**VISVESVARAYA TECHNOLOGICAL UNIVERSITY**

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



**LAB REPORT**

**on**

Analysis and Design of Algorithms

***Submitted by***

**Anagha K S (1BM21CS021)**

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

**BACHELOR OF ENGINEERING**

***in***

**COMPUTER SCIENCE AND ENGINEERING**



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

**(Autonomous Institution under VTU)**

**BENGALURU-560019**

**May-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 Anagha K S(1BM21CS021), 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 May-2023 to July-2023. The Lab report has been approved as it satisfies the academic requirements in respect of a Analysis and Design of Algorithms (22CS4PCADA) work prescribed for the said degree.**

**Sunayna S 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:**   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.** | **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.** | **20** |
| **6** | **Sort a given set of N integer elements using Heap Sort technique and compute its time taken.** | **23** |
| **7** | **Implement 0/1 Knapsack problem using dynamic programming.** | **28** |
| **8**  **9** | **Implement All Pair Shortest paths problem using Floyd’s algorithm.** | **30** |
| **Find Minimum Cost Spanning Tree of a given undirected graph using Prim’s and Kruskal’s algorithm.** | **32** |
| **10** | **From a given vertex in a weighted connected graph, find shortest paths to other vertices using Dijkstra’s algorithm.** | **38** |
| **11** | **Implement “N-Queens Problem” using Backtracking.** | **41** |

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

#include<stdio.h>

int i,j,n,visited[10],queue[10],front=0,rear=-1; int adj[10][10];

void bfs(int v)

{

for(i=1;i<=n;i++) if(adj[v][i] && !visited[i]) queue[++rear]=i; if(front<=rear) visited[queue[front]]=1; bfs(queue[front++]);

}

}

void main()

{

int v;

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

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

{

queue[i]=0; visited[i]=0;

}

printf("Enter the graph data in adjacent matrix form \n"); for(i=1;i<=n;i++)

{

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

{

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

}

}

printf("Enter the starting vertex\n"); scanf("%d",&v);

bfs(v); printf("Traversal ");

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

{

if(visited[i]) printf("%d\t",i); else

{

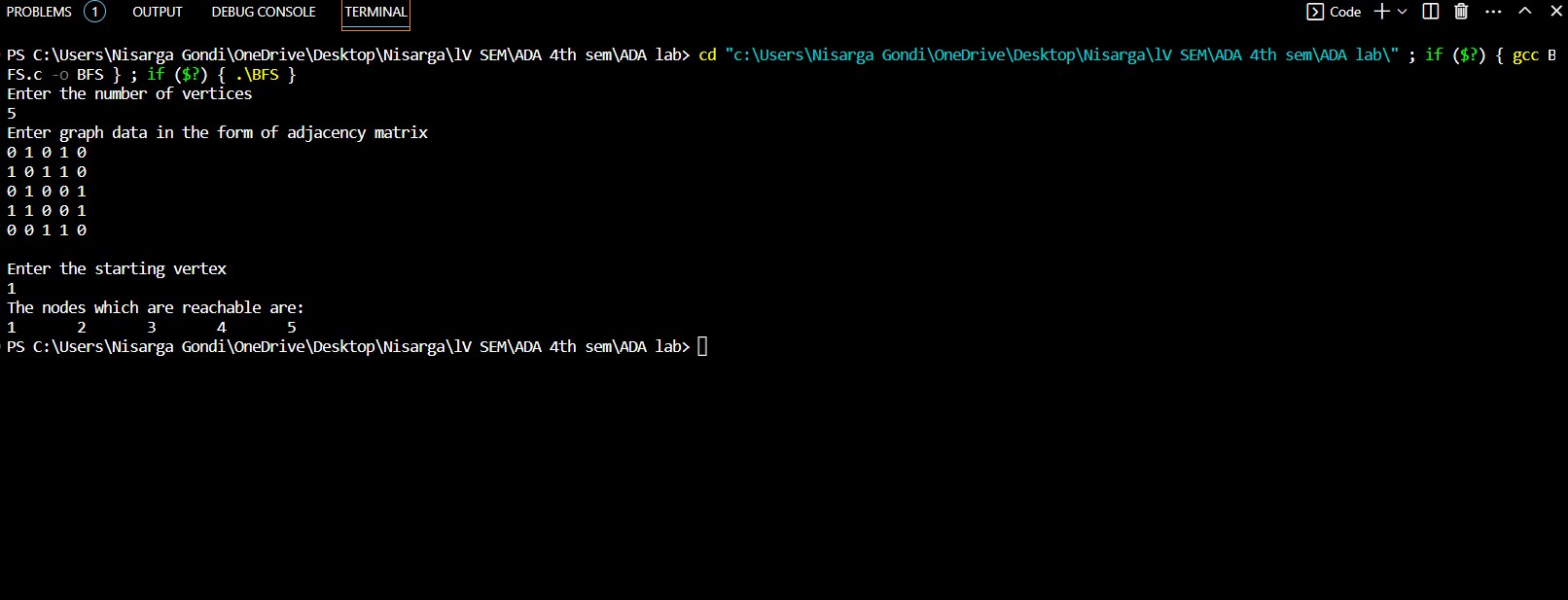
printf("BFS not possible\n"); break;

}

}

}

**OUTPUT:**



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

#include<stdio.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);

}

}

void main()

{

int i,j,count=0;

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

scanf("%d",&n); for(i=1;i<=n;i++)

{

reach[i]=0; for(j=1;j<=n;j++) a[i][j]=0;

}

printf("\n Enter the adjacency matrix:\n"); for(i=1;i<=n;i++)

for(j=1;j<=n;j++) scanf("%d",&a[i][j]); dfs(1);

printf("\n"); for(i=1;i<=n;i++)

{

if(reach[i]) count++;

}

if(count==n)

{

printf("\n Graph is connected");

}

else

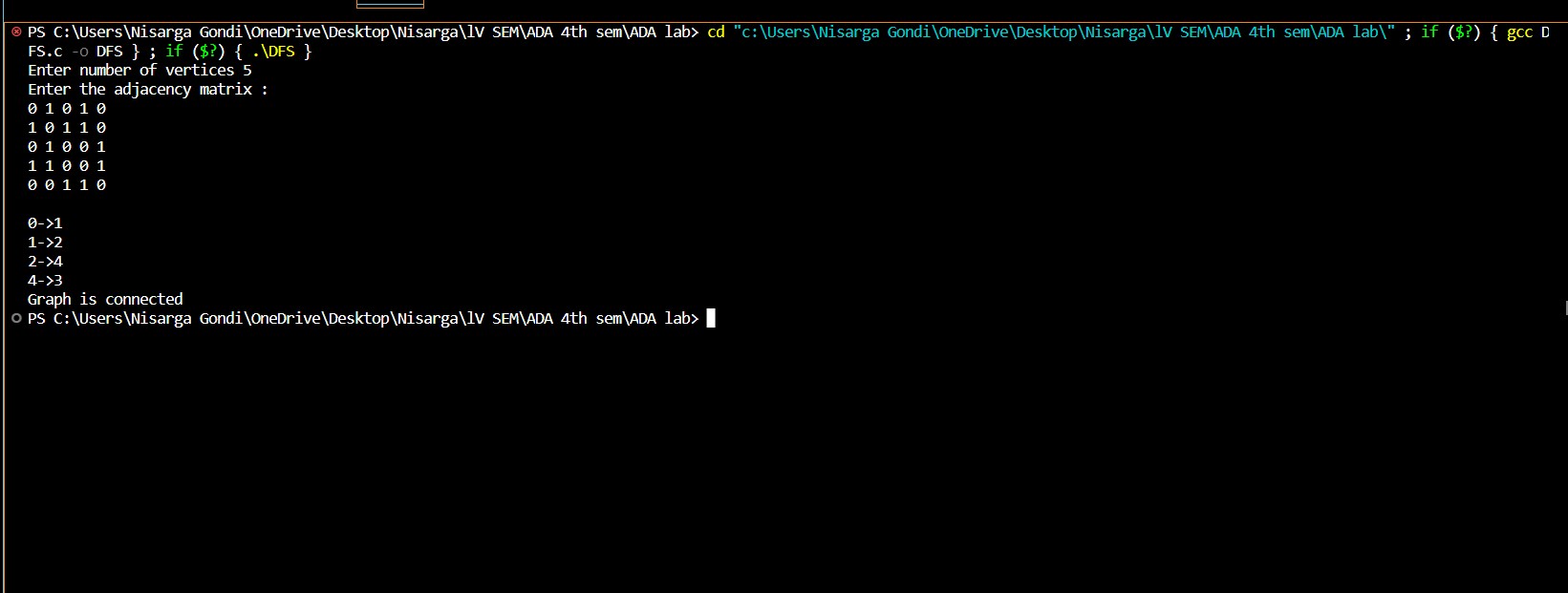
{

printf("\n Graph is not connected");

}

}

**OUTPUT:**



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

#include<stdio.h> #include<conio.h>

void dfs(int);

int a[10][10],vis[10],exp[10],n,j,m;

void main()

{

int i,x,y;

printf("enter the number of vertices\n"); scanf("%d",&n);

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

{

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

{ a[i][j]=0;

}

vis[i]=0;

}

printf("enter the number of edges\n"); scanf("%d",&m);

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

{

printf("enter an edge\n"); scanf("%d %d",&x,&y);

a[x][y]=1;

} j=0;

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

{

if(vis[i]==0)

dfs(i);

}

printf("topological sort\n"); for(i=n-1;i>=0;i--)

{

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

}

getch();

}

void dfs(int v)

{

int i; vis[v]=1;

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

{

if(a[v][i]==1 && vis[i]==0) dfs(i);

}

exp[j++]=v;

}

**OUTPUT:**



1. **Implement Johnson Trotter algorithm to generate permutations.**

#include <stdio.h>

#define RIGHT\_TO\_LEFT 0

#define LEFT\_TO\_RIGHT 1

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

int i;

for (i = 0; i < n; i++) if (a[i] == mobile) return i + 1;

return -1;

}

int getMobile(int a[], int dir[], int n) { int i;

int mobile\_prev = 0, mobile = 0; for (i = 0; i < n; i++) {

if (dir[a[i] - 1] == RIGHT\_TO\_LEFT && i != 0) { if (a[i] > a[i - 1] && a[i] > mobile\_prev) { mobile = a[i];

mobile\_prev = mobile;

}

}

if (dir[a[i] - 1] == LEFT\_TO\_RIGHT && i != n - 1) { if (a[i] > a[i + 1] && a[i] > mobile\_prev) {

mobile = a[i]; mobile\_prev = mobile;

}

}

}

return mobile;

}

void swap(int \*a, int \*b) { int temp = \*a;

\*a = \*b;

\*b = temp;

}

void printOnePerm(int a[], int dir[], int n) { int i;

int mobile = getMobile(a, dir, n); int pos = searchArr(a, n, mobile);

if (dir[a[pos - 1] - 1] == RIGHT\_TO\_LEFT) swap(&a[pos - 1], &a[pos - 2]);

else if (dir[a[pos - 1] - 1] == LEFT\_TO\_RIGHT) swap(&a[pos], &a[pos - 1]);

for (i = 0; i < n; i++) { if (a[i] > mobile) {

if (dir[a[i] - 1] == LEFT\_TO\_RIGHT) dir[a[i] - 1] = RIGHT\_TO\_LEFT;

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

}

}

for (i = 0; i < n; i++) printf("%d", a[i]); printf(" ");

}

int factorial(int n) { int i,res = 1;

for ( i = 1; i <= n; i++) res \*= i;

return res;

}

void printPermutation(int n) { int a[n];

int dir[n]; int i;

for (i = 0; i < n; i++) { a[i] = i + 1; printf("%d", a[i]);

}

printf("\n");

for (i = 0; i < n; i++) dir[i] = RIGHT\_TO\_LEFT;

for (i = 1; i < factorial(n); i++) printOnePerm(a, dir, n);

}

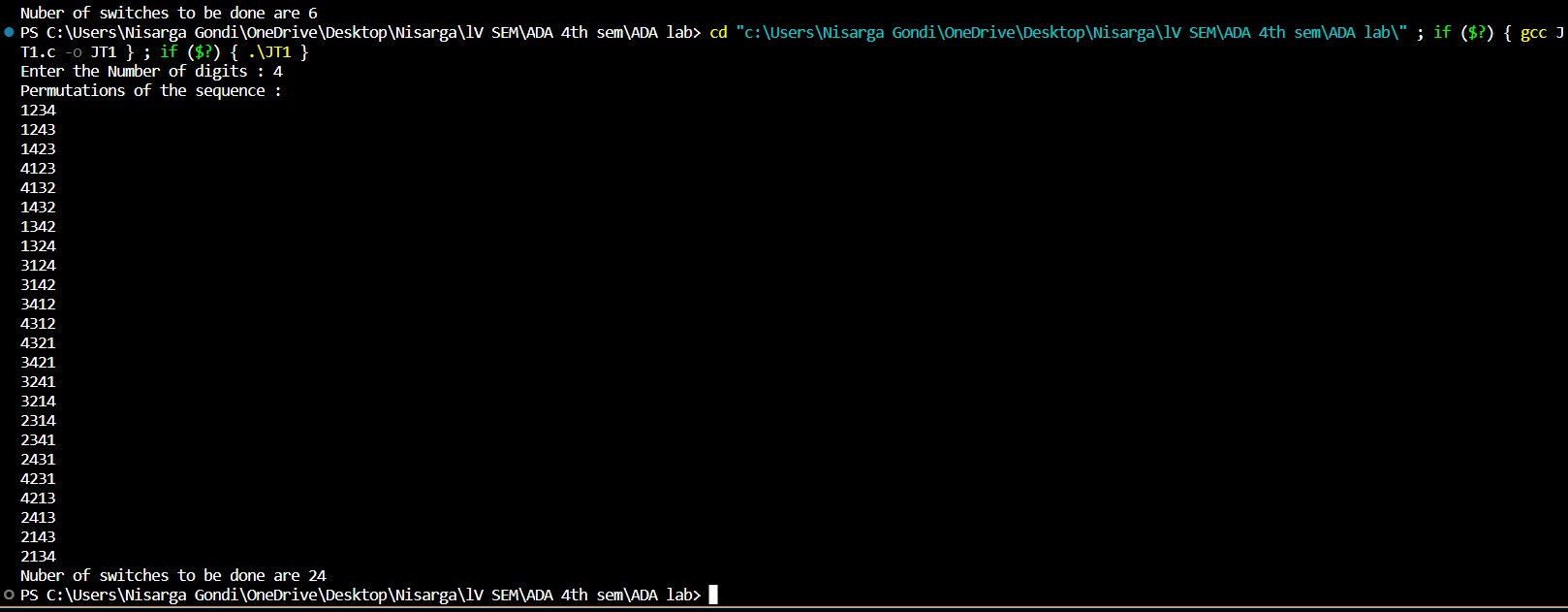
int main() { int n;

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

scanf("%d", &n); printf("Permutations:\n"); printPermutation(n); return 0;

}

**OUTPUT:**



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

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

void merge(int arr[],int l,int r,int m)

{

int i,j,k;

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

int left[n1], right[n2]; for(i=0;i<n1;i++)

{

left[i]=arr[l+i];

}

for(j=0;j<n2;j++)

{

right[j]=arr[m+1+j];

} i=0; j=0;

k=l;

while(i<n1 && j<n2)

{

if(left[i]<=right[j])

{

arr[k]=left[i]; i++;

}

else

{

arr[k]=right[j]; j++;

} k++;

}

while(i<n1)

{

arr[k]=left[i];

i++; k++;

}

while(j<n2)

{

arr[k]=right[j]; j++;

k++;

}

}

void mergesort(int arr[], int l, int r)

{

int mid; if(l<r)

{

mid=l+(r-l)/2; mergesort(arr,l,mid); mergesort(arr,mid+1,r); merge(arr,l,r,mid);

}

}

void print(int arr[],int n)

{

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

{

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

}

}

void main()

{

int arr[100000],n,i; float time\_taken; clock\_t st,et;

printf("Enter the size of the array\n"); scanf("%d",&n);

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

{

arr[i]=rand()%100;

}

printf("before sorting \n"); print(arr,n);

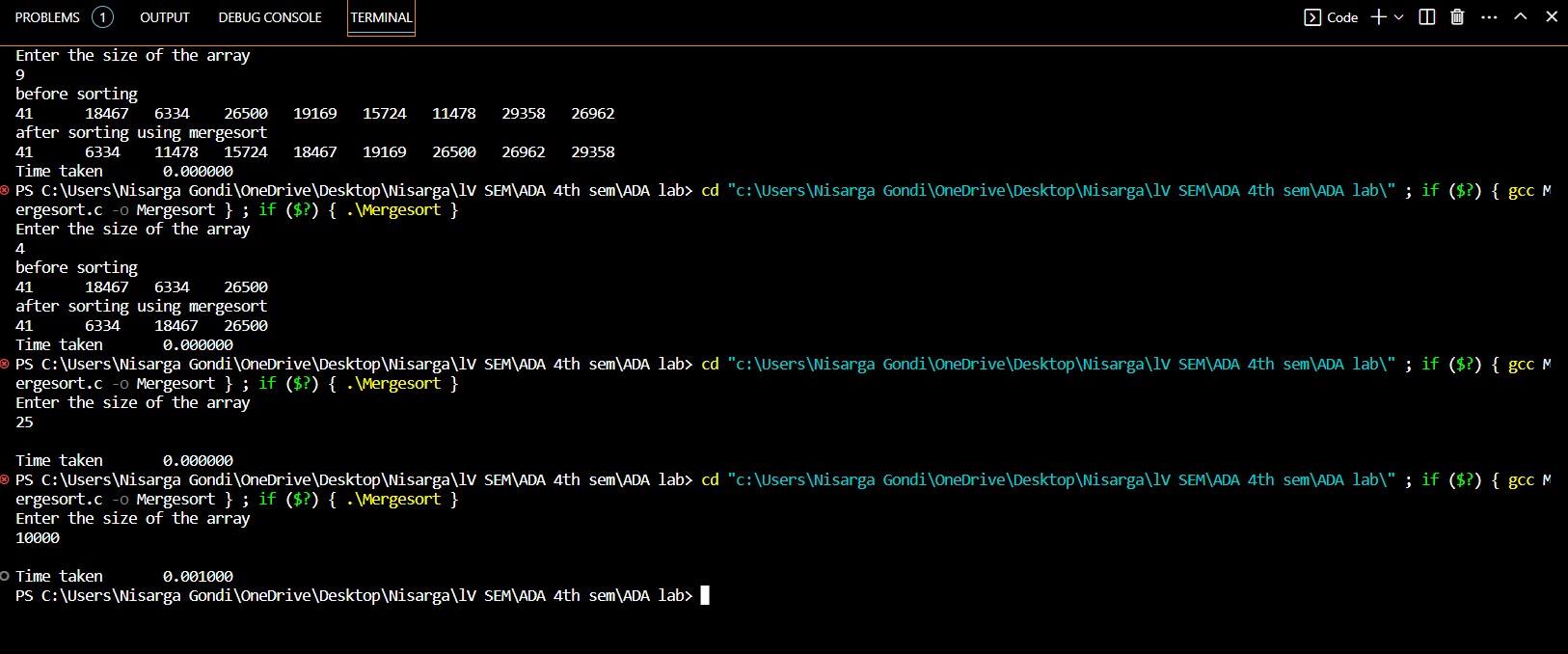
st=clock(); mergesort(arr,0,n-1); et=clock();

printf("\nafter sorting using mergesort\n"); print(arr,n);

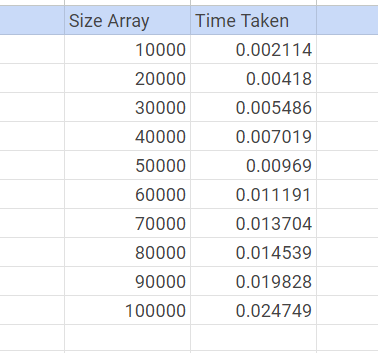
time\_taken = ((float)(et-st)/CLOCKS\_PER\_SEC); // in seconds printf("\nthe time taken is: %f Clocks per cycle",time\_taken);

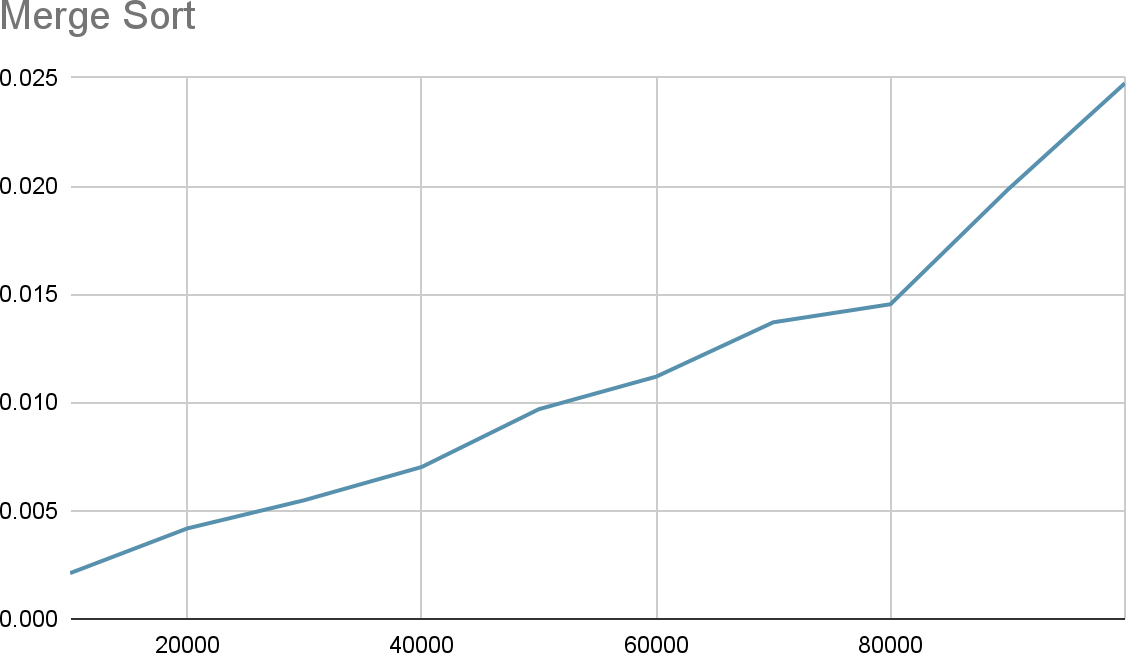
}

**OUTPUT:**



**GRAPH:**





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

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

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

{

int temp; temp=\*a;

\*a=\*b;

\*b=temp;

}

int partition(int arr[],int l,int r)

{

//ascending order int pivot=arr[r];

int i=l-1,j; for(j=l;j<=r-1;j++)

{

if(arr[j]<pivot)

{ i++;

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

}

}

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

}

void quicksort(int arr[],int l,int r)

{

int split; if(l<r)

{

split=partition(arr,l,r); quicksort(arr,l,split-1); quicksort(arr,split+1,r);

}

}

void print(int arr[],int n)

{

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

{

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

}

}

void main()

{

int arr[200000],n,i; float time\_taken; clock\_t st,et;

printf("Enter the size of the array\n"); scanf("%d",&n);

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

{

arr[i]=rand()%100;

}

printf("before sorting \n"); print(arr,n);

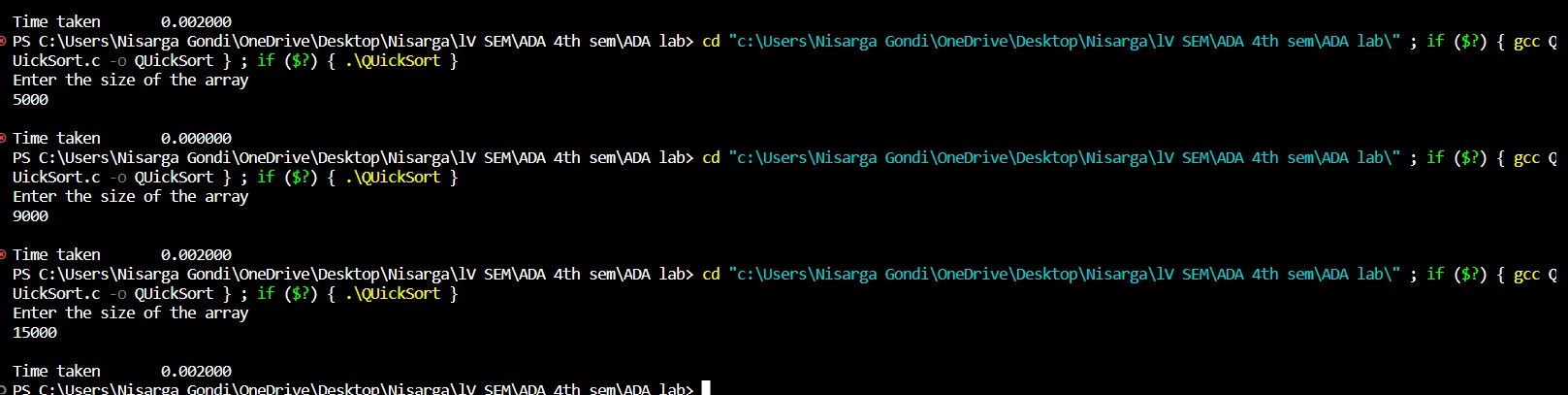
st=clock(); quicksort(arr,0,n-1); et=clock();

printf("\nafter sorting using quicksort\n"); print(arr,n);

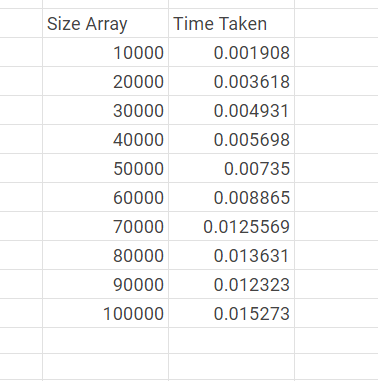
time\_taken = ((float)(et-st)/CLOCKS\_PER\_SEC); // in seconds printf("\nthe time taken is: %f Clocks per cycle",time\_taken);

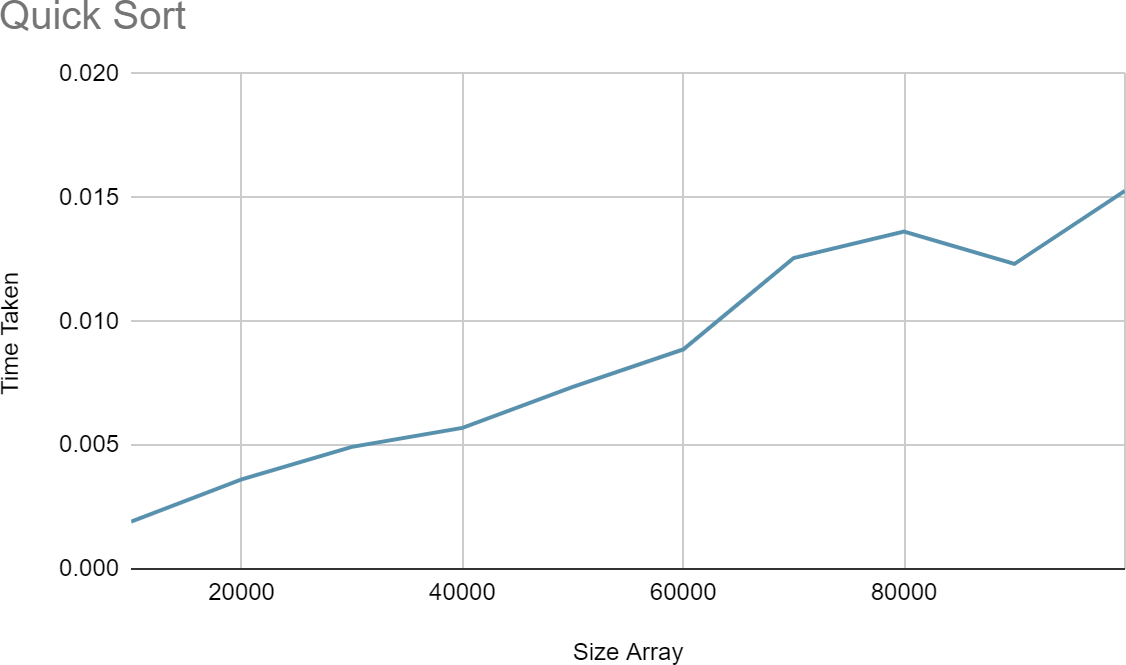
}

**OUTPUT:**



**Graph :**





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

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

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

{

int temp=\*a;

\*a=\*b;

\*b=temp;

}

void heapify(int a[],int n,int i)

{

int largest=i,l=2\*i,r=2\*i+1; while(l<n && a[l]>a[largest])

{

largest=l;

}

while (r<n && a[r]>a[largest])

{

largest=r;

}

if(largest!=i)

{

swap(&a[i],&a[largest]); heapify(a,n,largest);

}

}

void print(int a[],int n)

{

int i;

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

{

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

}

printf("\n");

}

void heapsort(int a[],int n)

{

int i;

//create max heap for(i=n/2;i>=1;i--)

{

heapify(a,n,i);

}

//sort using deletion for(i=n;i>=1;i--)

{

swap(&a[1],&a[i]); heapify(a,i,1);

}

}

int main() { int n, i; clock\_t st, et; float ts;

printf("Enter the number of elements\n"); scanf("%d", &n);

// Dynamically allocate the array

int \*a = (int \*)malloc(n \* sizeof(int));

if (a == NULL) {

printf("Memory allocation failed.\n"); return 1;

}

// Generate random values and place them in the array for (i = 0; i < n; i++) {

a[i] = rand();

}

st = clock(); heapsort(a, n); et = clock();

ts = (float)(et - st) / CLOCKS\_PER\_SEC; if (n <= 20) {

printf("\nAfter sorting elements are\n"); print(a, n);

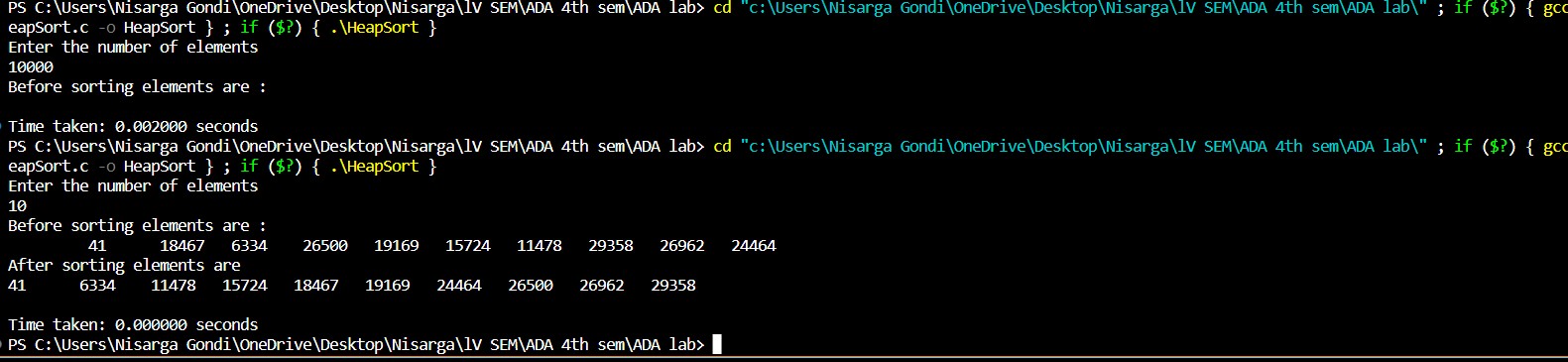
}

// Free dynamically allocated memory free(a);

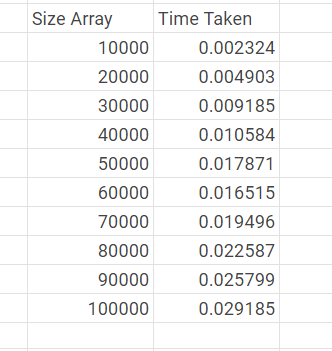
printf("\nTime taken: %f seconds\n", ts); return 0;

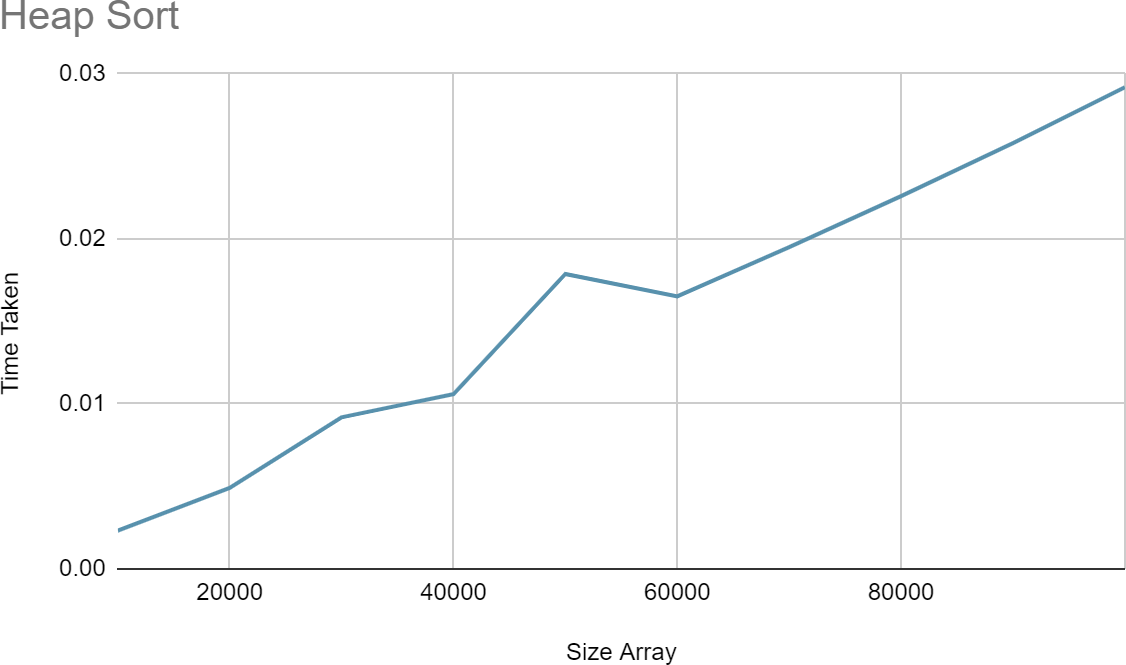
}

**OUTPUT:**



**Graph:**





1. **Implement 0/1 Knapsack problem using dynamic programming.**

#include<stdio.h> void main()

{

int i,j,w[10],p[10],opt[10][10],x[10],n,m; printf("Enter the number of items\n"); scanf("%d",&n);

printf("enter the weight and profit of each item\n"); for(i=1;i<=n;i++)

{

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

}

printf("enter the knapsack capacity\n"); scanf("%d",&m);

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

{

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

{

if(i==0 || j==0)

{

opt[i][j]=0;

}

else if(j-w[i]<0)

{

opt[i][j]=opt[i-1][j];

}

else

{

opt[i][j]=opt[i-1][j-w[i]]+p[i]>(opt[i-1][j])?opt[i-1][j-w[i]]+p[i]:(opt[i-1][j]);

}

}

}

//output

printf("\nknapsack table\n"); for(i=0;i<=n;i++)

{

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

{

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

}

printf("\n");}

for(i=n;i>=1;i--)

{

if(opt[i][m]!=opt[i-1][m])

{ x[i]=1;

m=m-w[i];

}

else

{ x[i]=0;

}

}

printf("\nitems selected are designated 1\n"); for(i=1;i<=n;i++)

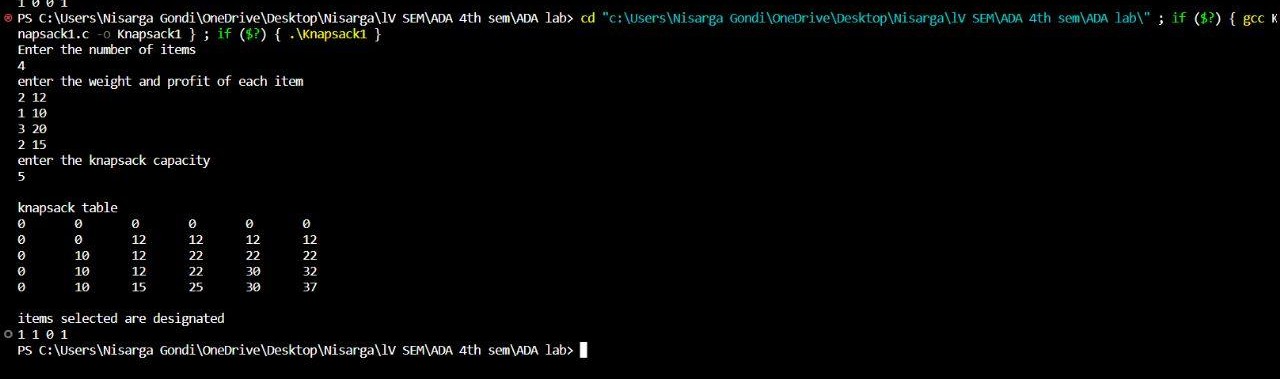
{

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

}

}

**OUTPUT:**



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

#include<stdio.h> void main()

{

int adj[10][10],n,i,j,k; int result[10][10];

printf("Floyd's algorithm\n"); printf("enter the number of vertices\n"); scanf("%d",&n);

printf("Enter the distance matrix for %d vertices\n",n); for(i=0;i<n;i++)

{

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

{

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

result[i][j]=adj[i][j];

}

}

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

{

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

{

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

{

result[i][j]=result[i][j]<(result[i][k]+result[k][j])?result[i][j]:(result[i][k]+result[k][j]);

}

}

}

printf("\nResult\n"); for(i=0;i<n;i++)

{

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

{

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

}

printf("\n");

}

}

**OUTPUT:**



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

**//prims**

#include <stdio.h>

int cost[10][10], vt[10], et[10][10], vis[10], j, n; int sum = 0;

int x = 1; int e = 0;

void prims(); void main()

{

int i;

printf("enter the number of vertices\n"); scanf("%d", &n);

printf("enter the cost adjacency matrix\n"); for (i = 1; i <= n; i++)

{

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

{

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

}

vis[i] = 0;

}

prims();

printf("edges of spanning tree\n"); for (i = 1; i <= e; i++)

{

printf("%d,%d\t", et[i][0], et[i][1]);

}

printf("weight=%d\n", sum);

}

void prims()

{

int s, min, m, k, u, v; vt[x] = 1;

vis[x] = 1;

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

{

j = x;

min = 999;

while (j > 0)

{

k = vt[j];

for (m = 2; m <= n; m++)

{

if (vis[m] == 0)

{

if (cost[k][m] < min)

{

min = cost[k][m]; u = k;

v = m;

}

}

}

j--;

}

vt[++x] = v; et[s][0] = u;

et[s][1] = v; e++;

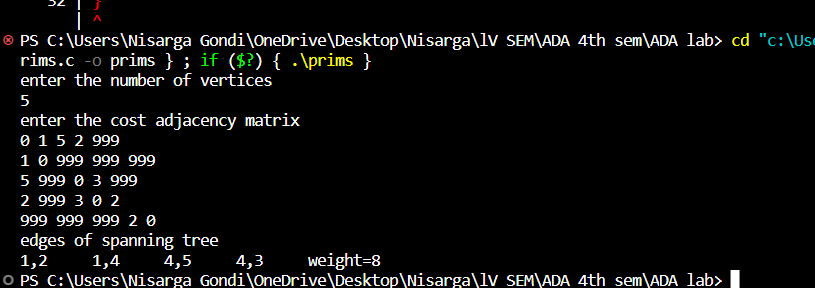
vis[v] = 1;

sum = sum + min;

}

}

**OUTPUT:**



**//kruskals**

#include<stdio.h>

int find(int v,int parent[10])

{

while(parent[v]!=v)

{

v=parent[v];

}

return v;

}

void union1(int i,int j,int parent[10])

{

if(i<j) parent[j]=i; else parent[i]=j;

}

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

{

int count,k,min,sum,i,j,t[10][10],u,v,parent[10]; count=0;

k=0;

sum=0; for(i=0;i<n;i++) parent[i]=i; while(count!=n-1)

{

min=999; for(i=0;i<n;i++)

{

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

{

if(a[i][j]<min && a[i][j]!=0)

{

min=a[i][j]; u=i;

v=j;

}

}

}

i=find(u,parent); j=find(v,parent); if(i!=j)

{

union1(i,j,parent); t[k][0]=u;

t[k][1]=v; k++;

count++; sum=sum+a[u][v];

} a[u][v]=a[v][u]=999;

}

if(count==n-1)

{

printf("spanning tree\n"); for(i=0;i<n-1;i++)

{

printf("%d %d\n",t[i][0],t[i][1]);

}

printf("cost of spanning tree=%d\n",sum);

}

else

printf("spanning tree does not exist\n");

}

void main()

{

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

printf("enter the number of nodes\n"); scanf("%d",&n);

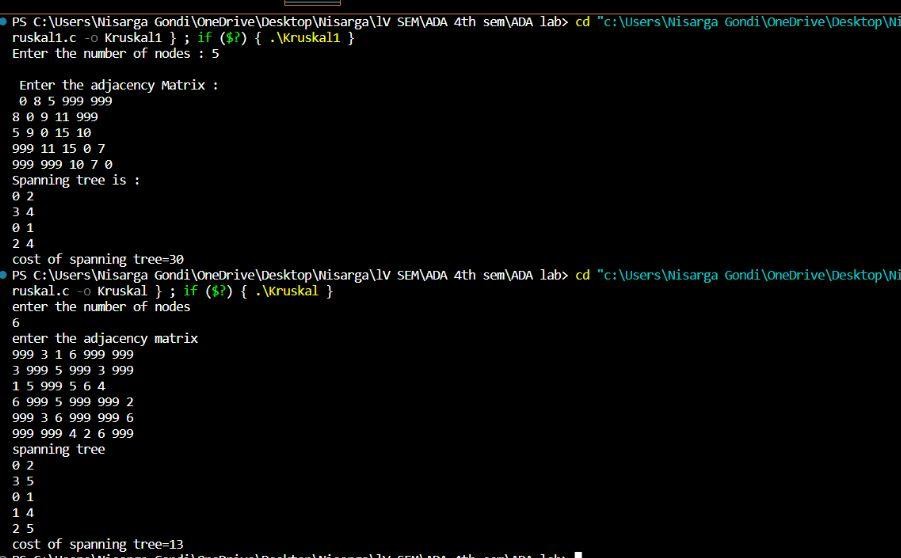
printf("enter the adjacency matrix\n"); for(i=0;i<n;i++)

for(j=0;j<n;j++) scanf("%d",&a[i][j]); kruskal(n,a);

getch();

}

**OUTPUT:**



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

#include <stdio.h> #define INFINITY 9999

#define MAX 10

void dijkstra(int G[MAX][MAX], int n, int startnode); int main()

{

int G[MAX][MAX], i, j, n, u; printf("Enter no. of vertices:"); scanf("%d", &n);

printf("\nEnter the adjacency matrix:\n"); for (i = 0; i < n; i++)

for (j = 0; j < n; j++) scanf("%d", &G[i][j]);

printf("\nEnter the starting node:"); scanf("%d", &u);

dijkstra(G, n, u); return 0;

}

void dijkstra(int G[MAX][MAX], int n, int startnode)

{

int cost[MAX][MAX], distance[MAX], pred[MAX]; int visited[MAX], count, mindistance, nextnode, i, j; for (i = 0; i < n; i++)

for (j = 0; j < n; j++) if (G[i][j] == 0) cost[i][j] = INFINITY; else

cost[i][j] = G[i][j]; for (i = 0; i < n; i++)

{

distance[i] = cost[startnode][i];

pred[i] = startnode; visited[i] = 0;

}

distance[startnode] = 0;

visited[startnode] = 1;

count = 1;

while (count < n - 1)

{

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

if (distance[i] < mindistance && !visited[i])

{

mindistance = distance[i]; nextnode = i;

}

visited[nextnode] = 1; for (i = 0; i < n; i++)

if (!visited[i])

if (mindistance + cost[nextnode][i] < distance[i])

{

distance[i] = mindistance + cost[nextnode][i]; pred[i] = nextnode;

}

count++;

}

for (i = 0; i < n; i++) if (i != startnode)

{

printf("\nDistance of node%d = %d", i, distance[i]);

printf("\nPath = %d", i); j = i;

do

{

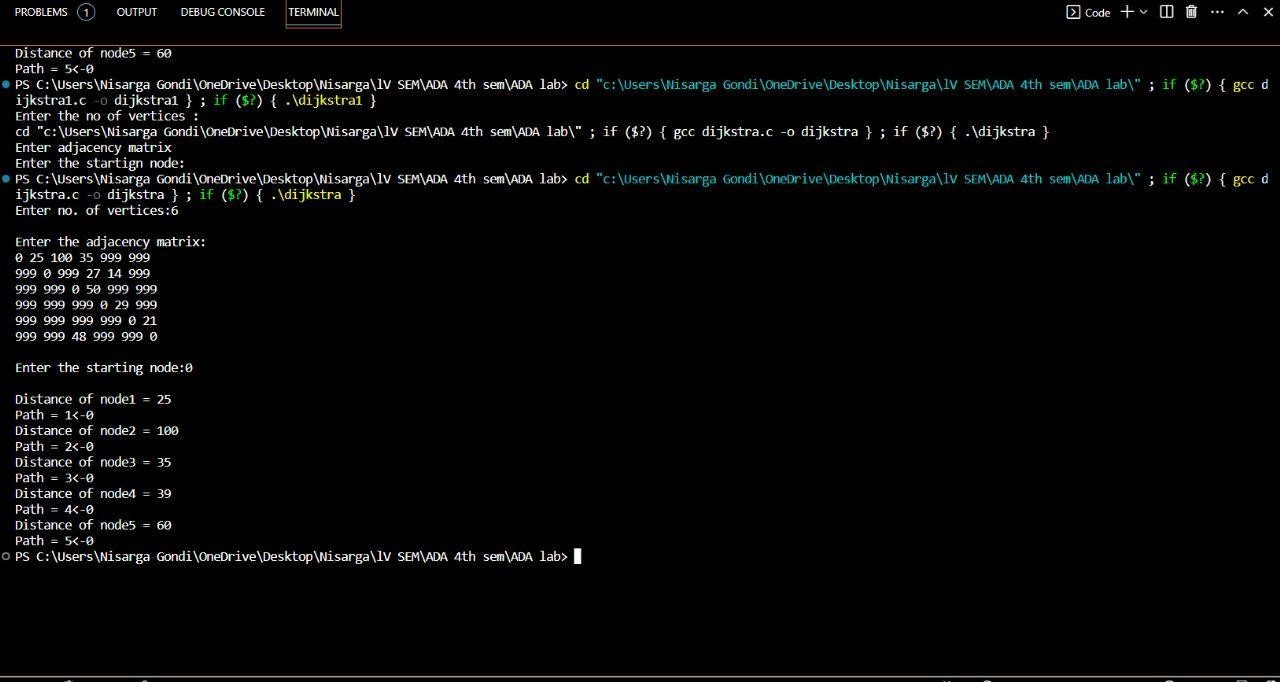
j = pred[j]; printf("<-%d", j);

} while (j != startnode);

}

}

**OUTPUT:**



1. **Implement “N-Queens Problem” using Backtracking.**

#include<stdio.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;

}

//function for printing the solution 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) //for nxn board

{

if(board[i]==j)

printf("\tQ"); //queen at i,j position else

printf("\t-"); //empty slot

}

}

}

/\*funtion to check conflicts

If no conflict for desired postion returns 1 otherwise returns 0\*/ int place(int row,int column)

{

int i;

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

{

//checking column and digonal conflicts if(board[i]==column)

return 0; else

if(abs(board[i]-column)==abs(i-row)) return 0;

}

return 1; //no conflicts

}

//function to check for proper positioning of queen void queen(int row,int n)

{

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

{

if(place(row,column))

{

board[row]=column; //no conflicts so place queen if(row==n) //dead end

print(n); //printing the board configuration else //try queen with next position queen(row+1,n);

}

}

}

**OUTPUT:**

