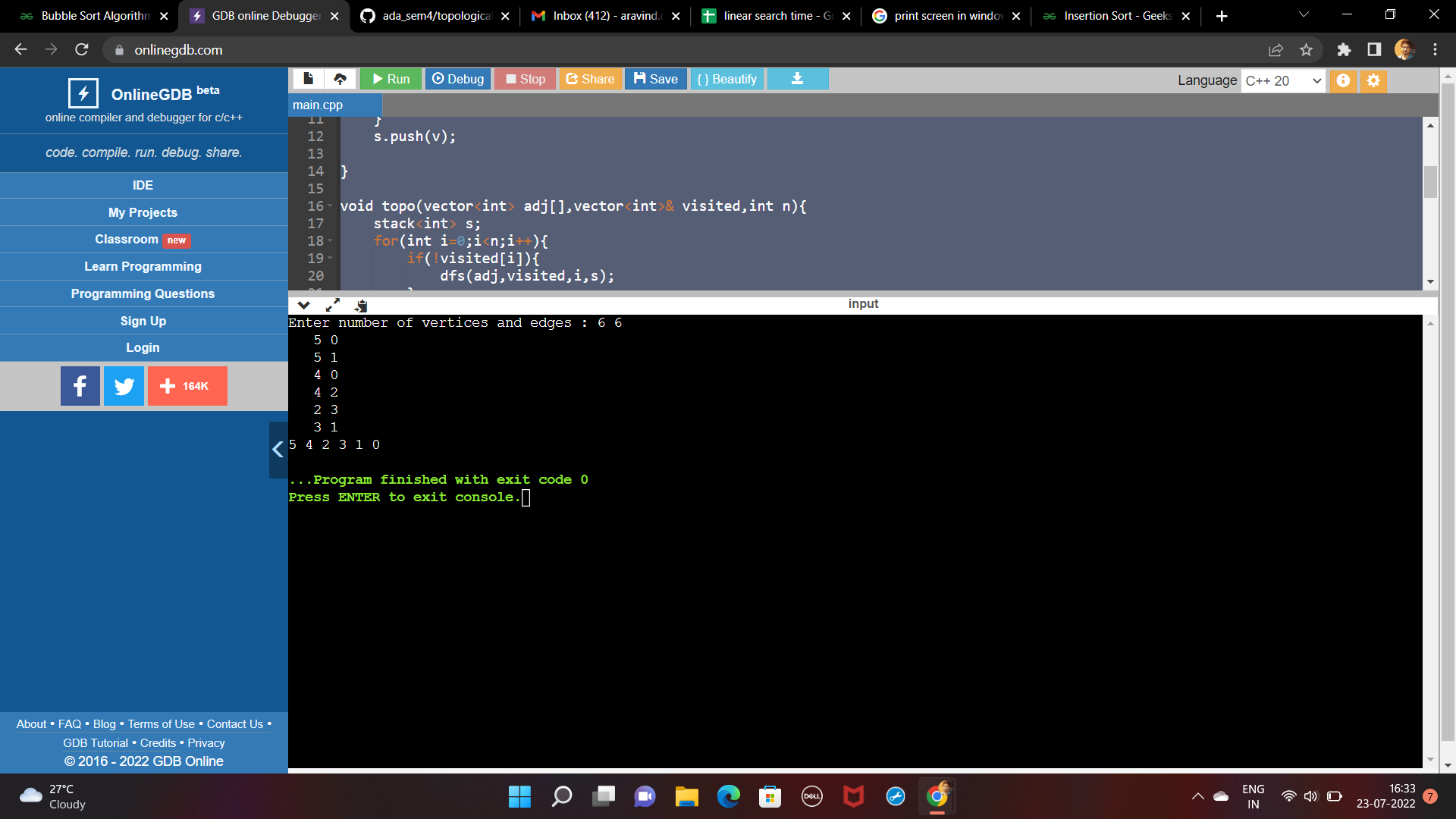
**}**

**vector<int> visited(n,0);**

**topo(adj,visited,n);**

**return 0;**

**}**

****

**7.Johnson Trotter Algorithm**

**#include <bits/stdc++.h>**

**using namespace std;**

**bool LEFT\_TO\_RIGHT = true;**

**bool RIGHT\_TO\_LEFT = false;**

**int searchArr(int a[], int n, int mobile)**

**{**

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

**if (a[i] == mobile)**

**return i + 1;**

**return 0;**

**}**

**int getMobile(int a[], bool dir[], int n)**

**{**

**int mobile\_prev = 0, mobile = 0;**

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

**{**

**if (dir[a[i]-1] == RIGHT\_TO\_LEFT && i!=0)**

**{**

**if (a[i] > a[i-1] && a[i] > mobile\_prev)**

**{**

**mobile = a[i];**

**mobile\_prev = mobile;**

**}**

**}**

**if (dir[a[i]-1] == LEFT\_TO\_RIGHT && i!=n-1)**

**{**

**if (a[i] > a[i+1] && a[i] > mobile\_prev)**

**{**

**mobile = a[i];**

**mobile\_prev = mobile;**

**}**

**}**

**}**

**if (mobile == 0 && mobile\_prev == 0)**

**return 0;**

**else**

**return mobile;**

**}**

**int printOnePerm(int a[], bool dir[], int n)**

**{**

**int mobile = getMobile(a, dir, n);**

**int pos = searchArr(a, n, mobile);**

**if (dir[a[pos - 1] - 1] == RIGHT\_TO\_LEFT)**

**swap(a[pos-1], a[pos-2]);**

**else if (dir[a[pos - 1] - 1] == LEFT\_TO\_RIGHT)**

**swap(a[pos], a[pos-1]);**

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

**{**

**if (a[i] > mobile)**

**{**

**if (dir[a[i] - 1] == LEFT\_TO\_RIGHT)**

**dir[a[i] - 1] = RIGHT\_TO\_LEFT;**

**else if (dir[a[i] - 1] == RIGHT\_TO\_LEFT)**

**dir[a[i] - 1] = LEFT\_TO\_RIGHT;**

**}**

**}**

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

**cout << a[i];**

**cout << " ";**

**return 0;**

**}**

**int fact(int n)**

**{**

**int res = 1;**

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

**res = res \* i;**

**return res;**

**}**

**void printPermutation(int n)**

**{**

**int a[n];**

**bool dir[n];**

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

**{**

**a[i] = i + 1;**

**cout << a[i];**

**}**

**cout << endl;**

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

**dir[i] = RIGHT\_TO\_LEFT;**

**for (int i = 1; i < fact(n); i++)**

**printOnePerm(a, dir, n);**

**}**

**int main()**

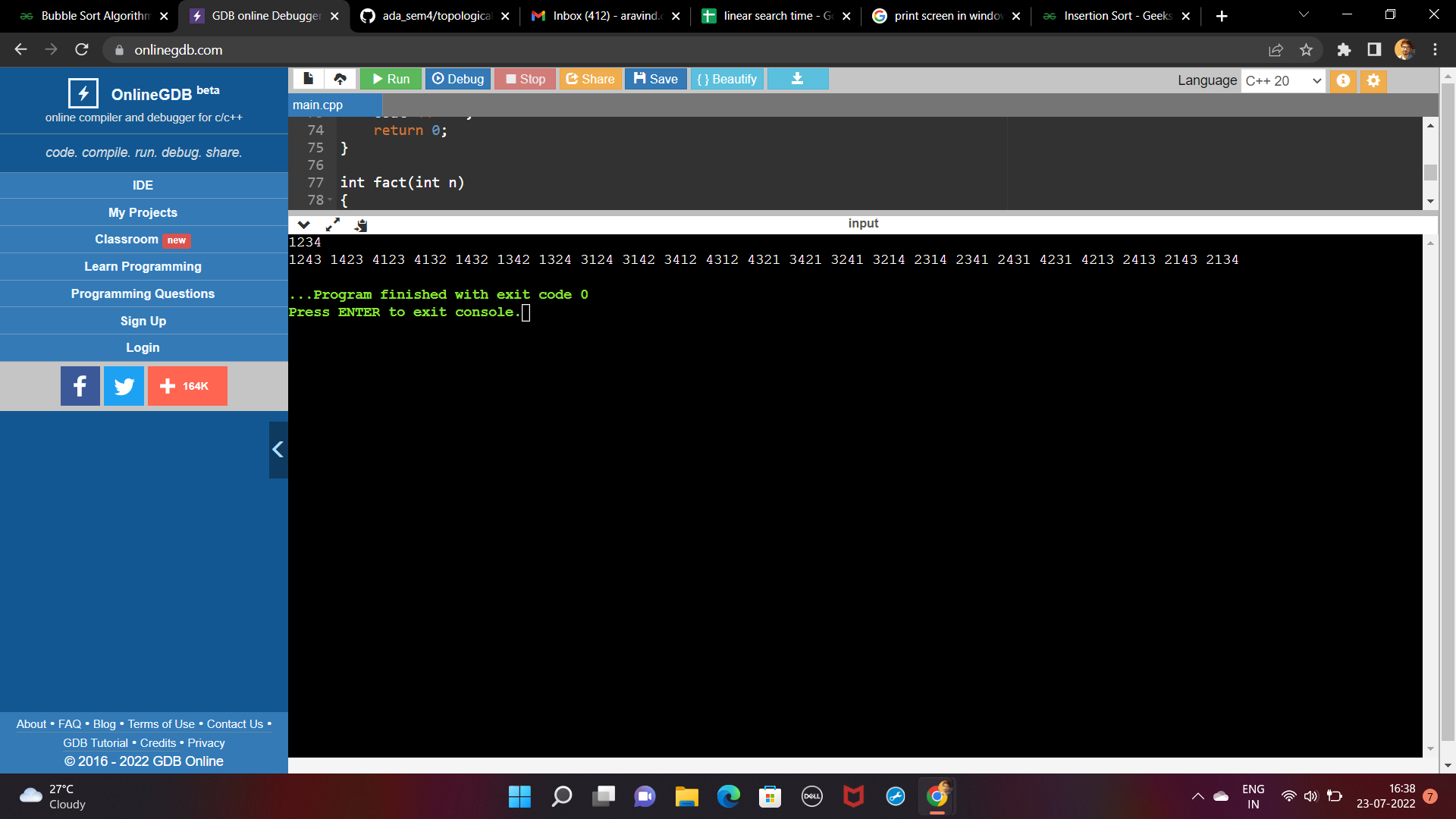
**{**

**int n = 4;**

**printPermutation(n);**

**return 0;**

**}**

****

**8.MergeSort**

**#include<stdio.h>**

**#include<time.h>**

**void merge(int arr[],int left,int mid,int right){**

**int n1 = mid-left+1;**

**int n2 = right - mid;**

**int a[n1];**

**int b[n2];**

**for(int i=0;i<n1;i++){**

**a[i] = arr[left+i];**

**}**

**for(int i=0;i<n2;i++){**

**b[i] = arr[mid+i+1];**

**}**

**int i=0,j=0,k=left;**

**while(i<n1 && j<n2){**

**if(a[i] < b[j]){**

**arr[k++] = a[i++];**

**}**

**else{**

**arr[k++] = b[j++];**

**}**

**}**

**while(i<n1){**

**arr[k++] = a[i++];**

**}**

**while(j<n2){**

**arr[k++] = b[j++];**

**}**

**}**

**void mergeSort(int arr[],int left,int right){**

**if(left>=right)return;**

**int mid = (left+right)/2;**

**mergeSort(arr,left,mid);**

**mergeSort(arr,mid+1,right);**

**merge(arr,left,mid,right);**

**}**

**void printArray(int arr[],int n){**

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

**printf("%d ",arr[i]);**

**}**

**printf("\n");**

**}**

**int main(){**

**int n=100;**

**clock\_t start,end;**

**while(n<=10000){**

**int arr[n];**

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

**arr[i] = n-i;**

**}**

**start = clock();**

**mergeSort(arr,0,n-1);**

**end = clock();**

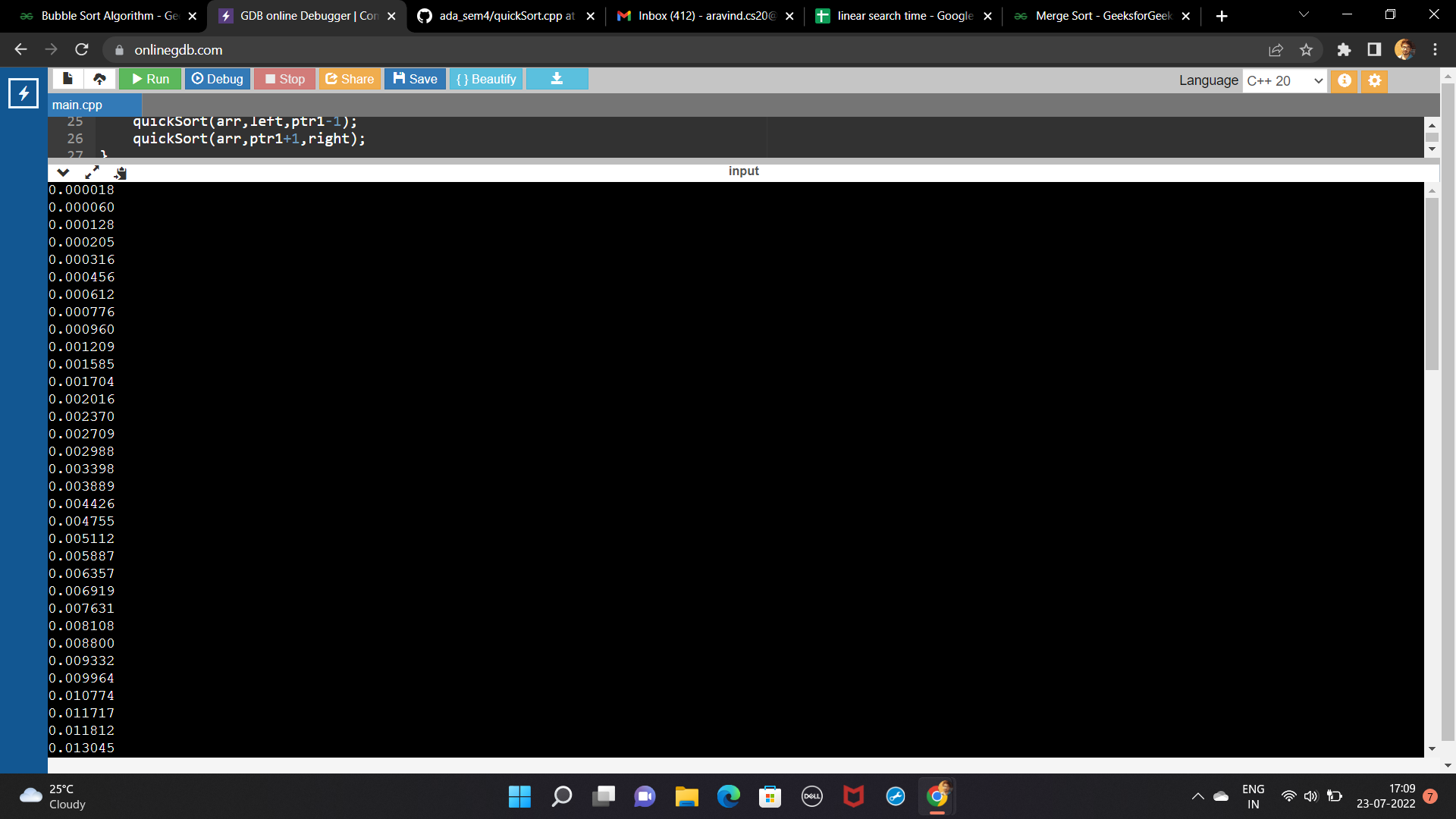
**printf("%f \n",(double)(end-start)/CLOCKS\_PER\_SEC);**

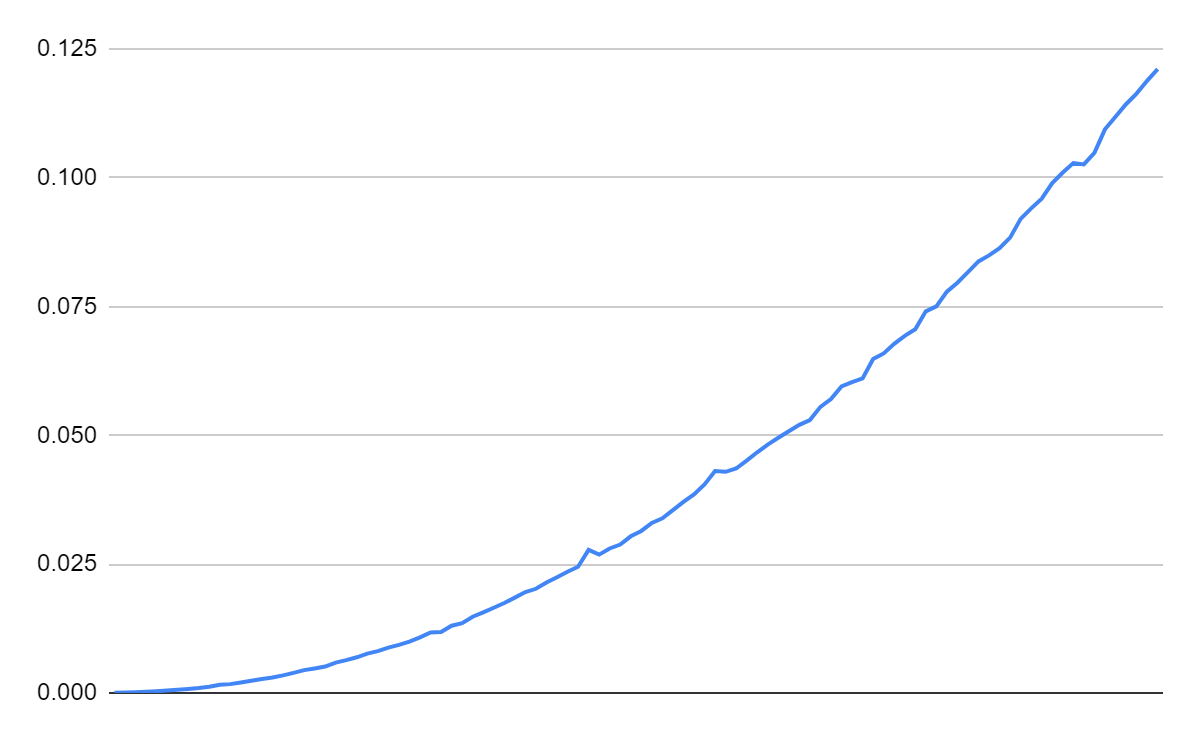
**n+=100;**

**}**

**return 0;**

**}**

****

****

**merge Sort**

**9.Quick Sort**

**#include<stdio.h>**

**void swap(int arr[],int num1,int num2){**

**int temp = arr[num1];**

**arr[num1] = arr[num2];**

**arr[num2] = temp;**

**}**

**void quickSort(int arr[],int left,int right){**

**int pivot = arr[right];**

**if(left>=right)return;**

**int ptr1 = left;**

**int ptr2 = right;**

**while(ptr1< ptr2){**

**while(arr[ptr1] <= pivot && ptr1<ptr2){**

**ptr1++;**

**}**

**while(arr[ptr2] >= pivot && ptr1<ptr2){**

**ptr2--;**

**}**

**swap(arr,ptr1,ptr2);**

**}**

**swap(arr,ptr1,right);**

**quickSort(arr,left,ptr1-1);**

**quickSort(arr,ptr1+1,right);**

**}**

**void printArray(int arr[],int n){**

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

**printf("%d ",arr[i]);**

**}**

**printf("\n");**

**}**

**int main(){**

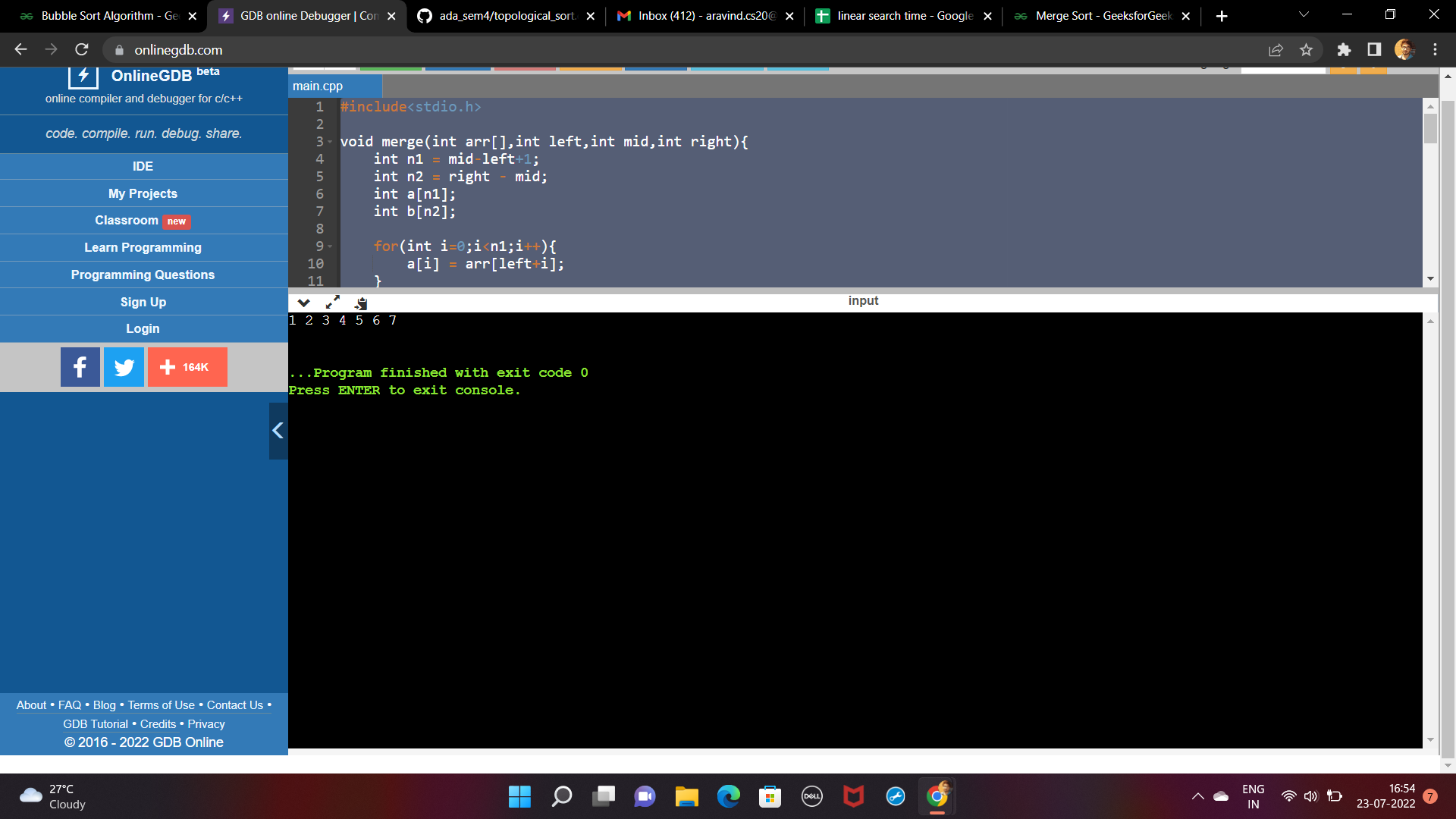
**int arr[7] = {7,6,5,4,3,2,1};**

**quickSort(arr,0,6);**

**printArray(arr,7);**

**return 0;**

**}**

****