VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT

on

Analysis and Design of Algorithms

Submitted by

Prakhyati Bansal (1BM21CS136)

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

June-2023 to September-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 Prakhyati Bansal (1BM21CS136), 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 academic semester June-2023 to September-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.

Name of the Lab-In charge:	Sowmya T	Dr. Jyothi S Nayak
Designation:	Assistant Professor	Professor and Head
	Department of CSE	Department of CSE
	BMSCE, Bengaluru	BMSCE, Bengaluru

Index Sheet

Course Outcome

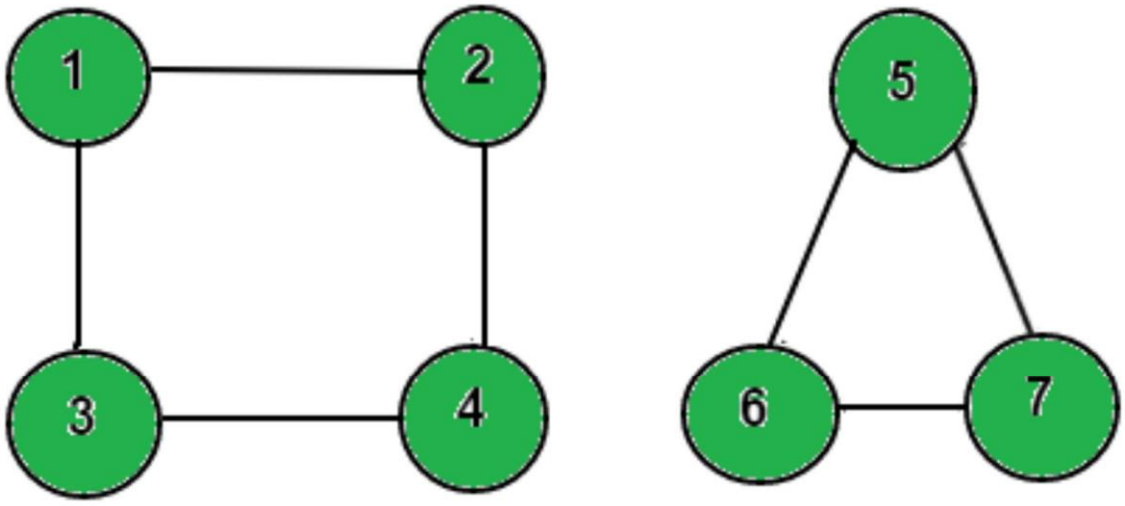
CO1	Analyze time complexity of Recursive and Non-recursive algorithms using asymptotic notations.
CO2	Apply various design techniques for the given problem.
CO3	Apply the knowledge of complexity classes P, NP, and NP-Complete and prove certain problems are NP-Complete
CO4	Design efficient algorithms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

1. a. Print all the nodes reachable from a given starting node in a digraph using BFS method

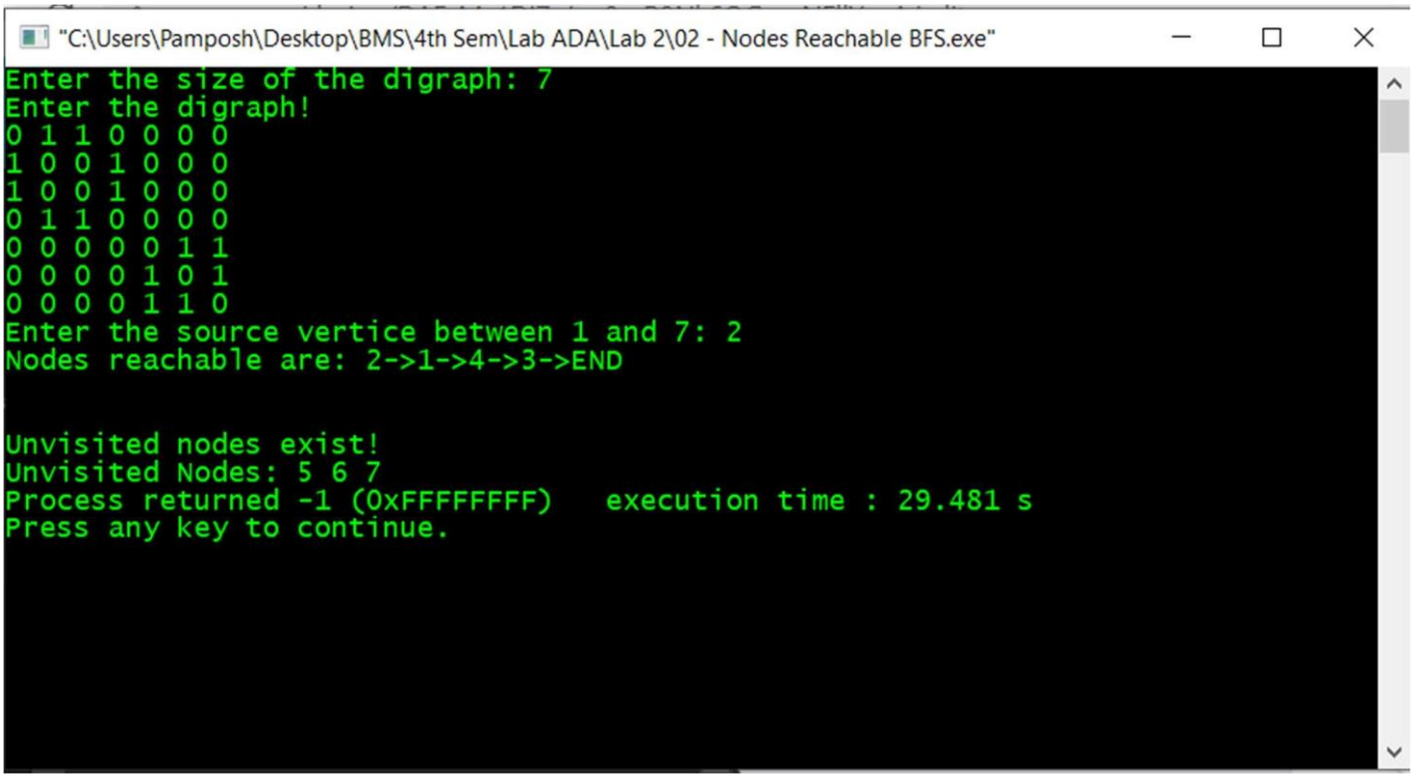
```c
/*
Print all the nodes reachable from a given starting node in a given digraph using BFS method
*/

#include<stdio.h>
#include<malloc.h>

//Making a queue
typedef struct queue{
    int *nodes;
    int f;
    int r;
}queue;

//Initiaiting the Queue
queue* initQueue(queue *q){
    q->f = 0;
    q->r = 0;
    return q;
}

//insert an element into the queue
queue* enterQueue(queue *q, int i){
    q->nodes[(q->r)++] = i;
    return q;
}

//remove an element from queue
int removeQueue(queue *q){
    return q->nodes[(q->f)++];
}

//Check if the queue is empty
int empty(queue *q){
    if (q->f == q->r)
        return 1;
    return 0;
}

//print the reachable nodes, and unreachable nodes, if any
void print(queue *q, int n, int *visited){
    int i, j, flag = 1;
    printf("Nodes reachable are: ");
    for(i=0; i<q->r; i++){
        printf("%d->",q->nodes[i]);
    }
    printf("END");
    for(i=0; i<n; i++){
        if (visited[i] == 0){
            flag = 0;
            printf("\n\nUnvisited nodes exist!\nUnvisited Nodes: ");
            for(j=i; j<n; j++){
                printf("%d ", visited[j]);
            }
            break;
        }
    }
    if(flag == 1){
        printf("\n\nNo unreachable nodes from given source node!\n");
    }
}

//Main BFS implementation function
void bfs(queue *q, int *visited, int **graph, int n){
    int source, i;
    printf("Enter the source vertex between 1 and %d: ", n);
    scanf("%d", &source);

    q = enterQueue(q, source);
    while(!empty(q)){
        source = removeQueue(q);
        visited[source-1] = 1;
        for(i=0; i<n; i++){
            if(graph[source-1][i]==1 && !visited[i]){
                q = enterQueue(q, i+1);
                visited[i] = 1;
            }
        }
    }
}

int main(){
    int **graph;
    int *visited;
```

```
    int n, i, j;
    queue *q = malloc(sizeof(*q));

    //Input taking begins
    printf("Enter the size of the digraph: ");
    scanf("%d", &n);

    printf("Enter the digraph:\n");
    graph = (int**)malloc(n*sizeof(int*));
    visited = (int*)malloc(n*sizeof(int));
    q = initQueue(q);
    q->nodes = (int*)malloc(n*sizeof(int));

    for(i=0; i<n; i++){
        graph[i] = (int*)malloc(n*sizeof(int));
        for(j=0; j<n; j++){
            scanf("%d", &graph[i][j]);
        }
        visited[i] = 0;
    }
    //Input taking ends

    bfs(q, visited, graph, n);

    print(q, n, visited);
    return -1;
}
```

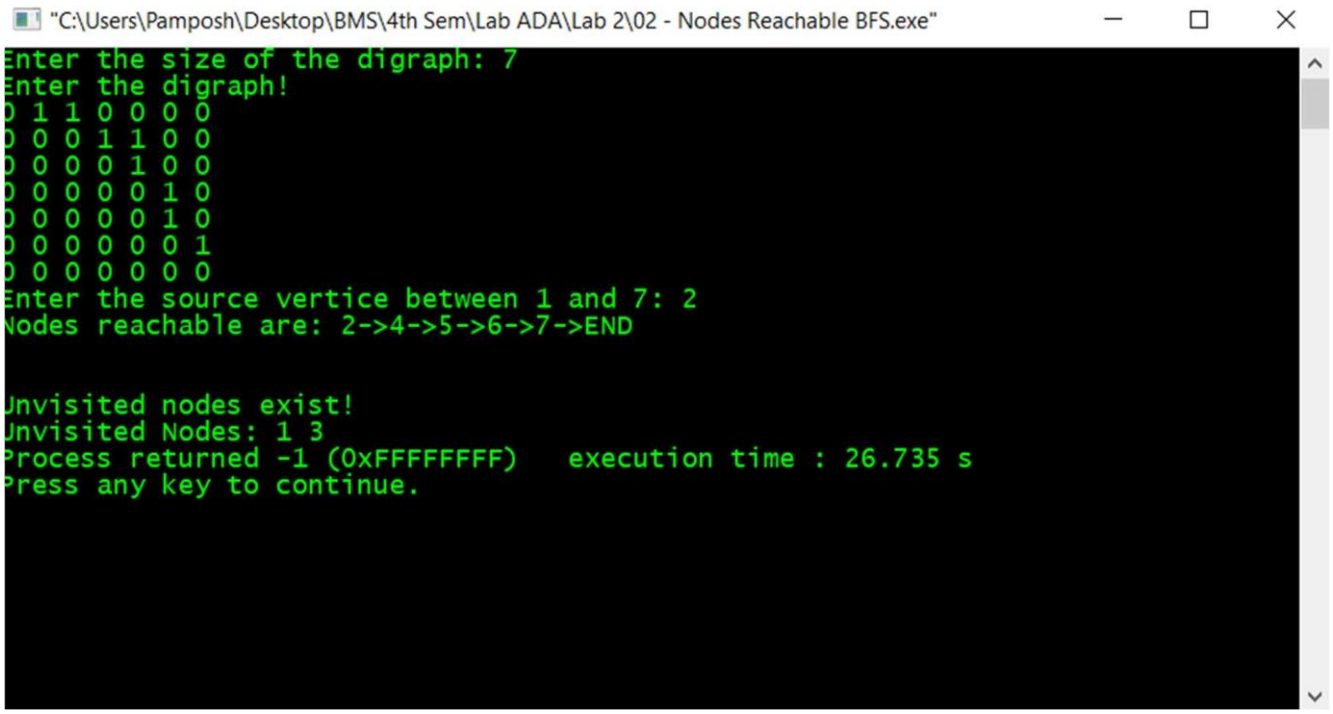
Graph 1



	1	2	3	4	5	6	7
1	0	1	1	0	0	0	0
2	1	0	0	1	0	0	0
3	1	0	0	1	0	0	0
4	0	1	1	0	0	0	0
5	0	0	0	0	0	1	1
6	0	0	0	0	1	0	1
7	0	0	0	0	1	1	0


"C:\Users\Pamposh\Desktop\BMS\4th Sem\Lab ADA\Lab 2\02 - Nodes Reachable BFS.exe"
Enter the size of the digraph: 7
Enter the digraph!
0 1 1 0 0 0 0
1 0 0 1 0 0 0
1 0 0 1 0 0 0
0 1 1 0 0 0 0
0 0 0 0 0 1 1
0 0 0 0 1 0 1
0 0 0 0 1 1 0
Enter the source vertice between 1 and 7: 2
Nodes reachable are: 2->1->4->3->END
Unvisited nodes exist!
Unvisited Nodes: 5 6 7
Process returned -1 (0xFFFFFFFF) execution time : 29.481 s
Press any key to continue.

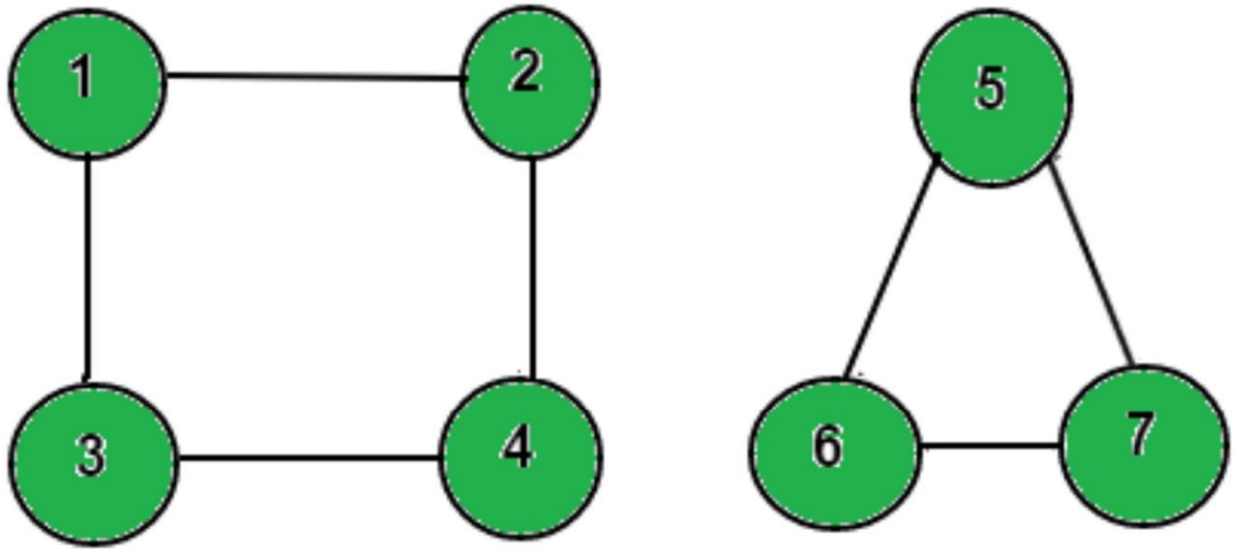
Graph 2


"C:\Users\Pamposh\Desktop\BMS\4th Sem\Lab ADA\Lab 2\02 - Nodes Reachable BFS.exe"
Enter the size of the digraph: 7
Enter the digraph!
0 1 1 0 0 0 0
0 0 0 1 1 0 0
0 0 0 0 1 0 0
0 0 0 0 0 1 0
0 0 0 0 0 1 0
0 0 0 0 0 0 1
0 0 0 0 0 0 0
Enter the source vertice between 1 and 7: 2
Nodes reachable are: 2->4->5->6->7->END
Unvisited nodes exist!
Unvisited Nodes: 1 3
Process returned -1 (0xFFFFFFFF)   execution time : 26.735 s
Press any key to continue.

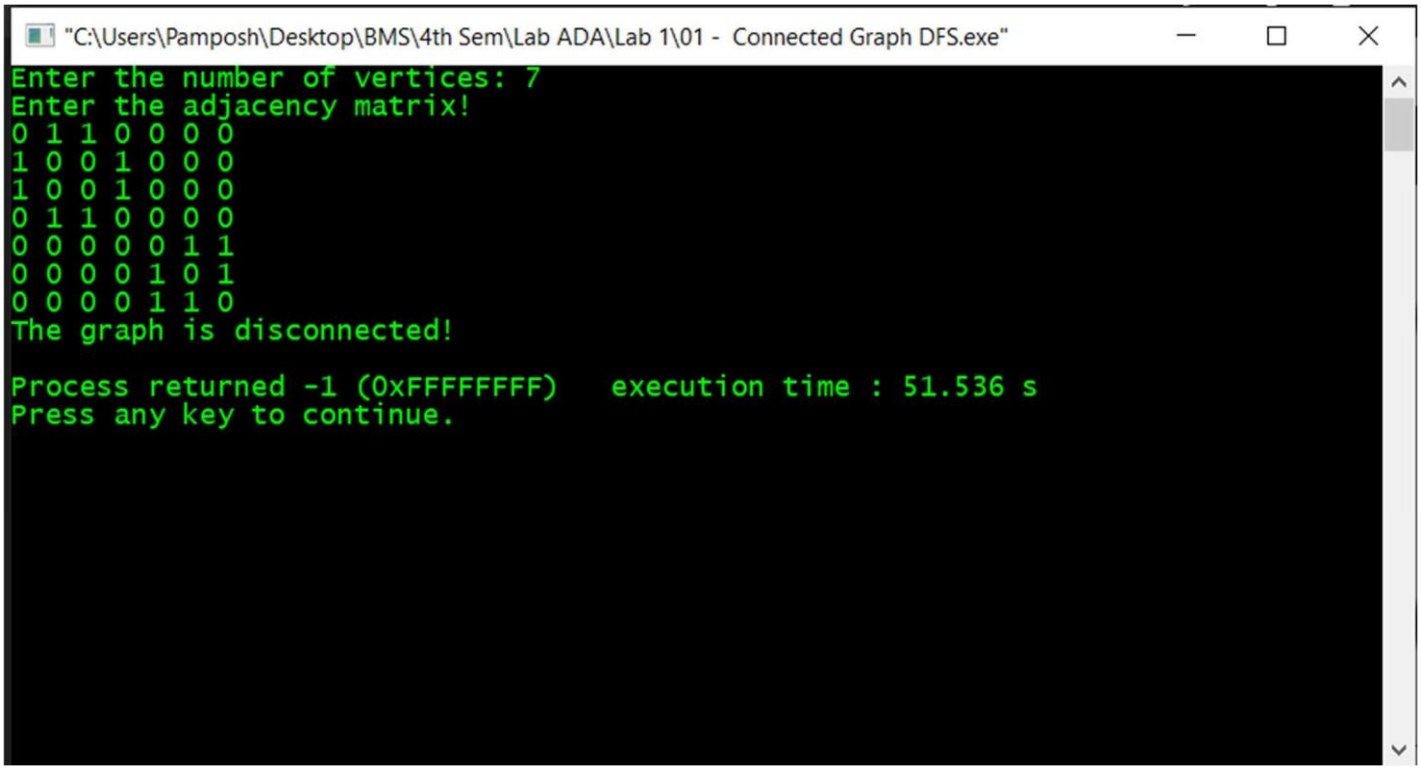
1. b. Check whether a given graph is connected or not using DFS method.

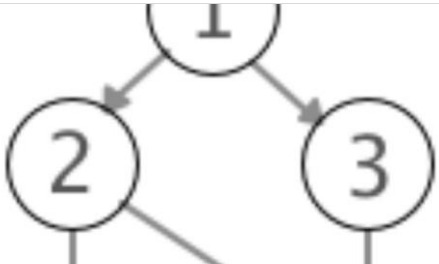
```c
/*
Check whether the given graph is connected or not using DFS traversal
*/

#include<stdio.h>
#include<malloc.h>

void dfs_second(int source, int **graph, int *visited, int n){
    int i;
    for(i=0; i<n; i++){
        if(graph[source][i] && !visited[i]){
            visited[i] = 1;
            dfs_second(i, graph, visited, n);
        }
    }
}

int dfs(int source, int **graph, int n){
    int *visited;
    int i, count = 0;

    visited = (int*)malloc(n*sizeof(int));
    for(i=0; i<n; i++){
        visited[i] = 0;
    }
    visited[source] = 1;

    for(i=0; i<n; i++){
        if(graph[source][i] && !visited[i]){
            visited[i] = 1;
            dfs_second(i, graph, visited, n);
        }
    }

    for(i=0; i<n; i++)
        if(visited[i])
            count += 1;
    return count;
}

int main(){

    //Input taking
    int n, i, j, count, flag = 0;
    int **graph;

    printf("Enter the number of vertices: ");
    scanf("%d", &n);

    graph = (int**)malloc(n*sizeof(int*));

    printf("Enter the adjacency matrix:\n");
    for(i=0; i<n; i++){
        graph[i] = (int*)malloc(n*sizeof(int));
        for(j=0; j<n; j++){
            scanf("%d", &graph[i][j]);
        }
    }
    //Inputs taken

    //Checking if nodes are reachable
    for(i=0; i<n; i++){
        count = dfs(i, graph, n);
        if(count == n){
            flag = 1;
            printf("All nodes are reachable via source %d\n", i+1);
            break;
        }
    }

    //If all nodes aren't reached, print it's disconnected
    if(!flag){
        printf("The graph is disconnected!\n");
    }

    return -1;
}
```

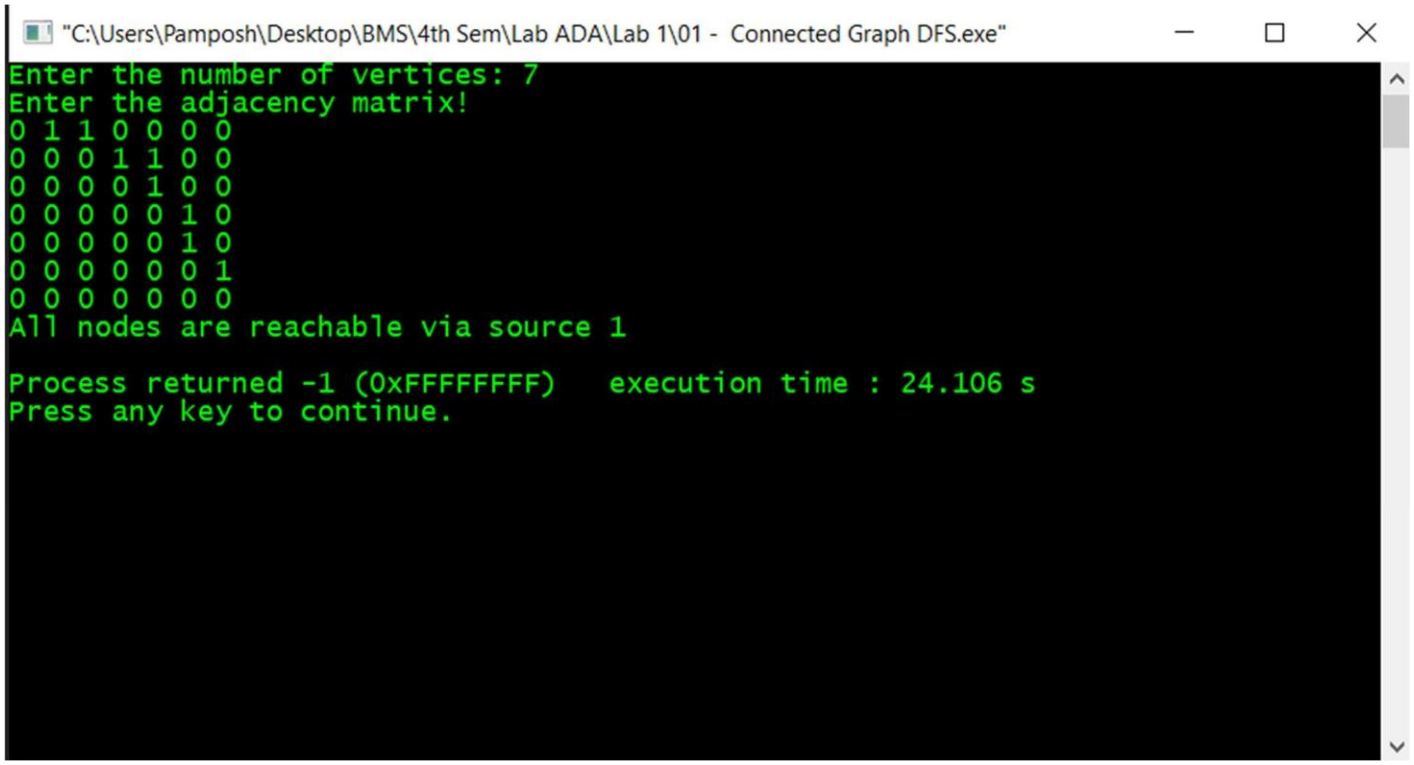
Graph 1



	1	2	3	4	5	6	7
1	0	1	1	0	0	0	0
2	1	0	0	1	0	0	0
3	1	0	0	1	0	0	0
4	0	1	1	0	0	0	0
5	0	0	0	0	0	1	1
6	0	0	0	0	1	0	1
7	0	0	0	0	1	1	0


"C:\Users\Pamposh\Desktop\BMS\4th Sem\Lab ADA\Lab 1\01 - Connected Graph DFS.exe"
Enter the number of vertices: 7
Enter the adjacency matrix!
0 1 1 0 0 0 0
1 0 0 1 0 0 0
1 0 0 1 0 0 0
0 1 1 0 0 0 0
0 0 0 0 0 1 1
0 0 0 0 1 0 1
0 0 0 0 1 1 0
The graph is disconnected!
Process returned -1 (0xFFFFFFFF) execution time : 51.536 s
Press any key to continue.

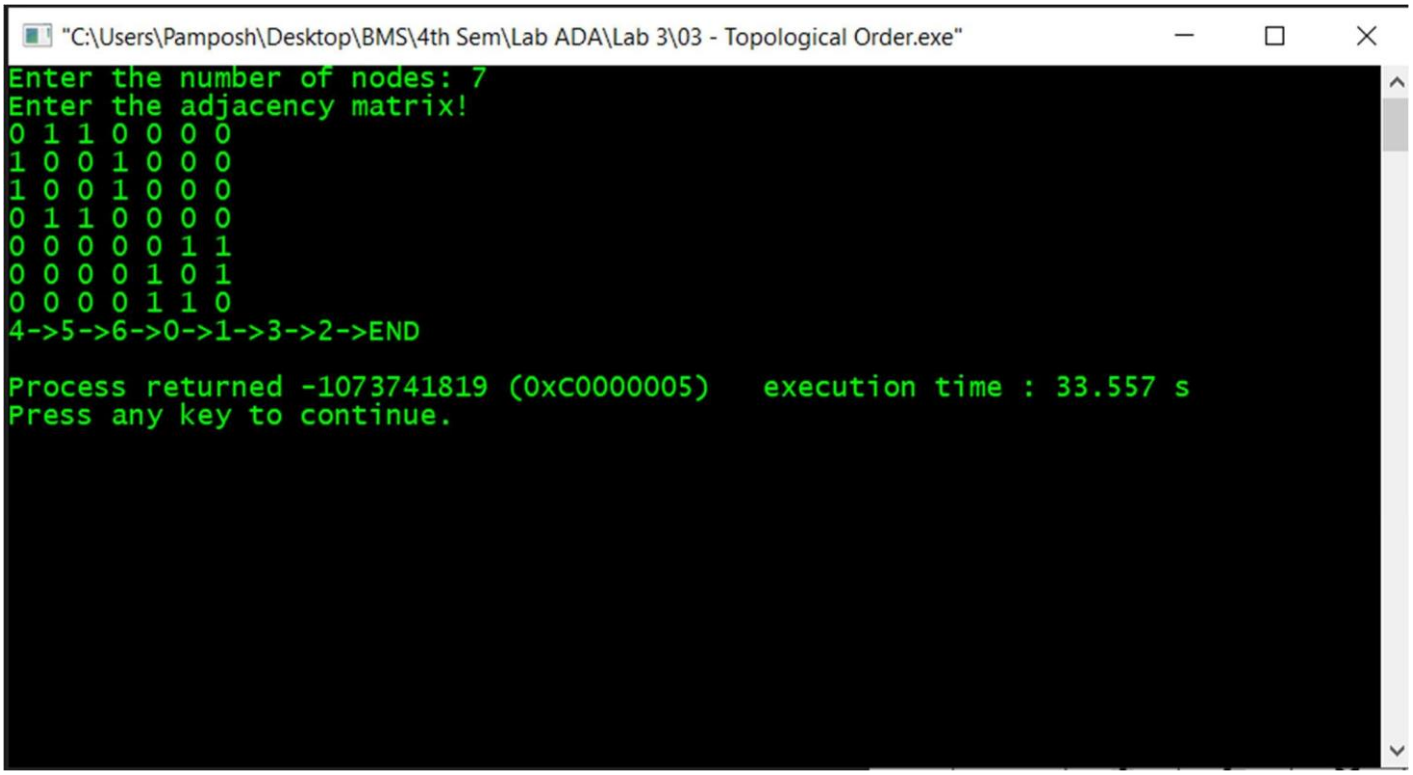

1
2
3


"C:\Users\Pamposh\Desktop\BMS\4th Sem\Lab ADA\Lab 1\01 - Connected Graph DFS.exe"
Enter the number of vertices: 7
Enter the adjacency matrix!
0 1 1 0 0 0 0
0 0 0 1 1 0 0
0 0 0 0 1 0 0
0 0 0 0 0 1 0
0 0 0 0 0 1 0
0 0 0 0 0 0 1
0 0 0 0 0 0 0
All nodes are reachable via source 1
Process returned -1 (0xFFFFFFFF) execution time : 24.106 s
Press any key to continue.

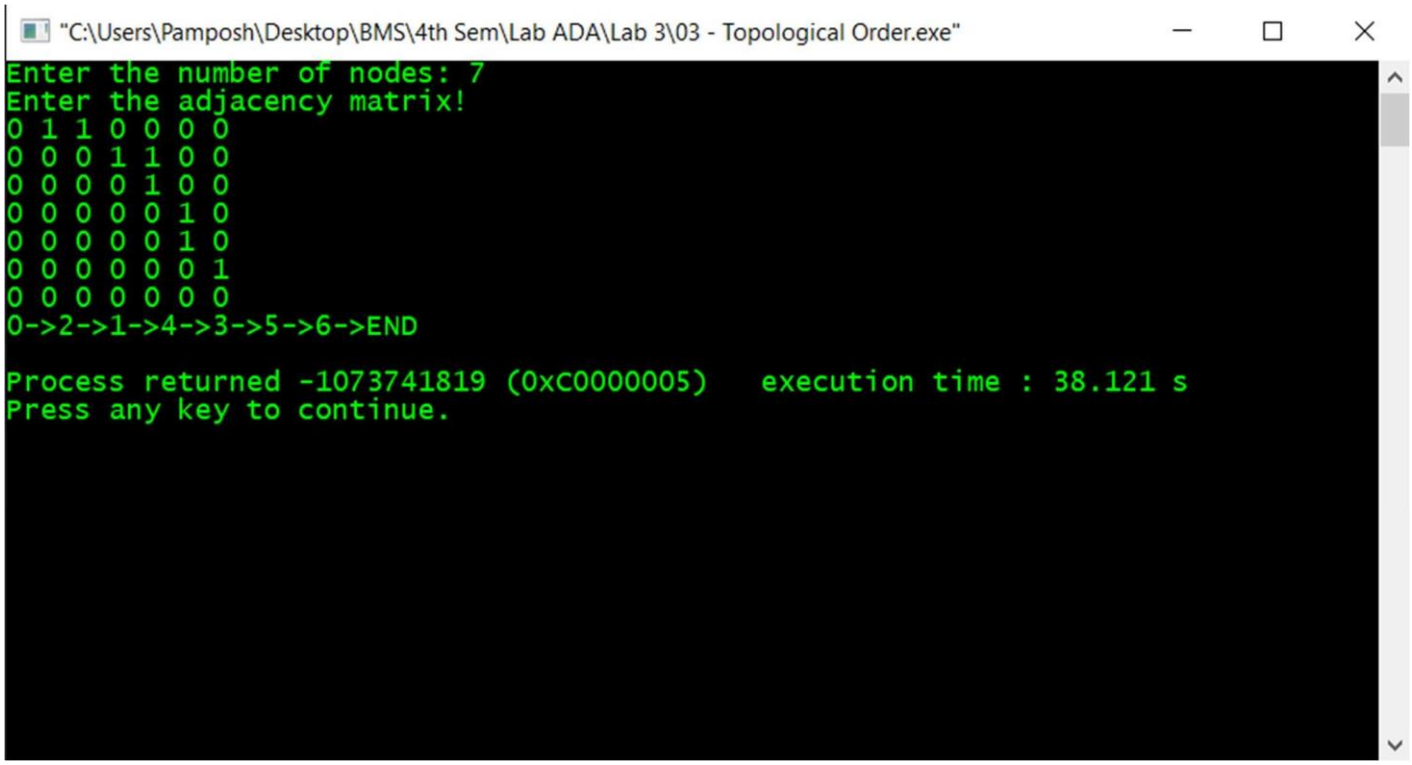
2. Write program to obtain the Topological ordering of vertices in a given digraph

```c
/*
Write a program to obtain the Topological ordering of vertices in a given digraph
*/

#include<stdio.h>
#include<malloc.h>

//Implementing the DFS function
void dfs(int source, int **graph, int *visited, int n, int *result, int *res_ind){
    int i;
    visited[source] = 1;

    for(i=0; i<n; i++){
        if(graph[source][i] && !visited[i]){
            dfs(i, graph, visited, n, result, res_ind);
        }
    }
    result[(*res_ind)++] = source;
}

//Implementing the topological ordering function
int* topological(int n, int **graph, int *visited){
    int i;
    int *res, res_ind = 0;

    res = (int*)malloc(sizeof(int));
    for(i=0; i<n; i++){
        if(!visited[i]){
            dfs(i, graph, visited, n, res, &res_ind);
        }
    }
    return res;
}

int main(){
    int i, j, n;
    int **graph;
    int *visited, *res;

    //Input entering begins
    printf("Enter the number of nodes: ");
    scanf("%d", &n);

    visited = (int*)malloc(n*sizeof(int));
    graph = (int**)malloc(n*sizeof(int*));

    printf("Enter the adjacency matrix:\n");
    for(i=0; i<n; i++){
        visited[i] = 0;
        graph[i] = (int*)malloc(n*sizeof(int));
        for(j=0; j<n; j++){
            scanf("%d", &graph[i][j]);
        }
    }
    //input intaking ends

    //Processing the graph for getting the topological order
    res = topological(n, graph, visited);
    for(i=n-1; i>=0; i--)
        printf("%d->", res[i]);
    printf("END\n");
    return -1;
}
```

Graph 1


"C:\Users\Pamposh\Desktop\BMS\4th Sem\Lab ADA\Lab 3\03 - Topological Order.exe"
Enter the number of nodes: 7
Enter the adjacency matrix!
0 1 1 0 0 0 0
1 0 0 1 0 0 0
1 0 0 1 0 0 0
0 1 1 0 0 0 0
0 0 0 0 0 1 1
0 0 0 0 1 0 1
0 0 0 0 1 1 0
4->5->6->0->1->3->2->END
Process returned -1073741819 (0xC0000005) execution time : 33.557 s
Press any key to continue.

Graph 2

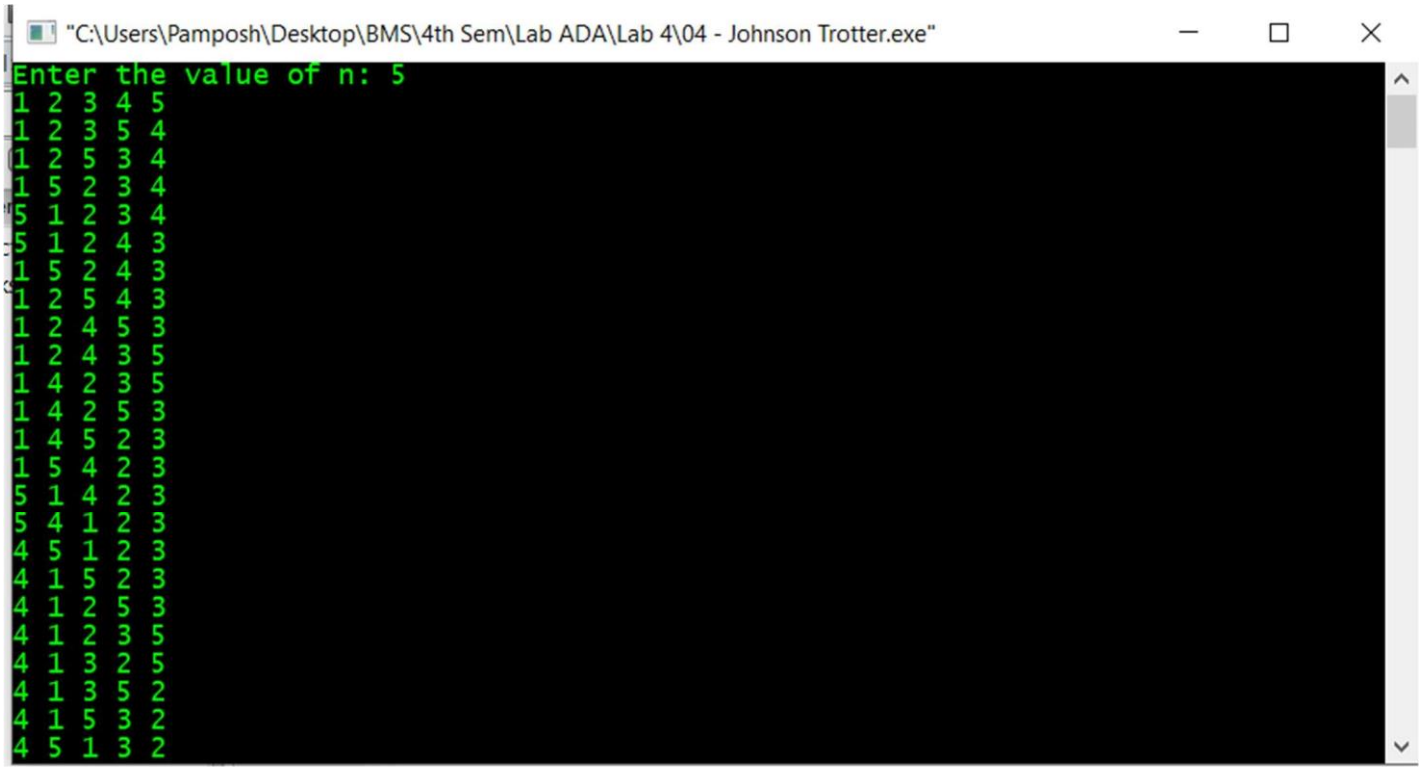

"C:\Users\Pamposh\Desktop\BMS\4th Sem\Lab ADA\Lab 3\03 - Topological Order.exe"
Enter the number of nodes: 7
Enter the adjacency matrix!
0 1 1 0 0 0 0
0 0 0 1 1 0 0
0 0 0 0 1 0 0
0 0 0 0 0 1 0
0 0 0 0 0 1 0
0 0 0 0 0 0 1
0 0 0 0 0 0 0
0->2->1->4->3->5->6->END
Process returned -1073741819 (0xC0000005) execution time : 38.121 s
Press any key to continue.

3. Implement Johnson Trotter algorithm to generate permutations

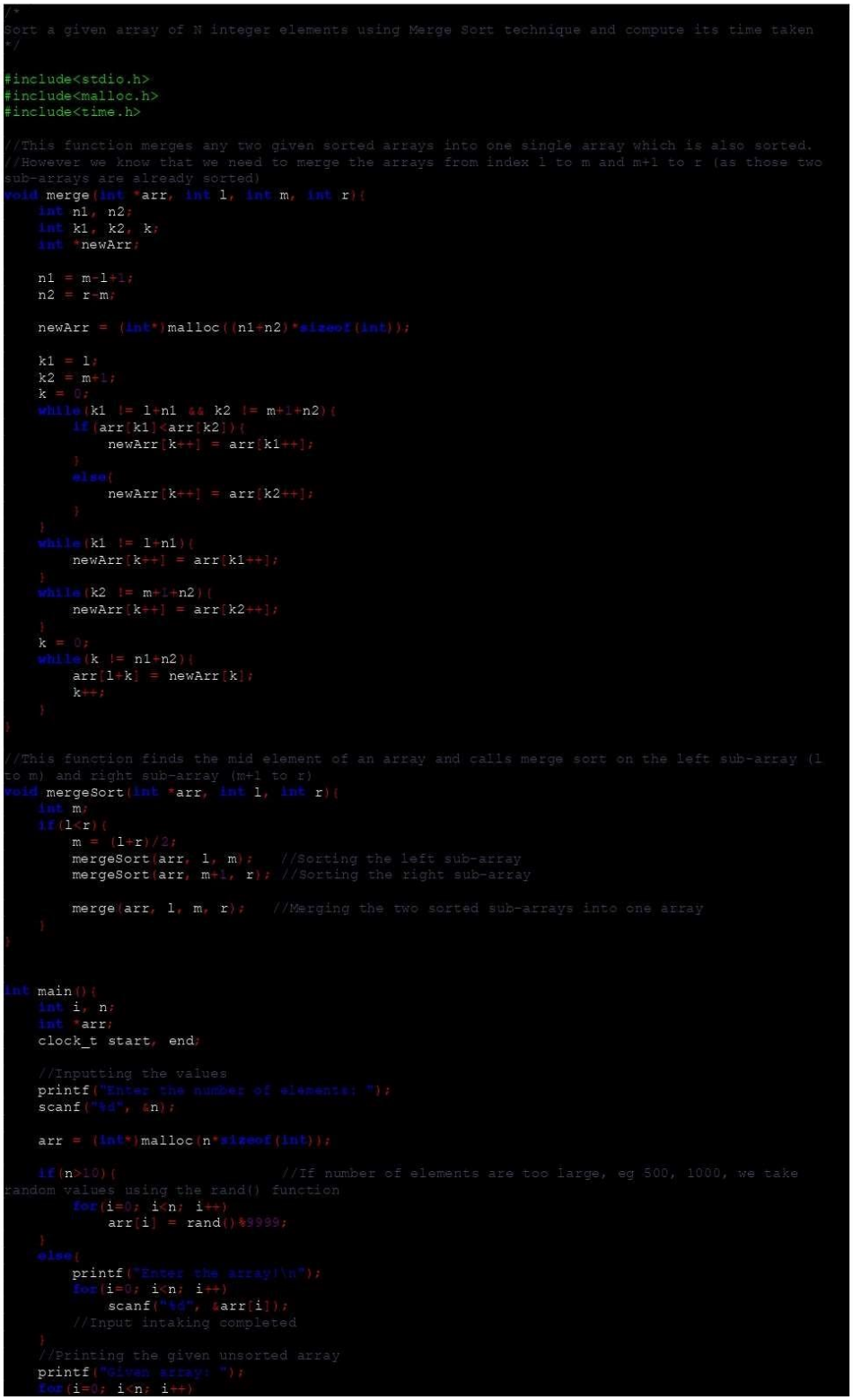
```c
/*
Implement Johnson Trotter algorithm to generate permutations
*/

#include<stdio.h>
#include<malloc.h>

/*
Algorithm of Johnson Trotter:
i) Look for the greatest element which is greater than the adjacent element its direction points
to. This is called the mobile element.
ii) Print the initial permutation and then swap the mobile element with its adjacent element to
get the next permutation.
iii) If the mobile element just selected it not the greatest element in the array, swap the
directions of the elements greater than the mobile element
iv) Repeat the process till you have no mobile element available
*/


//Function to swap the elements pointed to by a and b
void swap(int *a, int *b){
    int temp;
    temp = *a;
    *a = *b;
    *b = temp;
}

//Function to flip direction of all elements greater than the mobile element
void flipDirection(int *l, int *dir, int mobile, int n){
    int i;

    for(i=0; i<n; i++){
        if(l[i]>mobile){
            dir[i] = dir[i]*(-1);
        }
    }
}

//Function to get the mobile element in a given number array and a direction array
int getMobile(int *l, int *dir, int n){
    int i, maxMobile = -1, ind, k;   //Initialise maxMobile to -1, so that we can later compare
it to the elements in the array(as elements in array are always positive)
    for(i=0; i<n; i++){
        k = i + dir[i];       //k now represents the index of element in direction of i and
adjacent to it(since, dir[i] is -1 if element points to left, and +1 if it points to right)
        if(k>=0 && k<n){      //If this value of index (k) is within range of the array, ie it
IS NOT out of bounds, go to the next step
            if(l[i]>l[k]){    //Check if the element at i, is greater than the element at k (ie
element adjacent to it in direction of the element at i)
                if(maxMobile<l[i]){   //if previously selected maxMobile was greater, keep it
else replace it with the new mobile
                    maxMobile = l[i];
                }
            }
        }
    }
    if(maxMobile == -1) //If a mobile element was never found (ie, it remained on its
initialised value of -1, return -1 and stop the loop in the main function)
        return -1;
    for(i=0; i<n; i++){   //Else get the index of the maxMobile element and return the index
        if(l[i] == maxMobile){
            ind = i;
            break;
        }
    }
    return ind;
}

//Function to print the current permutation of the array and swap elements to get the next
permutation
int perm(int n, int *dir, int *l){
    int m = getMobile(l, dir, n);   //m now represents the CURRENT index of the mobile element
in the array (not yet swapped). It is -1 if no mobile element exists
    int i, pos;

    //Print the current permutation of the array
    for(i=0; i<n; i++)
        printf("%d ", l[i]);
    printf("\n");

    //If m == -1, ie no mobile element was found, return 0 and stop the while loop in the main
function
    if(m == -1)
        return 0;
```

```
    //pos now represents the new position of the mobile element AFTER swapping it with the
adjacent element
    pos = m + dir[m];

    swap(&l[m], &l[m+dir[m]]);      //Swap the mobile element in the number array
    swap(&dir[m], &dir[m+dir[m]]); //Swap the direction of the mobile element with its adjacent

    if (l[pos]<n)                   //If the value of the current mobile element is lesser than
the max element in the list, swap the directions of the elements greater than mobile
        flipDirection(l, dir, l[pos], n);
    return 1;
}

int main(){
    int n, i;
    int *l, *dir, cont = 1;

    //Input taking
    printf("Enter the value of n: ");
    scanf("%d", &n);

    l = (int*)malloc(n*sizeof(int));
    dir = (int*)malloc(n*sizeof(int));

    for(i=0; i<n; i++){
        l[i] = i+1;         //Creating the array of initial permutation, eg: if n = 4, l = [1, 2,
3, 4] initially
        dir[i] = -1;        //directions of all the elements is set to -1 (ie. left to right) first
    }

    while(cont){            //cont is initialised to 1
        cont = perm(n, dir, l);     //the funct perm returns 1 if a new permutation was
generated successfully, else -1 is returned and cont is set to -1, thus stopping the loop
    }
}
```

```
"C:\Users\Pamposh\Desktop\BMS\4th Sem\Lab ADA\Lab 4\04 - Johnson Trotter.exe"
Enter the value of n: 4
1 2 3 4
1 2 4 3
1 4 2 3
4 1 2 3
4 1 3 2
1 4 3 2
1 3 4 2
1 3 2 4
3 1 2 4
3 1 4 2
3 4 1 2
4 3 1 2
4 3 2 1
3 4 2 1
3 2 4 1
3 2 1 4
2 3 1 4
2 3 4 1
2 4 3 1
4 2 3 1
4 2 1 3
2 4 1 3
2 1 4 3
2 1 3 4
```

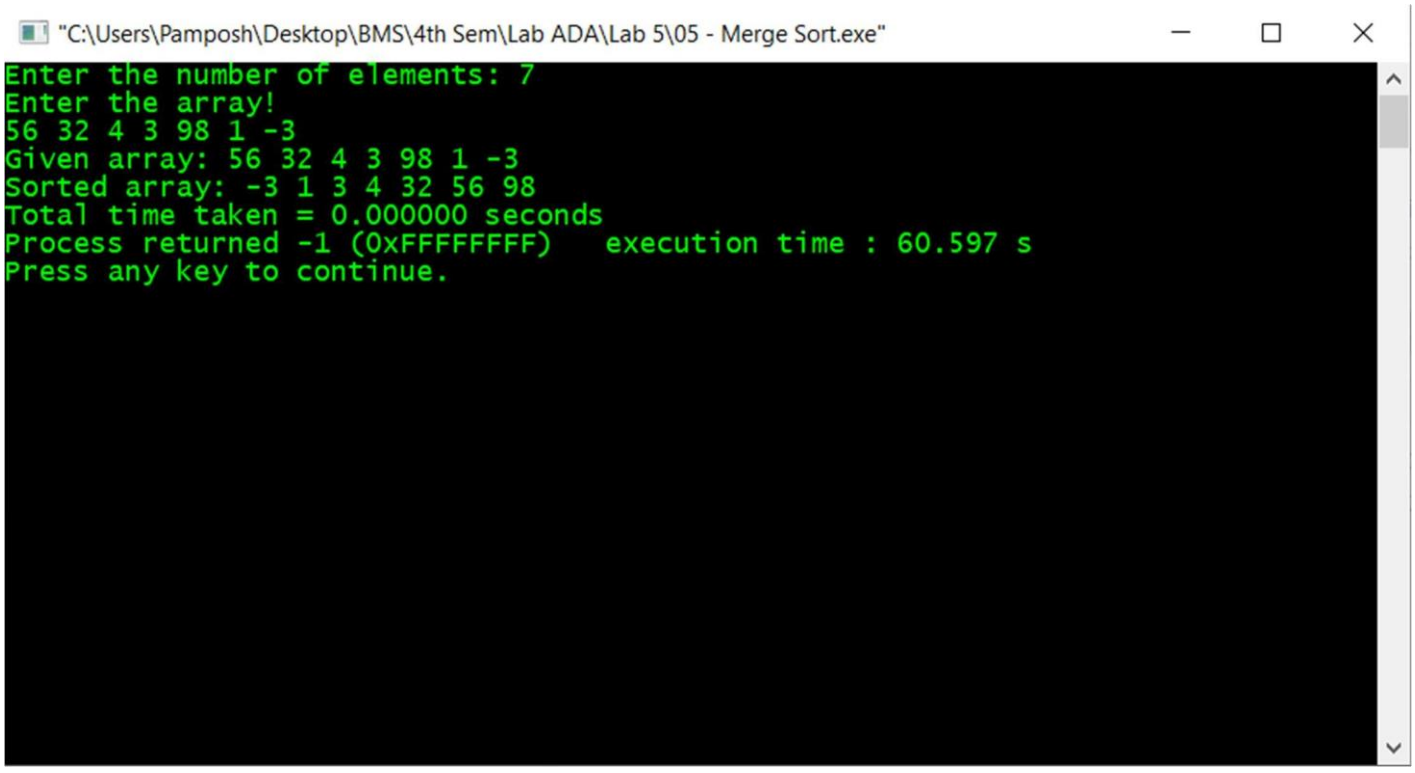

"C:\Users\Pamposh\Desktop\BMS\4th Sem\Lab ADA\Lab 4\04 - Johnson Trotter.exe"
Enter the value of n: 5
1 2 3 4 5
1 2 3 5 4
1 2 5 3 4
1 5 2 3 4
5 1 2 3 4
5 1 2 4 3
1 5 2 4 3
1 2 5 4 3
1 2 4 5 3
1 2 4 3 5
1 4 2 3 5
1 4 2 5 3
1 4 5 2 3
1 5 4 2 3
5 1 4 2 3
5 4 1 2 3
4 5 1 2 3
4 1 5 2 3
4 1 2 5 3
4 1 2 3 5
4 1 3 2 5
4 1 3 5 2
4 1 5 3 2
4 5 1 3 2

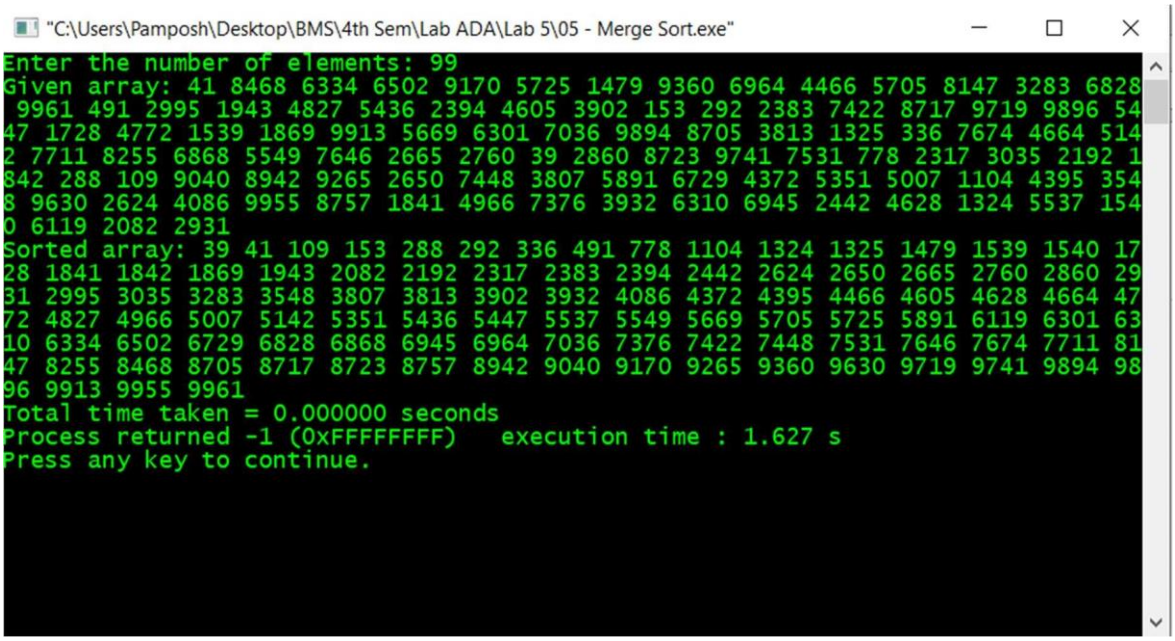
4. Sort a given set of N integer elements using Merge Sort technique and compute its time taken. Run the program for different values of N and record the time taken to sort



```
/*
Sort a given array of N integer elements using Merge Sort technique and compute its time taken
*/

#include<stdio.h>
#include<malloc.h>
#include<time.h>

//This function merges any two given sorted arrays into one single array which is also sorted.
//However we know that we need to merge the arrays from index l to m and m+1 to r (as those two
sub-arrays are already sorted)
void merge(int *arr, int l, int m, int r){
    int n1, n2;
    int k1, k2, k;
    int *newArr;

    n1 = m-l+1;
    n2 = r-m;

    newArr = (int*)malloc((n1+n2)*sizeof(int));

    k1 = l;
    k2 = m+1;
    k = 0;
    while(k1 != l+n1 && k2 != m+1+n2){
        if(arr[k1]<arr[k2]){
            newArr[k++] = arr[k1++];
        }
        else{
            newArr[k++] = arr[k2++];
        }
    }
    while(k1 != l+n1){
        newArr[k++] = arr[k1++];
    }
    while(k2 != m+1+n2){
        newArr[k++] = arr[k2++];
    }
    k = 0;
    while(k != n1+n2){
        arr[l+k] = newArr[k];
        k++;
    }
}

//This function finds the mid element of an array and calls merge sort on the left sub-array (l
to m) and right sub-array (m+1 to r)
void mergeSort(int *arr, int l, int r){
    int m;
    if(l<r){
        m = (l+r)/2;
        mergeSort(arr, l, m);   //Sorting the left sub-array
        mergeSort(arr, m+1, r); //Sorting the right sub-array

        merge(arr, l, m, r);   //Merging the two sorted sub-arrays into one array
    }
}


int main(){
    int i, n;
    int *arr;
    clock_t start, end;

    //Inputting the values
    printf("Enter the number of elements: ");
    scanf("%d", &n);

    arr = (int*)malloc(n*sizeof(int));

    if(n>10){                   //If number of elements are too large, eg 500, 1000, we take
random values using the rand() function
        for(i=0; i<n; i++)
            arr[i] = rand()%9999;
    }
    else{
        printf("Enter the array!\n");
        for(i=0; i<n; i++)
            scanf("%d", &arr[i]);
        //Input intaking completed
    }
    //Printing the given unsorted array
    printf("Given array: ");
    for(i=0; i<n; i++)
```

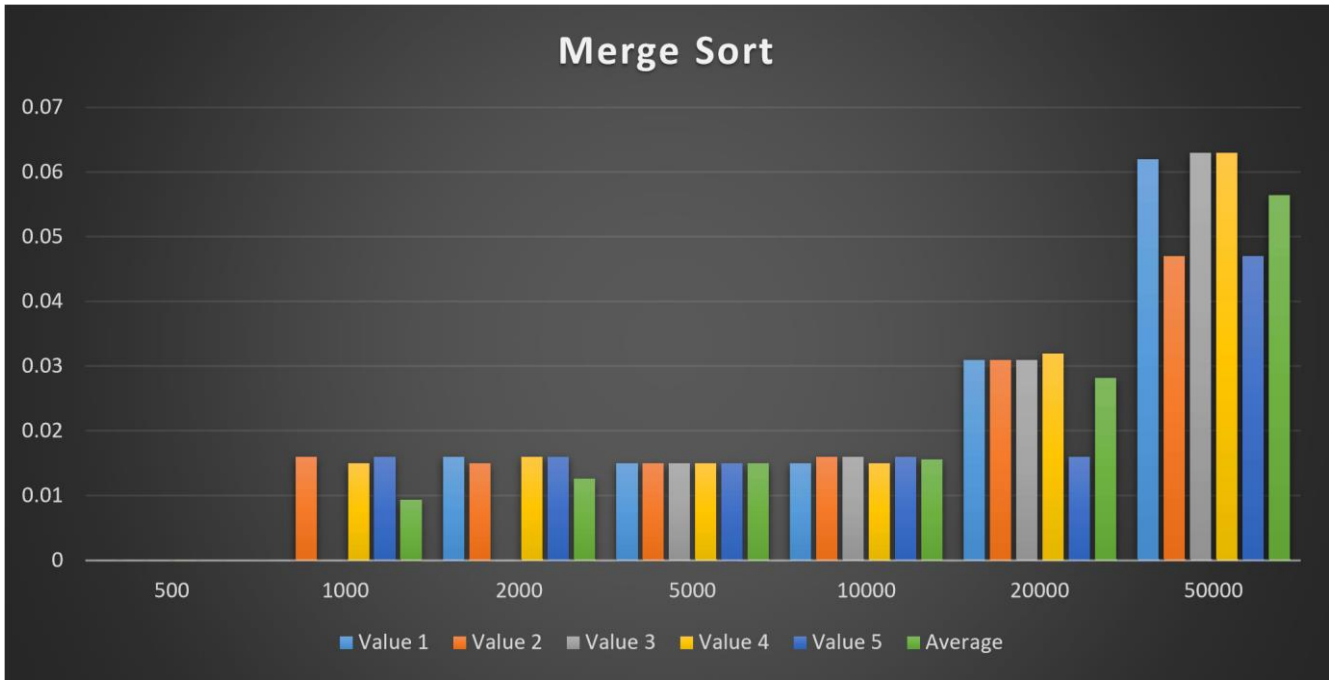
```
        printf("%d ", arr[i]);
    printf("\n");

    start = clock();

    //Implementing the mergeSort algorithm
    mergeSort(arr, 0, n-1);

    end = clock();

    //Printing the sorted array
    printf("Sorted array: ");
    for(i=0; i<n; i++)
        printf("%d ", arr[i]);
    printf("\n");

    printf("Total time taken = %f seconds", (double)(end-start)/CLOCKS_PER_SEC);
    return -1;
}
```





TIME TAKEN BY VARIOUS VALUES OF N

MERGE SORT	500	1000	2000	5000	10000	20000	50000
Value 1	0.0000001	0.0000001	0.016	0.015	0.015	0.031	0.062
Value 2	0.0000001	0.016	0.015	0.015	0.016	0.031	0.047
Value 3	0.0000001	0.0000001	0.0000001	0.015	0.016	0.031	0.063
Value 4	0.0000001	0.015	0.016	0.015	0.015	0.032	0.063
Value 5	0.0000001	0.016	0.016	0.015	0.016	0.016	0.047
Average	0.0000001	0.00940004	0.0126	0.015	0.0156	0.0282	0.0564



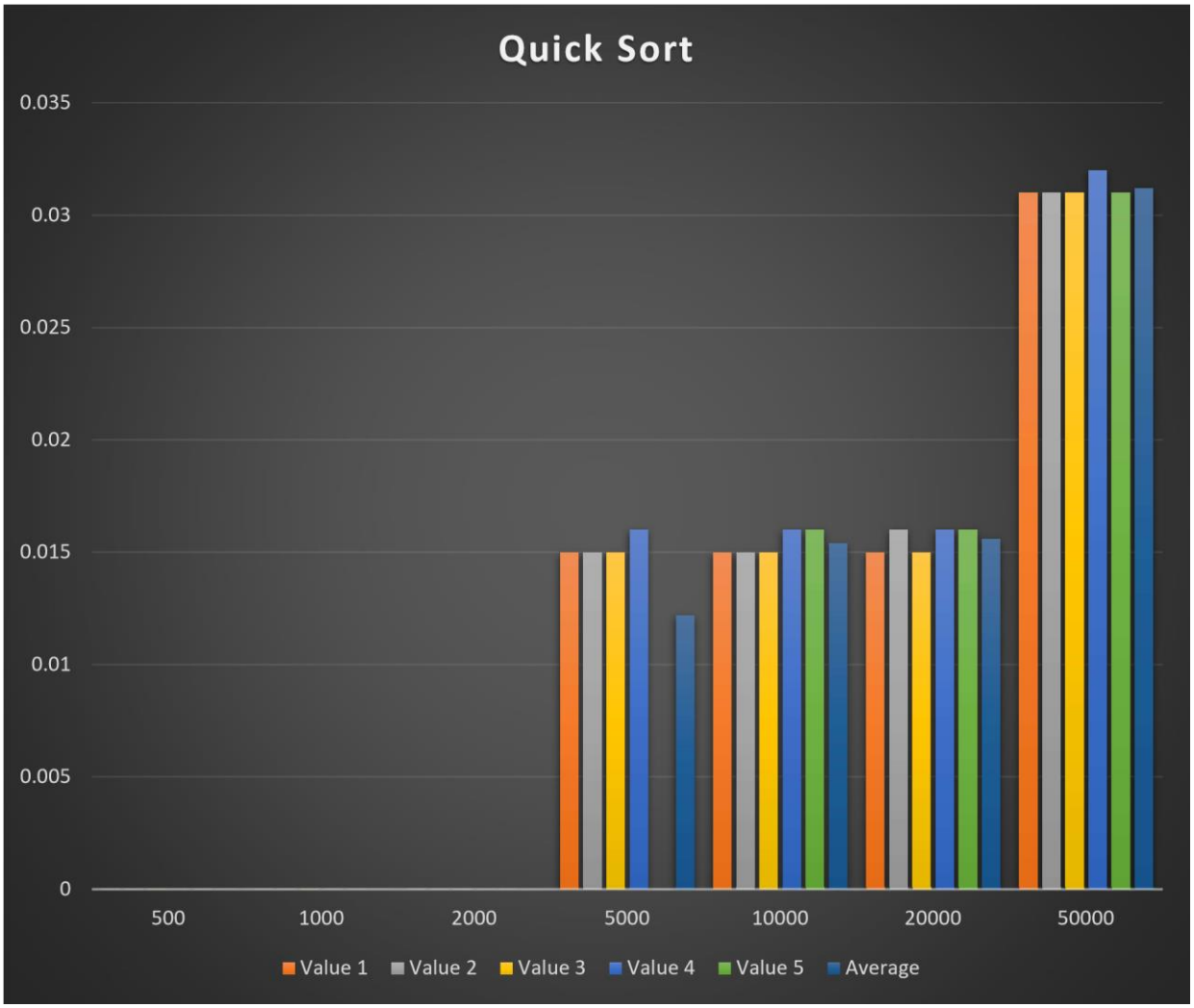
5. Sort a given set of N integer elements using Quick Sort technique and compute its time taken

```c
/*
Sort a given array of N integer elements using Quick Sort technique and compute its time taken
*/

#include<stdio.h>
#include<malloc.h>
#include<time.h>

//Function for swapping two values(pointers) given
void swap(int *a, int *b){
    int temp = *a;
    *a = *b;
    *b = temp;
}

//Main QuickSort function
void quickSort(int *arr, int l, int r){
    if(l!=r){ //If l==r, means only 1 element in the sub-array, so no need to sort
        int i = l, j = r;
        int pivot = l;
        while(i<j){
            while(arr[i]<=arr[pivot]){
                if (i==r)
                    break;
                i++;
            }
            while(arr[j]>arr[pivot]){
                if (j==l)
                    break;
                j--;
            }
            if(i<j){
                swap(&arr[i], &arr[j]);     //This swap brings all elements lesser than pivot
to the left, and all greater than it to the right
            }
        }
        swap(&arr[j], &arr[pivot]);    //This swap brings the pivot element to its correct
position
        if(j!=l)
            quickSort(arr, l, j-1);     //Calls the quicksort function on the left sub-array of
the pivot only if the current pivot is NOT the left most element
        if (j!=r)
            quickSort(arr, j+1, r);    //Calls the quicksort function on the right sub-array of
the pivot only if the current pivot is NOT the right most element
    }
}


int main(){
    int i, n;
    int *arr;
    time_t start_time, end_time;

    //Input entering begins
    printf("Enter the size of array: ");
    scanf("%d", &n);

    arr = (int*)malloc(n*sizeof(int));
    if(n>10){                   //If number of elements are too large, eg 500, 1000, we take
random values using the rand() function
        for(i=0; i<n; i++)
            arr[i] = rand()%9999;
    }
    else{
        printf("Enter the array:\n");
        for(i=0; i<n; i++)
            scanf("%d", &arr[i]);
        //Input intaking completed
    }

    //Printing the current array
    printf("Given array is: ");
    for(i=0; i<n; i++)
        printf("%d ", arr[i]);
    printf("\n");

    start_time = clock();

    //Calling the quicksort function on the whole array, ie. from 0 to n-1
    quickSort(arr, 0, n-1);

    end_time = clock();

    //Printing the sorted array
```

```
    printf("Sorted array is: ");
    for(i=0; i<n; i++)
        printf("%d ", arr[i]);
    printf("\n");

    printf("Total time taken = %f seconds", (double)(end_time - start_time)/CLOCKS_PER_SEC);
    return -1;
}
```

TIME TAKEN FOR VARIOUS VALUES OF N

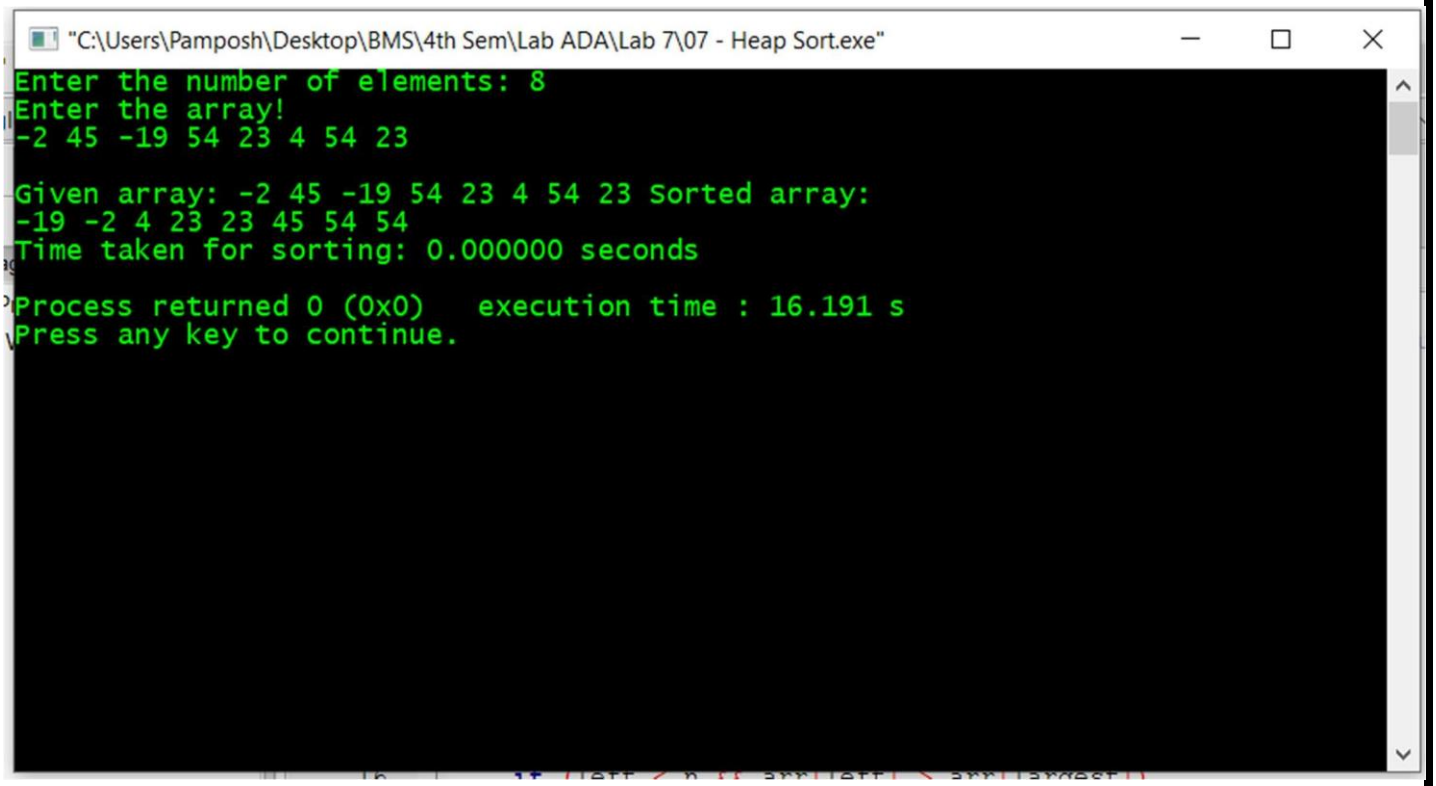
QUICK SORT	500	1000	2000	5000	10000	20000	50000
Value 1	0.000001	0.000001	0.000001	0.015	0.015	0.015	0.031
Value 2	0.000001	0.000001	0.000001	0.015	0.015	0.016	0.031
Value 3	0.000001	0.000001	0.000001	0.015	0.015	0.015	0.031
Value 4	0.000001	0.000001	0.000001	0.016	0.016	0.016	0.032
Value 5	0.000001	0.000001	0.000001	0.0000001	0.016	0.016	0.031
Average	0.000001	0.000001	0.000001	0.01220002	0.0154	0.0156	0.0312

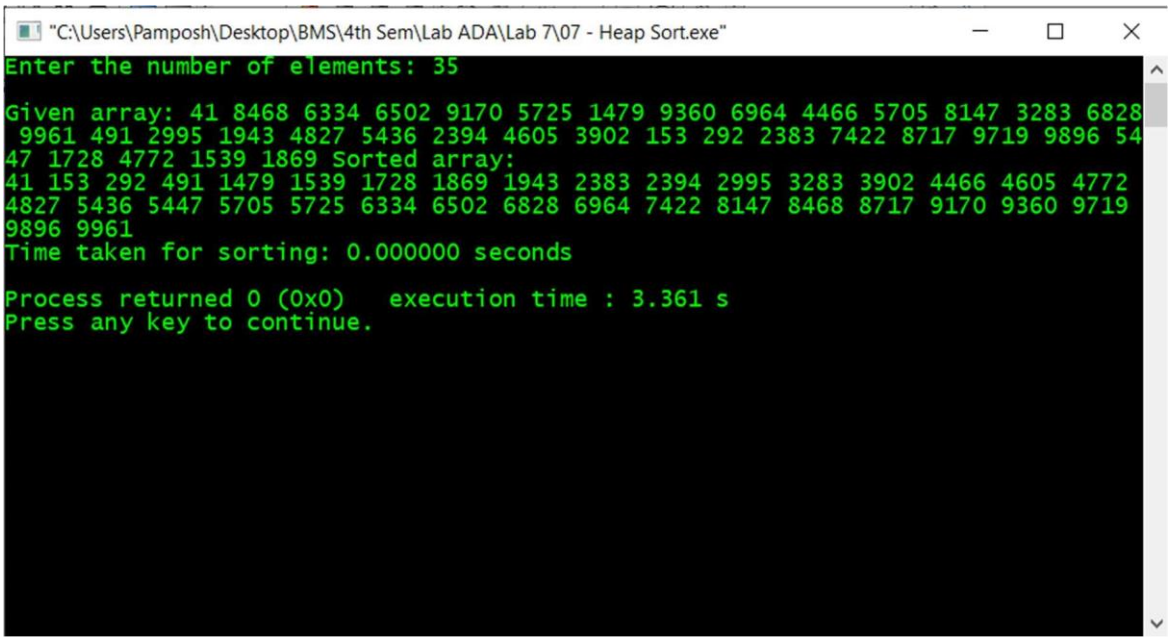


6. Sort a given set of N integer elements using Heap Sort technique and compute its time taken

```c
/*
Sort a given set of N integer elements using Heap Sort technique and compute its time taken
*/

#include <stdio.h>
#include <malloc.h>
#include <time.h>

// Function to heapify a subtree rooted at index i
void heapify(int *arr, int n, int i) {
    int largest = i; // Initialize largest as root
    int left = 2 * i + 1; // left child
    int right = 2 * i + 2; // right child

    // If left child is larger than root
    if (left < n && arr[left] > arr[largest])
        largest = left;

    // If right child is larger than largest so far
    if (right < n && arr[right] > arr[largest])
        largest = right;

    // If largest is not root
    if (largest != i) {
        int temp = arr[i];
        arr[i] = arr[largest];
        arr[largest] = temp;

        // Recursively heapify the affected sub-tree
        heapify(arr, n, largest);
    }
}

// Main function to perform Heap Sort
void heapSort(int *arr, int n) {
    int i, temp;

    // Build heap (rearrange array)
    for (i = n / 2 - 1; i >= 0; i--)
        heapify(arr, n, i);

    // One by one extract elements from heap
    for (i = n - 1; i >= 0; i--) {
        // Move current root to end
        temp = arr[0];
        arr[0] = arr[i];
        arr[i] = temp;

        // call max heapify on the reduced heap
        heapify(arr, i, 0);
    }
}

int main() {
    int n, i, *arr;
    clock_t start_time, end_time;
    printf("Enter the number of elements: ");
    scanf("%d", &n);

    arr = (int*)malloc(n*sizeof(n));

    if(n>10){                   //If number of elements are too large, eg 500, 1000, we take
random values using the rand() function
        for(i=0; i<n; i++)
            arr[i] = rand()%9999;
    }
    else{
        printf("Enter the array:\n");
        for(i=0; i<n; i++)
            scanf("%d", &arr[i]);
        //Input intaking completed
    }

    printf("\nGiven array: ");
    for(i=0; i<n; i++)
        printf("%d ", arr[i]);
    start_time = clock(); // Start time of sorting

    heapSort(arr, n);

    end_time = clock(); // End time of sorting

    printf("Sorted array:\n");
    for (i = 0; i < n; i++) {
```

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

    double time_taken = ((double)(end_time - start_time)) / CLOCKS_PER_SEC;
    printf("\nTime taken for sorting: %f seconds\n", time_taken);

    return 0;
}
```

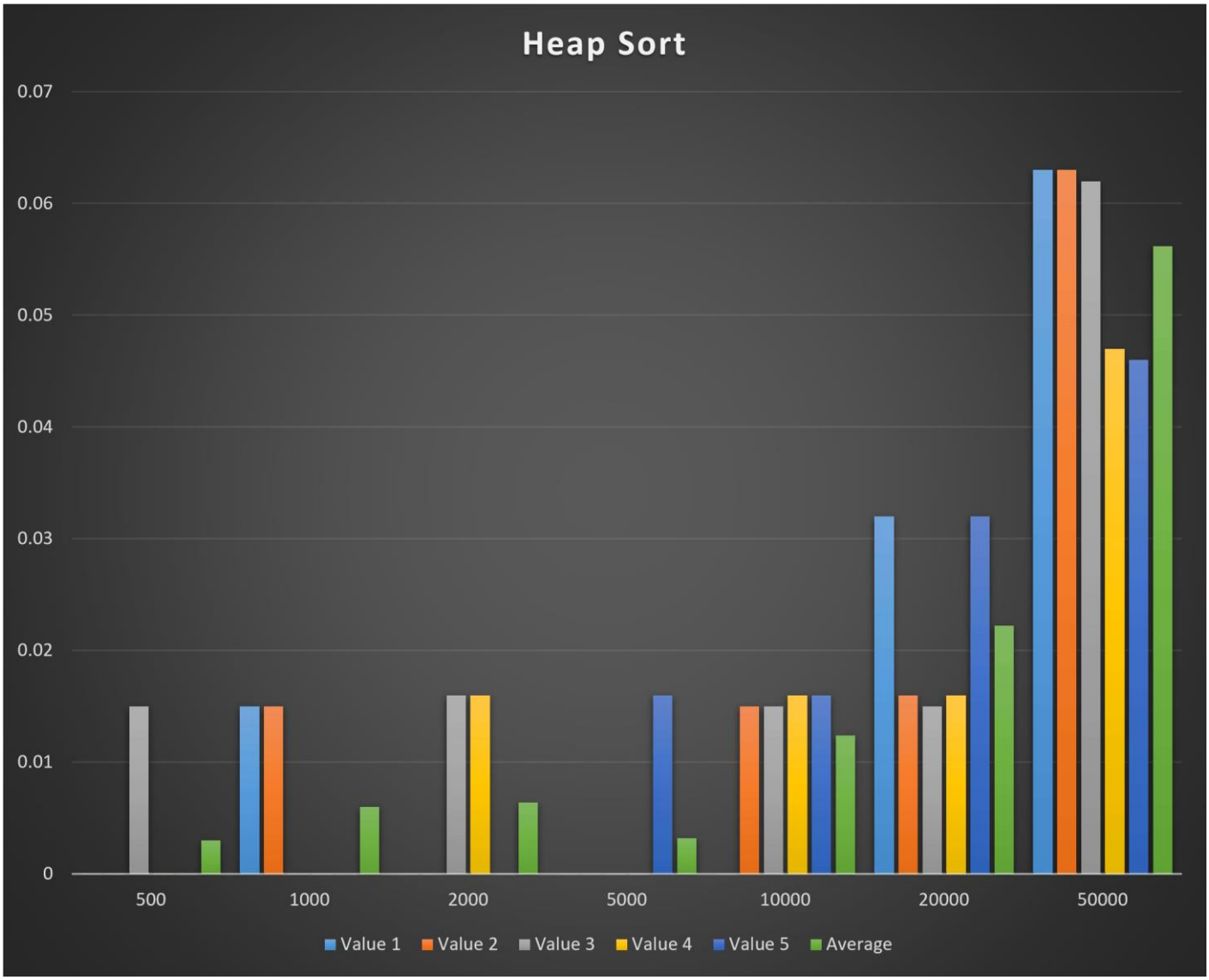




TIME TAKEN FOR VARIOUS VALUES OF N

HEAP SORT	500	1000	2000	5000	10000	20000	50000
Value 1	0.000001	0.015	0.0000001	0.0000001	1E-07	0.032	0.063
Value 2	0.0000001	0.015	0.0000001	0.0000001	0.015	0.016	0.063
Value 3	0.015	0.0000001	0.016	0.0000001	0.015	0.015	0.062

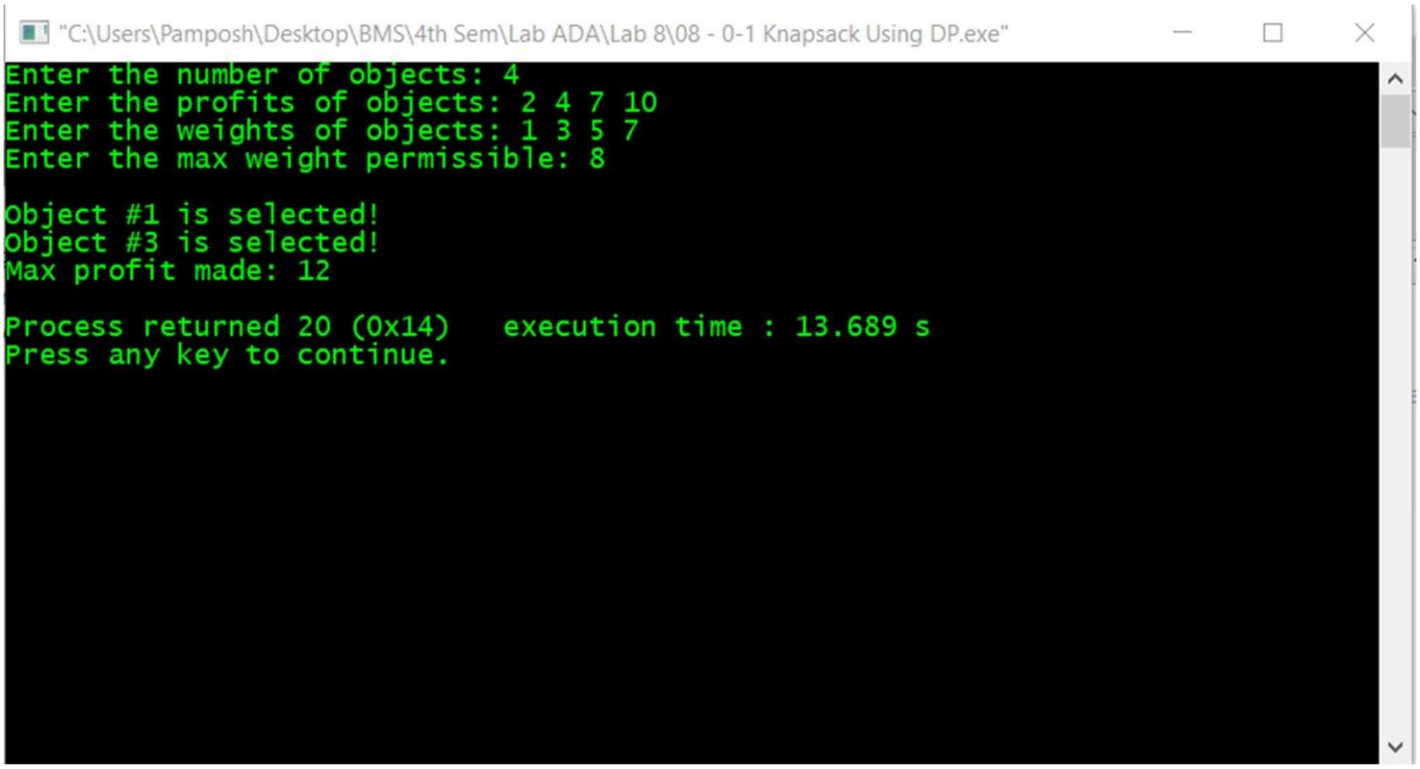
Value 4	0.0000001	0.0000001	0.016	0.0000001	0.016	0.016	0.047
Value 5	0.0000001	0.0000001	0.0000001	0.016	0.016	0.032	0.046
Average	0.0030003	0.00600006	0.0064001	0.00320008	0.0124	0.0222	0.0562

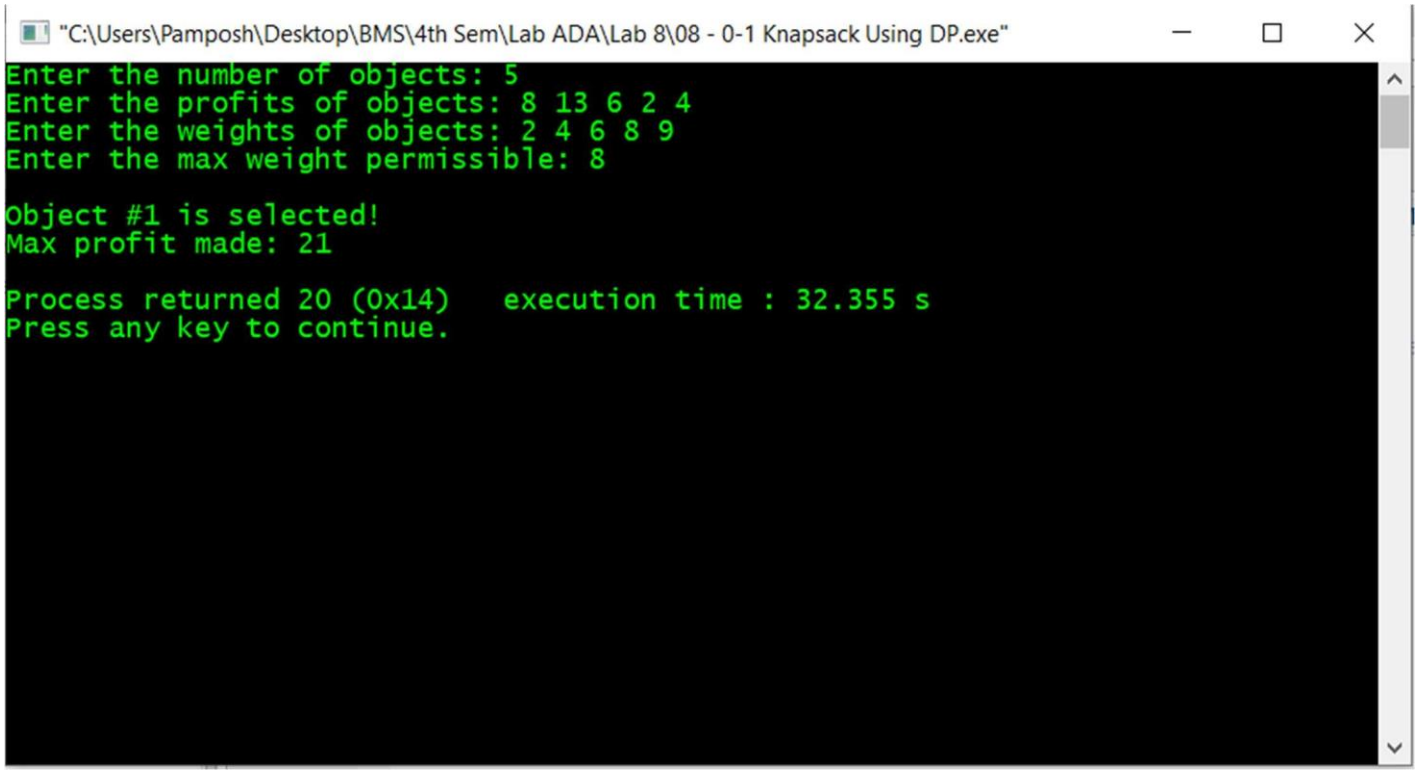


7. Implement 0/1 Knapsack problem using dynamic programming

```c
/*
Implement 0/1 Knapsack problem using dynamic programming
*/

#include<stdio.h>
#include<malloc.h>

//Function returns the maximum out of the two given values
int max(int a, int b){
    return a>=b?a:b;
}

//Function swaps two pointers whichhold a certain value
void swap(int *a, int *b){
    int temp = *a;
    *a = *b;
    *b = temp;
}

//The function is fed with two arrays. it sorts the second array and alongwith swaps the
positions in the first one as well
void sort(int *profit, int *weights, int n){
    int i, j;
    for(i=0; i<n-1; i++){
        for(j=0; j<n-i-1; j++){
            if (weights[j]>weights[j+1]){
                swap(&weights[j], &weights[j+1]);
                swap(&profit[j], &profit[j+1]);
            }
        }
    }
}

//Function to find the objects that are selected
void findObjects(int *weights, int **knapTable, int maxWeight, int n){
    int i, j;
    int obj[n];

    for(i=0; i<n; i++)
        obj[i] = -1;

    i=n;
    j=maxWeight;
    while(i>=0 && j>=0){
        if(knapTable[i][j]!=knapTable[i-1][j]){
            obj[i-2] = 1;
            j -= weights[i-2];
        }
        i -= 1;
    }
    printf("\n");
    for(i=0; i<n; i++){
        if(obj[i]!=-1){
            printf("Object #%d is selected!\n", i+1);
        }
    }
}

//Actual program implementing the knapsack problem
int knapSack(int W, int *weights, int *profit, int n){
    int **knapTable;
    int i, j;

    knapTable = (int**)malloc((n+1)*sizeof(int*));  //Declaring a 2-D matrix for creating the
DP table for the knapsack problem
    for(i=0; i<n+1; i++){
        knapTable[i] = (int*)malloc((W+1)*sizeof(int));
        for(j=0; j<W+1; j++){
            if(i==0 || j == 0)            //If we are in the first row or column (row 0 or col
0), we set that cell to zero
                knapTable[i][j] = 0;
            else{
                if(weights[i-1]<=j){
                    knapTable[i][j] = max(knapTable[i-1][j],
profit[i-1]+knapTable[i-1][j-weights[i-1]]);  //If the weight of the object of the current row is
                    //less or equal to the permissible weight for that column (ie the val of j),
then we have a choice to make. we choose the larger of the profits
                    //among the profit earned in that column till the previous row (ie by not
selecting the current item) and by selecting the item and getting
                    // the rest of the profit(ie W-weight[current row]) from the previous row
items
                }
                else{
                    knapTable[i][j] = knapTable[i-1][j];  //If the current row item can't be
```

```
included as its weight is more than the permissible weight for that col,
                    //we simply add the profit earned till the last row in that column as the
profit for this row as well
                }
            }
        }
    }

    //Calling the function to print all the objects selected
    findObjects(weights, knapTable, W, n);

    return knapTable[n][W];  //return the last cell of the last row as that's the answer
}

void main(){
    int  n, i, maxWeight;
    int  *profit, *weights;

    //Inputting the values
    printf("Enter the number of objects: ");
    scanf("%d", &n);

    profit = (int *)malloc(n*sizeof(int ));
    weights = (int *)malloc(n*sizeof(int ));

    printf("Enter the profits of objects: ");
    for(i=0; i<n; i++)
        scanf("%d", &profit[i]);
    printf("Enter the weights of objects: ");
    for(i=0; i<n; i++)
        scanf("%d", &weights[i]);
    printf("Enter the max weight permissible: ");
    scanf("%d", &maxWeight);
    //Inputting the values ends

    //sort the objects based on their weight
    sort(profit, weights, n);

    //print the max profit made!
    printf("Max profit made: %d\n", knapSack(maxWeight, weights, profit, n));
}
```

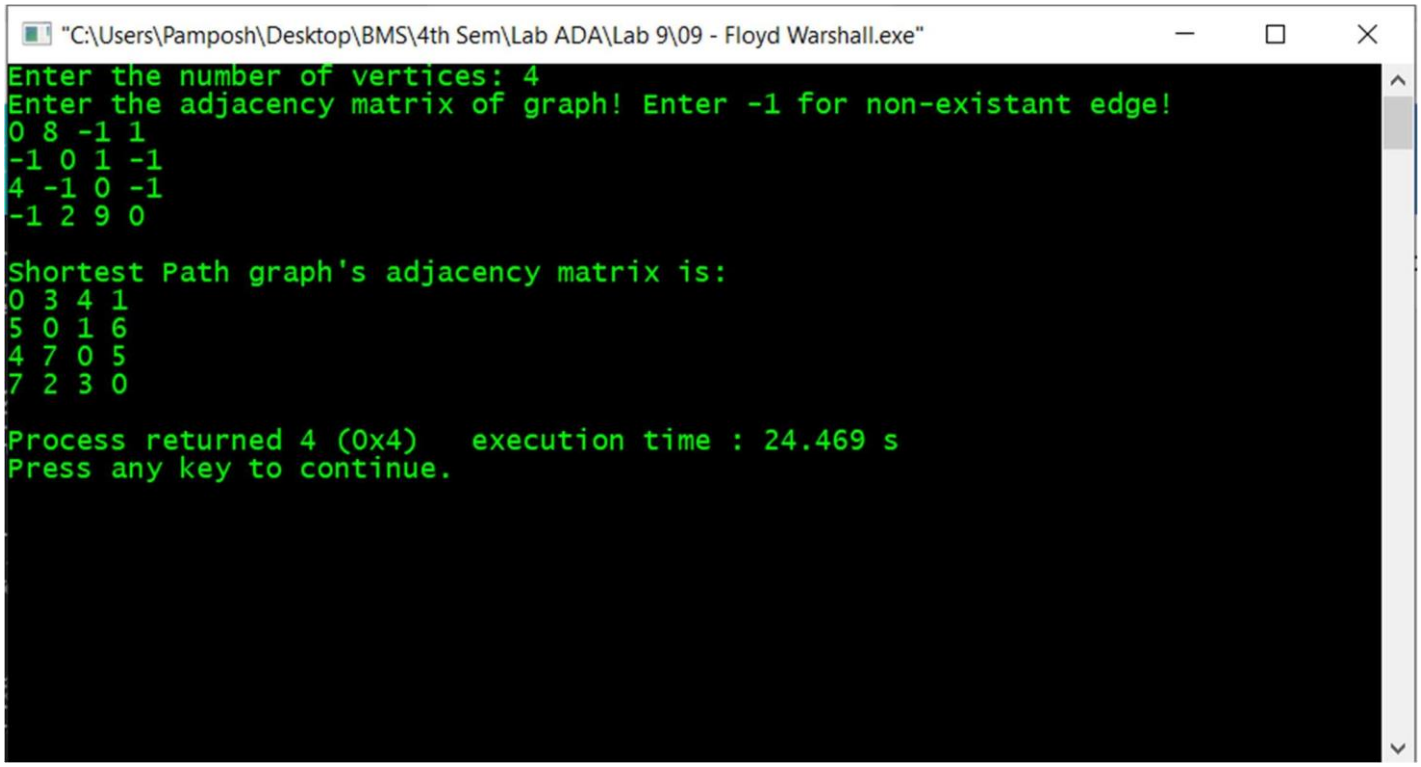

"C:\Users\Pamposh\Desktop\BMS\4th Sem\Lab ADA\Lab 8\08 - 0-1 Knapsack Using DP.exe"
Enter the number of objects: 4
Enter the profits of objects: 2 4 7 10
Enter the weights of objects: 1 3 5 7
Enter the max weight permissible: 8
Object #1 is selected!
Object #3 is selected!
Max profit made: 12
Process returned 20 (0x14) execution time : 13.689 s
Press any key to continue.


"C:\Users\Pamposh\Desktop\BMS\4th Sem\Lab ADA\Lab 8\08 - 0-1 Knapsack Using DP.exe"
Enter the number of objects: 5
Enter the profits of objects: 8 13 6 2 4
Enter the weights of objects: 2 4 6 8 9
Enter the max weight permissible: 8
Object #1 is selected!
Max profit made: 21
Process returned 20 (0x14) execution time : 32.355 s
Press any key to continue.

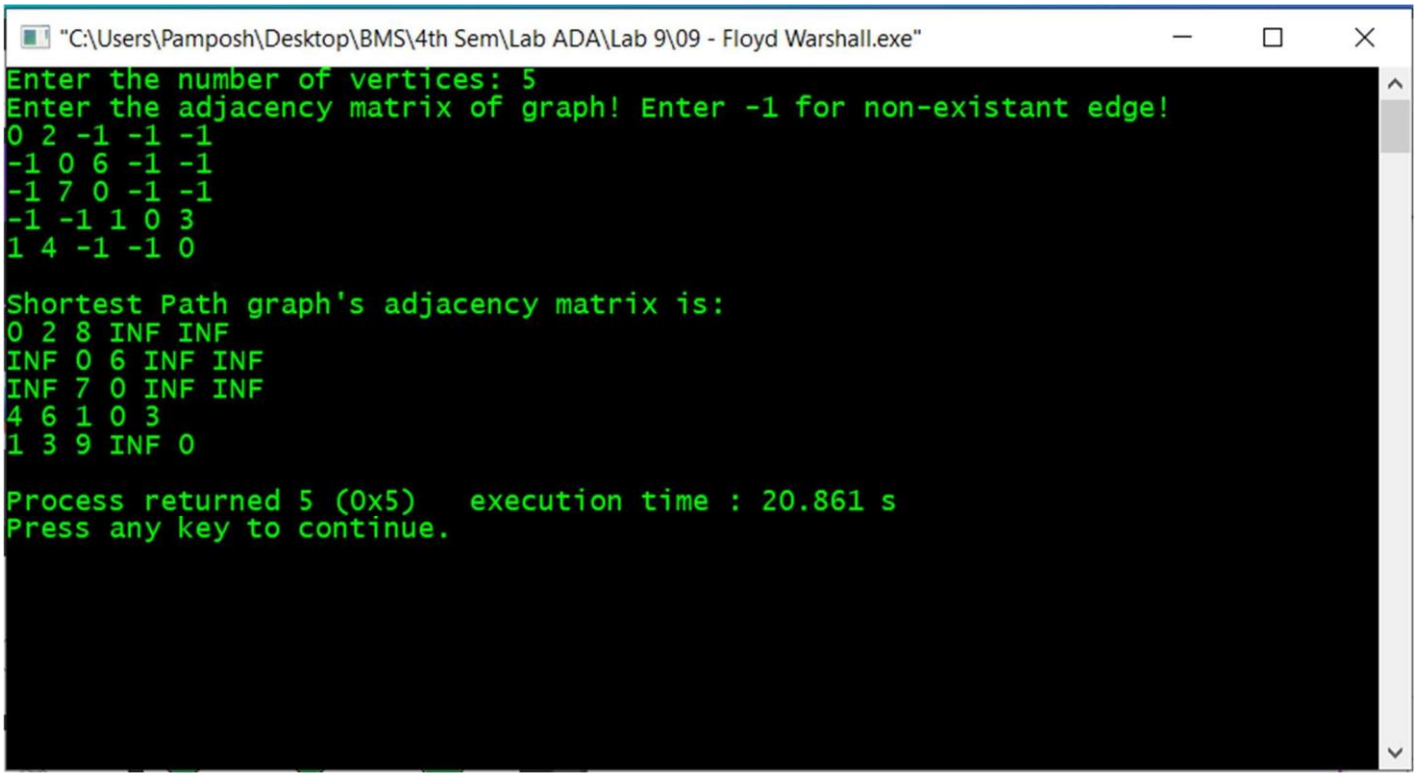
8. Implement All Pair Shortest paths problem using Floyd's algorithm

```c
/*
Implement all pair shortest paths problem using Floyd's algorithm
*/


#include<stdio.h>
#include<malloc.h>
#define INF 99999

//Finding the minimum of two elements
int min(int a, int b){
    return a>b?b:a;
}

//Implementing the main Floyd-Warshall algorithm
void floydWarshall(int **graph, int n){
    int i, j, k;

    for(k=0; k<n; k++){
        for(i=0; i<n; i++){
            for(j=0; j<n; j++){
                graph[i][j] = min(graph[i][j], graph[i][k]+graph[k][j]);
            }
        }
    }
}

//Printing the matrix consisting of minimum distances between any 2 vertices
void printGraph(int **graph, int n){
    int i, j;

    for(i=0; i<n; i++){
        for(j=0; j<n; j++){
            if(graph[i][j] == INF){
                printf("INF ");
            }
            else{
                printf("%d ", graph[i][j]);
            }
        }
        printf("\n");
    }
}

void main(){
    int n, i, j;
    int **graph;

    //Input entering begins
    printf("Enter the number of vertices: ");
    scanf("%d", &n);

    graph = (int**)malloc(n*sizeof(int*));

    printf("Enter the adjacency matrix of graph! Enter -1 for non-existent edge!\n");
    for(i=0; i<n; i++){
        graph[i] = (int*)malloc(n*sizeof(int));
        for(j=0; j<n; j++){
            scanf("%d", &graph[i][j]);
            if(graph[i][j] == -1)
                graph[i][j] = INF;
        }
    }
    //Inputs entering ends

    //Main algorithm implementation
    floydWarshall(graph, n);
    printf("\nShortest Path graph's adjacency matrix is:\n");
    printGraph(graph, n);
}
```

Graph 1


"C:\Users\Pamposh\Desktop\BMS\4th Sem\Lab ADA\Lab 9\09 - Floyd Warshall.exe"
Enter the number of vertices: 4
Enter the adjacency matrix of graph! Enter -1 for non-existant edge!
0 8 -1 1
-1 0 1 -1
4 -1 0 -1
-1 2 9 0
Shortest Path graph's adjacency matrix is:
0 3 4 1
5 0 1 6
4 7 0 5
7 2 3 0
Process returned 4 (0x4) execution time : 24.469 s
Press any key to continue.

Graph 2


"C:\Users\Pamposh\Desktop\BMS\4th Sem\Lab ADA\Lab 9\09 - Floyd Warshall.exe"
Enter the number of vertices: 5
Enter the adjacency matrix of graph! Enter -1 for non-existant edge!
0 2 -1 -1 -1
-1 0 6 -1 -1
-1 7 0 -1 -1
-1 -1 1 0 3
1 4 -1 -1 0
Shortest Path graph's adjacency matrix is:
0 2 8 INF INF
INF 0 6 INF INF
INF 7 0 INF INF
4 6 1 0 3
1 3 9 INF 0
Process returned 5 (0x5) execution time : 20.861 s
Press any key to continue.

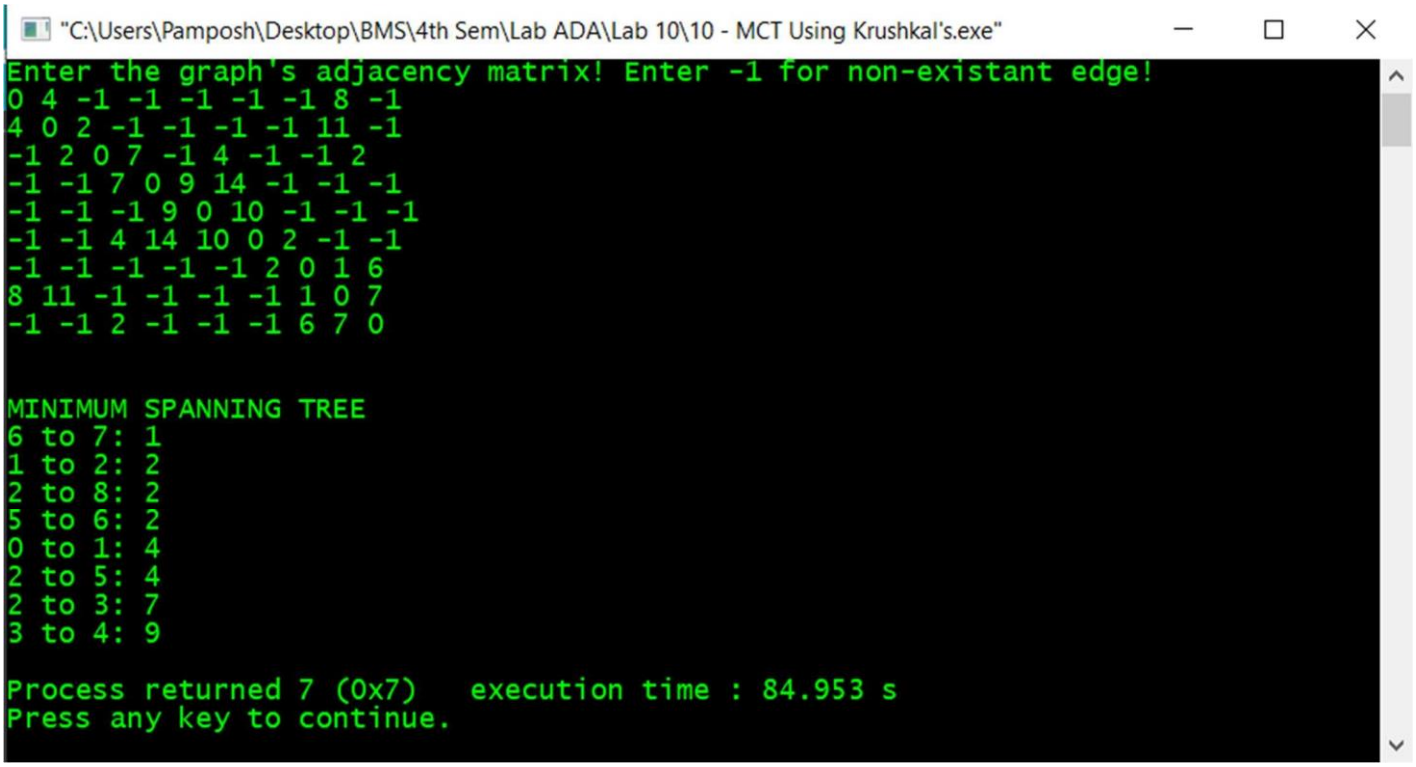
9. Find Minimum Cost Spanning Tree of a given undirected graph using Prim's and Kruskal's algorithm

PRIM's

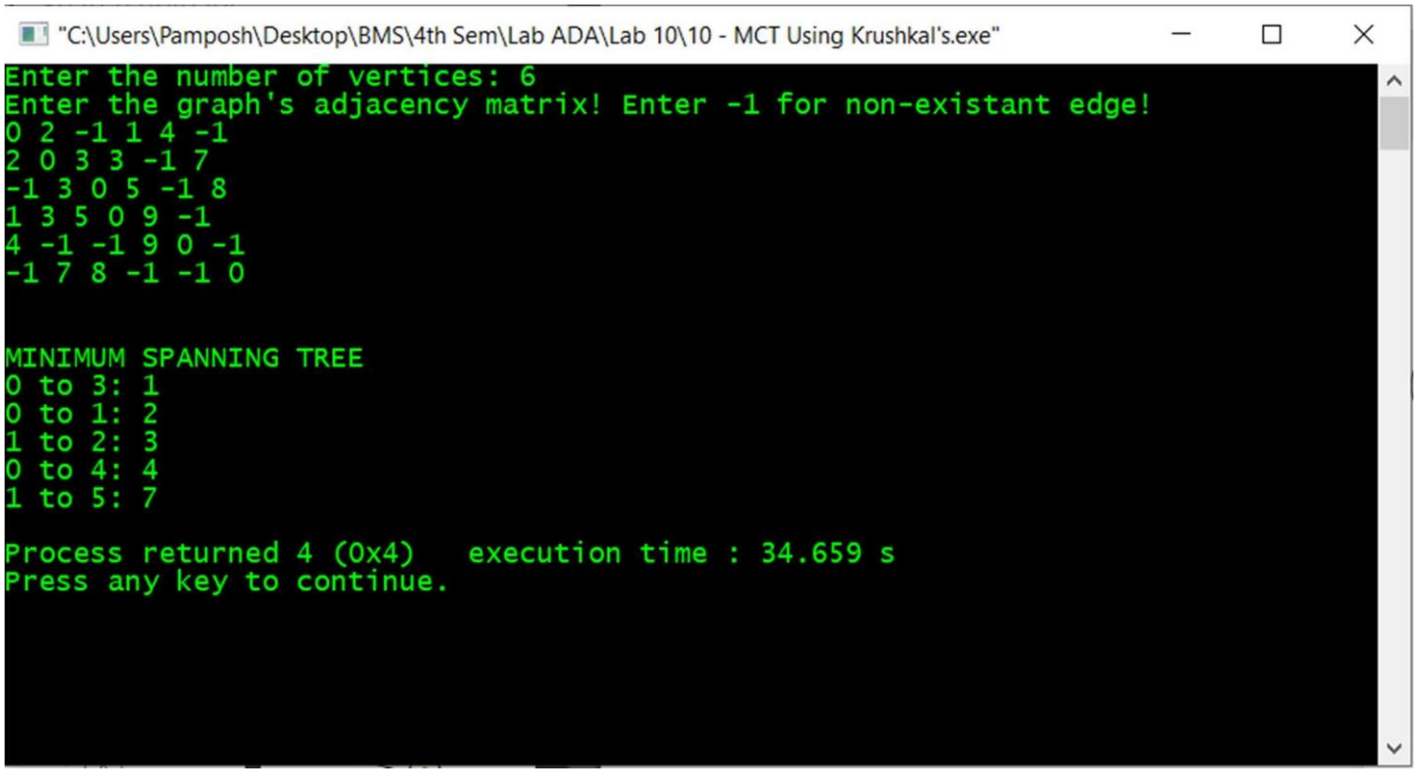
```c
/*
Find Minimum Cost Spanning Tree of a given undirected graph using Krushkal's algorithm
*/

#include<stdio.h>
#include<malloc.h>
#define INF 99999

//This program used DSU (Disjoint Set Union) Data structure

//One of these structures represents one edge (ie. the two vertices and the weight of that edge
typedef struct edge{
    int i, j, w;
}edge;

//This function swaps two edge structures in an array
void swap(edge *a, edge *b){
    edge t = *a;
    *a = *b;
    *b = t;
}

//Function sorts an array of edge structure based on their edge weight using bubble sort
void sort(edge **el, int maxEdges){
    int i, j;
    for(i=0; i<maxEdges+1; i++){
        for(j=0; j<maxEdges-i-1; j++){
            if(el[j]->w > el[j+1]->w){

                swap(el[j], el[j+1]);
            }
        }
    }
}


//Takes the adjacency matrix as input (graph) and converts it into a list of edge structures (ie
it stores the edge in the array el) and returns the number of edges
int convertToEdgeList(int **graph, int n, edge **el){
    int i, j, k=-1;
    for(i=0; i<n; i++){
        for(j=0; j<n; j++){
            if(i<j){        //since we are inputting an undirected graph, the value of edge is
same from u to v as it is from v to u, so we consider only lower triangular matrix
                if(graph[i][j]!=INF){
                    el[++k]->i = i;
                    el[k]->j = j;
                    el[k]->w = graph[i][j];
                }
            }
        }
    }
    return k;   //Returns the index of the last edge inputted
}


//This is used to perform the union of two sets by changing their respective heads
void replace(int v1, int v2, int *set, int n){
    int i, x;

    //There are 4 possibilities for the vertices passed
    //Both can be -1, both can be a certain value, one -1 and other a certain value

    if(set[v1] == -1 && set[v2] == -1){  //If both are -1, set both of their heads to the
lesser value vertex
        set[v1] = v1;
        set[v2] = v1;
    }
    else if(set[v1] == -1){             //If the smaller one is -1, set its head to the other
value
        set[v1] = set[v2];
    }
    else if(set[v2] == -1){             //If the larger one is -1, set its head to the other
value
        set[v2] = set[v1];
    }
    else{                               //If both of them are non-negative, we are merging two
disjoint sets. So we change the head of all those vertices whose head is same as v2 to the head
of v1
        x = set[v2];    //We store the actual value of set[v2] because it will be changed later
and we can no longer use it then
        for(i=0; i<n; i++){
            if(set[i] == x){
                set[i] = set[v1];
```

```
            }
        }
    }
}

void main(){
    int n, i, j, k, maxEdges, head1, head2;
    int **graph, *set;
    edge **el;      //We create a pointer which points to an array of pointers, each of which
are pointing to a structure containing the definition of an edge

    //Inputting the values
    printf("Enter the number of vertices: ");
    scanf("%d", &n);

    //If there are n vertices, at max there are nC2 edges, or max n*(n-1)/2 edges
    maxEdges = n*(n-1)/2;
    el = (edge**)malloc(maxEdges*sizeof(edge*));
    set = (int*)malloc(n*sizeof(int));
    for(i=0; i<maxEdges; i++)
        el[i] = (edge*)malloc(sizeof(edge));

    graph = (int**)malloc(n*sizeof(int*));
    printf("Enter the graph's adjacency matrix! Enter -1 for non-existent edge!\n");
    for(i=0; i<n; i++){
        graph[i] = (int*)malloc(n*sizeof(int));
        for(j=0; j<n; j++){
            scanf("%d", &graph[i][j]);
            if(graph[i][j] == -1){
                graph[i][j] = INF;
            }
        }
        set[i] = -1;            //Setting all vertices as not selected!
    }
    //Inputting completed!

    maxEdges = convertToEdgeList(graph, n, el) + 1;   //The function converts the given
adjacency matrix of graph into a list of edges and returns the index of the last edge
    //maxEdges now represents the number of edges in the graphs

    //Sorts the edges based on their weights in increasing order
    sort(el, maxEdges);

    set[el[0]->i] = el[0]->i;
    set[el[0]->j] = el[0]->i;  //We mark the 2 vertices of the minimum edge as selected

    printf("\n\nMINIMUM SPANNING TREE\n");
    printf("%d to %d: %d\n", el[0]->i, el[0]->j, el[0]->w);

    k=0;
    while(k<n-2){   //Number of edges in minimum spanning tree is n-1, and one of them is
already selected, is el[0]!
        for(i=0; i<maxEdges; i++){
            head1 = set[el[i]->i];
            head2 = set[el[i]->j];
            if((head1 != head2) || (head1 == -1 && head2 == -1)){
                printf("%d to %d: %d\n", el[i]->i, el[i]->j, el[i]->w);
                replace(el[i]->i, el[i]->j, set, n);
                k++;
                if(k==n-2)
                    break;
            }
        }
    }
}
```

Graph 1


"C:\Users\Pamposh\Desktop\BMS\4th Sem\Lab ADA\Lab 10\10 - MCT Using Krushkal's.exe"
Enter the graph's adjacency matrix! Enter -1 for non-existant edge!
0 4 -1 -1 -1 -1 -1 8 -1
4 0 2 -1 -1 -1 -1 11 -1
-1 2 0 7 -1 4 -1 -1 2
-1 -1 7 0 9 14 -1 -1 -1
-1 -1 -1 9 0 10 -1 -1 -1
-1 -1 4 14 10 0 2 -1 -1
-1 -1 -1 -1 -1 2 0 1 6
8 11 -1 -1 -1 -1 1 0 7
-1 -1 2 -1 -1 -1 6 7 0
MINIMUM SPANNING TREE
6 to 7: 1
1 to 2: 2
2 to 8: 2
5 to 6: 2
0 to 1: 4
2 to 5: 4
2 to 3: 7
3 to 4: 9
Process returned 7 (0x7) execution time : 84.953 s
Press any key to continue.

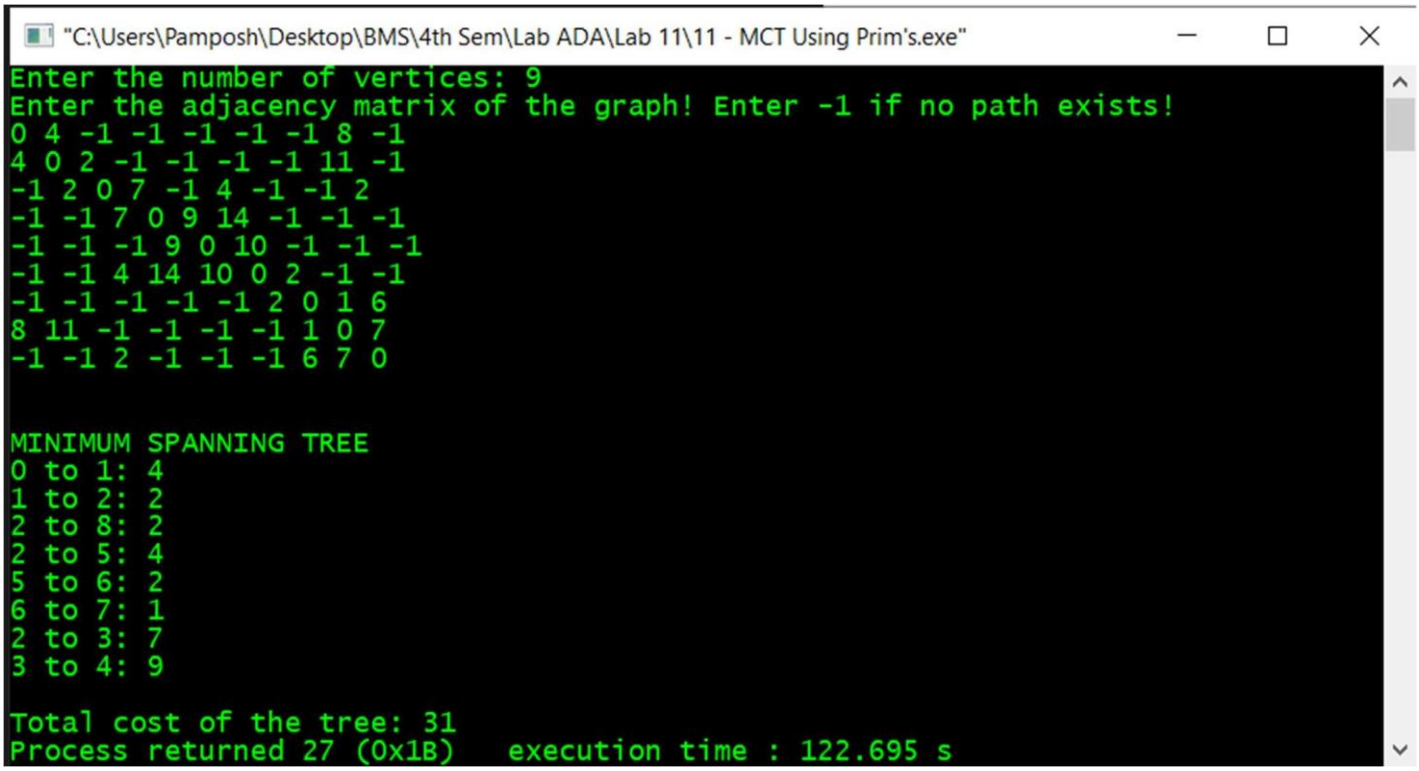
Graph 2


"C:\Users\Pamposh\Desktop\BMS\4th Sem\Lab ADA\Lab 10\10 - MCT Using Krushkal's.exe"
Enter the number of vertices: 6
Enter the graph's adjacency matrix! Enter -1 for non-existant edge!
0 2 -1 1 4 -1
2 0 3 3 -1 7
-1 3 0 5 -1 8
1 3 5 0 9 -1
4 -1 -1 9 0 -1
-1 7 8 -1 -1 0
MINIMUM SPANNING TREE
0 to 3: 1
0 to 1: 2
1 to 2: 3
0 to 4: 4
1 to 5: 7
Process returned 4 (0x4) execution time : 34.659 s
Press any key to continue.

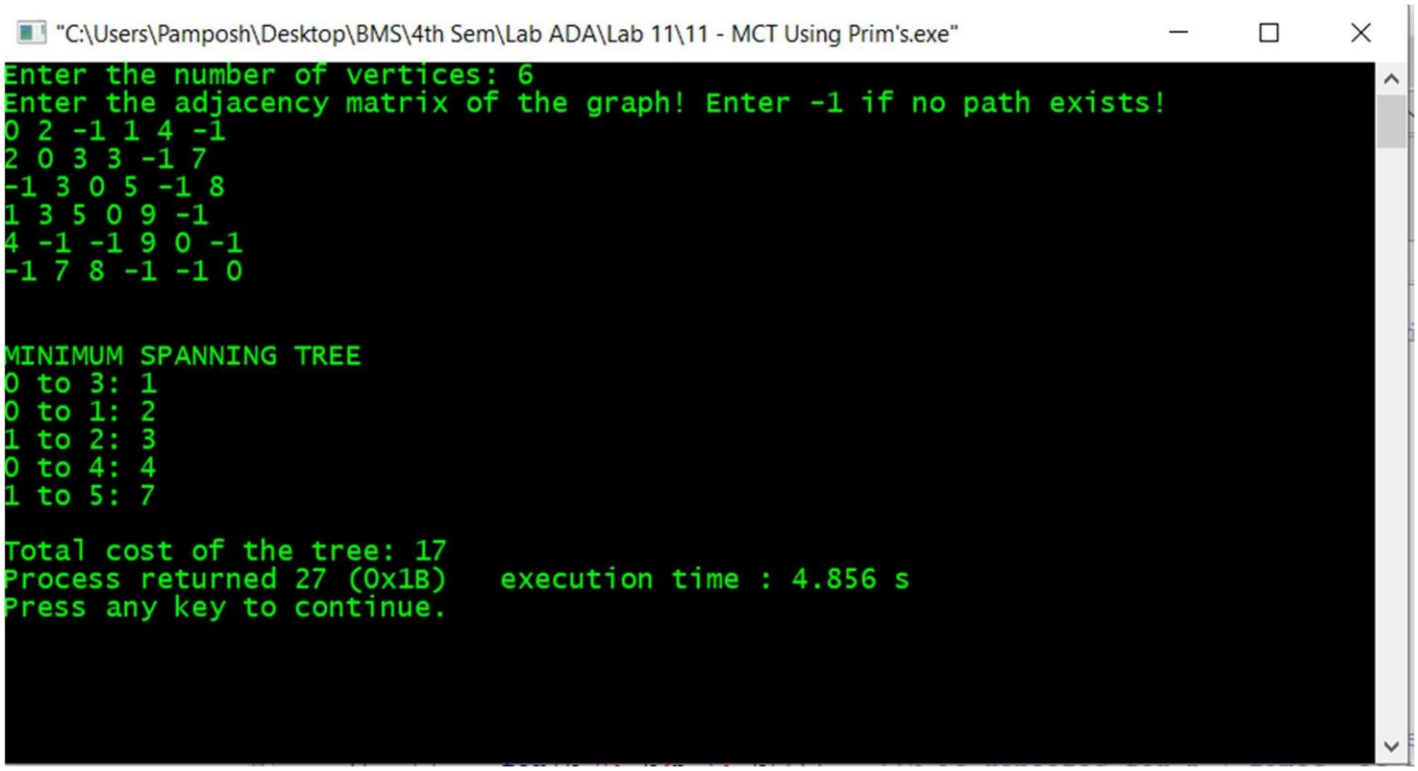
Krushkal's

```c
/*
Find Minimum Cost Spanning tree of a given undirected graph using Prim's Algorithm
*/

#include<stdio.h>
#include<malloc.h>
#define INF 99999

int prims(int **graph, int *selected, int n){
    int k, i, j, minDistance, x, y, sum = 0;

    selected[0] = 1;    //Make the 1st vertex as selected

    printf("\n\nMINIMUM SPANNING TREE\n");
    //In each iteration of this for loop we add one edge to the minimum spanning tree
    for(k=0; k<n-1; k++){    //k is repeated for n-1 times, as the number of edges in a minimum
spanning tree is n-1

        //We need to get the vertice adjacent to the currently selected vertex
        minDistance = INF;
        x = 0;
        y = 0;

        for(i=0; i<n; i++){
            if(selected[i]){
                //This checks all those vertices that are already selected and the adjacent
vertices to them
                for(j=0; j<n; j++){     //This loop now finds the distances between the current
i vertex and every other non-selected vertex
                    if(!selected[j] && graph[i][j]){
                        if(graph[i][j]<minDistance){
                            minDistance = graph[i][j];    //Minimum distance for the current
selected vertex (ie k vertex)
                            x = i;              //x is one of the vertices of the minimum
distance
                            y = j;              //y is other vertex of the minimum distance
                        }
                    }
                }
            }
        }

        printf("%d to %d: %d\n", x, y, graph[x][y]);
        sum += graph[x][y];
        selected[y] = 1;
    }
    return sum;
}

void main(){
    int **graph, *selected;
    int n, i, j;

    printf("Enter the number of vertices: ");
    scanf("%d", &n);

    selected = (int*)malloc(n*sizeof(int));

    printf("Enter the adjacency matrix of the graph! Enter -1 if no path exists!!\n");
    graph = (int**)malloc(n*sizeof(int*));
    for(i=0; i<n; i++){
        graph[i] = (int*)malloc(n*sizeof(int));
        for(j=0; j<n; j++){
            scanf("%d", &graph[i][j]);
            if(graph[i][j] == -1){
                graph[i][j] = INF;
            }
        }
        selected[i] = 0;
    }
    printf("\nTotal cost of the tree: %d", prims(graph, selected, n));
}
```

Graph 1


"C:\Users\Pamposh\Desktop\BMS\4th Sem\Lab ADA\Lab 11\11 - MCT Using Prim's.exe"
Enter the number of vertices: 9
Enter the adjacency matrix of the graph! Enter -1 if no path exists!
0 4 -1 -1 -1 -1 -1 8 -1
4 0 2 -1 -1 -1 -1 11 -1
-1 2 0 7 -1 4 -1 -1 2
-1 -1 7 0 9 14 -1 -1 -1
-1 -1 -1 9 0 10 -1 -1 -1
-1 -1 4 14 10 0 2 -1 -1
-1 -1 -1 -1 -1 2 0 1 6
8 11 -1 -1 -1 -1 1 0 7
-1 -1 2 -1 -1 -1 6 7 0
MINIMUM SPANNING TREE
0 to 1: 4
1 to 2: 2
2 to 8: 2
2 to 5: 4
5 to 6: 2
6 to 7: 1
2 to 3: 7
3 to 4: 9
Total cost of the tree: 31
Process returned 27 (0x1B) execution time : 122.695 s

Graph 2

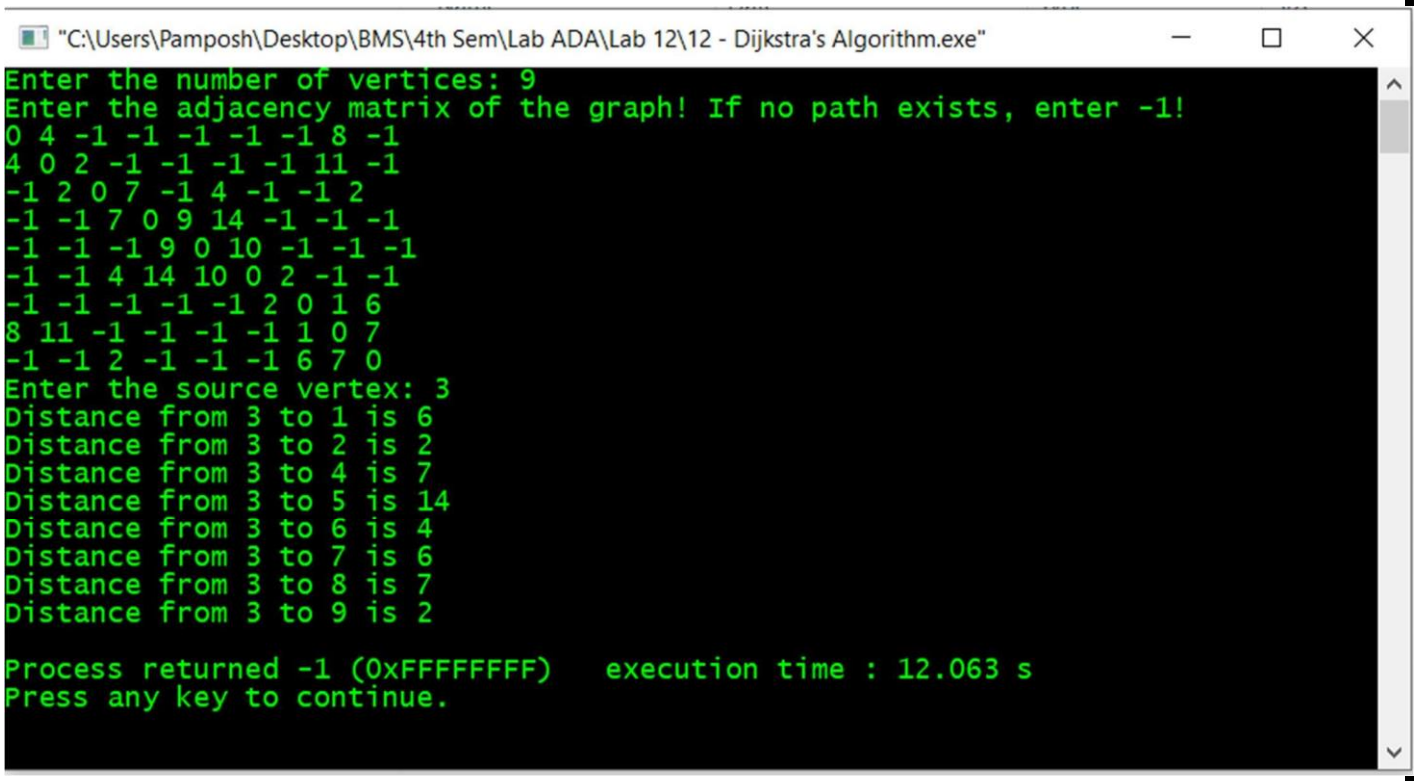

"C:\Users\Pamposh\Desktop\BMS\4th Sem\Lab ADA\Lab 11\11 - MCT Using Prim's.exe"
Enter the number of vertices: 6
Enter the adjacency matrix of the graph! Enter -1 if no path exists!
0 2 -1 1 4 -1
2 0 3 3 -1 7
-1 3 0 5 -1 8
1 3 5 0 9 -1
4 -1 -1 9 0 -1
-1 7 8 -1 -1 0
MINIMUM SPANNING TREE
0 to 3: 1
0 to 1: 2
1 to 2: 3
0 to 4: 4
1 to 5: 7
Total cost of the tree: 17
Process returned 27 (0x1B) execution time : 4.856 s
Press any key to continue.

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

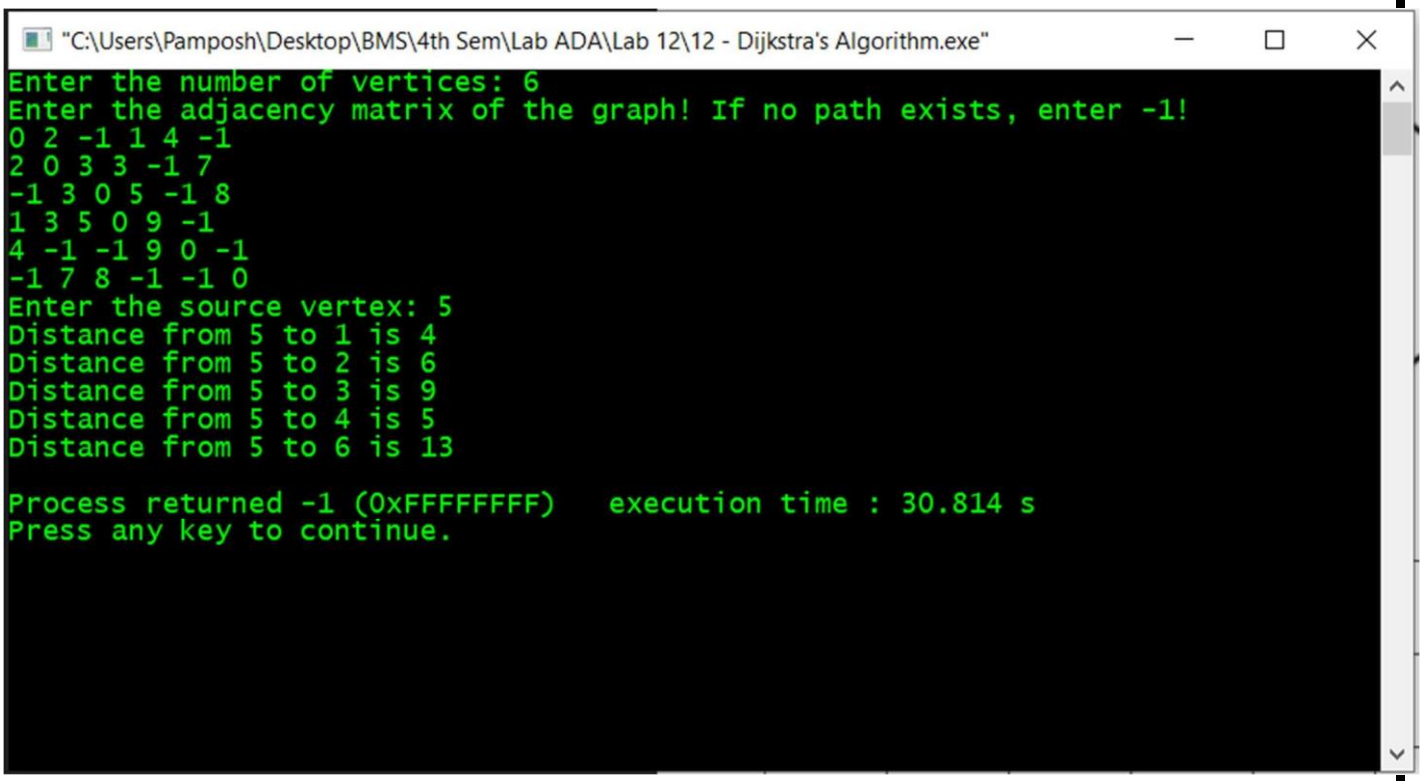
```c
/*
From a given vertex in a weighted connected graph, find shortest paths to other vertices using
Dijkstra's algorithm
*/

#include<stdio.h>
#include<malloc.h>
#define INF 99999

//Function to return the minimum of 2 integers
int min(int a, int b){
    return a<=b?a:b;
}

//Function to implement the Dijkstra's algorithm
void dijkstra(int **graph, int source, int n){
    int distance[n], visited[n];    //Distance matrix stores the values of distances between the
source and every other vertex
    //visited array stores whether a node has been visited or not

    int i, j, minDistance, nextNode; /*minDistance is set to infinity at start and during the
iteration,
    the vertices which is closest to the currently selected vertice is found*/

    //Now we set the distance of every vertice from the source as per the input matrix (ie
direct distance between source and vertex
    for(i=0; i<n; i++){
        distance[i] = graph[source][i];
        visited[i] = 0;                 //Set every vertex as 0 (not visited)
    }

    distance[source] = 0;          //distance from source to source is 0
    visited[source] = 1;         //the source vertex is now visited

    /*this for loop runs for n-2 times (ie from 0 to n-3).
    We don't run it for n times because the source vertex is already selected and the last node
will be automatically selected*/
    for(i=0; i<n-2; i++){
        minDistance = INF;

        //This for loop finds the vertex nearest to the source (since distance[i] is distance
between i and source)
        for(j=0; j<n; j++){
            if(distance[j] < minDistance && !visited[j]){
                minDistance = distance[j];
                nextNode = j;
            }
        }
        //nextNode here now represents the node closest to the source

        //nextNode is now visited
        visited[nextNode] = 1;

        /*Next for loop checks whether distance between source and any other vertex
        is lesser (ie distance[i]) or distance from source to nextNode + nextNode to other
vertex is lesser*/
        for(j=0; j<n; j++){
            if(!visited[j]){
                distance[j] = min(distance[j], minDistance + graph[nextNode][j]);
            }
        }

        //With completion of this iteration, one vertex (closest to the source is selected and
visited completed. Next the second closest will be selected
        //This is similar to the Floyd-Warshall algorithm
    }


    //Printing the distance between any two vertices
    for(i=0; i<n; i++){
        if(i != source){
            printf("Distance from %d to %d is %d\n", source+1, i+1, distance[i]);
        }
    }
}

int main(){
    int **graph;
    int n, i, j, source;

    //Inputting the graph
    printf("Enter the number of vertices: ");
    scanf("%d", &n);
```

```
    graph = (int**)malloc(n*sizeof(int*));

    printf("Enter the adjacency matrix of the graph! If no path exists, enter -1!\n");
    for(i=0; i<n; i++){
        graph[i] = (int*)malloc(n*sizeof(int));
        for(j=0; j<n; j++){
            scanf("%d", &graph[i][j]);
            if(graph[i][j] == -1){
                graph[i][j] = INF;
            }
        }
    }

    printf("Enter the source vertex: ");
    scanf("%d", &source);
    //Inputting ends


    dijkstra(graph, source-1, n);   //we subtract 1 from the source to adjust for the
0-indexing in the arrays

    return -1;
}
```

Graph 1

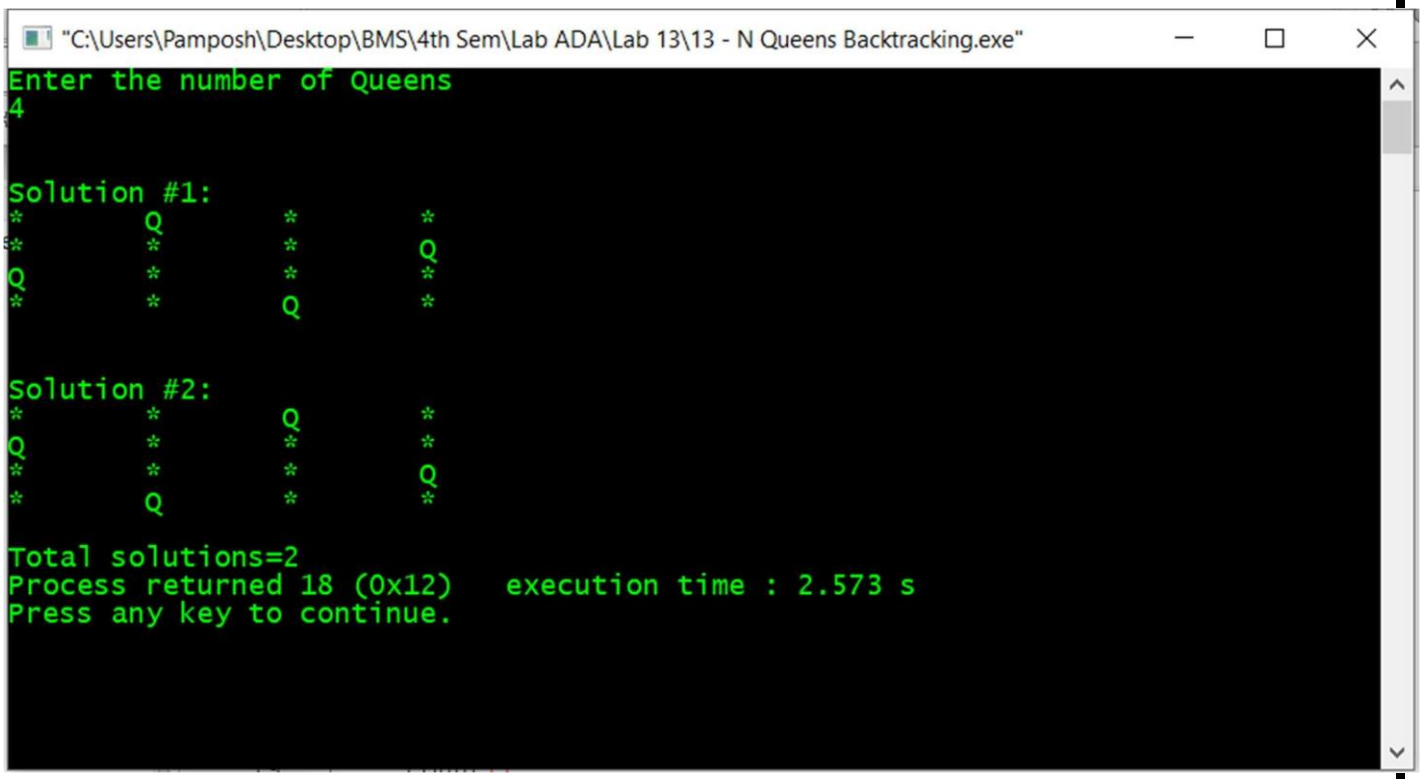

"C:\Users\Pamposh\Desktop\BMS\4th Sem\Lab ADA\Lab 12\12 - Dijkstra's Algorithm.exe"
Enter the number of vertices: 9
Enter the adjacency matrix of the graph! If no path exists, enter -1!
0 4 -1 -1 -1 -1 -1 8 -1
4 0 2 -1 -1 -1 -1 11 -1
-1 2 0 7 -1 4 -1 -1 2
-1 -1 7 0 9 14 -1 -1 -1
-1 -1 -1 9 0 10 -1 -1 -1
-1 -1 4 14 10 0 2 -1 -1
-1 -1 -1 -1 -1 2 0 1 6
8 11 -1 -1 -1 -1 1 0 7
-1 -1 2 -1 -1 -1 6 7 0
Enter the source vertex: 3
Distance from 3 to 1 is 6
Distance from 3 to 2 is 2
Distance from 3 to 4 is 7
Distance from 3 to 5 is 14
Distance from 3 to 6 is 4
Distance from 3 to 7 is 6
Distance from 3 to 8 is 7
Distance from 3 to 9 is 2
Process returned -1 (0xFFFFFFFF) execution time : 12.063 s
Press any key to continue.

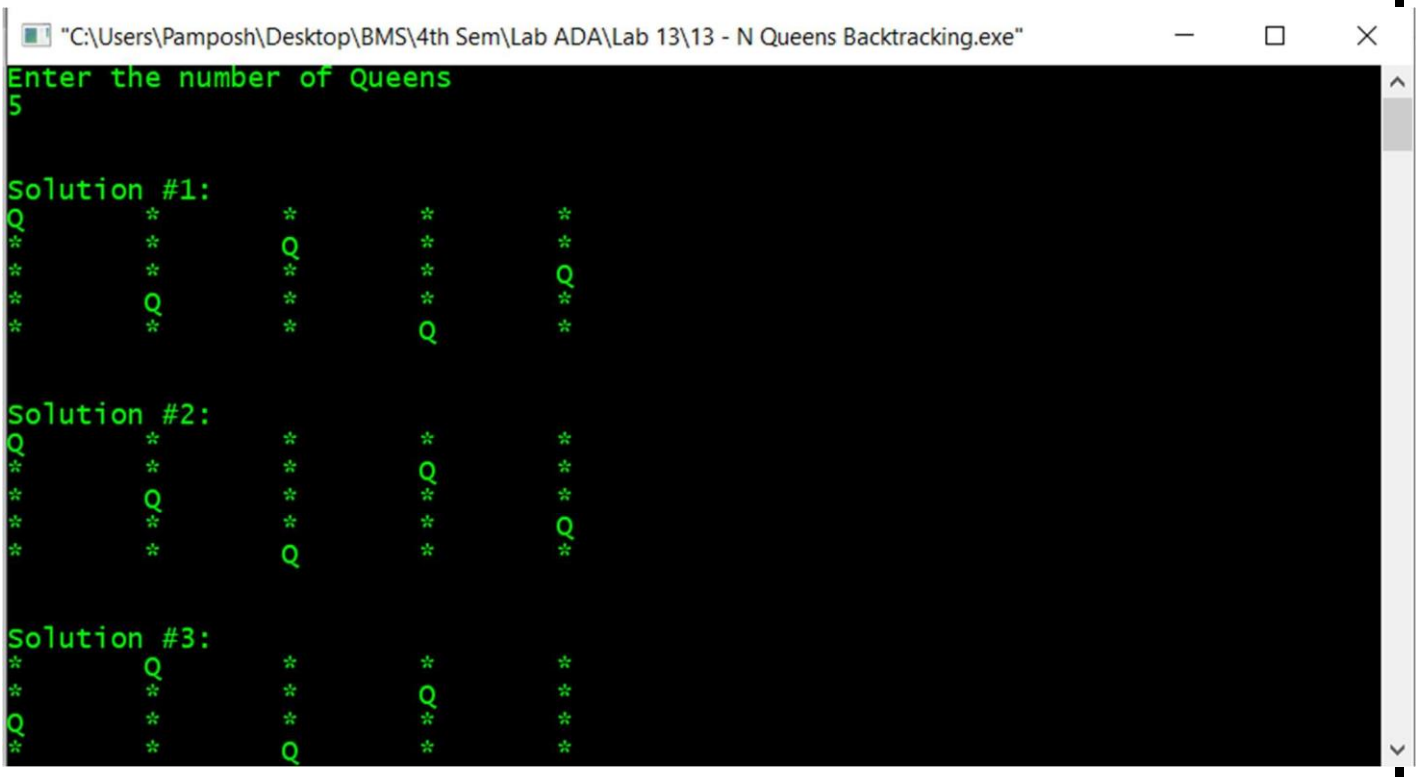
Graph 2

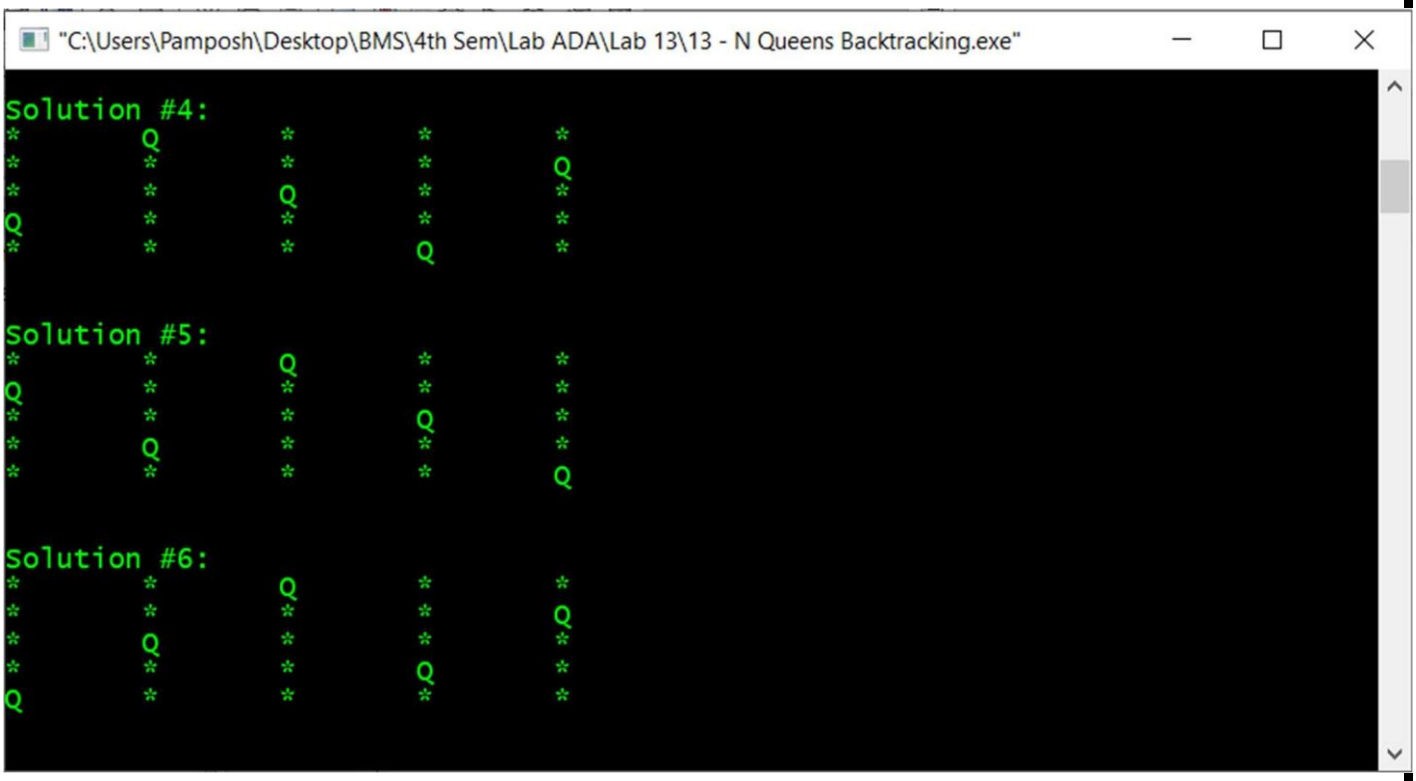

"C:\Users\Pamposh\Desktop\BMS\4th Sem\Lab ADA\Lab 12\12 - Dijkstra's Algorithm.exe"
Enter the number of vertices: 6
Enter the adjacency matrix of the graph! If no path exists, enter -1!
0 2 -1 1 4 -1
2 0 3 3 -1 7
-1 3 0 5 -1 8
1 3 5 0 9 -1
4 -1 -1 9 0 -1
-1 7 8 -1 -1 0
Enter the source vertex: 5
Distance from 5 to 1 is 4
Distance from 5 to 2 is 6
Distance from 5 to 3 is 9
Distance from 5 to 4 is 5
Distance from 5 to 6 is 13
Process returned -1 (0xFFFFFFFF) execution time : 30.814 s
Press any key to continue.

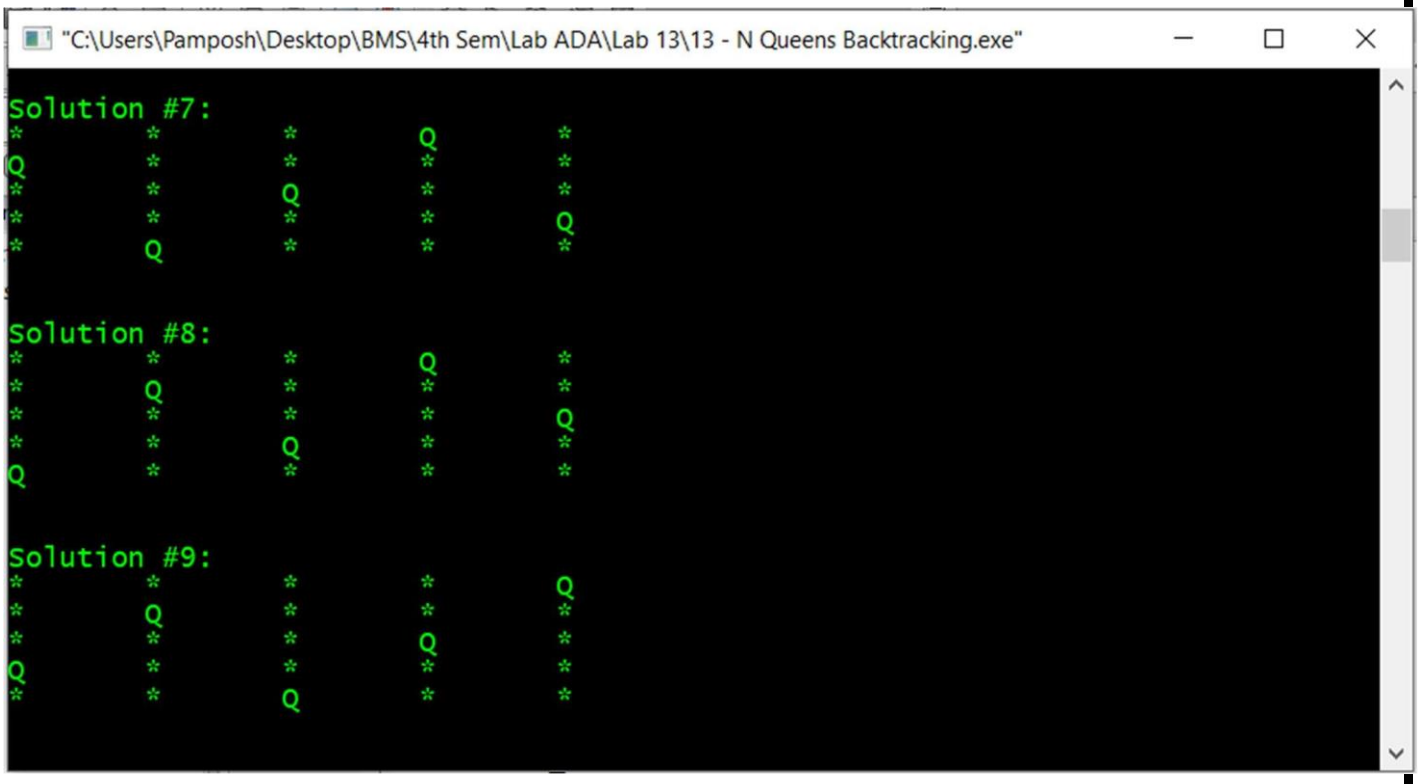
11. Implement “N-Queens Problem” using Backtracking

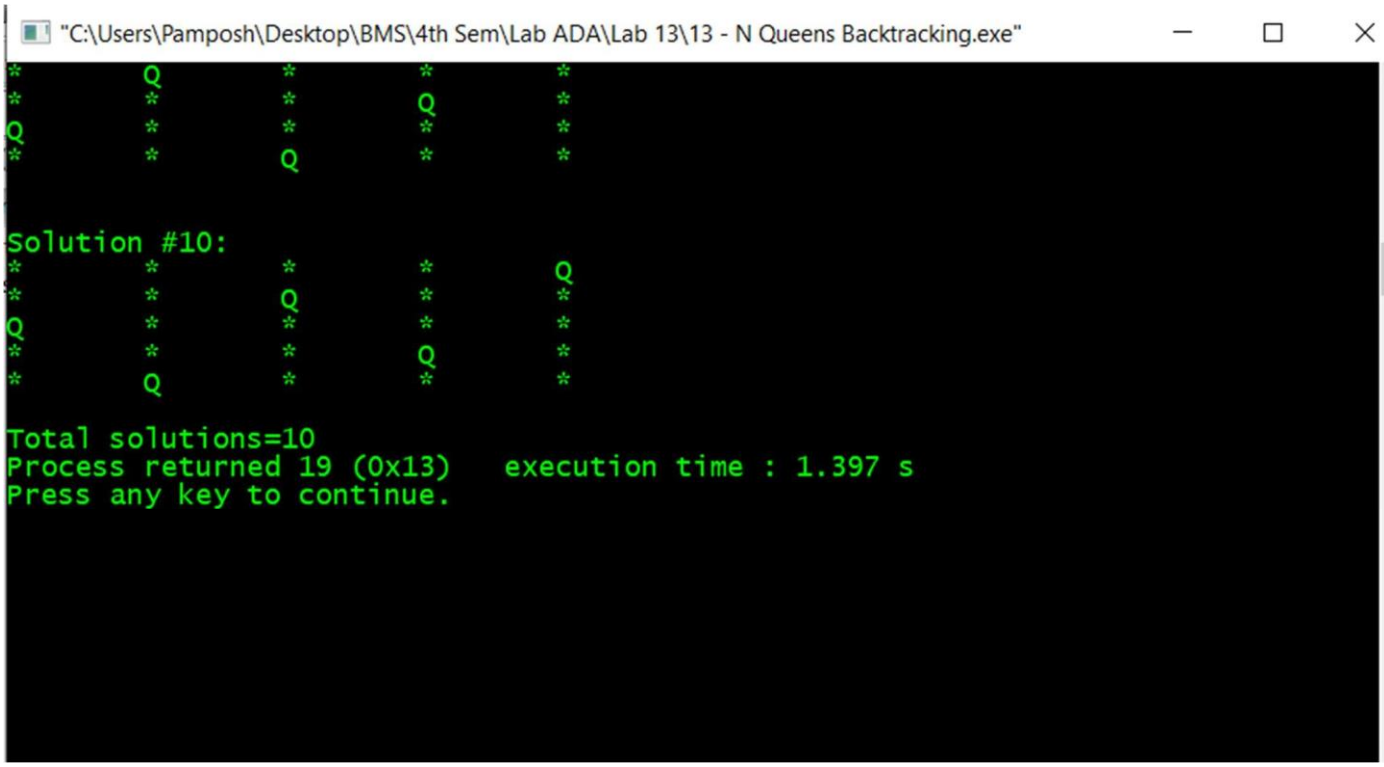
```c
/*
Implement "N-Queens Problem" using Backtracking
*/

#include<stdio.h>
#include<conio.h>
#include<math.h>
int a[30],count=0;
int place(int pos) {
    int i;
    for (i=1;i<pos;i++) {
        if((a[i]==a[pos])||((abs(a[i]-a[pos])==abs(i-pos))))
            return 0;
    }
    return 1;
}
void print_sol(int n) {
    int i,j;
    count++;
    printf("\n\nSolution #%d:\n",count);
    for (i=1;i<=n;i++) {
        for (j=1;j<=n;j++) {
            if(a[i]==j)
                printf("Q\t"); else
                printf("*\t");
        }
        printf("\n");
    }
}
void queen(int n) {
    int k=1;
    a[k]=0;
    while(k!=0) {
        a[k]=a[k]+1;
        while((a[k]<=n)&&(place(k)))
            a[k]++;
        if(a[k]<=n) {
            if(k==n)
                print_sol(n); else {
                k++;
                a[k]=0;
            }
        } else
            k--;
    }
}
void main() {
    int i,n;
    printf("Enter the number of Queens\n");
    scanf("%d",&n);
    queen(n);
    printf("\nTotal solutions=%d",count);
}
```


"C:\Users\Pamposh\Desktop\BMS\4th Sem\Lab ADA\Lab 13\13 - N Queens Backtracking.exe"
Enter the number of Queens
4
Solution #1:
* Q * *
* * * Q
Q * * *
* * Q *
Solution #2:
* * Q *
Q * * *
* * * Q
* Q * *
Total solutions=2
Process returned 18 (0x12) execution time : 2.573 s
Press any key to continue.

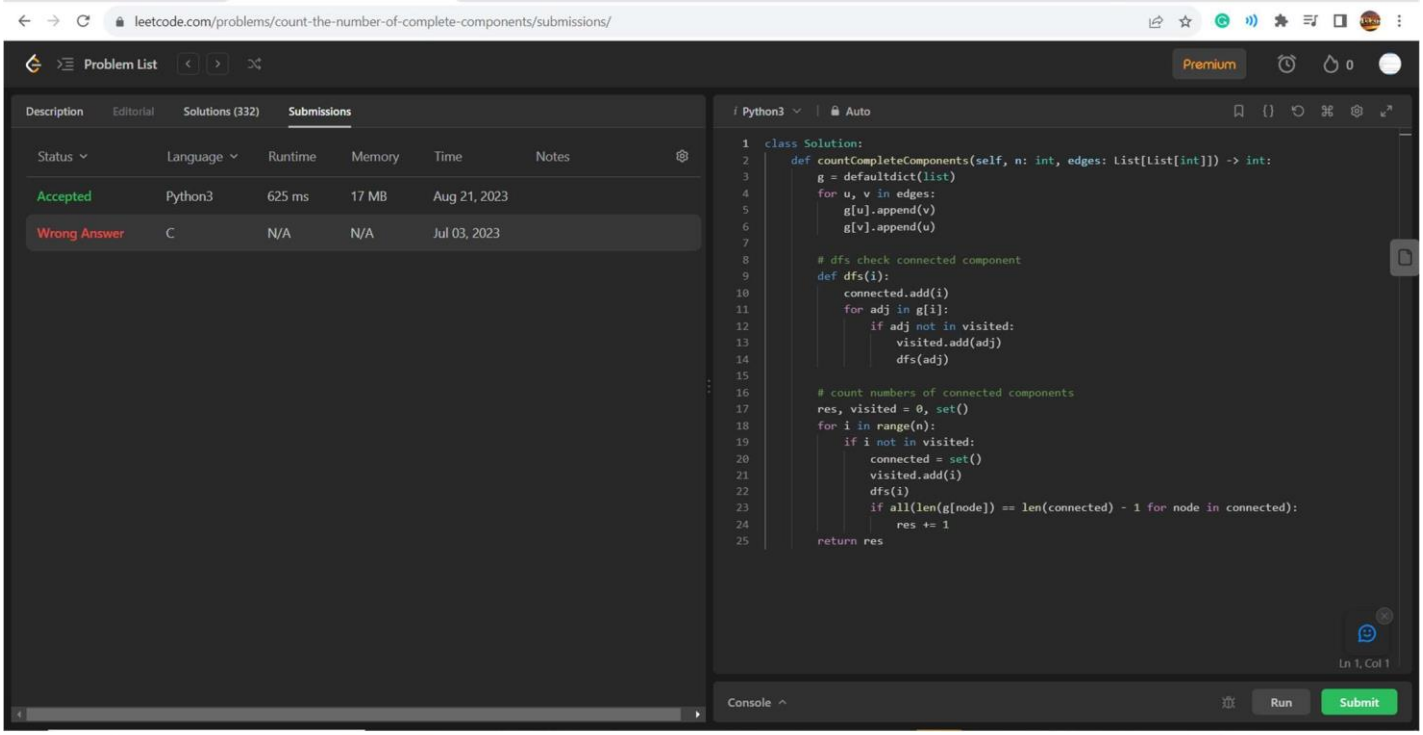

"C:\Users\Pamposh\Desktop\BMS\4th Sem\Lab ADA\Lab 13\13 - N Queens Backtracking.exe"
Enter the number of Queens
5
Solution #1:
Q * * * *
* * Q * *
* * * * Q
* Q * * *
* * * Q *
Solution #2:
Q * * * *
* * * Q *
* Q * * *
* * * * Q
* * Q * *
Solution #3:
* Q * * *
* * * Q *
Q * * * *
* * Q * *

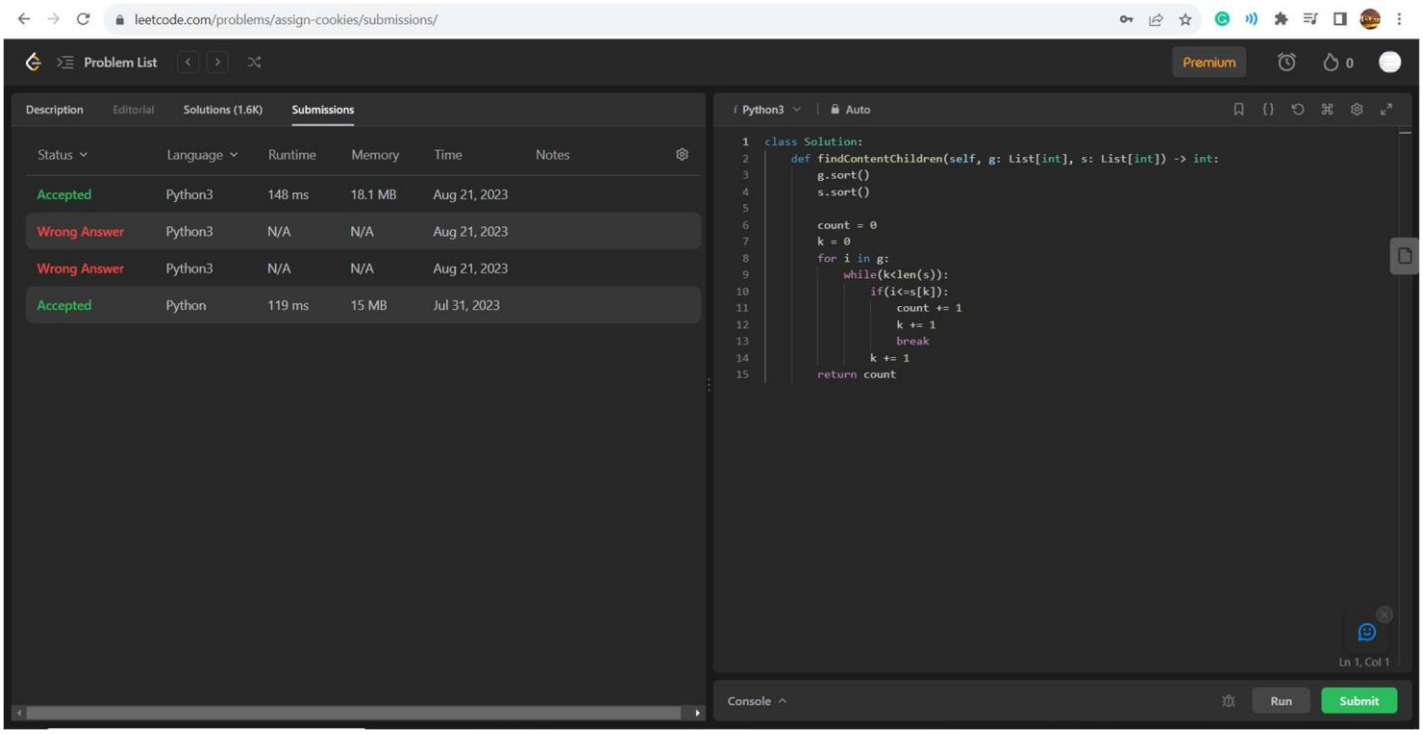

"C:\Users\Pamposh\Desktop\BMS\4th Sem\Lab ADA\Lab 13\13 - N Queens Backtracking.exe"
Solution #4:
* Q * * *
* * * * Q
* * Q * *
Q * * * *
* * * Q *
Solution #5:
* * Q * *
Q * * * *
* * * Q *
* Q * * *
* * * * Q
Solution #6:
* * Q * *
* * * * Q
* Q * * *
* * * Q *
Q * * * *

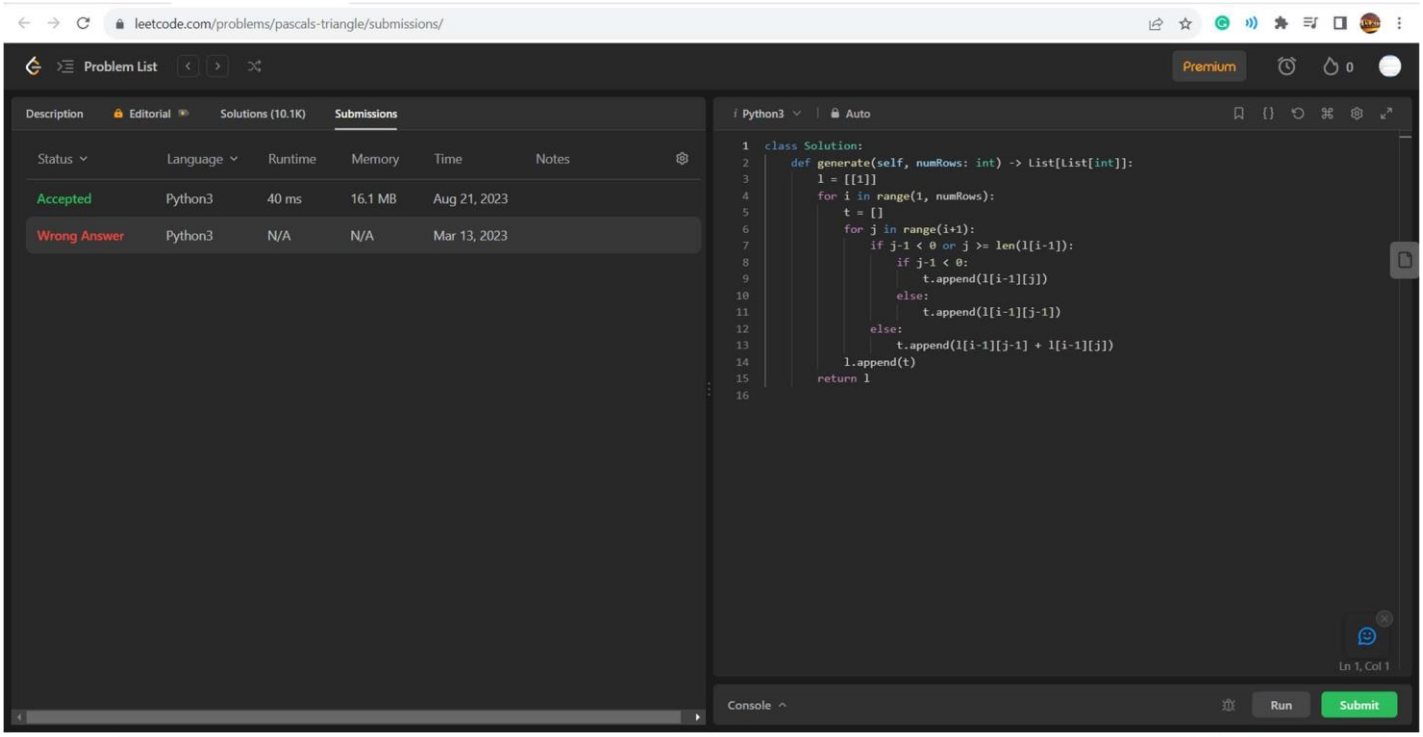

"C:\Users\Pamposh\Desktop\BMS\4th Sem\Lab ADA\Lab 13\13 - N Queens Backtracking.exe"
Solution #7:
* * * Q *
Q * * * *
* * Q * *
* * * * Q
* Q * * *
Solution #8:
* * * Q *
* Q * * *
* * * * Q
* * Q * *
Q * * * *
Solution #9:
* * * * Q
* Q * * *
* * * Q *
Q * * * *
* * Q * *

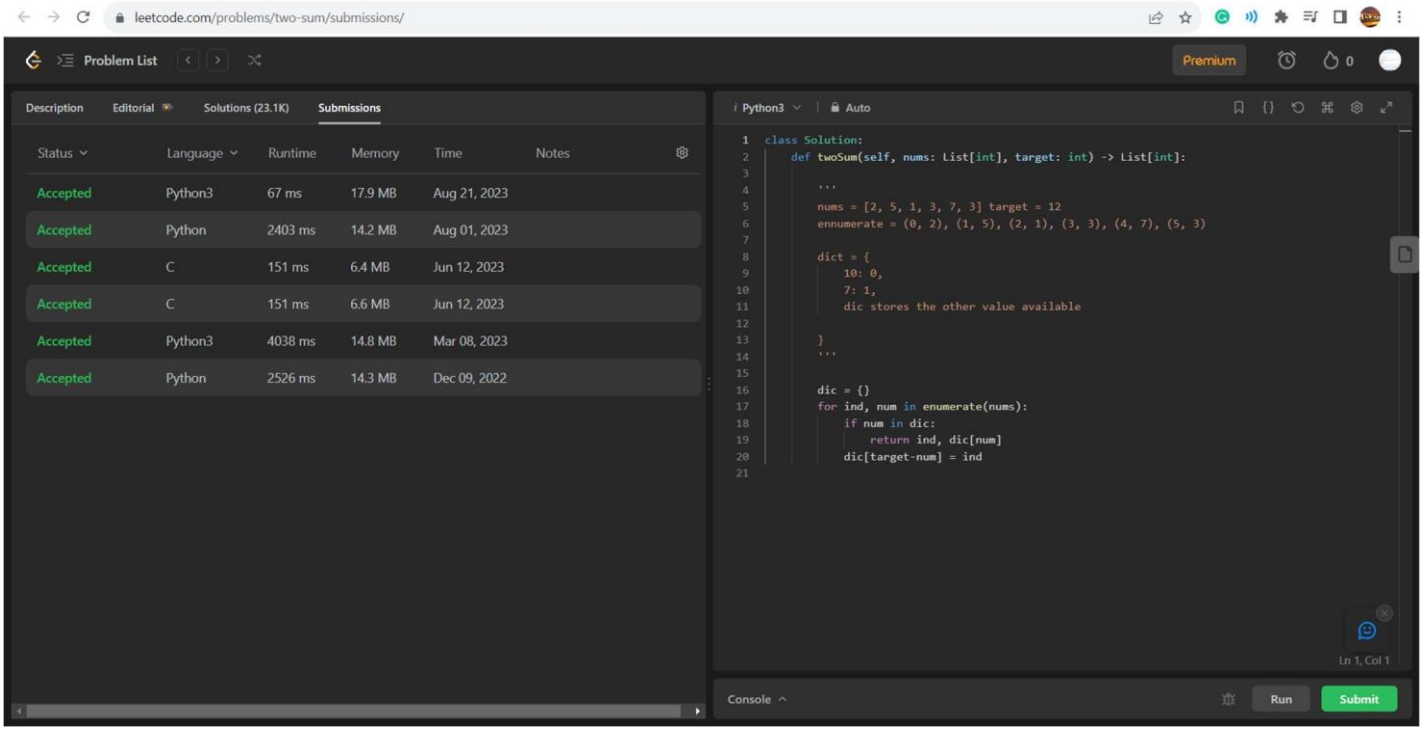

"C:\Users\Pamposh\Desktop\BMS\4th Sem\Lab ADA\Lab 13\13 - N Queens Backtracking.exe"
* Q * * *
* * * Q *
Q * * * *
* * Q * *
Solution #10:
* * * * Q
* * Q * *
Q * * * *
* * * Q *
* Q * * *
Total solutions=10
Process returned 19 (0x13) execution time : 1.397 s
Press any key to continue.

LEET CODE QUESTIONS






leetcode.com/problems/pascals-triangle/submissions/
Problem List
Premium
Description
Editorial
Solutions (10.1K)
Submissions
Status
Language
Runtime
Memory
Time
Notes
Accepted
Python3
40 ms
16.1 MB
Aug 21, 2023
Wrong Answer
Python3
N/A
N/A
Mar 13, 2023
Python3
Auto
class Solution:
def generate(self, numRows: int) -> List[List[int]]:
l = [[1]]
for i in range(1, numRows):
t = []
for j in range(i+1):
if j-1 < 0 or j >= len(l[i-1]):
if j-1 < 0:
t.append(l[i-1][j])
else:
t.append(l[i-1][j-1])
else:
t.append(l[i-1][j-1] + l[i-1][j])
l.append(t)
return l
Ln 1, Col 1
Console
Run
Submit


leetcode.com/problems/two-sum/submissions/
Problem List
Premium
Description
Editorial
Solutions (23.1K)
Submissions
Status
Language
Runtime
Memory
Time
Notes
Accepted
Python3
67 ms
17.9 MB
Aug 21, 2023
Accepted
Python
2403 ms
14.2 MB
Aug 01, 2023
Accepted
C
151 ms
6.4 MB
Jun 12, 2023
Accepted
C
151 ms
6.6 MB
Jun 12, 2023
Accepted
Python3
4038 ms
14.8 MB
Mar 08, 2023
Accepted
Python
2526 ms
14.3 MB
Dec 09, 2022
Python3
Auto
class Solution:
def twoSum(self, nums: List[int], target: int) -> List[int]:
'''
nums = [2, 5, 1, 3, 7, 3] target = 12
ennumerate = (0, 2), (1, 5), (2, 1), (3, 3), (4, 7), (5, 3)
dict = {
10: 0,
7: 1,
dic stores the other value available
}
'''
dic = {}
for ind, num in enumerate(nums):
if num in dic:
return ind, dic[num]
dic[target-num] = ind
Ln 1, Col 1
Console
Run
Submit