**VISVESVARAYA TECHNOLOGICAL UNIVERSITY**

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

****

**LAB REPORT**

**On**

**Analysis and Design of Algorithms**

***Submitted by***

**Harshala Rani**

**(1BM21CS074)**

***In partial fulfilment 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-2023 to July-2023**

**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 **Harshala Rani(1BM21CS074),** who is bonafide student of **B.M.S. College of Engineering.** It is in partial fulfilment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum during the academic semester May-2023 to July-2023.  The Lab report has been approved as it satisfies the academic requirements in respect of an **Analysis and Design of Algorithms (22CS4PCADA)** work prescribed for the said degree.

Radhika A D                    Dr. Jyothi S Nayak

Assistant Professor                              Professor and Head

Department of CSE                 Department of CSE

BMSCE, Bengaluru                 BMSCE, Bengaluru

**Index Sheet**

|  |  |  |
| --- | --- | --- |
| **Lab Program No.** | **Program Details** | **Page No.** |
| 1 | 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 |
| 2 | Write program to obtain the Topological ordering of vertices in a given digraph. | 9 |
| 3 | Implement Johnson Trotter algorithm to generate permutations. | 11 |
| 4 | 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. | 15 |
| 5 | Sort a given set of N integer elements using Quick Sort technique and compute its time taken. | 18 |
| 6 | Sort a given set of N integer elements using Heap Sort technique and compute its time taken. | 22 |
| 7 | Implement 0/1 Knapsack problem using dynamic programming. | 24 |
| 8 | Implement All Pair Shortest paths problem using Floyd’s algorithm. | 26 |
| 9 | Find Minimum Cost Spanning Tree of a given undirected graph using Prim’s and Kruskal’s algorithm. | 28 |
| 10 | From a given vertex in a weighted connected graph, find shortest paths to other vertices using Dijkstra’s algorithm. | 32 |
| 11 | Implement “N-Queens Problem” using Backtracking. | 34 |

**Course Outcome**

|  |  |
| --- | --- |
| CO1 | Analyze time complexity of Recursive and Non-recursive algorithms using asymptotic notations. |
| CO2 | Apply various design techniques for the given problem. |
| CO3 | Apply the knowledge of complexity classes P, NP, and NP-Complete and prove certain  problems are NP-Complete |
| CO4 | Design efficient algorithms and conduct practical experiments to solve problems. |

1. **Experiments**
   1. **Experiment - 1**
      1. **Question:**

Write program to do the following:

1. Print all the nodes reachable from a given starting node in a digraph using BFS method.
2. Check whether a given graph is connected or not using DFS method.
   * 1. **Code:**
3. **BFS method:**

#include<stdio.h>

#include<conio.h>

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

void bfs(int v)

{

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

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

q[++r]=i;

if(f<=r)

{

visited[q[f]]=1;

bfs(q[f++]);

}

}

void main()

{

int v;

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

scanf("%d",&n);

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

{

q[i]=0;

visited[i]=0;

}

printf("\n Enter graph data in matrix form:\n");

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

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

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

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

scanf("%d",&v);

bfs(v);

printf("\n The node which are reachable are:\n");

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

if(visited[i])

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

getch();

}

1. **DFS method:**

#include<stdio.h>

#include<conio.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("\n%d->%d",v,i);

dfs(i);

}

}

int main()

{

int i,j,count=0;

printf("\nEnter no of vertices : ");

scanf("%d",&n);

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

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

{

reach[i]=0;

a[i][j]=0;

}

printf("\nEnter adjacency matrix : \n");

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

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

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

dfs(1);

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

if(reach[i])

count++;

if(count==n)

printf("\nGraph is connected.");

else

printf("\nGraph is disconnected.");

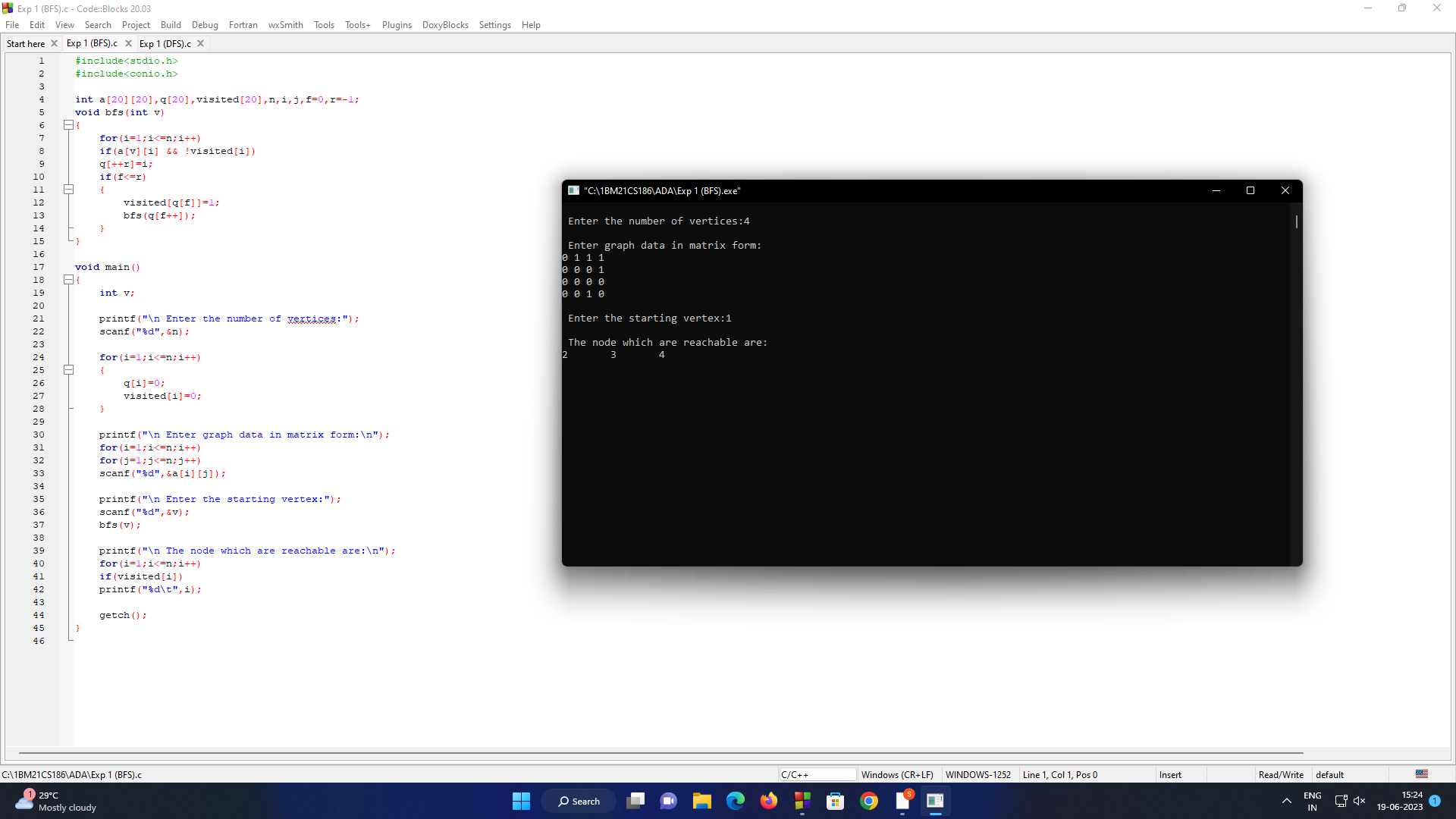
getch();

return(0);

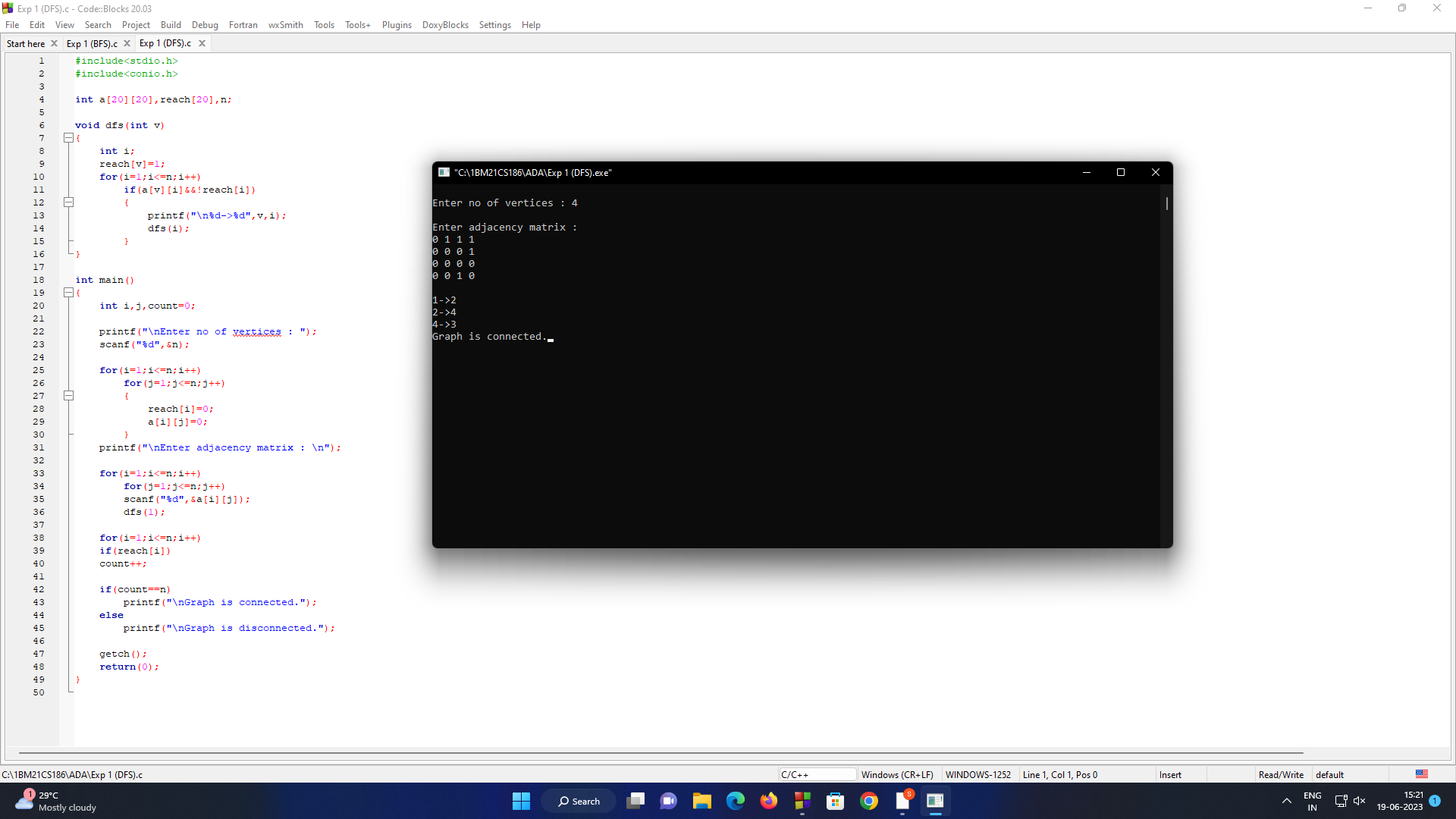
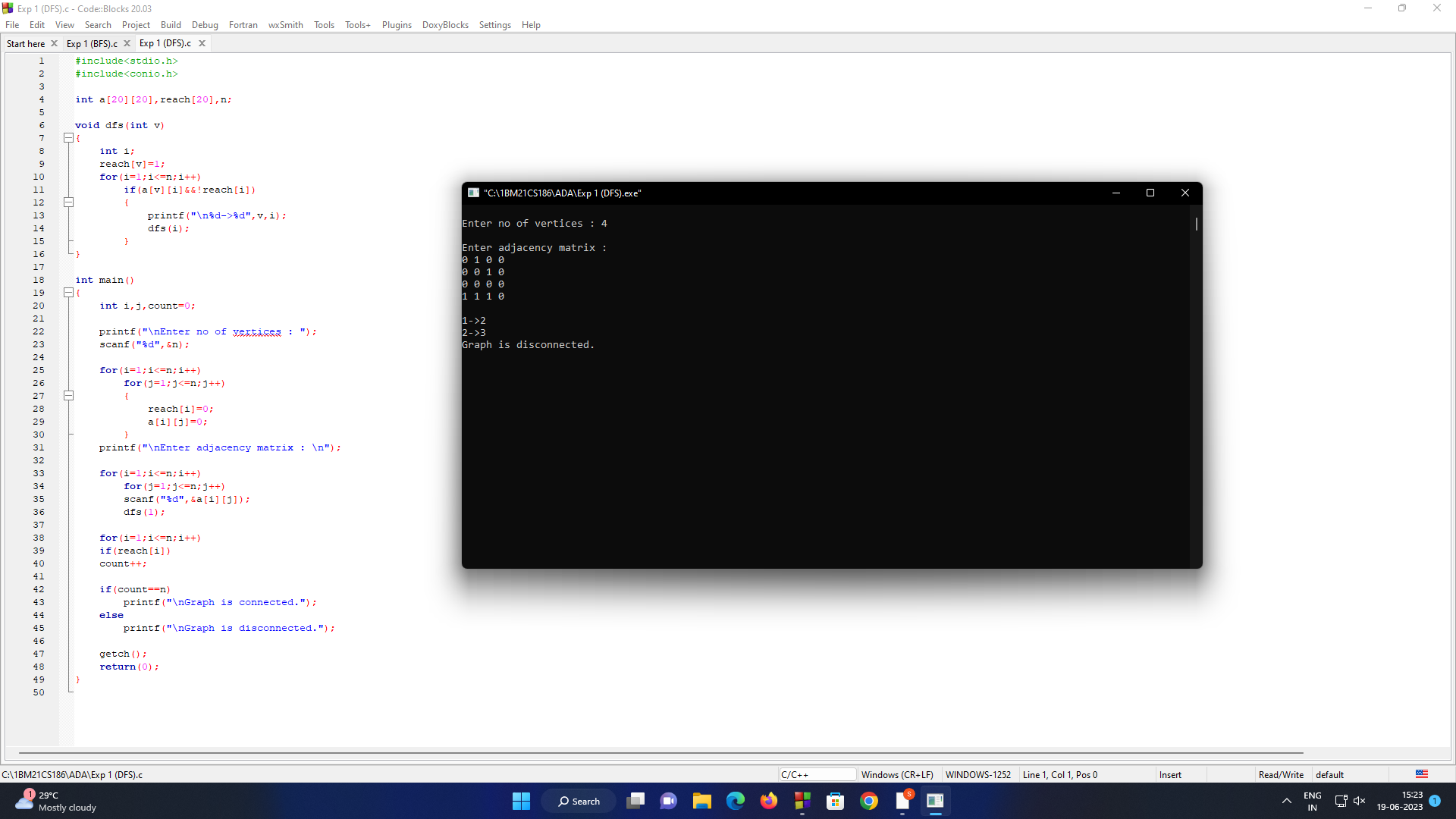
}

* + 1. **Output:**

1. **BFS method:**

****

**DFS method**

* 1. **Experiment - 2**
     1. **Question:**

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

* + 1. **Code:**

#include <stdio.h>

int main()

{

int i,j,k,n,a[10][10],indeg[10],visited[10],count=0;

printf("Enter the no of vertices:\n");

scanf("%d",&n);

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

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

{

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

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

}

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

{

indeg[i]=0;

visited[i]=0;

}

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

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

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

printf("\nThe topological order is:");

while(count<n)

{

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

{

if((indeg[k]==0) && (visited[k]==0))

{

printf("%d ",(k+1));

visited [k]=1;

}

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

{

if(a[i][k]==1)

indeg[k]--;

}

}

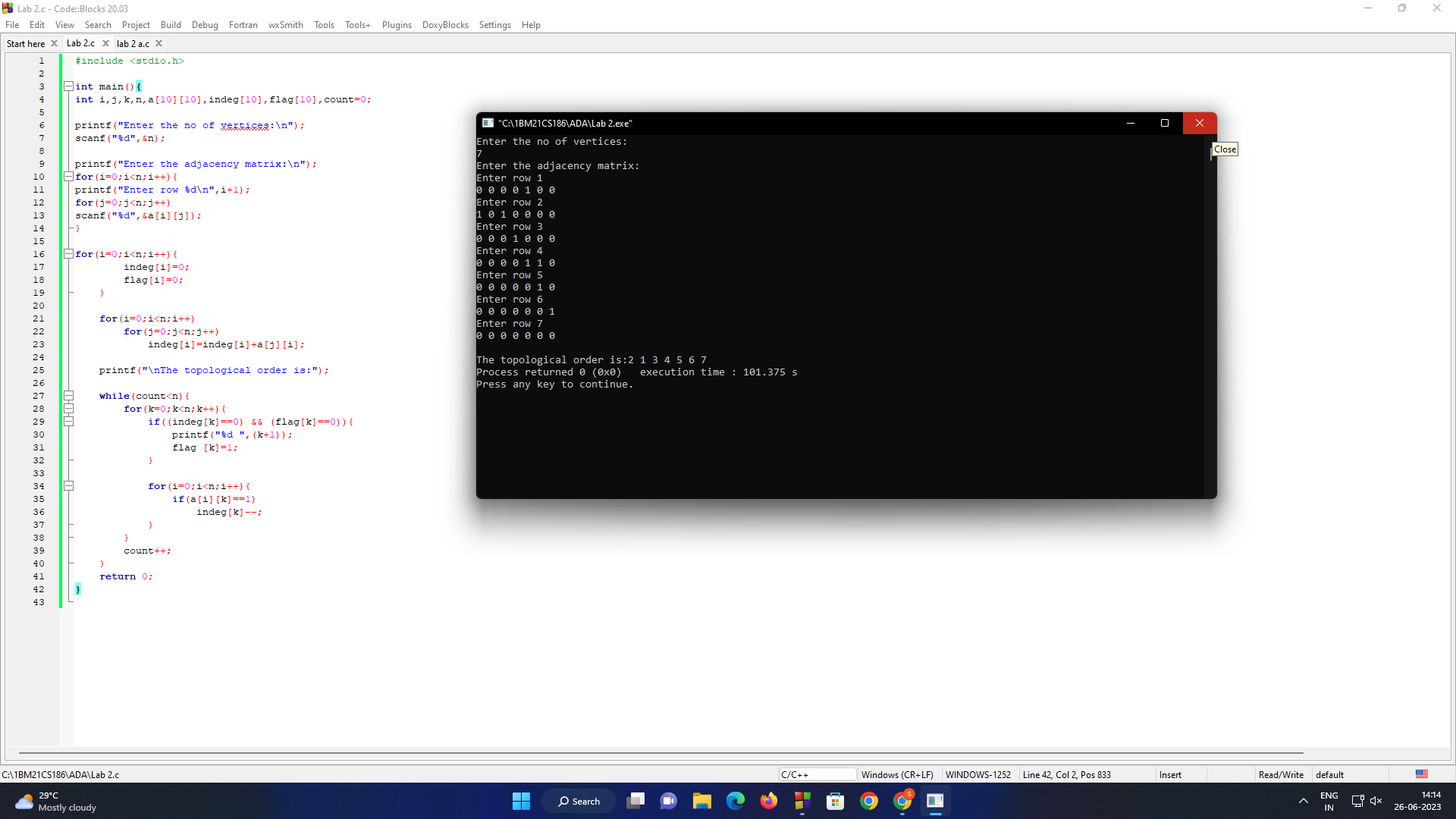
count++;

}

return 0;

}

* + 1. **Output**:



* 1. **Experiment - 3**
     1. **Question:**

Implement Johnson Trotter algorithm to generate permutations.

* + 1. **Code:**

#include <stdio.h>

#include <stdbool.h>

#define MAX\_N 10

void swap(int \*a, int \*b)

{

int temp = \*a;

\*a = \*b;

\*b = temp;

}

void printPermutation(int permutation[], int direction[], int n)

{

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

{

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

}

printf("\n");

}

void generatePermutations(int n)

{

int permutation[MAX\_N];

int direction[MAX\_N];

bool mobile[MAX\_N];

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

{

permutation[i] = i + 1;

direction[i] = -1;

mobile[i] = true;

}

printPermutation(permutation, direction, n);

int mobileElement, mobileIndex, temp;

while (true)

{

mobileElement = -1;

mobileIndex = -1;

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

{

if (direction[i] == -1 && i > 0 && permutation[i] > permutation[i - 1] && mobile[i])

{

if (mobileElement == -1 || permutation[i] > mobileElement)

{

mobileElement = permutation[i];

mobileIndex = i;

}

}

if (direction[i] == 1 && i < n - 1 && permutation[i] > permutation[i + 1] && mobile[i])

{

if (mobileElement == -1 || permutation[i] > mobileElement)

{

mobileElement = permutation[i];

mobileIndex = i;

}

}

}

if (mobileIndex == -1)

{

break;

}

if (direction[mobileIndex] == -1)

{

swap(&permutation[mobileIndex], &permutation[mobileIndex - 1]);

swap(&direction[mobileIndex], &direction[mobileIndex - 1]);

}

else

{

swap(&permutation[mobileIndex], &permutation[mobileIndex + 1]);

swap(&direction[mobileIndex], &direction[mobileIndex + 1]);

}

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

{

if (permutation[i] > mobileElement)

{

direction[i] \*= -1;

}

}

printPermutation(permutation, direction, n);

}

}

int main()

{

int n;

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

scanf("%d", &n);

if (n < 1 || n > MAX\_N)

{

printf("Invalid input!\n");

return 0;

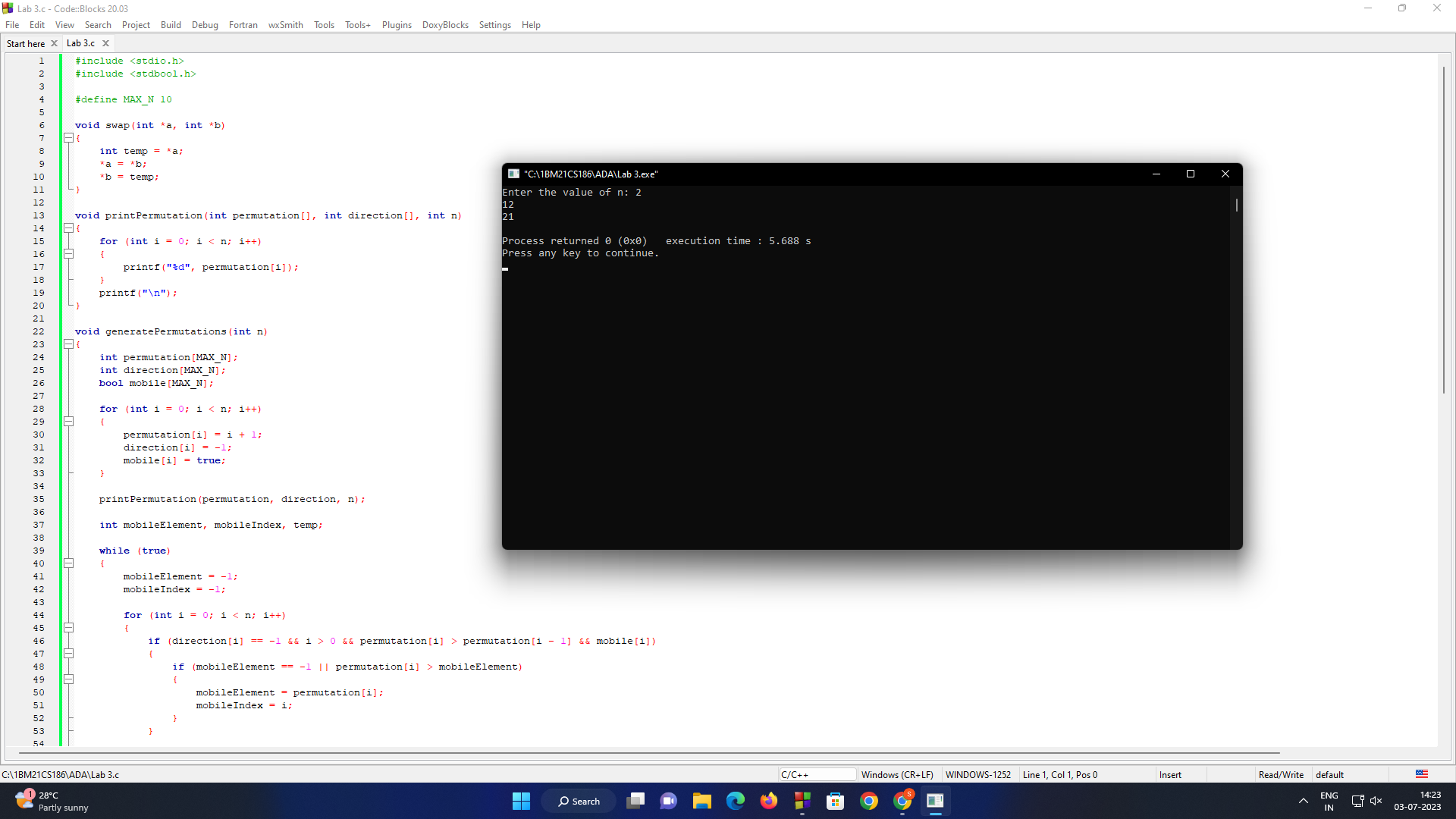
}

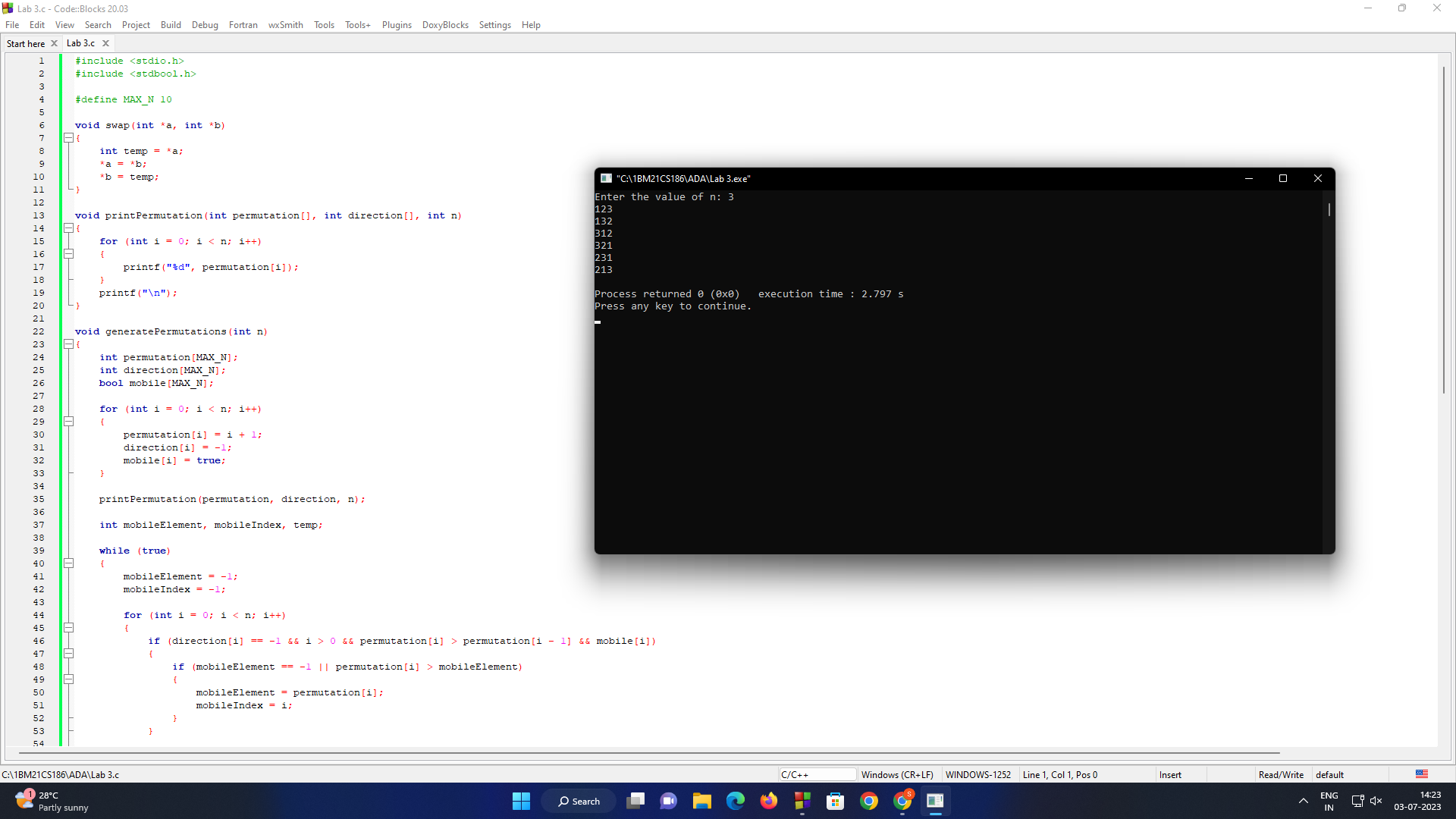
generatePermutations(n);

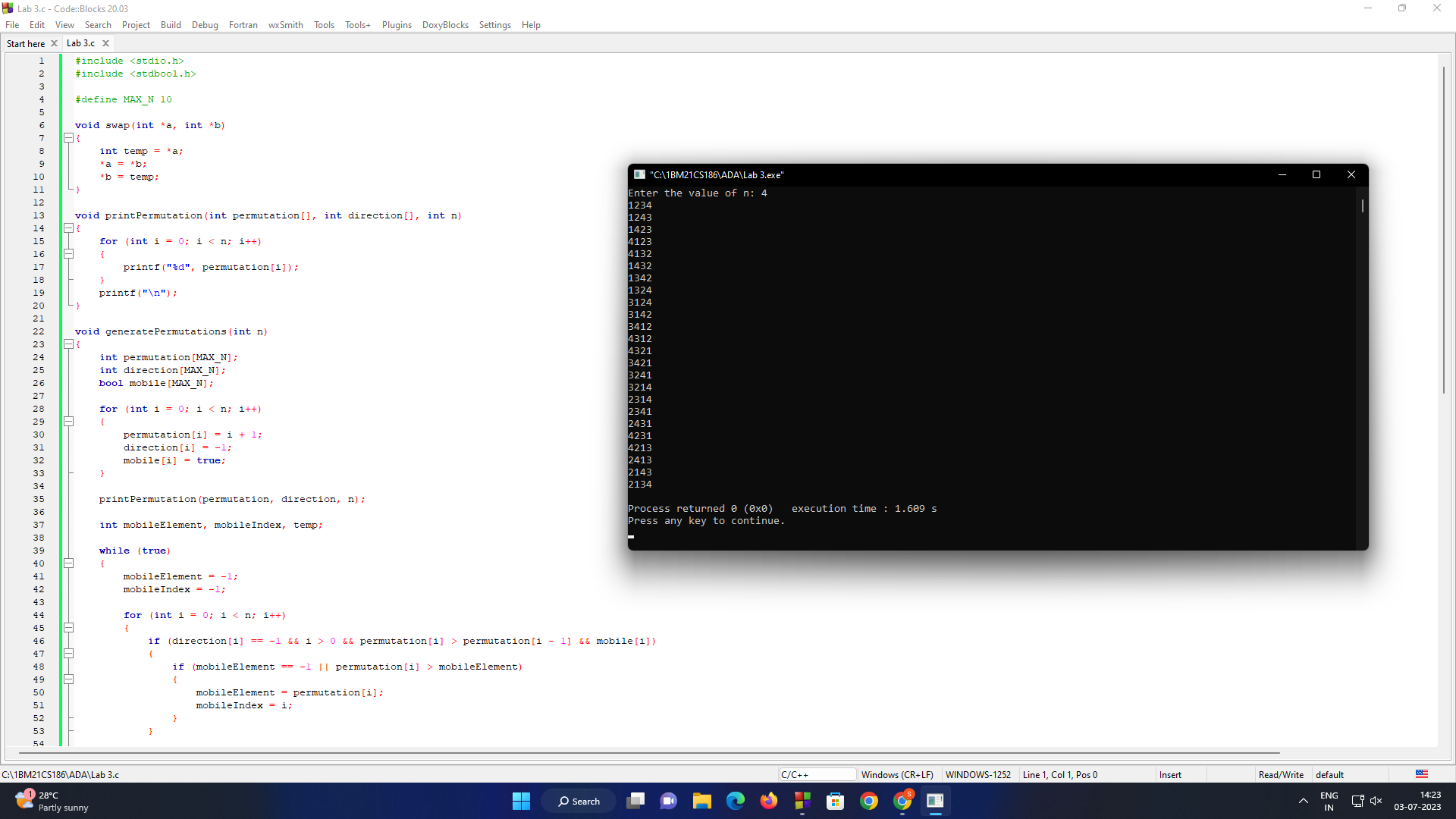
return 0;

}

* + 1. **Output:**







* 1. **Experiment - 4**
     1. **Question:**

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.

* + 1. **Code:**

#include<stdio.h>

#include<time.h>

#include<stdlib.h>

void mergesort(int a[],int i,int j);

void merge(int a[],int i1,int j1,int i2,int j2);

int main()

{

int a[50000],n,i;

clock\_t start\_t, end\_t;

double total\_t;

srand(time(NULL));

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

scanf("%d",&n);

printf("Enter array elements:");

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

a[i]=rand()%10000;

start\_t = clock();

printf("Starting of the program, start\_t = %ld\n", start\_t);

mergesort(a,0,n-1);

end\_t = clock();

printf("End of the program, end\_t = %ld\n", end\_t);

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

printf("Total time taken by CPU: %f\n", total\_t );

printf("\nSorted array is :");

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

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

return 0;

}

void mergesort(int a[],int i,int j)

{

int mid;

if(i<j)

{

mid=(i+j)/2;

mergesort(a,i,mid);

mergesort(a,mid+1,j);

merge(a,i,mid,mid+1,j);

}

}

void merge(int a[],int i1,int j1,int i2,int j2)

{

int temp[50000];

int i,j,k;

i=i1;

j=i2;

k=0;

while(i<=j1 && j<=j2)

{

if(a[i]<a[j])

temp[k++]=a[i++];

else

temp[k++]=a[j++];

}

while(i<=j1)

temp[k++]=a[i++];

while(j<=j2)

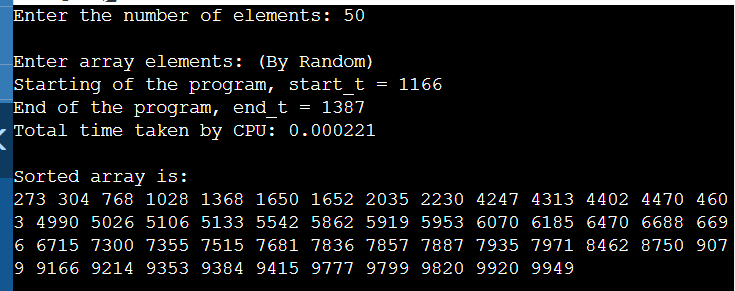
temp[k++]=a[j++];

for(i=i1,j=0;i<=j2;i++,j++)

a[i]=temp[j];

}

* + 1. **Output:**



* 1. **Experiment - 5**
     1. **Question:**

Sort a given set of N integer elements using Quick Sort technique and compute its time taken.

* + 1. **Code:**

#include<stdio.h>

#include<stdlib.h>

#include<time.h>

void swap (int \*a, int \*b)

{

int t = \*a;

\*a = \*b;

\*b = t;

}

int partition (int arr[], int low, int high)

{

int pivot = arr[high];

int i = (low - 1);

for (int j = low; j <= high - 1; j++)

{

if (arr[j] <= pivot)

{

i++;

swap (&arr[i], &arr[j]);

}

}

swap (&arr[i + 1], &arr[high]);

return (i + 1);

}

void quickSort (int arr[], int low, int high)

{

if (low < high)

{

int pi = partition (arr, low, high);

quickSort (arr, low, pi - 1);

quickSort (arr, pi + 1, high);

}

}

int main ()

{

int n;

clock\_t start\_t, end\_t;

double total\_t;

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

scanf("%d", &n);

int arr[n];

printf("Enter the maximum value of the elements: ");

int max;

scanf("%d", &max);

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

{

arr[i] = rand() % max;

}

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

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

{

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

}

start\_t = clock();

printf("\n\nStarting of the program: %ld\n", start\_t);

quickSort (arr, 0, n - 1);

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

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

{

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

}

end\_t = clock();

printf("\n\nEnd of the program: %ld\n", end\_t);

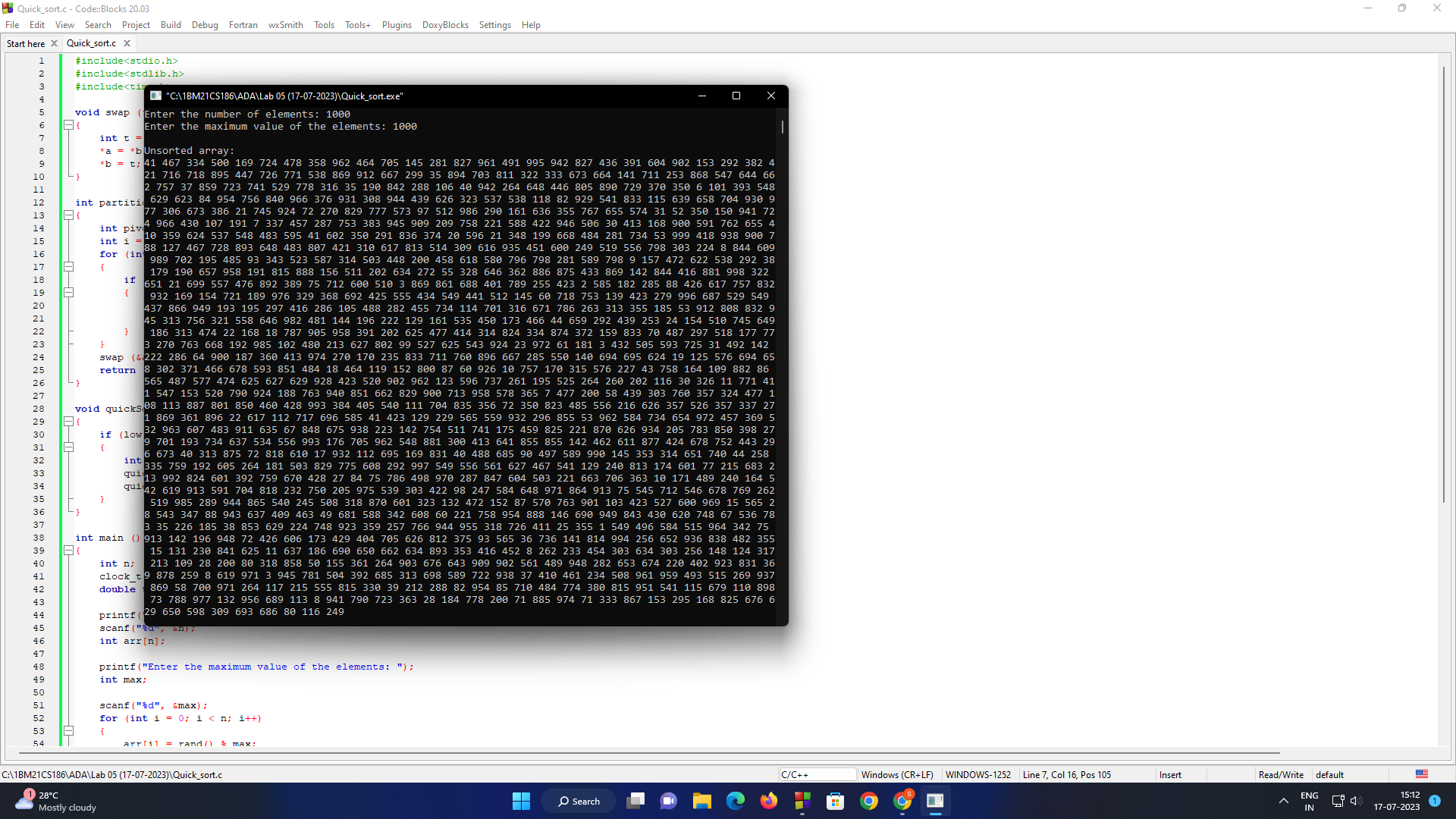
total\_t = (double)(end\_t - start\_t)/CLOCKS\_PER\_SEC;

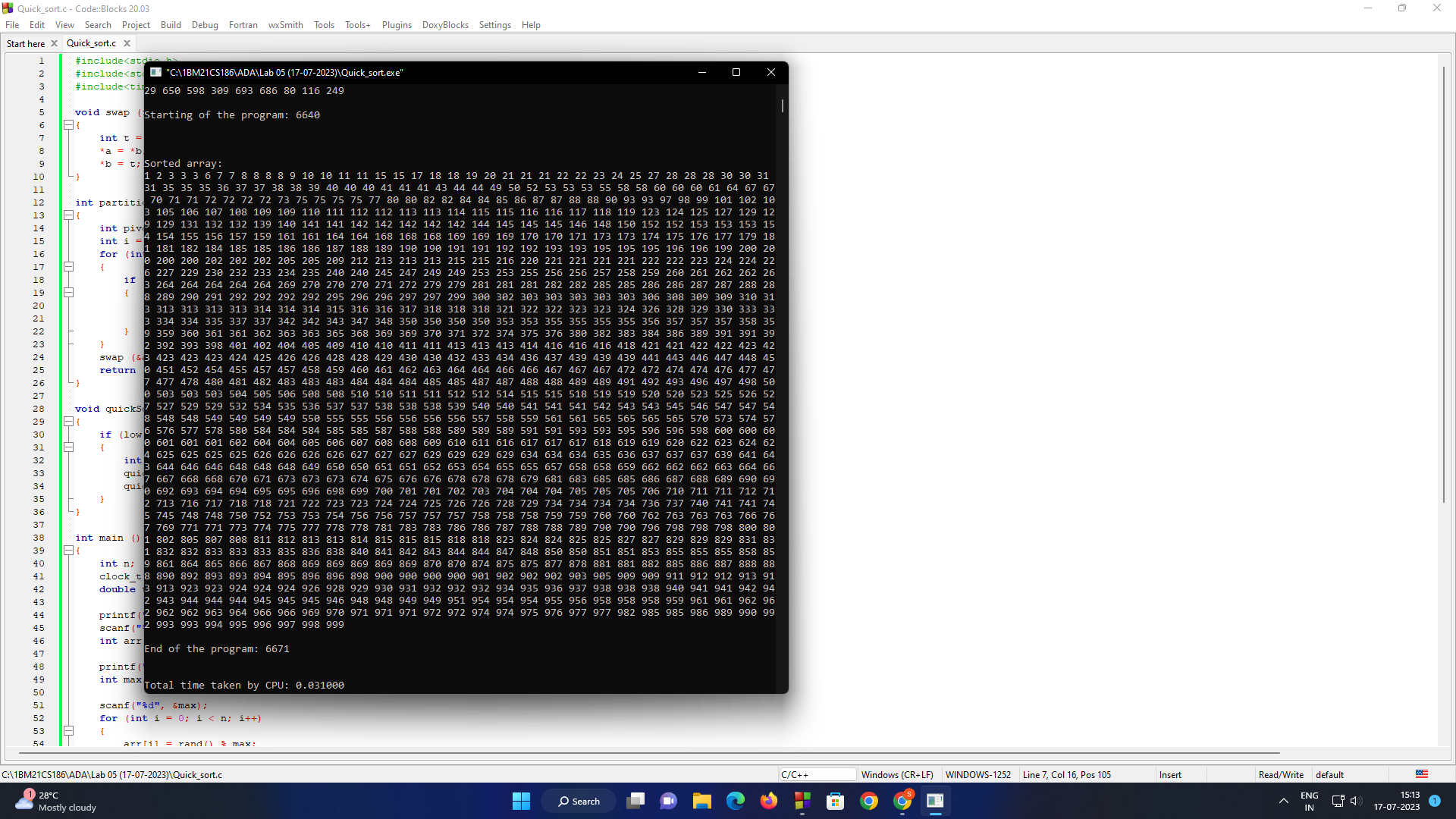
printf("\n\nTotal time taken by CPU: %f\n", total\_t);

return 0;

}

* + 1. **Output:**





* 1. **Experiment - 6**
     1. **Question:**

Sort a given set of N integer elements using Heap Sort technique and compute its time taken.

* + 1. **Code:**

#include <stdio.h>

#include <time.h>

#include <stdlib.h>

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

int largest = i;

int left = 2 \* i + 1;

int right = 2 \* i + 2;

if (left < n && arr[left] > arr[largest])

largest = left;

if (right < n && arr[right] > arr[largest])

largest = right;

if (largest != i) {

int temp = arr[i];

arr[i] = arr[largest];

arr[largest] = temp;

heapify(arr, n, largest);

}

}

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

for (int i = n / 2 - 1; i >= 0; i--)

heapify(arr, n, i);

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

int temp = arr[0];

arr[0] = arr[i];

arr[i] = temp;

heapify(arr, i, 0);

}

}

int main() {

int n;

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

scanf("%d", &n);

int arr[n];

// Generate and fill array with random numbers

srand(time(NULL)); // Seed the random number generator

printf("\n\nRandomly generated elements:\n");

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

arr[i] = rand() % 1000; // Generate random numbers between 0 and 999

printf("Original array: \n");

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

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

clock\_t start\_time = clock(); // Record the start time

heapSort(arr, n);

clock\_t end\_time = clock(); // Record the end time

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

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

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

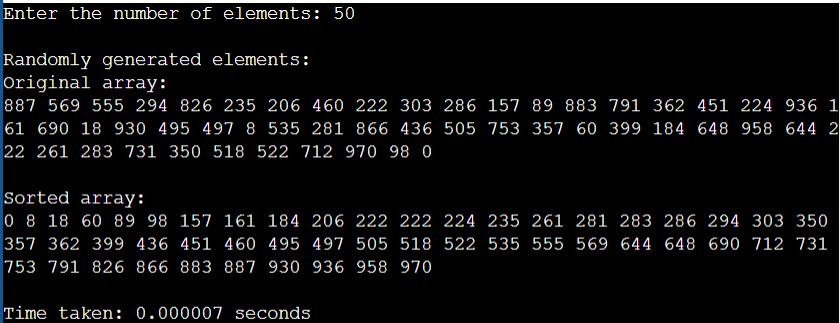
double time\_taken = (double)(end\_time - start\_time) / CLOCKS\_PER\_SEC;

printf("\n\nTime taken: %f seconds\n", time\_taken);

return 0;

}

* + 1. **Output:**



* 1. **Experiment - 7**
     1. **Question:**

Implement 0/1 Knapsack problem using dynamic programming.

* + 1. **Code:**

#include <stdio.h>

int max(int a, int b) {

return (a > b) ? a : b;

}

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

{

int i, w;

int K[n + 1][W + 1];

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

{

for (w = 0; w <= W; w++)

{

if (i == 0 || w == 0)

K[i][w] = 0;

else if (wt[i - 1] <= w)

K[i][w] = max(val[i - 1] + K[i - 1][w - wt[i - 1]], K[i - 1][w]);

else

K[i][w] = K[i - 1][w];

}

}

int res = K[n][W];

w = W;

for (i = n; i > 0 && res > 0; i--)

{

if (res == K[i - 1][w])

continue;

else {

selected[i - 1] = 1;

res = res - val[i - 1];

w = w - wt[i - 1];

}

}

return K[n][W];

}

int main()

{

int n, W, i;

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

scanf("%d", &n);

int val[n], wt[n], selected[n];

printf("Enter the values of the items: ");

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

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

printf("Enter the weights of the items: ");

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

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

printf("Enter the capacity of the knapsack: ");

scanf("%d", &W);

int max\_profit = knapSack(W, wt, val, n, selected);

printf("The maximum profit is %d\n", max\_profit);

printf("The objects selected for the optimal solution are: ");

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

if (selected[i]==1)

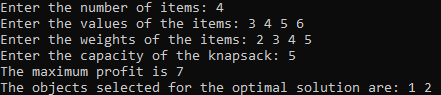
printf("%d ", i + 1);

}

return 0;

}

* + 1. **Output:**



* 1. **Experiment - 8**
     1. **Question:**

Implement All Pair Shortest paths problem using Floyd’s algorithm.

* + 1. **Code:**

#include <stdio.h>

#define INFINITY 999

int nV;

void printMatrix(int matrix[][nV]);

void floyd(int graph[][nV])

{

int matrix[nV][nV], i, j, k;

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

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

matrix[i][j] = graph[i][j];

for (k = 0; k < nV; k++)

{

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

{

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

{

if (matrix[i][k] + matrix[k][j] < matrix[i][j])

matrix[i][j] = matrix[i][k] + matrix[k][j];

}

}

}

printMatrix(matrix);

}

void printMatrix(int matrix[][nV])

{

printf("\nAll Pairs Shortest Path is :\n");

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

{

for (int j = 0; j < nV; j++)

{

if (matrix[i][j] == INFINITY)

printf("%4s", "INF");

else

printf("%4d", matrix[i][j]);

}

printf("\n");

}

}

int main()

{

printf("Enter the number of vertices in the graph: ");

scanf("%d", &nV);

int graph[nV][nV];

printf("Enter the weight of edges in the graph:\n");

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

{

for (int j = 0; j < nV; j++)

{

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

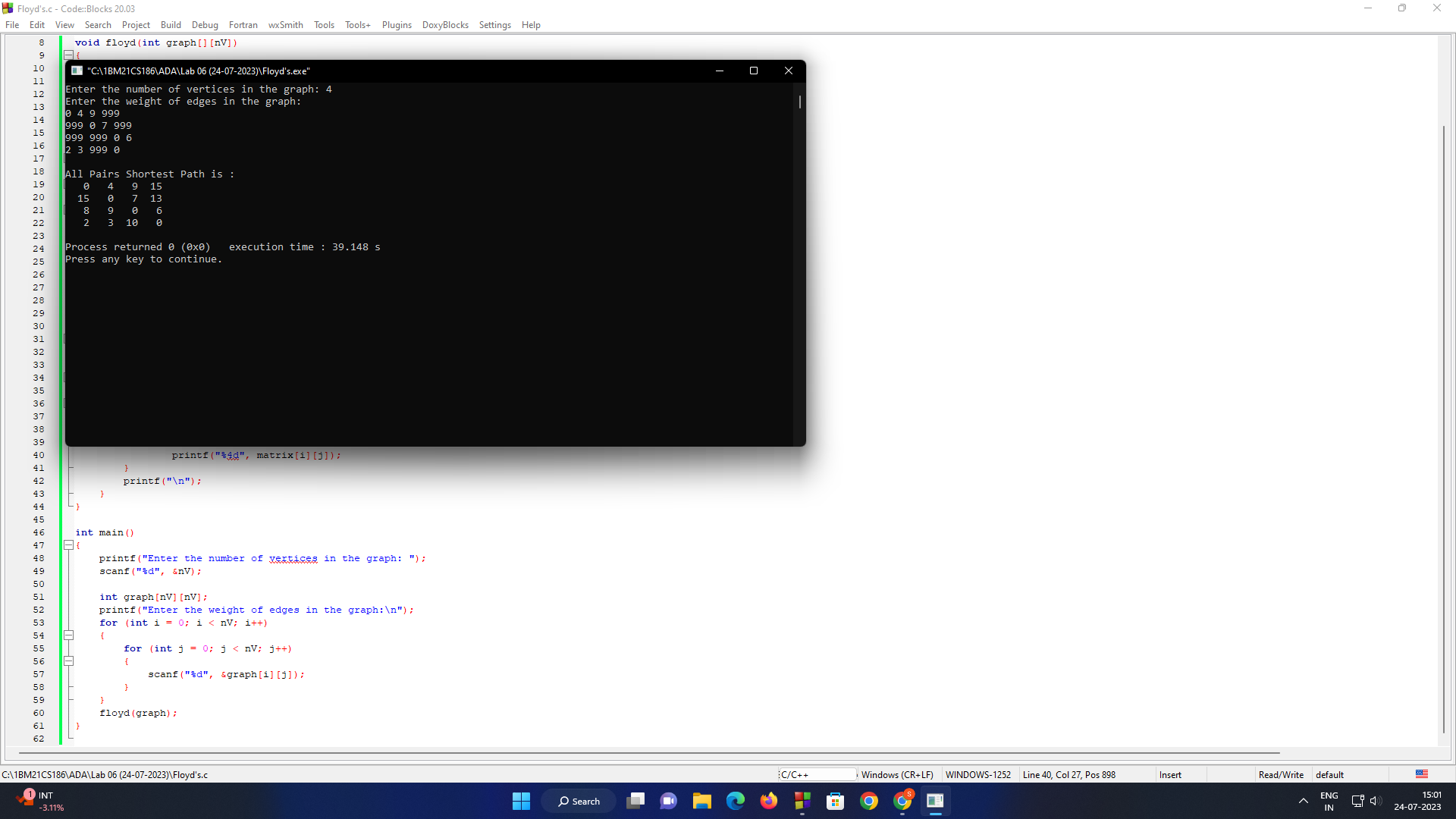
}

}

floyd(graph);

}

* + 1. **Output:**



* 1. **Experiment - 9**
     1. **Question:**

Find Minimum Cost Spanning Tree of a given undirected graph using Prim’s and Kruskal’s algorithm.

* + 1. **Code:**

1. **Prim’s Algorithm:**

#include<stdio.h>

int a,b,u,v,n,i,j,ne=1;

int visited[10]={0},min,mincost=0,cost[10][10];

void main()

{

printf("Prim's algorithm:\n");

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

scanf("%d",&n);

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

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

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

{

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

if(cost[i][j]==0)

cost[i][j]=999;

}

visited[1]=1;

printf("\n");

printf("\nThe edges of Minimum Cost Spanning Tree are:");

while(ne < n)

{

for(i=1,min=999;i<=n;i++)

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

if(cost[i][j]< min)

if(visited[i]!=0)

{

min=cost[i][j];

a=u=i;

b=v=j;

}

if(visited[u]==0 || visited[v]==0)

{

printf("\nEdge %d:(%d,%d) Weight:%d",ne++,a,b,min);

mincost+=min;

visited[b]=1;

}

cost[a][b]=cost[b][a]=999;

}

printf("\n\nMinimun Cost=%d",mincost);

}

1. **Kruskal’s Algorithm:**

#include <stdio.h>

#include <stdlib.h>

int i, j, k, a, b, u, v, n, ne = 1;

int min, mincost = 0, cost[10][10], parent[9];

int find(int);

int uni(int, int);

void main()

{

printf("Kruskal's algorithm:\n");

printf("Enter the no. of vertices:\n");

scanf("%d", &n);

printf("\nEnter the cost adjacency matrix:\n");

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

{

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

{

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

if (cost[i][j] == 0)

cost[i][j] = 999;

}

}

printf("\nThe edges of Minimum Cost Spanning Tree are\n");

while (ne < n)

{

for (i = 1, min = 999; i <= n; i++)

{

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

{

if (cost[i][j] < min)

{

min = cost[i][j];

a = u = i;

b = v = j;

}

}

}

u = find(u);

v = find(v);

if (uni(u, v))

{

printf("Edge %d:(%d,%d) Weight:%d\n", ne++, a, b, min);

mincost += min;

}

cost[a][b] = cost[b][a] = 999;

}

printf("\nMinimum cost = %d\n", mincost);

}

int find(int i)

{

while (parent[i])

i = parent[i];

return i;

}

int uni(int i, int j)

{

if (i != j)

{

parent[j] = i;

return 1;

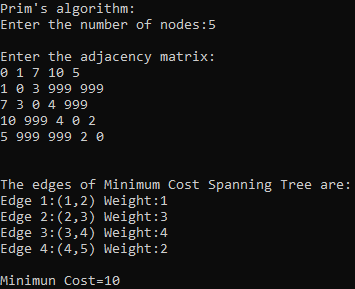
}

return 0;

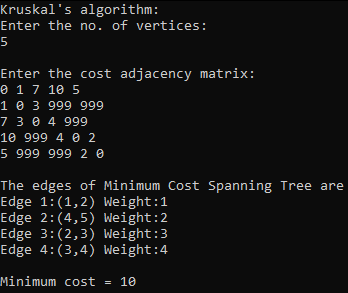
}

* + 1. **Output:**

1. **Prim’s Algorithm:**



1. **Kruskal’s Algorithm:**



* 1. **Experiment - 10**
     1. **Question:**

From a given vertex in a weighted connected graph, find shortest paths to other vertices using Dijkstra’s algorithm.

* + 1. **Code:**

#include<stdio.h>

#include<conio.h>

void dijkstra(int n,int cost[10][10],int src)

{

int i,j,u,dis[10],vis[10],min;

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

{

dis[i]=cost[src][i];

vis[i]=0;

}

vis[src]=1;

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

{

min=999;

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

{

if(vis[j]==0 && dis[j]<min)

{

min=dis[j];

u=j;

}

}

vis[u]=1;

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

{

if(vis[j]==0 && dis[u]+cost[u][j]<dis[j])

{

dis[j]=dis[u]+cost[u][j];

}

}

}

printf("Shortest Path:\n");

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

printf("%d->%d=%d\n",src,i,dis[i]);

}

void main()

{

int src,j,cost[10][10],n,i;

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

scanf("%d",&n);

printf("\nEnter the cost adjacency matrix: \n");

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

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

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

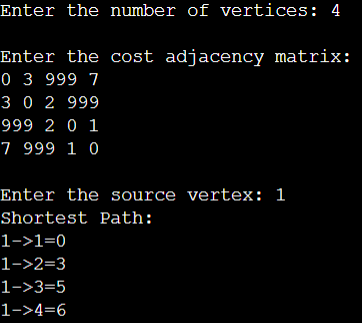
printf("\nEnter the source vertex: ");

scanf("%d",&src);

dijkstra(n,cost,src);

}

* + 1. **Output:**



* 1. **Experiment - 11**
     1. **Question:**

Implement “N-Queens Problem” using Backtracking.

* + 1. **Code:**

#include<stdio.h>

#include<stdlib.h>

#include<math.h>

int board[20],count;

int main()

{

int n,i,j;

void queen(int row,int n);

printf("N Queens Problem Using Backtracking:");

printf("\n\nEnter number of Queens:");

scanf("%d",&n);

queen(1,n);

return 0;

}

void print(int n)

{

int i,j;

printf("\n\nSolution %d:\n\n",++count);

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

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

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

{

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

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

{

if(board[i]==j)

printf("\tQ");

else

printf("\t-");

}

}

}

int place(int row,int column)

{

int i;

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

{

if(board[i]==column)

return 0;

else

if(abs(board[i]-column)==abs(i-row))

return 0;

}

return 1;

}

void queen(int row,int n)

{

int column;

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

{

if(place(row,column))

{

board[row]=column;

if(row==n)

print(n);

else

queen(row+1,n);

}

}

}

* + 1. **Output:**

