|  |  |
| --- | --- |
| Question | Page no. |
| Q1)i)Insertion Sort | 3 – 11 |
| ii)Merge Sort | 11 - 20 |
| Q2) Heap Sort | 20 - 28 |
| Q3) Randomized Quick Sort | 28 - 36 |
| Q4) Radix Sort | 36 - 38 |
| Q5) Bucket Sort | 39 - 40 |
| Q6) Randomized Select | 40 - 44 |
| Q7) Breath First Search | 44 – 48 |
| Q8) Depth First Search | 48 - 53 |
| Q9) i) Prims | 53 - 60 |
| ii) Krushkals | 60 – 64 |
| Q10) Weighted Interval Scheduling | 64 - 69 |
| Q11) 0 – 1 Knapsack Problem | 69 - 70 |

//1. i. Implement Insertion Sort (The program should report the number of comparisons)

//Test run the algorithm on 100 different inputs of sizes varying from //30 to 1000. Count the number of comparisons and draw the graph.

#include<iostream>

#include<bits/stdc++.h>

using namespace std;

int y = 0;

void insertionsort(int arr[],int n){

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

int curr = arr[i];

int prev = i-1;

while(prev >= 0 && arr[prev]>curr){

arr[prev+1] = arr[prev];

prev--;

y++;

}

arr[prev+1] = curr;

}

}

void display(int array[], int size) {

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

cout << array[i] << " ";

cout << endl;

}

void fun(int arr[],int s){

int size = s;

int u = 0;

int b = 1000;

arr[s];

mt19937 num(random\_device{}());

uniform\_int\_distribution<int> dist(u,b);

for(int i= 0;i<s;i++){

arr[i] = dist(num);

}

insertionsort(arr, size-1); //(n-1) for last index

cout << "Array after Sorting: ";

display(arr, size);

}

int main(){

int ub = 30;

int lb = 1000;

int size = 100;

int u = 0;

int arr[size] ;

mt19937

num(random\_device{}());

uniform\_int\_distribution<int> dist(ub, lb);

for (auto& i : arr) {

i = dist(num);

}

for (auto i : arr) {

cout << i << " ";

}

cout << endl;

int c = 0;

cout<<"No. of comparisons = "<<y<<endl;

ofstream myfile;

myfile.open ("kartik.csv");

myfile <<"Iteration No.,Array Size,Time ,Comparisons " <<" \n";

for(int j=0;j<100;j++){

int s = arr[j];

int ar[s];

cout<<endl;

cout<<"size of array = "<<s<<endl;

c++;

clock\_t time\_req;

time\_req = clock();

fun(ar,s);

time\_req = clock()- time\_req;

cout << "Processor time taken for iteration "<<j+1<<" : "

<< (float)time\_req/CLOCKS\_PER\_SEC << " seconds" << endl;

u = y - u;

cout<<"No. of comparisons = "<<u<<endl;

myfile <<j+1<<","<<s<<","<<(float)time\_req/CLOCKS\_PER\_SEC<<","<<u<<" \n";

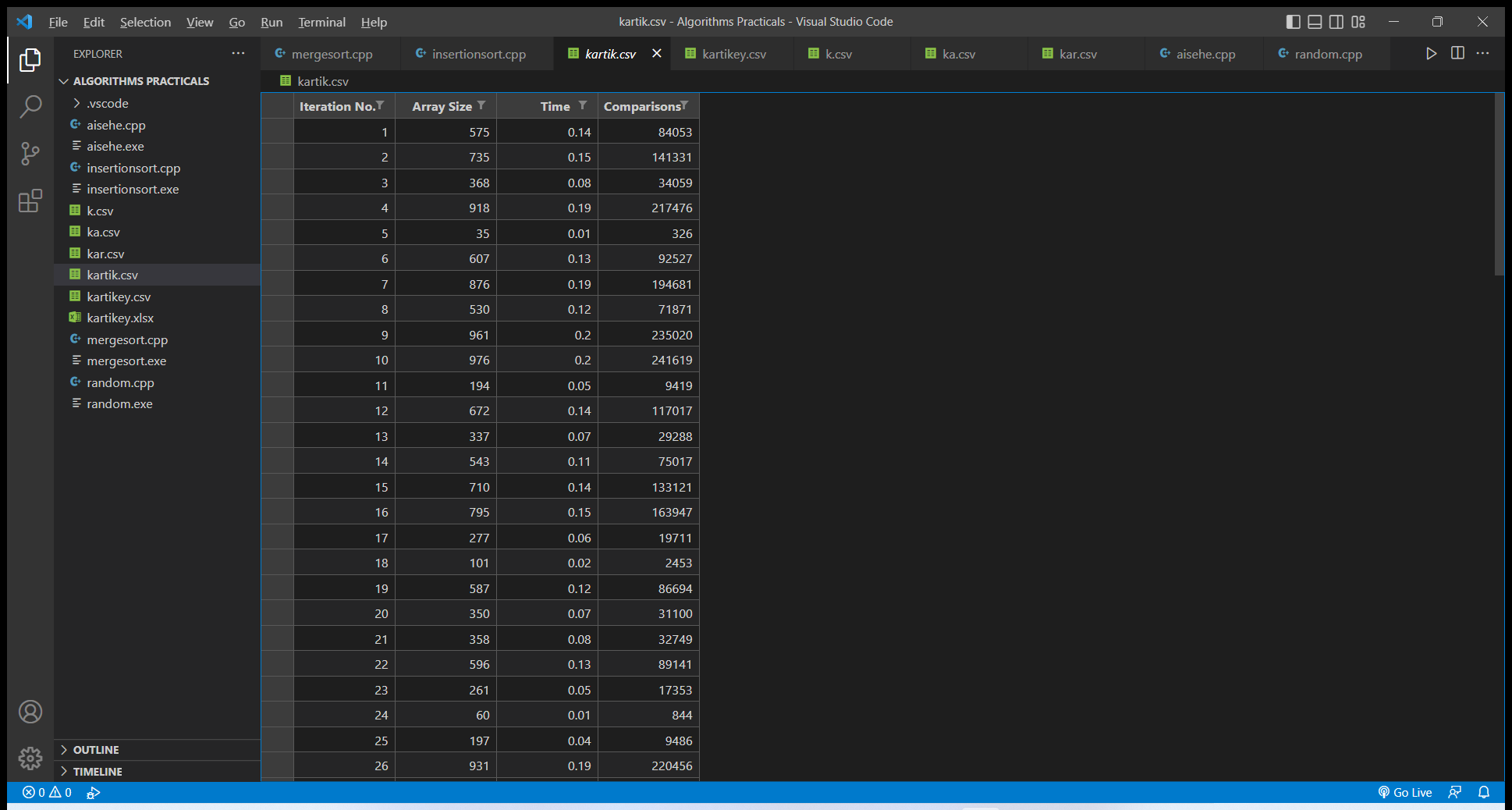
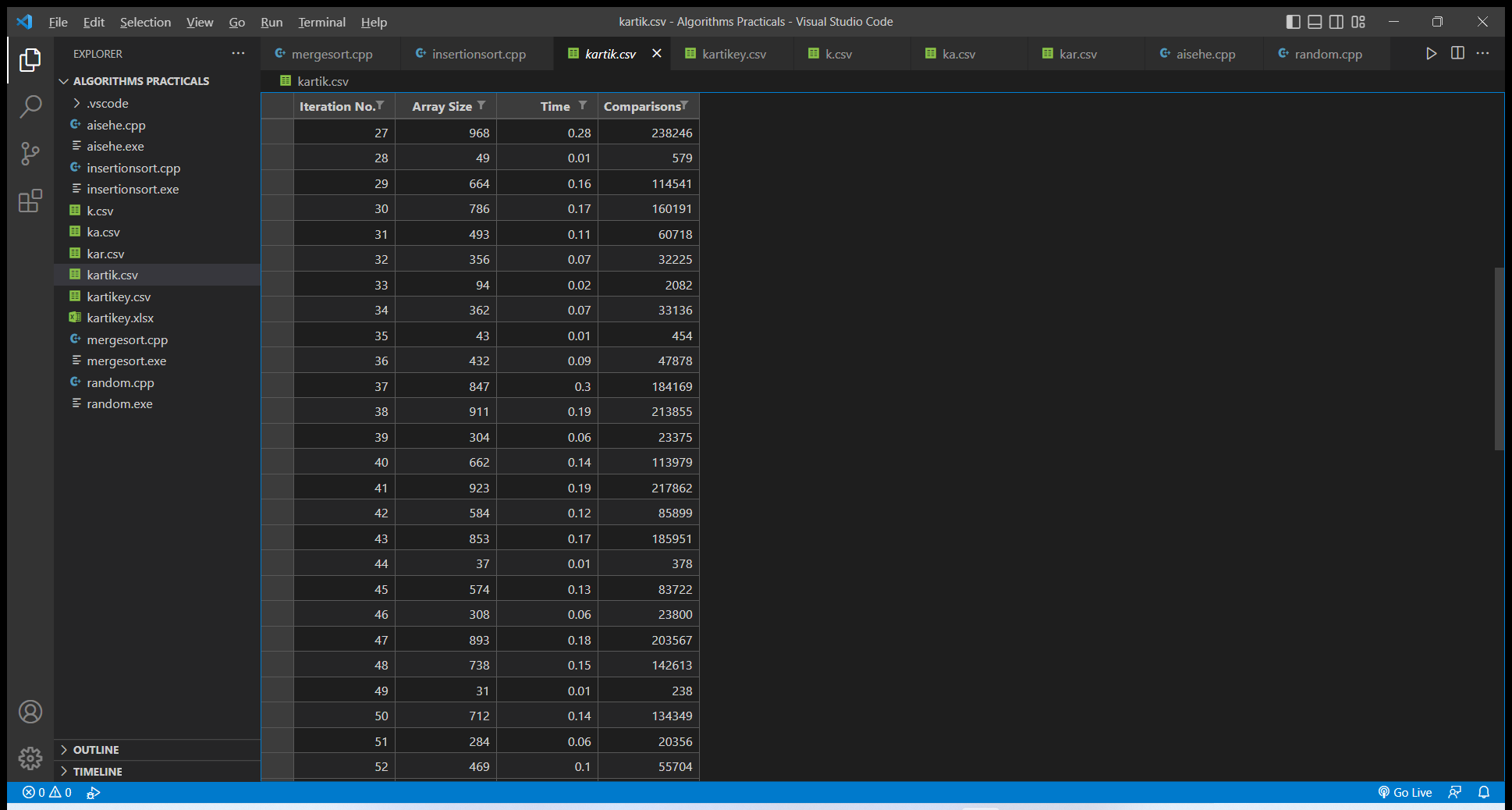
u = y;}

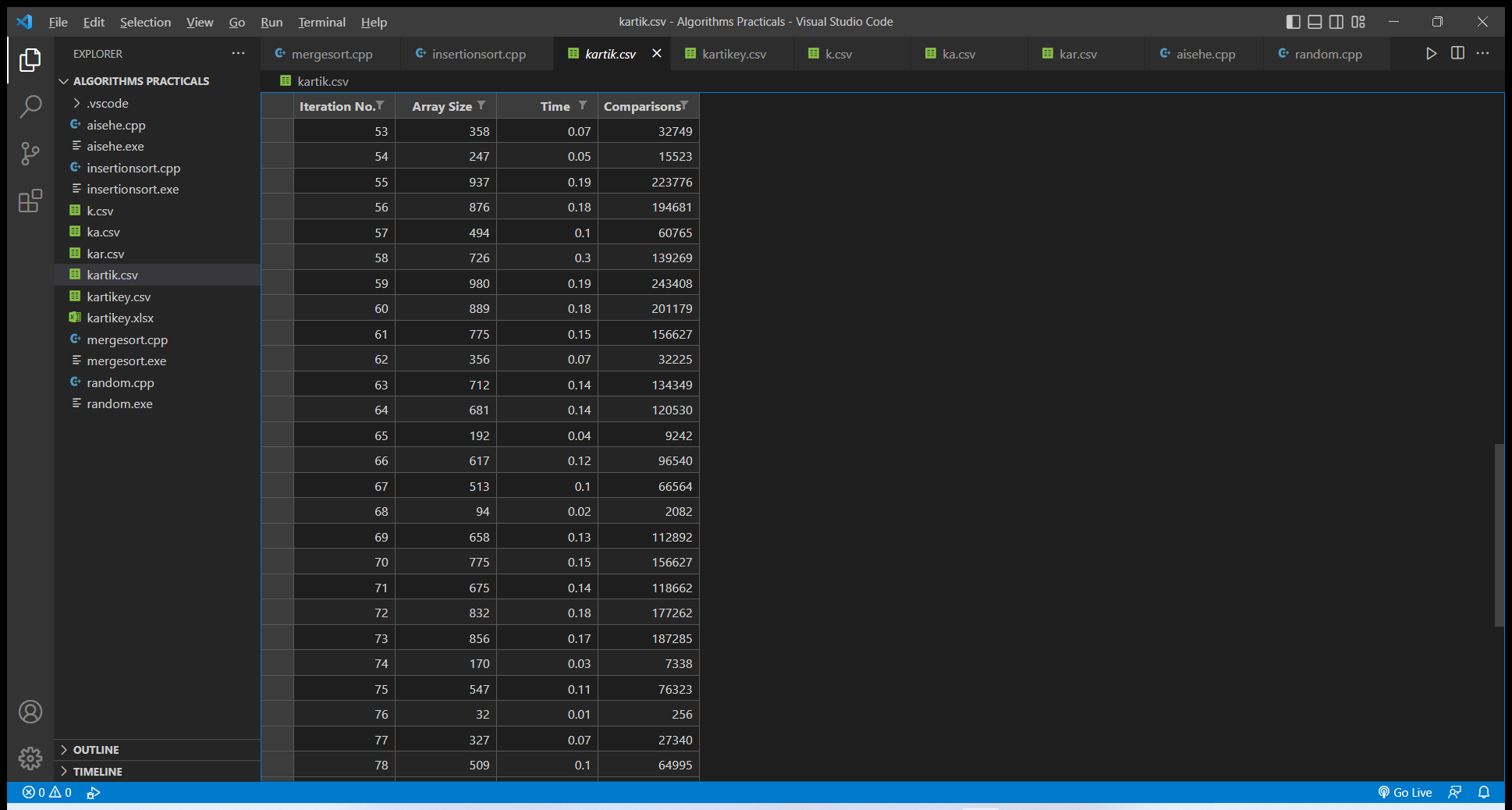
myfile.close();

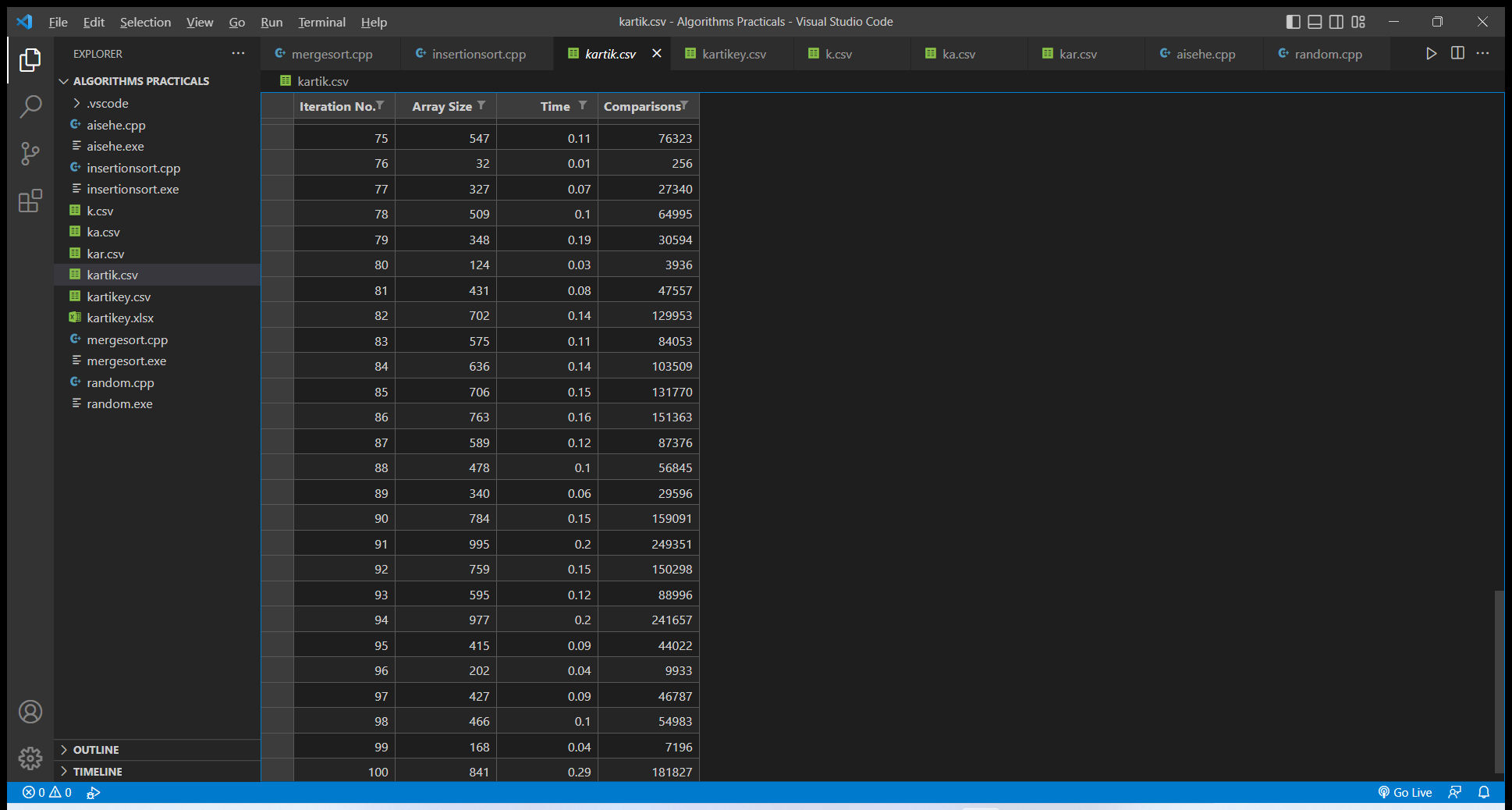
cout<<endl;

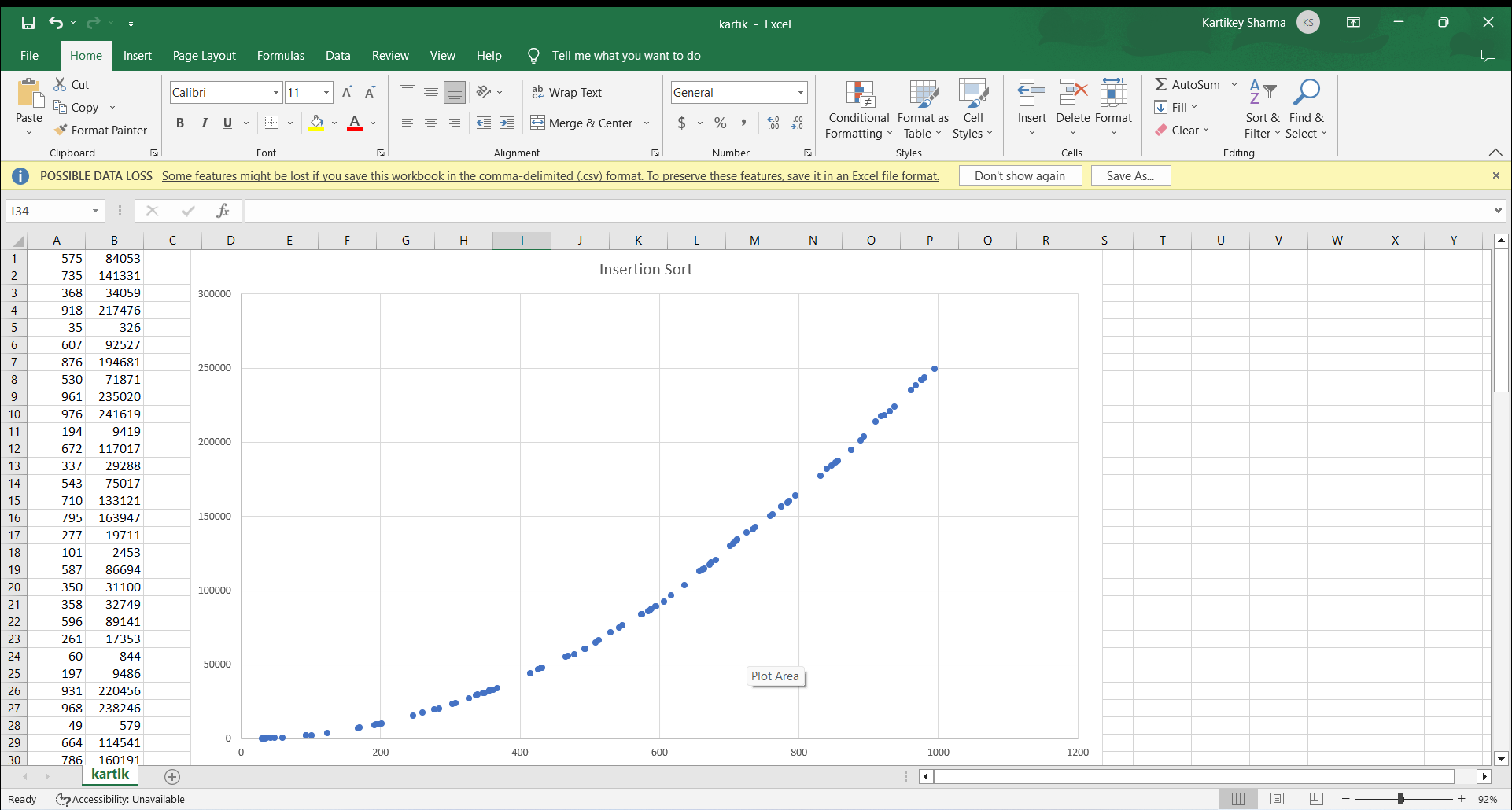
cout<<"number of inputs sorted = "<<c;

}







// ii. Implement Merge Sort (The program should report the number of comparisons)

#include<iostream>

#include<bits/stdc++.h>

#include <random>

using namespace std;

int y = 0;

void display(int arr[], int s) {

for(int i = 0; i<s; i++)

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

cout << endl;

}

void merge(int arr[], int si, int mid, int ei) {

int i, j, k, nl, nr;

nl = mid-si+1; nr = ei-mid;

int larr[nl], rarr[nr];

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

larr[i] = arr[si+i];

for(j = 0; j<nr; j++)

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

i = 0; j = 0; k = si;

while(i < nl && j<nr) {

if(larr[i] <= rarr[j]) {

arr[k] = larr[i];

i++;

}else{

arr[k] = rarr[j];

j++;

}

k++;

y++;

}

while(i<nl) {

arr[k] = larr[i];

i++; k++;

}

while(j<nr) {

arr[k] = rarr[j];

j++; k++;

}

}

void mergeSort(int arr[], int si, int r) {

int mid;

if(si < r) {

int mid = si+(r-si)/2;

mergeSort(arr, si, mid);

mergeSort(arr, mid+1, r);

merge(arr, si, mid, r);

}

}

void fun(int arr[],int s){

int size = s;

int u = 0;

int b = 1000;

arr[s];

mt19937 pr(random\_device{}());

uniform\_int\_distribution<int> dist(u,b);

for(int i= 0;i<s;i++){

arr[i] = dist(pr);

}

mergeSort(arr, 0, size-1);

cout << "Array after Sorting: ";

display(arr, size);

}

int main() {

int ub = 30;

int lb = 1000;

int size = 100;

int u = 0;

int arr[size] ;

mt19937

num(random\_device{}());

uniform\_int\_distribution<int> dist(ub, lb);

for (auto& i: arr) {

i = dist(num);

}

for (auto i: arr) {

cout << i << " ";

}

cout << endl;

int c = 0;

ofstream myfile;

myfile.open ("kartikey.csv");

myfile <<"Iteration No.,Array Size,Time ,Comparisons " <<" \n";

clock\_t time\_req;

for(int j=0;j<100;j++){

int s = arr[j];

int ar[s];

cout<<endl;

cout<<"size of array = "<<s<<endl;

c++;

time\_req = clock();

fun(ar,s);

time\_req = clock()- time\_req;

cout << "Processor time taken for iteration "<<j+1<<" : "

<< (float)time\_req/CLOCKS\_PER\_SEC << " seconds" << endl;

u = y - u;

cout<<"No. of comparisons = "<<u<<endl;

myfile <<j+1<<","<<s<<","<<(float)time\_req/CLOCKS\_PER\_SEC<<","<<u<<" \n";

u = y;

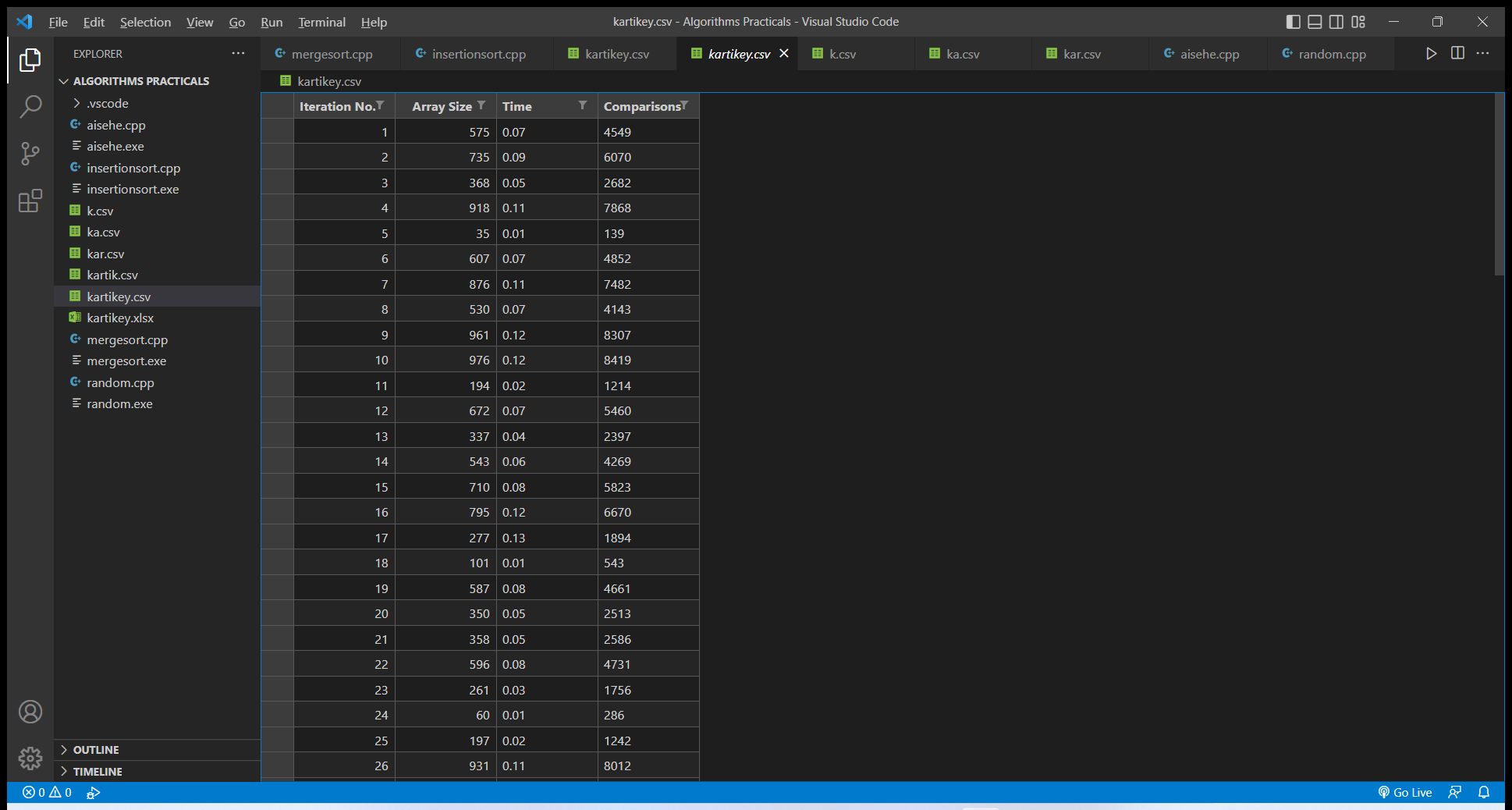
}

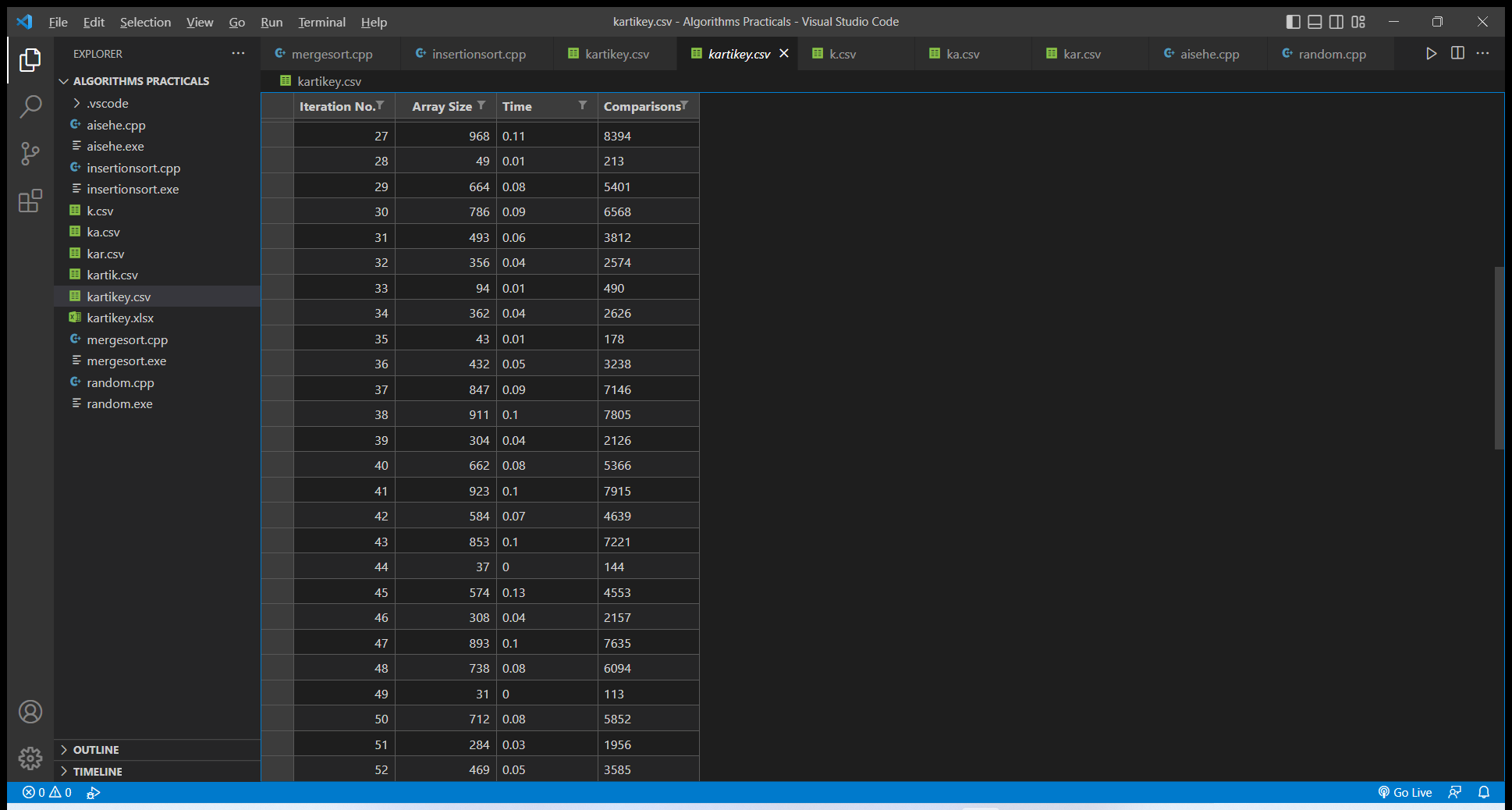
myfile.close();

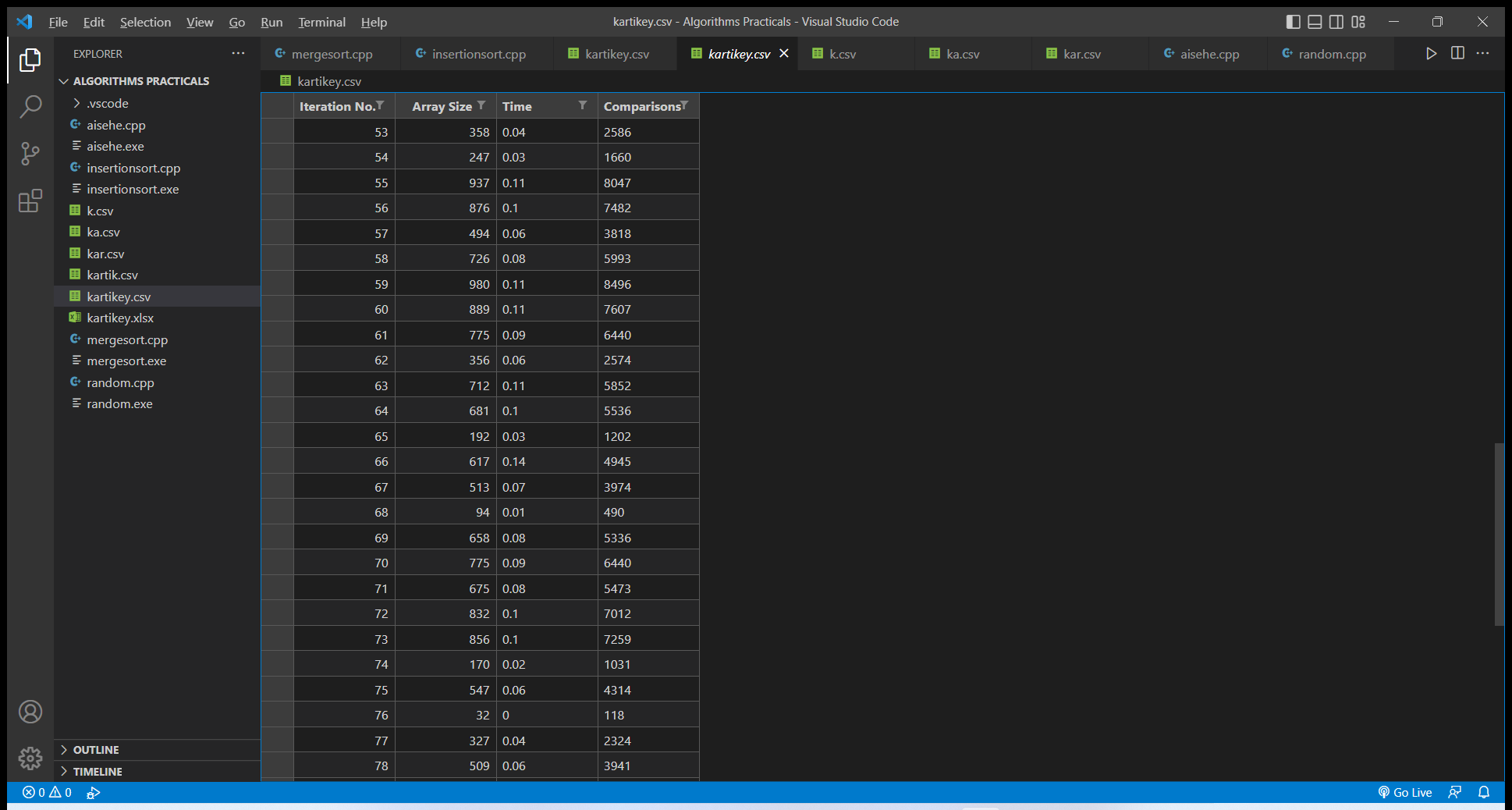
cout<<endl;

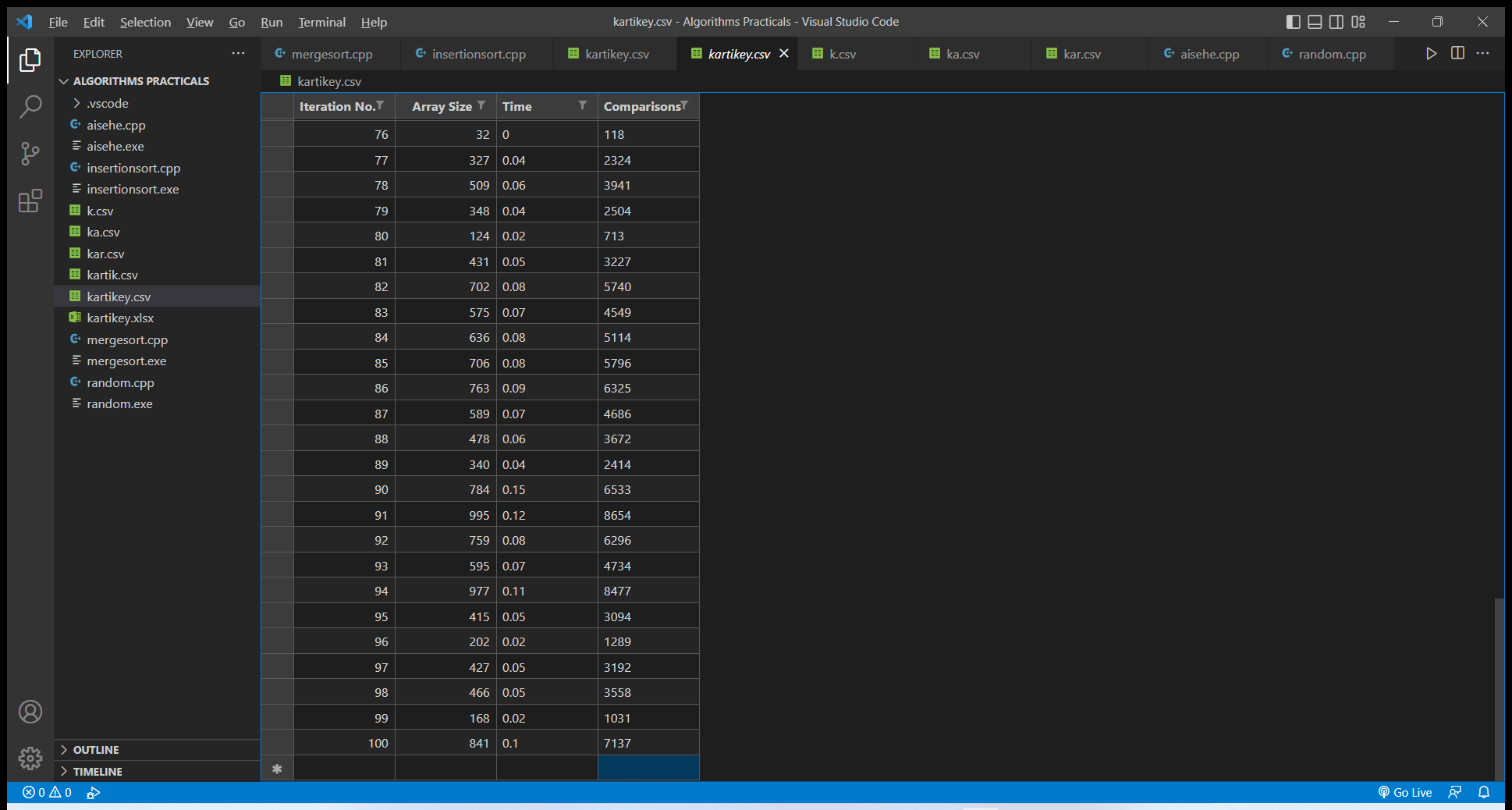
cout<<"number of inputs sorted = "<<c;

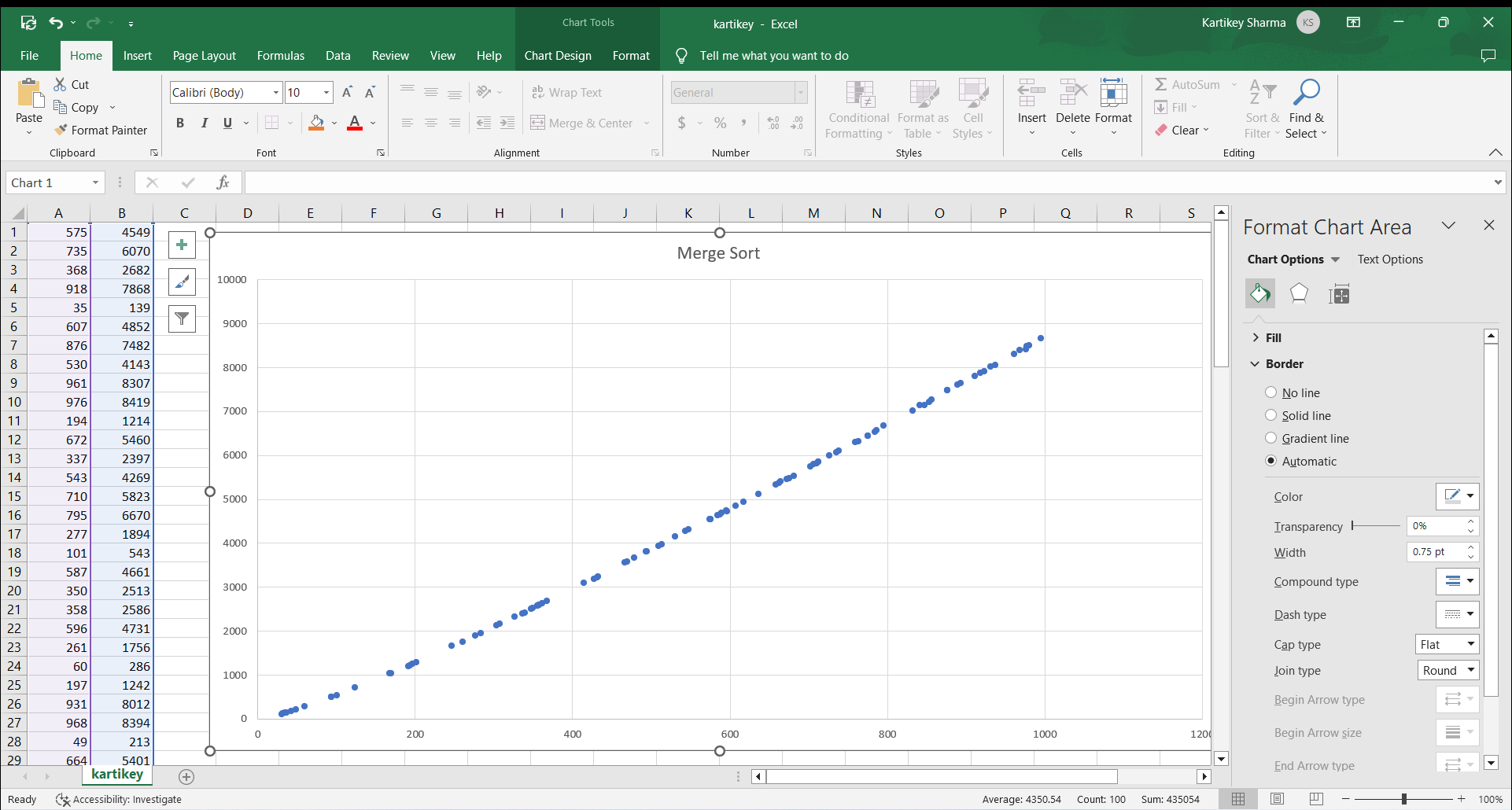
}











//same for heap sort

#include<iostream>

#include<bits/stdc++.h>

#include <random>

using namespace std;

int y = 0;

void heapify(int arr[],int i,int size){

int left = 2\*i + 1;

int right = 2\*i + 2;

int maxidx = i;

if(left<size && arr[left]>arr[maxidx]){

maxidx = left;

y++;

}

if(right<size && arr[right]>arr[maxidx]){

maxidx = right;

y++;

}

if( maxidx != i){

int temp = arr[i];

arr[i] = arr[maxidx];

arr[maxidx] = temp;

heapify(arr,maxidx,size);

}

}

void display(int arr[], int s) {

for(int i = 0; i<s; i++)

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

cout << endl;

}

void heapsort(int arr[],int n){

// Build heap (rearrange array)

for(int i=n/2;i>=0;i--){

heapify(arr,i,n);

}

// One by one extract an element from heap and Moving current root to end

for(int i=n-1;i>=0;i--){

int temp = arr[0];

arr[0] = arr[i];

arr[i] = temp;

// heapify on the reduced heap

heapify(arr,0,i);

}

}

void fun(int arr[],int s){

int size = s;

int u = 0;

int b = 1000;

arr[s];

//generating random numbers and string them in the arr[s]

// where s is the particular index of another 100 size array

mt19937 pr(random\_device{}());

uniform\_int\_distribution<int> dist(u,b);

for(int i= 0;i<s;i++){

arr[i] = dist(pr);

}

heapsort(arr, size-1);

cout << "Array after Sorting: ";

display(arr, size);

}

int main(){

int ub = 30;

int lb = 1000;

int size = 100;

int u = 0;

int arr[size] ;

//generating 100 random numbers and storing them in an array

mt19937 num(random\_device{}());

uniform\_int\_distribution<int> dist(ub, lb);

for (auto& i: arr) {

i = dist(num);

}

for (auto i: arr) {

cout << i << " ";

}

cout << endl;

int c = 0;

ofstream myfile;

myfile.open ("kartikeyh.csv");

myfile <<"Iteration No.,Time,Array Size ,Comparisons " <<" \n";

clock\_t time\_req;

for(int j=0;j<100;j++){

int s = arr[j];

int ar[s];

cout<<endl;

cout<<"size of array = "<<s<<endl;

c++;

time\_req = clock();

fun(ar,s);

time\_req = clock()- time\_req;

cout << "Processor time taken for iteration "<<j+1<<" : "

<< (float)time\_req/CLOCKS\_PER\_SEC << " seconds" << endl;

u = y - u;

cout<<"No. of comparisons = "<<u<<endl;

myfile <<j+1<<","<<(float)time\_req/CLOCKS\_PER\_SEC<<","<<s<<","<<u<<" \n";

u = y;

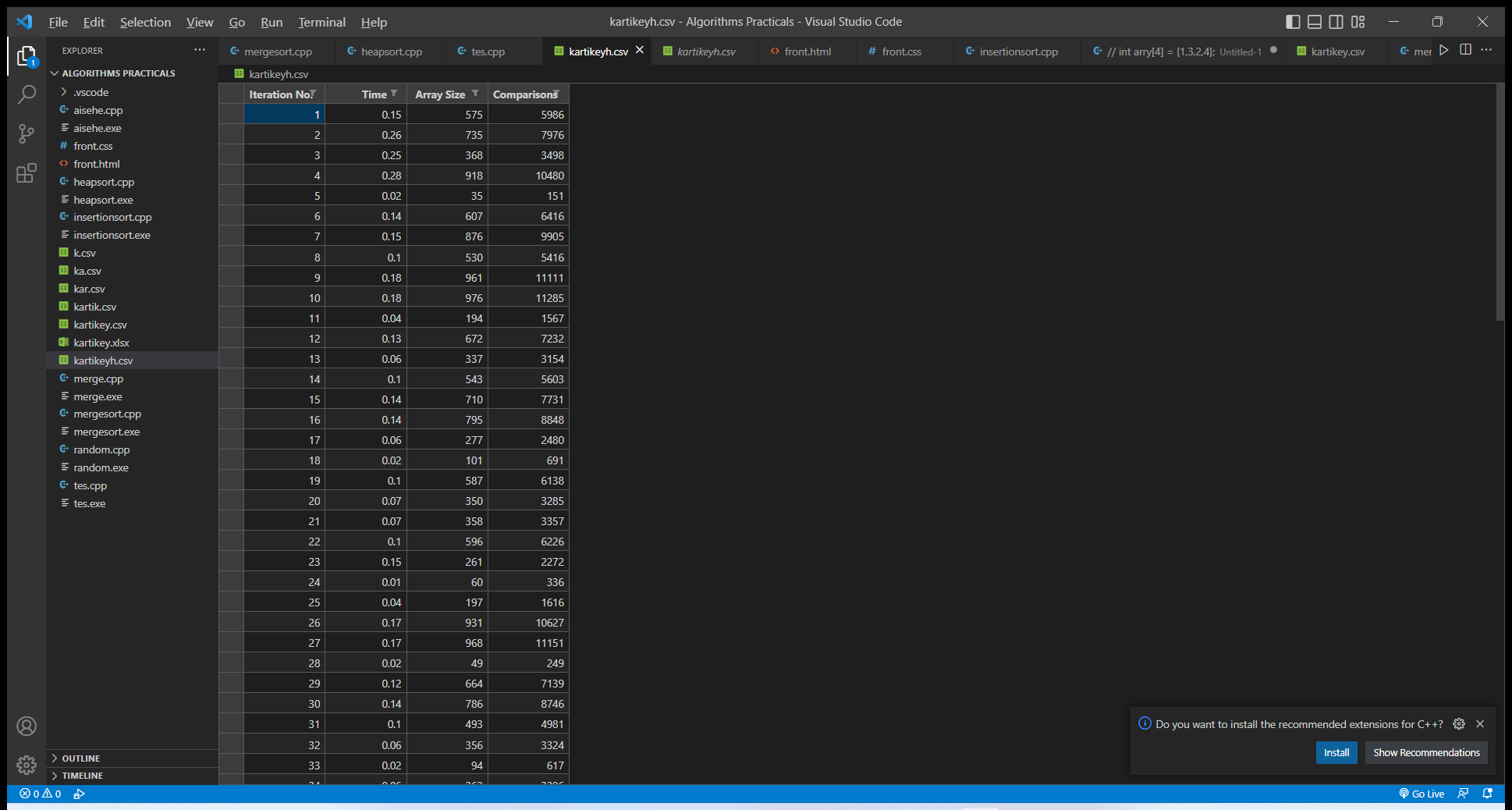
}

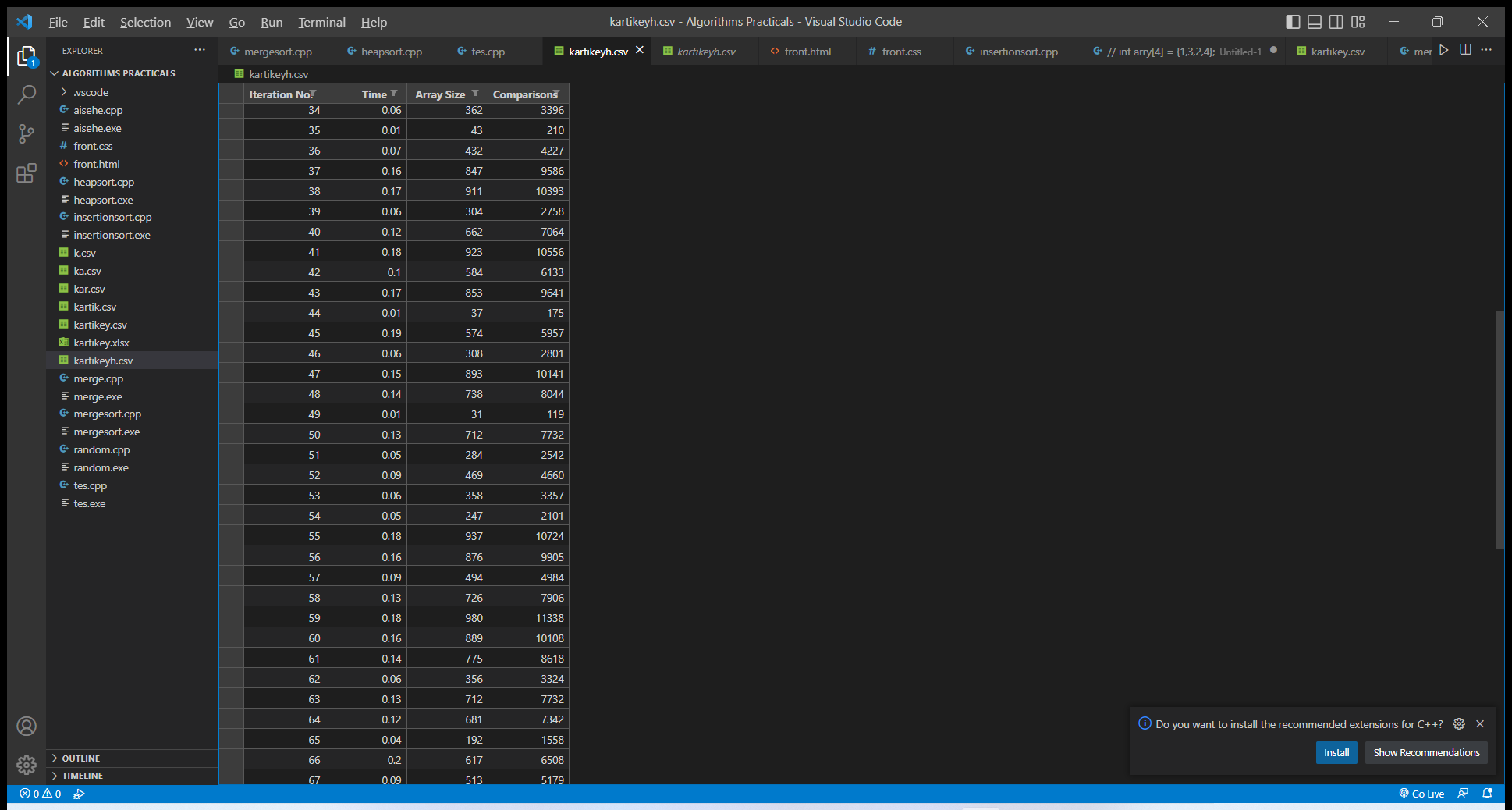
myfile.close();

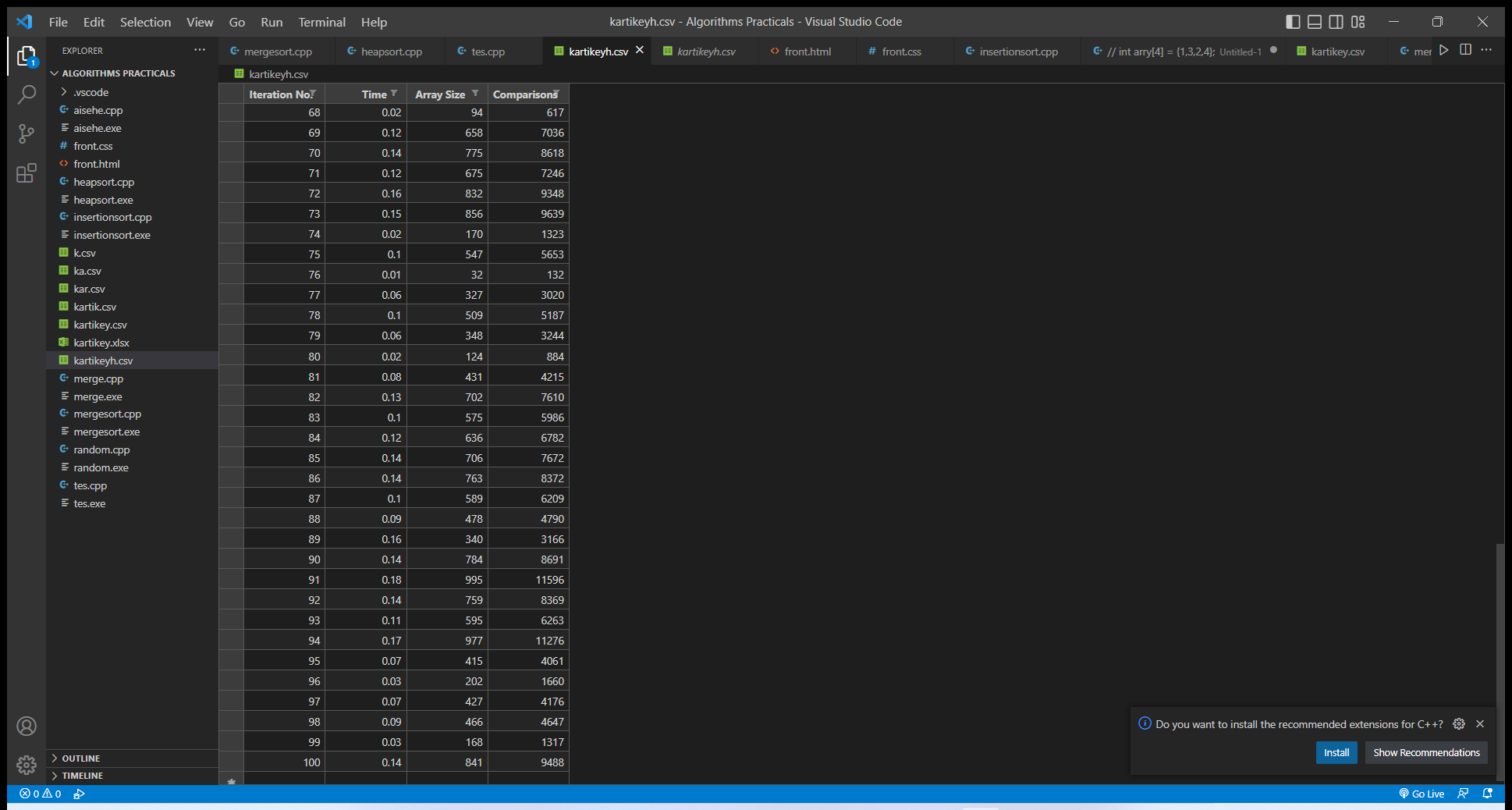
cout<<endl;

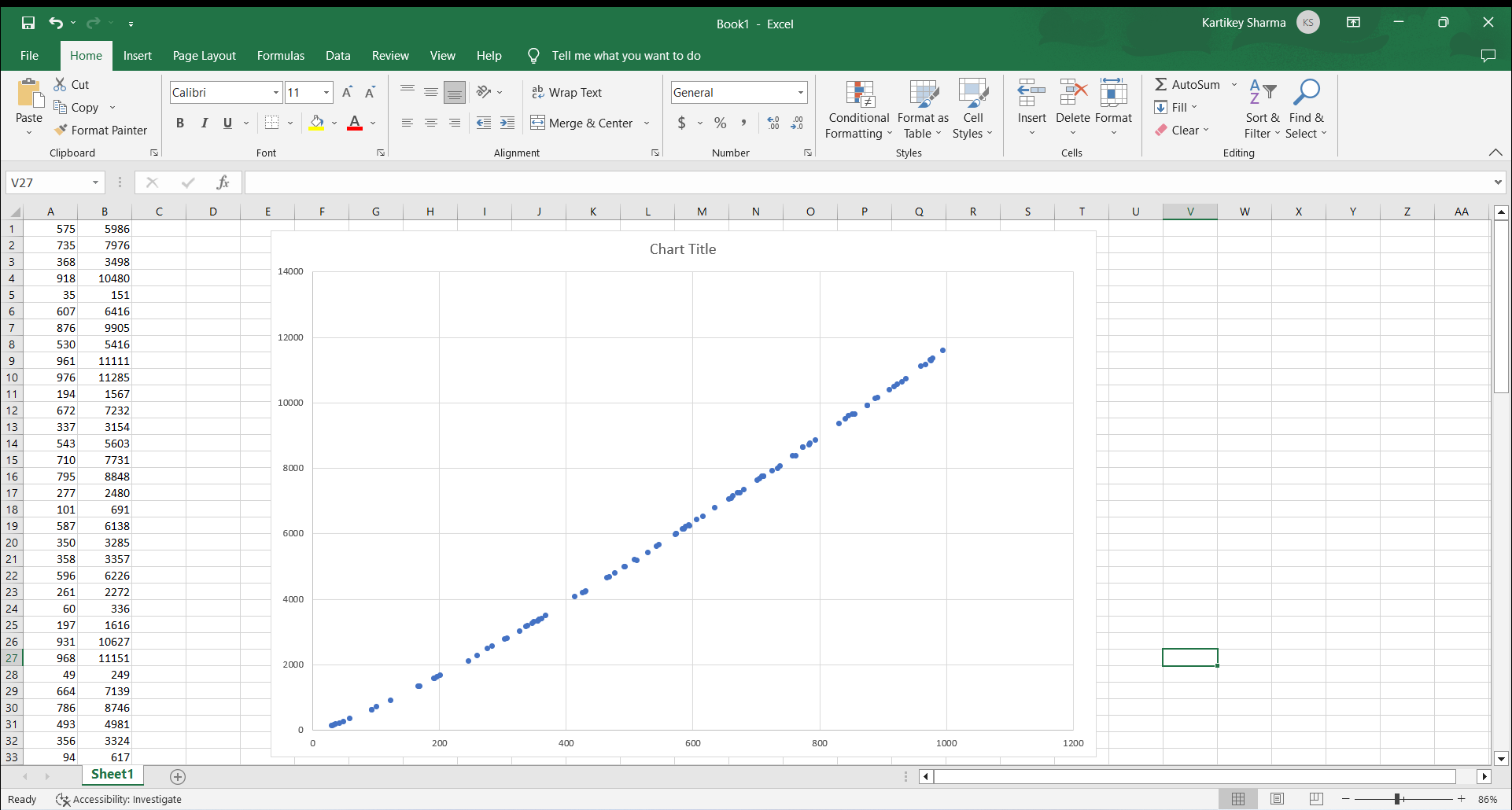
cout<<"number of inputs sorted = "<<c;

}









Q3)//implementing randomized quicksort :-

#include<iostream>

#include<bits/stdc++.h>

using namespace std;

int y=0;

int partition(int arr[],int left,int right){

int p = arr[right];

int i = left-1; //holding elements smaller then pivot

for(int j=left;j<right;j++){

if( arr[j]<=p){

i++;

y++;

int temp = arr[j];

arr[j] = arr[i];

arr[i] = temp;

}

}

i++;

int temp = p;

arr[right] = arr[i];

arr[i] = temp;

return i;

}

int partition\_r(int arr[], int left, int right)

{

// Generate a random number in between

// low .. high

srand(time(NULL));

int random = left + rand() % (right - left);

// Swap A[random] with A[high]

swap(arr[random], arr[right]);

return partition(arr, left, right);

}

void display(int arr[], int s) {

for(int i = 0; i<s; i++)

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

cout << endl;

}

void quicksort(int arr[],int left,int right){

if(left>=right){

return;

}

int pivot\_index = partition\_r(arr,left,right);

//cout<<pivot\_index<<endl;

quicksort(arr,left,pivot\_index-1);

quicksort(arr,pivot\_index+1,right);

}

void fun(int arr[],int s){

int size = s;

int u = 0;

int b = 1000;

arr[s];

//generating random numbers and string them in the arr[s]

// where s is the particular index of another 100 size array

mt19937 pr(random\_device{}());

uniform\_int\_distribution<int> dist(u,b);

for(int i= 0;i<s;i++){

arr[i] = dist(pr);

}

quicksort(arr,0,s-1);

cout << "Array after Sorting: ";

display(arr, size);

}

int main(){

int ub = 30;

int lb = 1000;

int size = 100;

int u = 0;

int arr[size] ;

//generating 100 random numbers and storing them in an array

mt19937 num(random\_device{}());

uniform\_int\_distribution<int> dist(ub, lb);

for (auto& i: arr) {

i = dist(num); }

for (auto i: arr) {

cout << i << " "; }

cout << endl;

int c = 0;

ofstream myfile;

myfile.open ("quick.csv");

myfile <<"Iteration No.,Time,Array Size ,Comparisons " <<" \n";

clock\_t time\_req;

for(int j=0;j<100;j++){

int s = arr[j];

int ar[s];

cout<<endl;

cout<<"size of array = "<<s<<endl;

c++;

time\_req = clock();

fun(ar,s);

time\_req = clock()- time\_req;

cout << "Processor time taken for iteration "<<j+1<<" : "

<< (float)time\_req/CLOCKS\_PER\_SEC << " seconds" << endl;

u = y - u;

cout<<"No. of comparisons = "<<u<<endl;

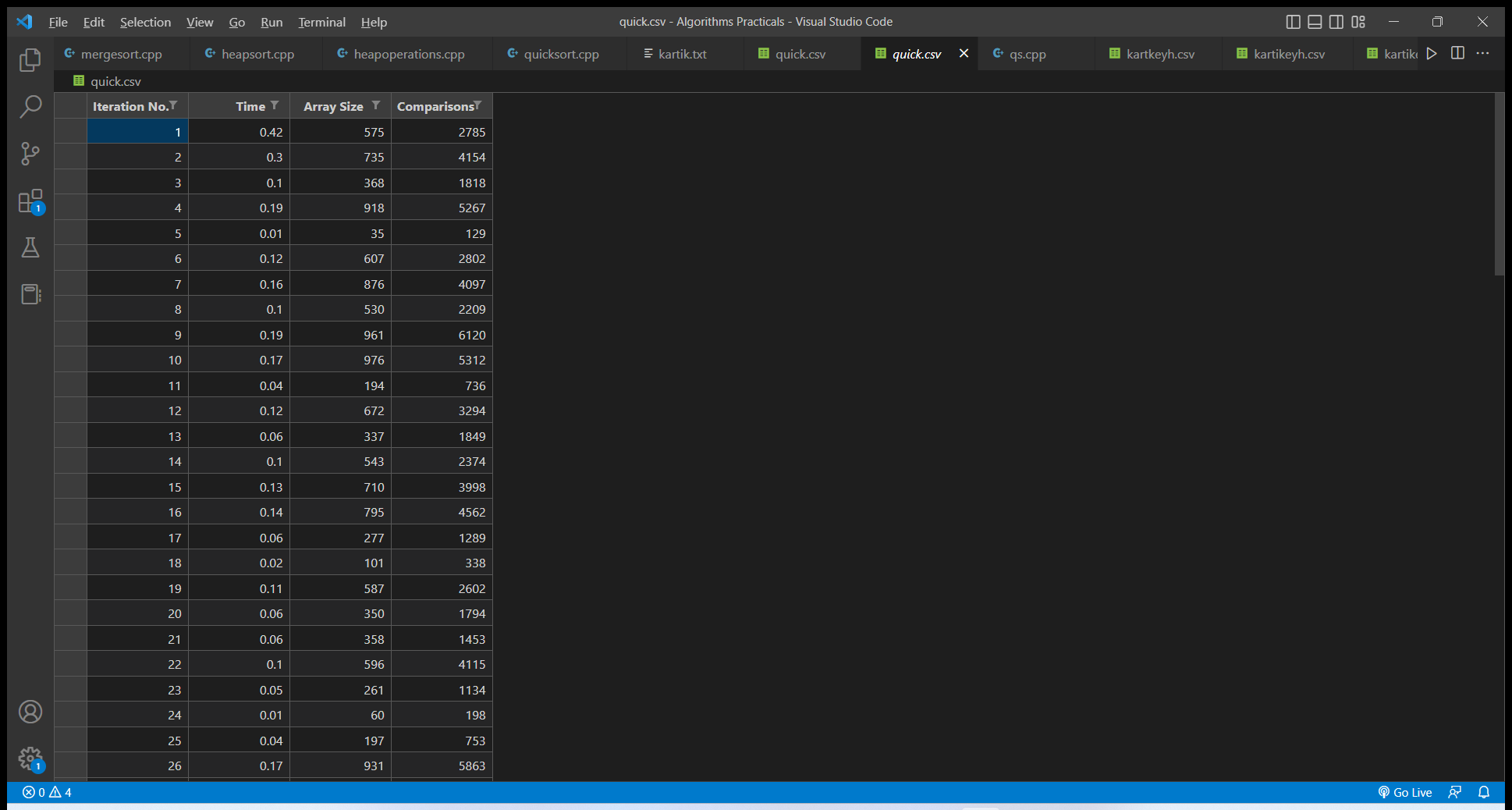
myfile <<j+1<<","<<(float)time\_req/CLOCKS\_PER\_SEC<<","<<s<<","<<u<<" \n";

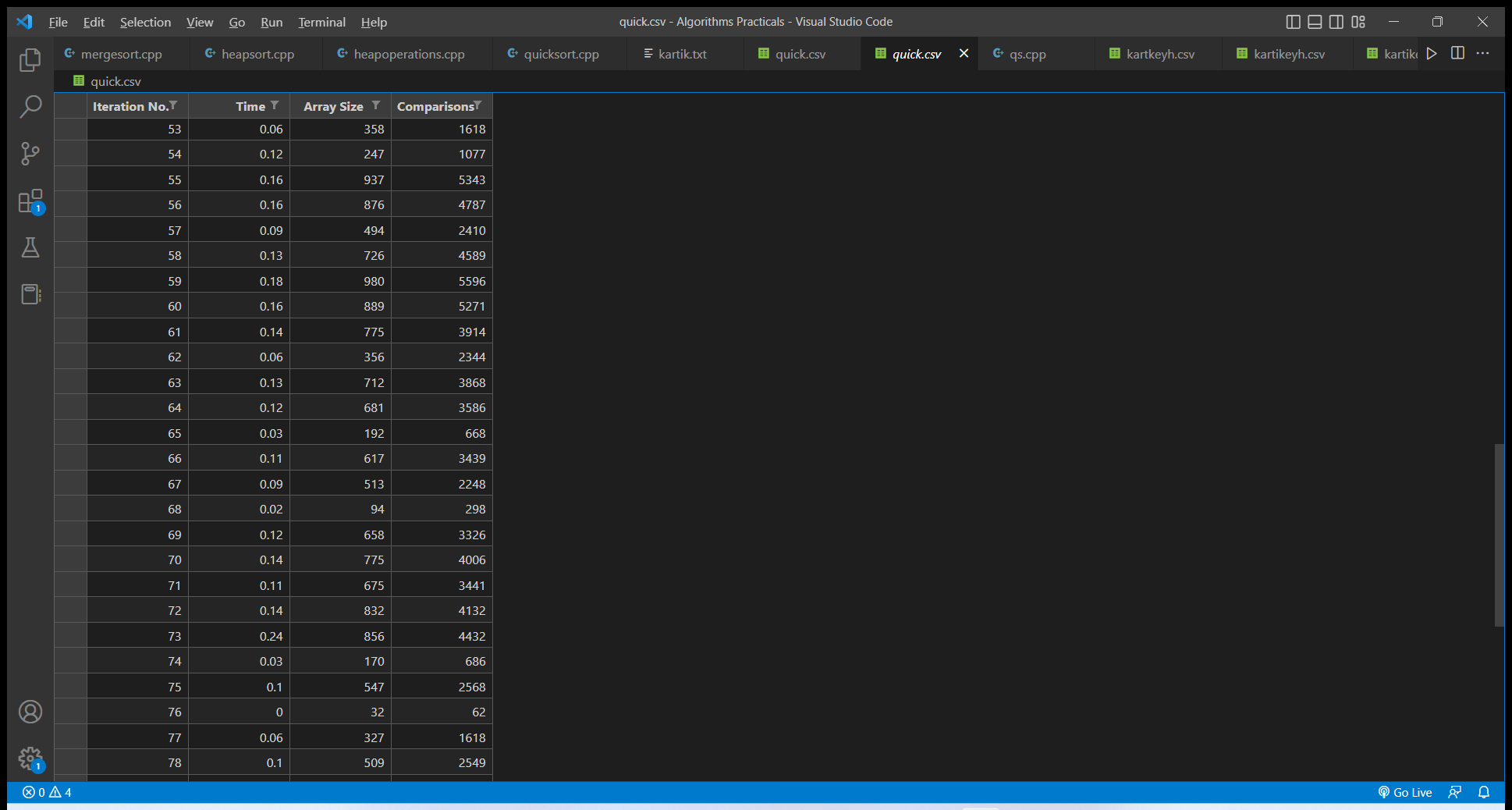
u = y; }

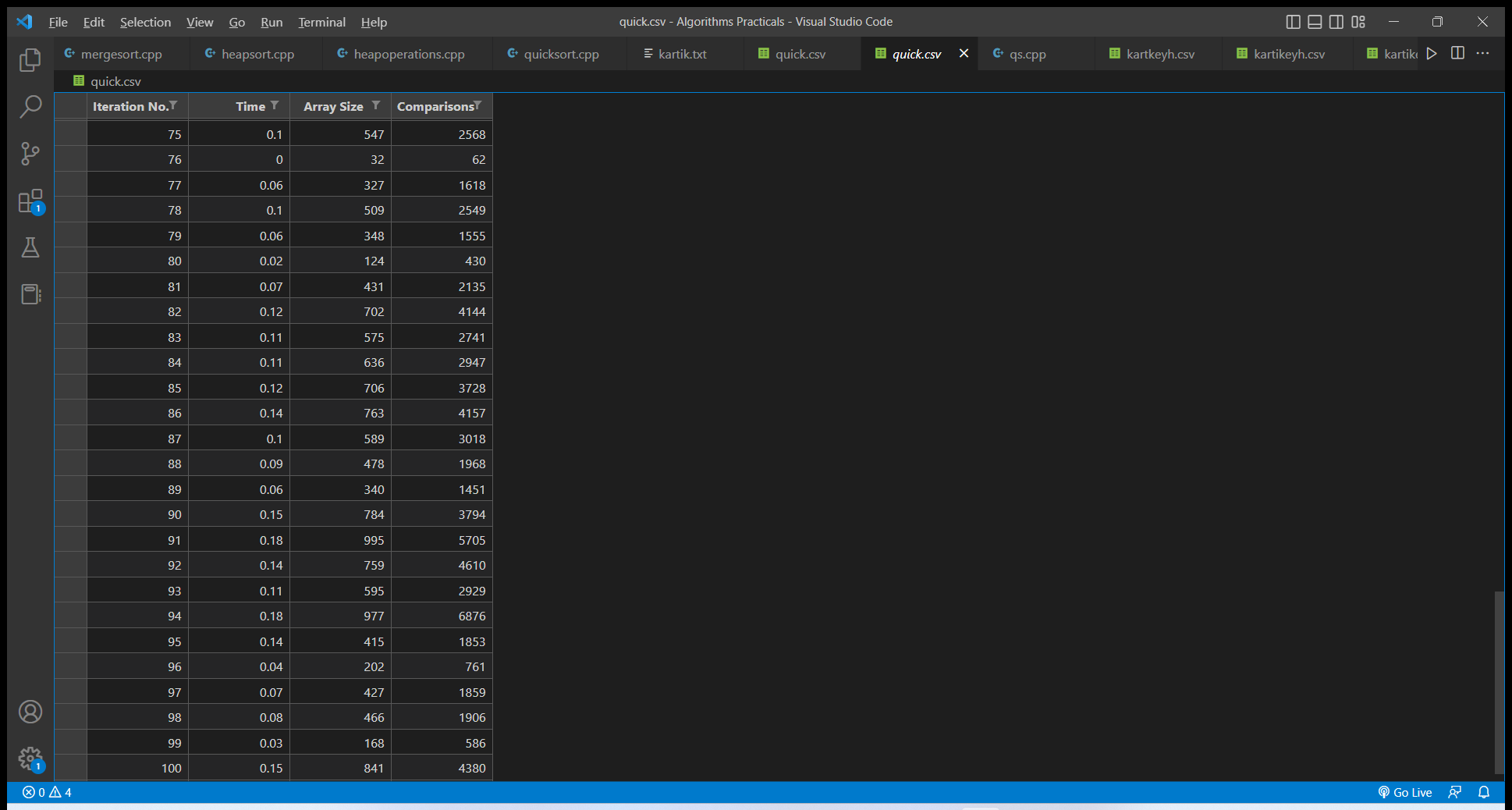
myfile.close();

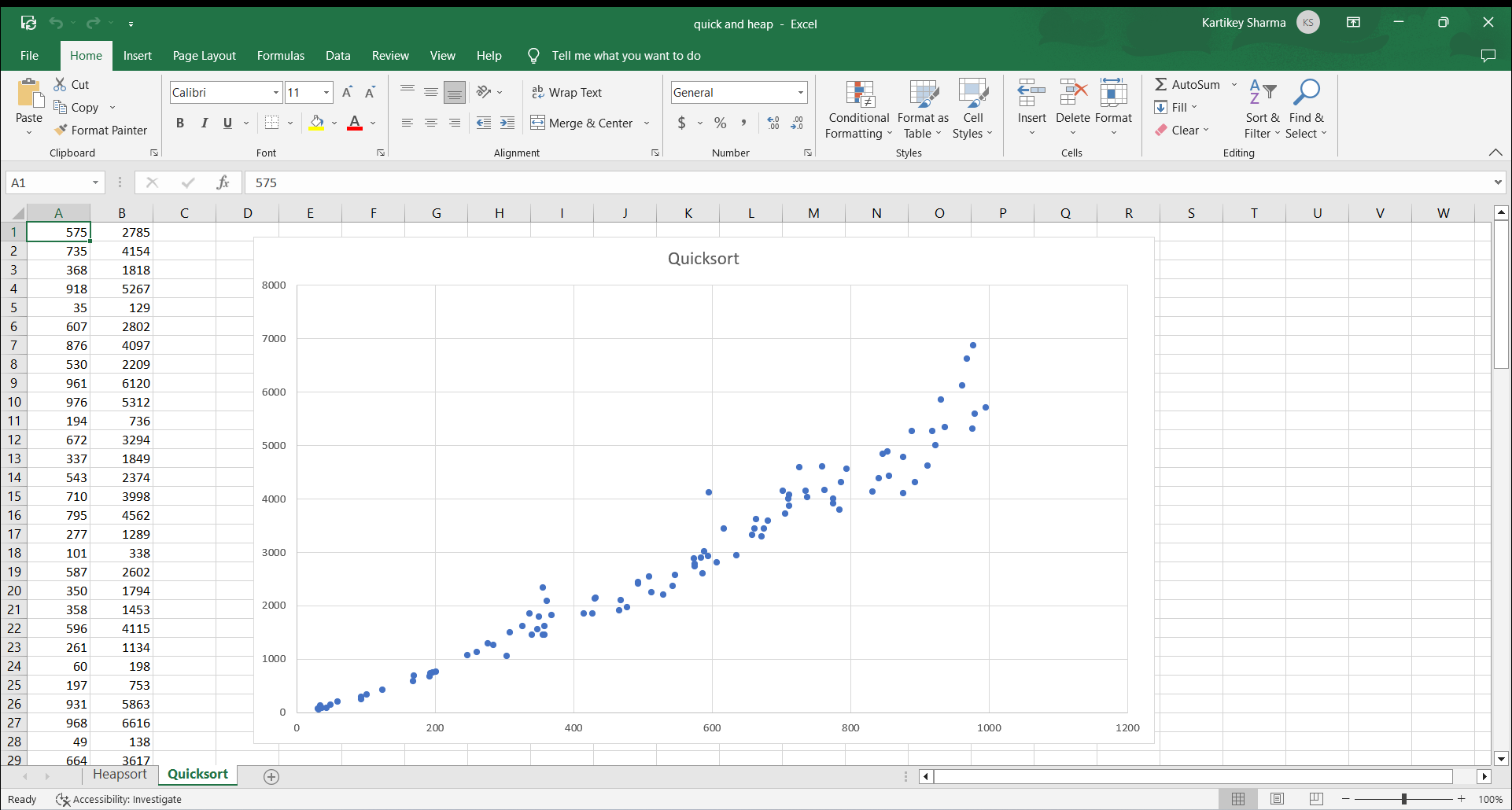
cout<<endl;

cout<<"number of inputs sorted = "<<c; }









Q4) Implement Radix Sort :-

#include<iostream>

using namespace std;

int getmax(int arr[],int n){

int m = arr[0];

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

if(arr[i]>m){

m = arr[i];

}

}

return m;

}

void countsort(int arr[],int n,int p){

int b[n];

int c[10] = {0};

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

++c[(arr[i]/p)%10];

}

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

c[i]=c[i]+c[i-1];

}

for(int i=n-1;i>=0;i--){

b[--c[(arr[i]/p)%10]] = arr[i];

}

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

arr[i]=b[i];

}

}

void radixsort(int arr[],int n){

int m = getmax(arr,n);

for(int i=1;m/i>0;i\*=10){

countsort(arr,n,i);

}

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

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

}

}

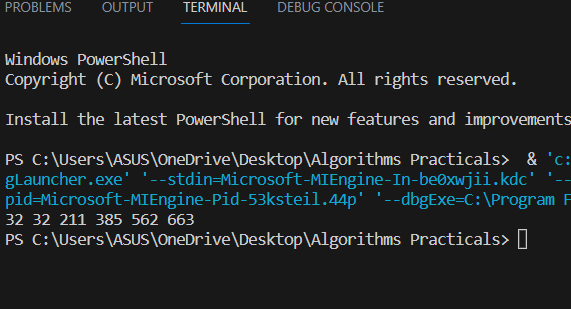
int main(){

int arr[] = {663,562,385,211,906,328};

int n = 6;

radixsort(arr,n);

}



Q5) Implement Bucket Sort :-

#include <algorithm>

#include <iostream>

#include <vector>

using namespace std;

void bucketSort(float arr[], int n)

{

vector<float> b[n];

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

int bi = n \* arr[i];

b[bi].push\_back(arr[i]);

}

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

sort(b[i].begin(), b[i].end());

int index = 0;

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

for (int j = 0; j < b[i].size(); j++)

arr[index++] = b[i][j];

}

int main()

{

float arr[]

= { 0.837, 0.565, 0.65, 0.124, 0.065, 0.344 };

int n = sizeof(arr) / sizeof(arr[0]);

bucketSort(arr, n);

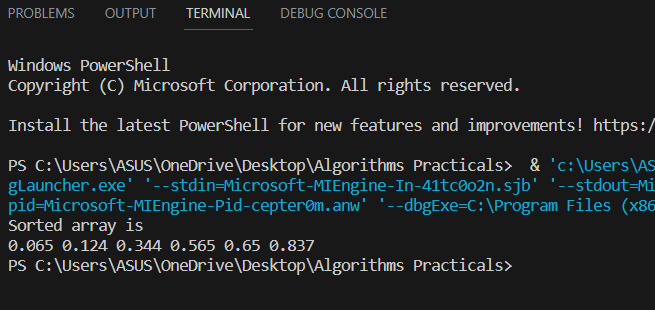
cout << "Sorted array is \n";

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

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

return 0;

}



Q6) Implement Randomized Select :-

#include<iostream>

#include<time.h>

#include<ctime>

using namespace std;

// Creating the Exchange function :

void exchange(int &a,int &b){

int temp=a;

a=b;

b=temp;

}

// function to select a random number between two given numbers :

/\*int Random(int p,int r){

srand(time(NULL));

int x= rand()%r;

return x;

}\*/

// Creating the Partition Function :

int Partiton(int A[], int p,int r){

int x=A[r];

int i=(p-1);

for(int j=p;j<=r-1;j++){

if(A[j] <=x){

i=i+1;

exchange(A[i],A[j]);

}

}

exchange(A[i+1],A[r]);

return (i+1);

}

// Creating the random partiton function :

int Randomized\_Partiton(int A[],int p,int r){

int i=r;

exchange(A[r],A[i]);

return Partiton(A,p,r);

}

// Randomized Select :

int Randomized\_Select(int A[],int p,int r,int i){

if(p==r)

return A[p];

int q=Randomized\_Partiton(A,p,r);

int k=q-p+1;

if (i==k){

return A[q];

}

else if (i<k){

return Randomized\_Select(A,p,q-1,i);

}

else{

return Randomized\_Select(A,q+1,r,i-k);

}

}

//Driver Code :

int main(){

int A[]={5,2,7,9,6,12,4,3,16,14};

cout<<"\nThe array is : ";

for(int e:A){

cout<<e<<" ";

}

cout<<endl;

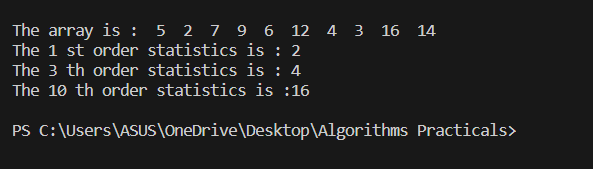
cout<<"The 1 st order statistics is : "<<Randomized\_Select(A,0,9,1)<<endl;

cout<<"The 3 th order statistics is : "<<Randomized\_Select(A,0,9,3)<<endl;

cout<<"The 10 th order statistics is :"<<Randomized\_Select(A,0,9,10)<<endl<<endl;

return 0;

}



Q7) Implement Breadth-First Search in a graph :-

#include <bits/stdc++.h>

#include <iostream>

#include<algorithm>

#include <iomanip>

using namespace std;

const int s = 100;

void bfs(queue<int>b){

while(!b.empty()){

cout<<b.front()<<" ";

b.pop();

}

}

void visited(vector<int>& v,queue<int>& b,int admatrrix[][s],int size){

int c=0;

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

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

if(admatrrix[i][j]!=0 && c==0){

v.push\_back(i);

b.push(i);

v.push\_back(j);

b.push(j);

c++;

}

else if(admatrrix[i][j]!=0){

int c=count(v.begin(),v.end(),j);

if(c==0){

v.push\_back(j);

b.push(j);

}

}

}

}

int main()

{

int size, size1, size2;

cout << "Enter the number of verteces";

cin >> size;

cout << endl;

int admattrix[s][s];

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

{

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

{

admattrix[i][j] = 0;

}

}

cout << "Enter the number of edges ";

cin >> size1;

int s1, s2, w;

int w1, u, v;

queue<int>b;

vector<int>v1;

cout << "Enter the vertex1 and vertex2";

for (int i = 1; i <= size1; i++)

{

cin >> u >> v;

admattrix[u][v] = 1;

admattrix[v][u] = 1;

}

cout<<"\n adjacency matrix \n";

cout << " ";

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

{

cout << setw(4) << " " << i;

}

cout << endl;

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

{

cout << i;

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

{

cout << setw(5) << admattrix[i][j];

}

cout << endl;

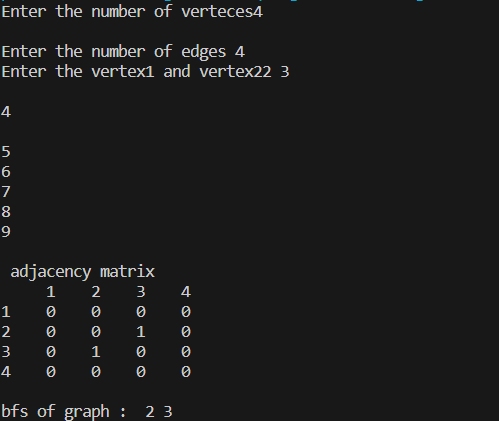
}cout<<"\nbfs of graph : ";

visited(v1,b,admattrix,size);

bfs(b);

return 0;

}



Q8) Implement Depth-First Search in a graph :-

#include <bits/stdc++.h>

#include <iostream>

#include<algorithm>

#include <iomanip>

using namespace std;

const int s = 100;

void dfs(queue<int>b){

while(!b.empty()){

cout<<b.front()<<" ";

b.pop();

}

}

void visited(vector<int>& v,queue<int>& b,stack<int>& s1,int admatrrix[][s],int size){

int c=0;

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

for (int j=1;j<=size;j++){

if(admatrrix[i][j]!=0 && c==0){

s1.push(i);

s1.push(j);

v.push\_back(i);

b.push(i);

b.push(j);

v.push\_back(j);

i=j;

j=0;

c++;

}

else if(admatrrix[i][j]!=0){

int c=count(v.begin(),v.end(),j);

if(c==0){

v.push\_back(j);

b.push(j);

s1.push(j);

i=j;

j=0;

}

}

if(j==size && !s1.empty()){

int p=s1.top();

s1.pop();

i=p;

j=0;

}

}

}

}

int main()

{

int size, size1, size2;

cout << "Enter the number of verteces";

cin >> size;

cout << endl;

int admattrix[s][s];

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

{

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

{

admattrix[i][j] = 0;

}

}

cout << "Enter the number of edges ";

cin >> size1;

int w1, u, v;

queue<int>b;

vector<int>v1;

stack<int>s1;

cout << "Enter the vertex1 and vertex2";

for (int i = 1; i <= size1; i++)

{

cin >> u >> v;

admattrix[u][v] = 1;

admattrix[v][u] = 1;

}

cout<<"\n adjacency matrix \n";

cout << " ";

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

{

cout << setw(4) << " " << i;

}

cout << endl;

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

{

cout << i;

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

{

cout << setw(5) << admattrix[i][j];

}

cout << endl;

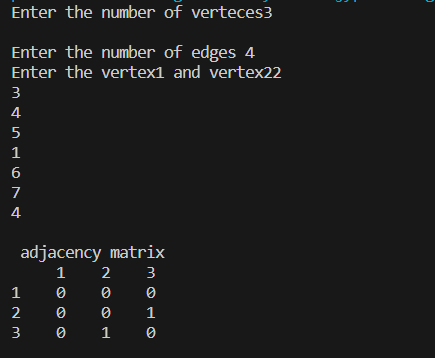
}cout<<"\nDFS of graph : ";

visited(v1,b,s1,admattrix,size);

dfs(b);

return 0;

}



Q9) Write a program to determine the minimum spanning tree of a graph using both Prims and Kruskals algorithm :-

Prims :-

#include <iostream>

using namespace std;

class Edge

{

public:

int src;

int dest;

int weight;

};

class Subset

{

public:

int p;

int rank;

};

int compEdges(const void \*a, const void \*b)

{

return ((Edge \*)(a))->weight > ((Edge \*)(b))->weight;

}

class Graph

{

public:

int V, E;

Edge \*edges;

Subset \*subsets;

Graph(int V, int E)

{

this->V = V;

this->E = E;

this->edges = new Edge[E];

for (int i = 0; i < E; i++)

{

int src, dest, weight;

cout << "Edge " << (i + 1)

<< "\n=============\n";

cout << "Source Node: ";

cin >> src;

cout << "Destination Node: ";

cin >> dest;

cout << "Edge Weight: ";

cin >> weight;

cout << endl;

if (src < 1 || src > V || dest < 1 || dest > V)

{

cout << "Invalid Node" << endl;

exit(-1);

}

this->edges[i].src = src - 1;

this->edges[i].dest = dest - 1;

this->edges[i].weight = weight;

}

}

void makeSet()

{

this->subsets = new Subset[(this->V \* sizeof(Subset))];

for (int v = 0; v < this->V; ++v)

{

this->subsets[v].p = v;

this->subsets[v].rank = 0;

}

}

int findSet(int i)

{

if (this->subsets[i].p != i)

{

this->subsets[i].p = this->findSet(this->subsets[i].p);

}

return this->subsets[i].p;

}

void link(int x, int y)

{

if (this->subsets[x].rank > this->subsets[y].rank)

{

this->subsets[y].p = x;

}

else

{

this->subsets[x].p = y;

if (this->subsets[x].rank == this->subsets[y].rank)

{

this->subsets[y].rank++;

}

}

}

void Union(int x, int y)

{

this->link(this->findSet(x), this->findSet(y));

}

void KruskalMST()

{

int e = 0, i = 0;

Edge next, result[this->V];

qsort(this->edges, this->E, sizeof(Edge), compEdges);

this->makeSet();

while (e < this->V - 1 && i < this->E)

{

next = this->edges[i++];

int x = this->findSet(next.src);

int y = this->findSet(next.dest);

if (x != y)

{

result[e++] = next;

this->Union(x, y);

}

}

qsort(result, this->V - 1, sizeof(Edge), compEdges);

cout << "Edges in Minimum Spanning Tree:"

<< "\n===============================\n";

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

{

cout << "Edge (" << (result[i].src + 1)

<< ", " << (result[i].dest + 1)

<< ") ==> " << result[i].weight

<< endl;

}

return;

}

};

int main()

{

int V, E;

cout << "Enter Number of Vertices: ";

cin >> V;

cout << "Enter Number of Edges: ";

cin >> E;

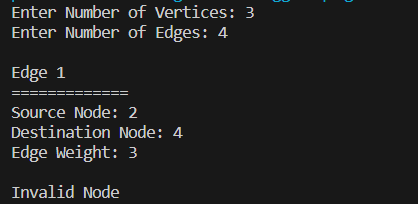
cout << endl;

Graph graph(V, E);

graph.KruskalMST();

return 0;

}



Krushkals :-

#include <algorithm>

#include <iostream>

#include <vector>

using namespace std;

#define edge pair<int, int>

class Graph {

private:

vector<pair<int, edge> > G; // graph

vector<pair<int, edge> > T; // mst

int \*parent;

int V; // number of vertices/nodes in graph

public:

Graph(int V);

void AddWeightedEdge(int u, int v, int w);

int find\_set(int i);

void union\_set(int u, int v);

void kruskal();

void print();

};

Graph::Graph(int V) {

parent = new int[V];

//i 0 1 2 3 4 5

//parent[i] 0 1 2 3 4 5

for (int i = 0; i < V; i++)

parent[i] = i;

G.clear();

T.clear();

}

void Graph::AddWeightedEdge(int u, int v, int w) {

G.push\_back(make\_pair(w, edge(u, v)));

}

int Graph::find\_set(int i) {

if (i == parent[i])

return i;

else

return find\_set(parent[i]);

}

void Graph::union\_set(int u, int v) {

parent[u] = parent[v];

}

void Graph::kruskal() {

int i, uRep, vRep;

sort(G.begin(), G.end());

for (i = 0; i < G.size(); i++) {

uRep = find\_set(G[i].second.first);

vRep = find\_set(G[i].second.second);

if (uRep != vRep) {

T.push\_back(G[i]);

union\_set(uRep, vRep);

}

}

}

void Graph::print() {

cout << "Edge :"

<< " Weight" << endl;

for (int i = 0; i < T.size(); i++) {

cout << T[i].second.first << " - " << T[i].second.second << " : "

<< T[i].first;

cout << endl;

}

}

int main() {

Graph g(6);

g.AddWeightedEdge(0, 1, 4);

g.AddWeightedEdge(0, 2, 4);

g.AddWeightedEdge(1, 2, 2);

g.AddWeightedEdge(1, 0, 4);

g.AddWeightedEdge(2, 0, 4);

g.AddWeightedEdge(2, 1, 2);

g.AddWeightedEdge(2, 3, 3);

g.AddWeightedEdge(2, 5, 2);

g.AddWeightedEdge(2, 4, 4);

g.AddWeightedEdge(3, 2, 3);

g.AddWeightedEdge(3, 4, 3);

g.AddWeightedEdge(4, 2, 4);

g.AddWeightedEdge(4, 3, 3);

g.AddWeightedEdge(5, 2, 2);

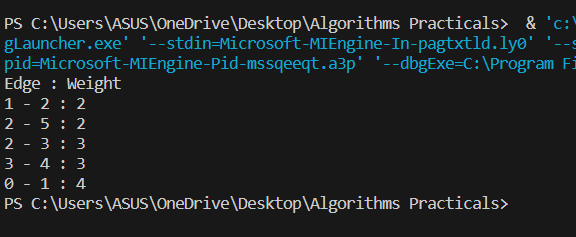
g.AddWeightedEdge(5, 4, 3);

g.kruskal();

g.print();

return 0;

}



Q10) Write a program to solve the weighted interval scheduling problem :-

#include<iostream>

using namespace std;

#define MAX 20

int M[MAX];

struct Interval

{

int startTime;

int finishTime;

int weight;

};

class WIS{

Interval I[MAX];

public:

int n;

WIS(){

for(int i=0;i<=MAX;i++)

M[i]= 0;

}

void sortIntervals();

int mComputeOpt(int);

void input();

int p(int);

};

int WIS::p(int j)

{

for(int i=j-1;i>0;i--){

if(I[i].finishTime <= I[j].startTime)

return i;

}

return 0;

}

void WIS::input(){

cout<<"enter number of intervals: ";

cin>>n;

cout<<"Enter the starting time ,finishing rime and weight value for the intervals : ";

cout<<"\n\nSi Fi Vi\n";

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

cin>>I[i].startTime;

cin>>I[i].finishTime;

cin>>I[i].weight;

}

}

void WIS::sortIntervals(){

int i,flag=1;

Interval temp;

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

flag =0 ;

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

if(I[j+1].finishTime < I[j].finishTime){

temp = I[j];

I[j]=I[j+1];

I[j+1]=temp;

flag = 1;

}

}

}

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

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

if(I[i].finishTime == I[j].finishTime && I[i].startTime>I[j].startTime)

{

temp=I[i];

I[i] = I[j];

I[j]=temp;

}

}

}

cout<<"I<i>\t\ts<i>\t\tF<i>\t\tv<i>\n";

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

cout<<"\t\t"<<i<<"\t\t"<<I[i].startTime<<"\t\t"<<I[i].finishTime<<"\t\t"<<I[i].weight<<"\n";

}

}

int WIS::mComputeOpt(int j){

if(j==0){

return 0;

}else if(M[j]){

return M[j];

}

else{

M[j]=max((I[j].weight+mComputeOpt(p(j))),mComputeOpt(j-1));

}

return M[j];

}

int main(){

WIS job;

job.input();

cout<<"\nSorted input: ";

job.sortIntervals();

cout<<endl;

for(int i=1;i<=job.n;i++)

cout<<"opt["<<i<<"]\t";

cout<<endl;

for(int i=1;i<=job.n;i++){

cout<<job.mComputeOpt(i)<<"\t";

if(i==job.n){

cout<<endl;

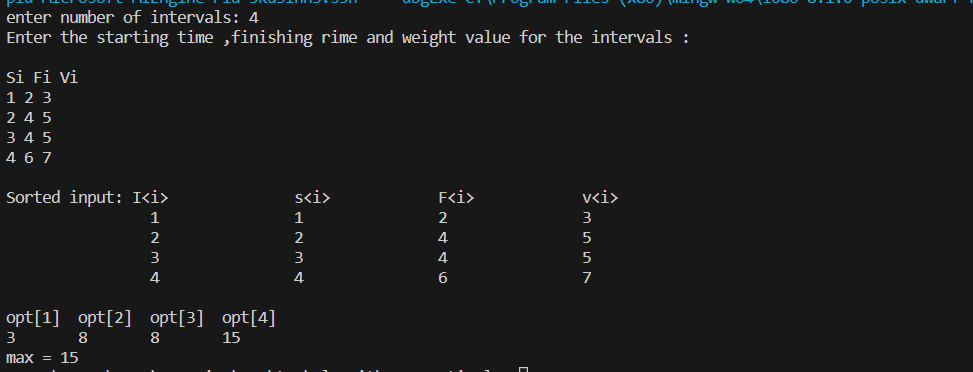
cout<<"max = "<<job.mComputeOpt(i);

}

}

return 0;

}



Q11) Write a program to solve the 0-1 knapsack problem :-

#include <bits/stdc++.h>

using namespace std;

int max(int a, int b) { return (a > b) ? a : b; }

int knapSack(int W, int wt[], int val[], int n)

{

if (n == 0 || W == 0)

return 0;

if (wt[n - 1] > W)

return knapSack(W, wt, val, n - 1);

else

return max(val[n - 1] + knapSack(W - wt[n - 1], wt, val, n - 1),knapSack(W, wt, val, n - 1));

}

int main()

{

int profit[] = { 90, 20, 40 };

int weight[] = { 10, 30, 30 };

int W = 50;

int n = sizeof(profit) / sizeof(profit[0]);

cout << knapSack(W, weight, profit, n);

return 0;

}

