**VISVESVARAYA TECHNOLOGICAL UNIVERSITY**

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



# LAB REPORT

**on**

Analysis and Design of Algorithms

***Submitted by***

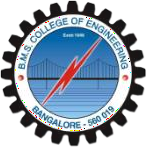
**ADNAN ANWAR(1BM21CS008)**

***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 **ADNAN ANWAR(1BM21CS008),** 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.

Prof Antara Roy Choudhury 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 the BFS method. 2. Check whether a given graph is connected or not using the DFS method. | 5-8 |
| 2 | Write a program to obtain the Topological ordering of vertices in a given digraph. | 9-10 |
| 3 | Implement Johnson Trotter algorithm to generate permutations. | 11-14 |
| 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. | 15-17 |
| 5 | Sort a given set of N integer elements using Quick Sort technique and compute its time taken. | 18-21 |
| 6 | Sort a given set of N integer elements using Heap Sort technique and compute its time taken. | 22-25 |
| 7 | Implement 0/1 Knapsack problem using dynamic programming. | 26-28 |
| 8 | Implement All Pair Shortest paths problem using Floyd’s algorithm. | 29-31 |
| 9 | Find Minimum Cost Spanning Tree of a given undirected graph using Prim’s and Kruskal’s algorithm. | 32-37 |
| 10 | From a given vertex in a weighted connected graph, find shortest paths to other vertices using Dijkstra’s algorithm. | 38-40 |
| 11 | Implement “N-Queens Problem” using Backtracking. | 41-43 |

**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. |

# PROGRAM 1

Write program to do the following:

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

## CODE:

**BFS:**

#include<stdio.h>

int a[10][10],vis[10],vert,f=1,r=1,q[10],src;

void bfs(int v);

void main(){

int i,j;

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

scanf("%d",&vert);

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

for(j=1;j<=vert;j++){

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

}

vis[i]=0;

}

printf("traversal \n");

printf("Enter the source vertex: ");

scanf("%d",&src);

bfs(src);

}

void bfs(int v){

int n=v;

vis[v]=1;

q[r]=v;

while(f<=r){

printf("%d\t",q[f]);

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

if(a[q[f]][j]==1&&vis[j]==0){

q[++r]=j;

vis[j]=1;

}

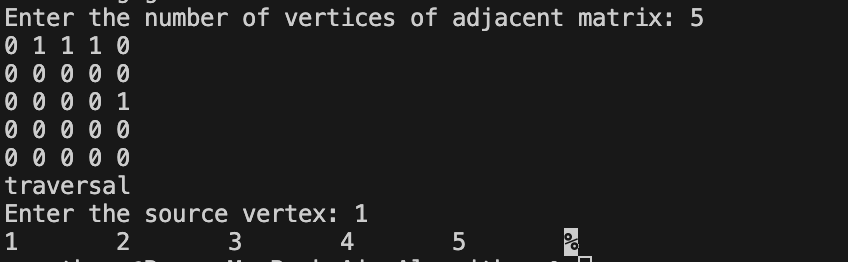
}

f++;

}

}

**OUTPUT:**



## DFS:

#include<stdio.h>

int a[10][10],vis[10],vert,count=1;

void dfs(int);

void main(){

    int i,j;

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

    scanf("%d",&vert);

    for(i=1;i<=vert;i++){

        for(j=1;j<=vert;j++){

            scanf("%d",&a[i][j]);

        }

        vis[i]=0;

    }

    printf("traversal \n");

    for(i=1;i<vert;i++){

        if(vis[i]==0){

            dfs(i);

        }

    }

    if(count==vert){

        printf("Connected ");

    }

    else{

        printf("Disconnected b");

    }

}

void dfs(int v){

    int n=v;

    vis[n]=1;

    printf("%d\t",n);

    for(int j=1;j<=vert;j++){

        if(a[n][j]==1&&vis[j]==0){

            dfs(j);

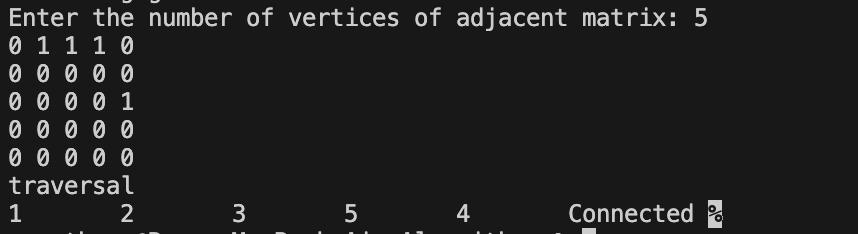
            count++;

        }

    }

}

## OUTPUT:

****

# PROGRAM 2

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

## CODE:

#include <stdio.h>

int v[10], adj[10][10];

void dfs(int);

int n;

int sort[10];

int sindex = 0;

int main() {

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

scanf("%d", &n);

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

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

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

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

}

}

printf("\nThe topological sorting is:\n");

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

if (v[i] == 0) {

dfs(i);

}

}

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

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

}

return 0;

}

void dfs(int ver) {

v[ver] = 1;

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

if (adj[ver][j] == 1 && v[j] == 0) {

dfs(j);

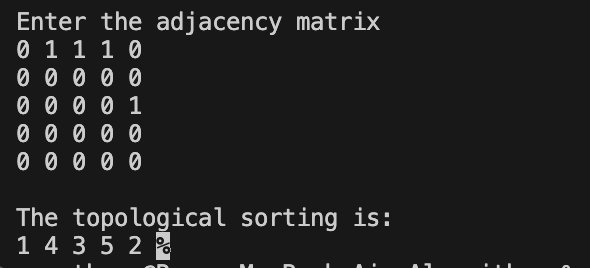
}

}

sort[sindex++] = ver;

}

**OUTPUT:**

****

# PROGRAM 3

Implement Johnson Trotter algorithm to generate permutations.

## CODE:

#include <stdio.h>

#define RIGHT\_TO\_LEFT 0

#define LEFT\_TO\_RIGHT 1

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

\*a = \*b;

\*b = temp;

}

int searchArr(int a[], int n, int mobile) { for (int i = 0; i < n; i++) {

if (a[i] == mobile) { return i + 1;

}

}

return -1; // Mobile not found

}

int getMobile(int a[], int dir[], int n) { int mobile\_prev = 0, mobile = 0; for (int i = 0; i < n; i++) {

// Direction 0 represents RIGHT TO LEFT.

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

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

}

}

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

mobile = a[i];

mobile\_prev = mobile;

}

}

}

if (mobile == 0 && mobile\_prev == 0) { return 0; // No mobile element found

} else {

return mobile;

}

}

void printOnePerm(int a[], int dir[], int n) { int mobile = getMobile(a, dir, n);

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

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

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

}

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

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

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

}

}

}

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

}

printf("\n");

}

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

for (int i = 1; i <= n; i++) { res = res \* i;3

}

return res;

}

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

int dir[n];

for (int i = 0; i < n; i++) { a[i] = i + 1;

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

}

printf("\n");

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

}

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

}

}

int main() { int n;

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

printPermutation(n);

return 0;

}

**OUTPUT:**



# PROGRAM 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.

## CODE:

#include<stdio.h>

#include<time.h>

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

void merge(int a[],int low,int mid,int mid1,int high);

int main()

{

int a[30],n,i;

clock\_t start\_t, end\_t;

double total\_t;

printf("Enter no of elements:");

scanf("%d",&n);

start\_t = clock();

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

printf("Enter array elements:");

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

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

mergesort(a,0,n-1);

end\_t = clock();

printf("End of the big loop, 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[50];

int i,j,k;

i=i1;

j=i2;

k=i1;

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)

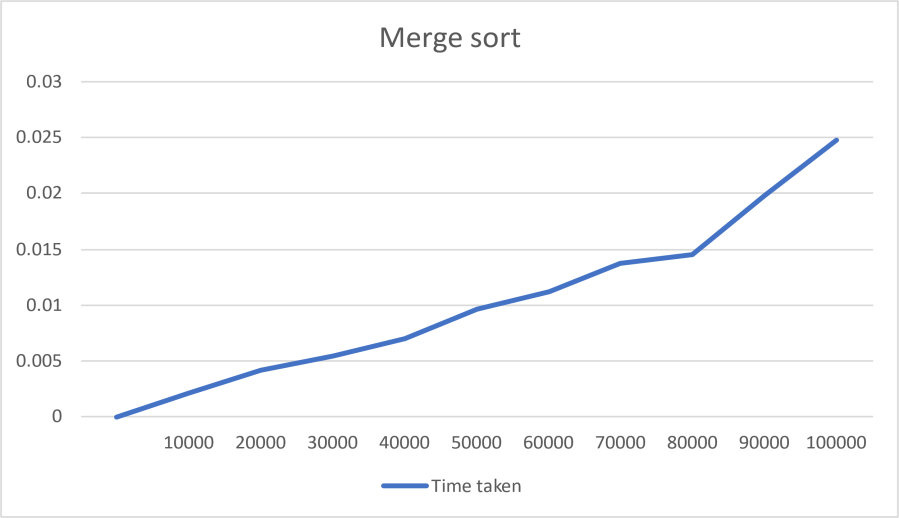
temp[k++]=a[j++];

for(k=i1;k<=j2;k++)

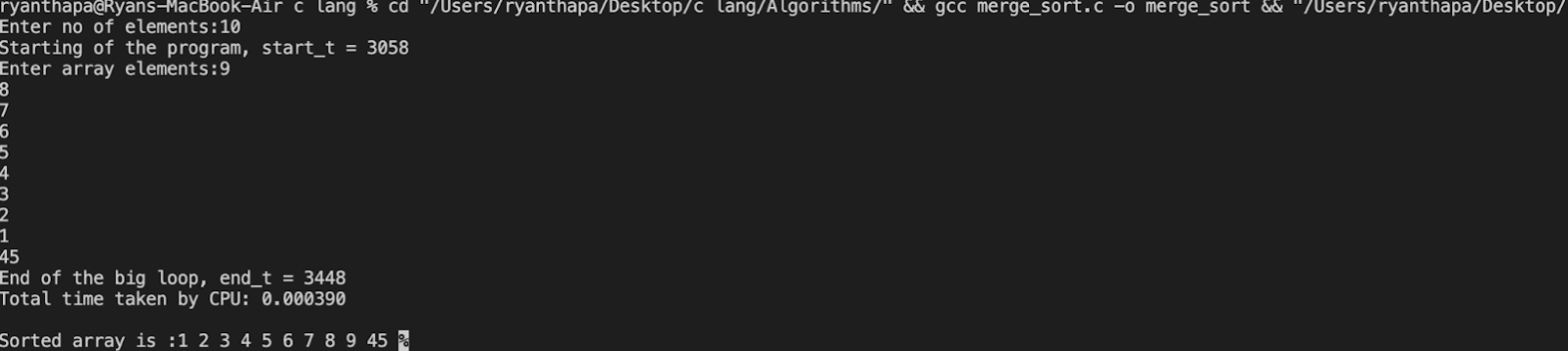
a[k]=temp[k];

}

## GRAPH:



**OUTPUT:**



# PROGRAM 5

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

## CODE:

#include<stdio.h>

#include<time.h>

#include<stdlib.h>

void quicksort(int a[], int low, int high);

int partition(int a[], int low, int high);

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

int main(){

int a[30000];

int low, high;

int n;

clock\_t end, start;

srand(time(NULL));

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

scanf("%d",&n);

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

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

//a[i] = rand()%10000;

}

low = 0;

high = n-1;

start = clock();

quicksort(a, low, high);

end = clock();

printf("Sorted array:\n");

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

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

}

printf("\nThe time taken to sort %d elements is %lf", n, (double)(end-start)/CLOCKS\_PER\_SEC);

return 0;

}

void quicksort(int a[], int low, int high){

if(low<high){

int j = partition(a, low, high);

quicksort(a,low,j-1);

quicksort(a, j+1, high);

}

}

int partition(int a[], int low, int high){

int i = low;

int j = high;

int pivot = a[low];

while(i<=j){

while(pivot >= a[i] && i<=high){

i++;

};

while(pivot < a[j] && j>low){

j--;

};

if(i<=j){

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

}

}

swap(&a[low],&a[j]);

return j;

}

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

int temp;

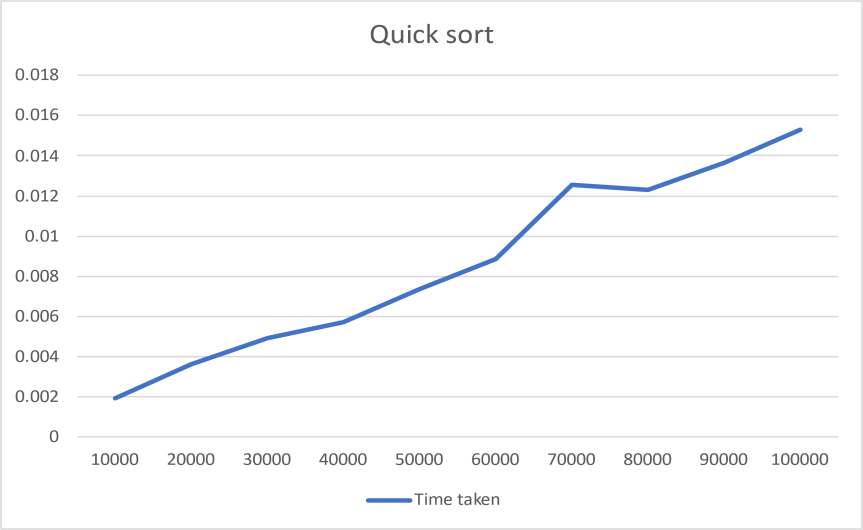
temp = \*a;

\*a=\*b;

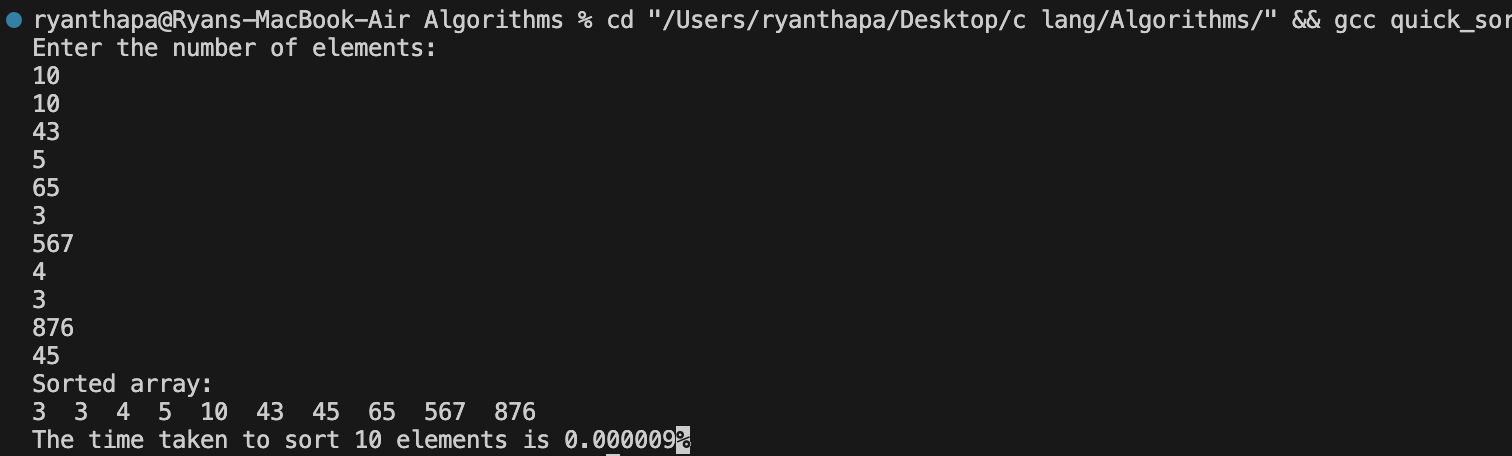
\*b=temp;

}

## GRAPH:



**OUTPUT:**



# PROGRAM 6

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

**CODE:**

#include <stdio.h>

#include<time.h>

#include<stdlib.h>

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

int temp = \*a;

\*a = \*b;

\*b = temp;

}

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) {

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

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--) {

swap(&arr[0], &arr[i]);

heapify(arr, i, 0);

}

}

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

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

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

printf("\n");

}

int main() {

printf("Enter the no. of elements: ");

clock\_t start\_t, end\_t;

double total\_t;

int n;

srand(time(NULL));

scanf("%d",&n);

int arr[n];

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

arr[i]=rand()%10000;

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

}

start\_t = clock();

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

heapSort(arr, n);

end\_t = clock();

printf("End of the big loop, 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("Sorted array is \n");

printArray(arr, n);

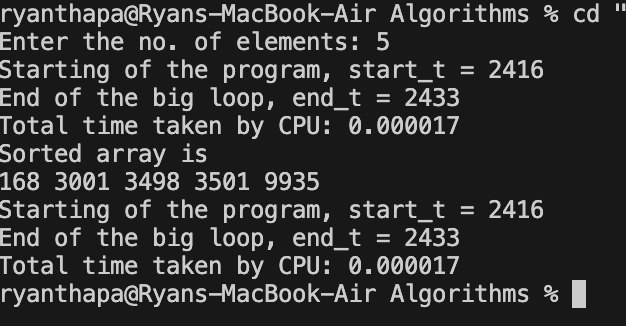
printf("Starting of the program, start\_t = %ld\n", start\_t);

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

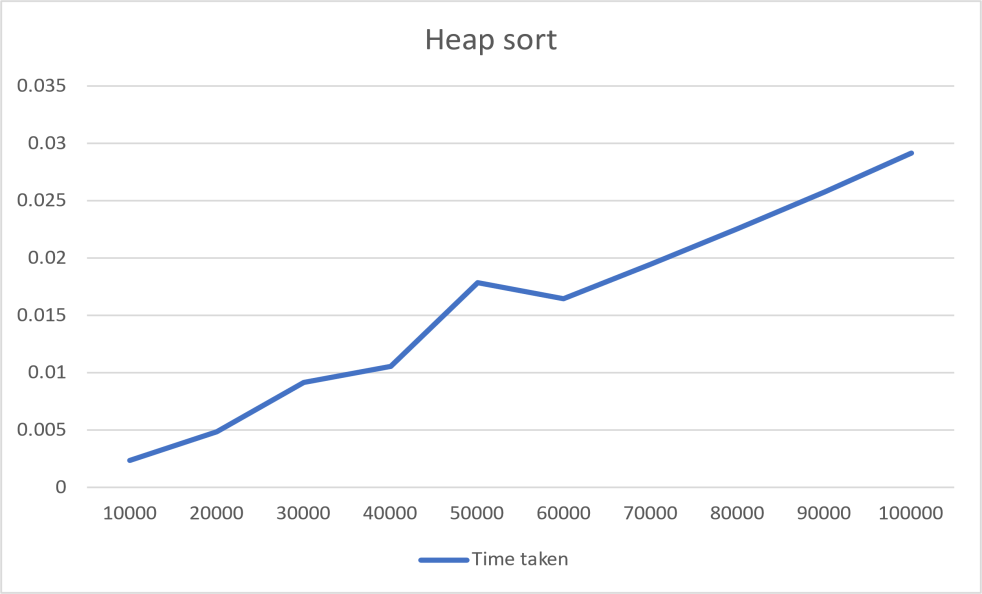
printf("Total time taken by CPU: %f\n", total\_t );

}

**OUTPUT:**



**Graph:**



# PROGRAM 7

Implement 0/1 Knapsack problem using dynamic programming.

## CODE:

#include<stdio.h>

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

void printknapSack(int W, int wt[], int val[], int n)

{

    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];

    printf("Value is: %d\n", res);

    printf("Weights are: ");

    w = W;;

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

        if (res == K[i - 1][w])

            continue;

        else {

            printf("%d\t ", wt[i - 1]);

            res = res - val[i - 1];

            w = w - wt[i - 1];

        }

    }

}

int main()

{

    int W;

    int n;

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

    scanf("%d",&n);

    int val[n];

    int wt[n];

    for(int i=0;i<n;i++){

        printf("Enter the value of item %d: ",i+1);

        scanf("%d",&val[i]);

        printf("Enter the weight of item %d: ",i+1);

        scanf("%d",&wt[i]);

    }

    printf("Enter the capacity: ");

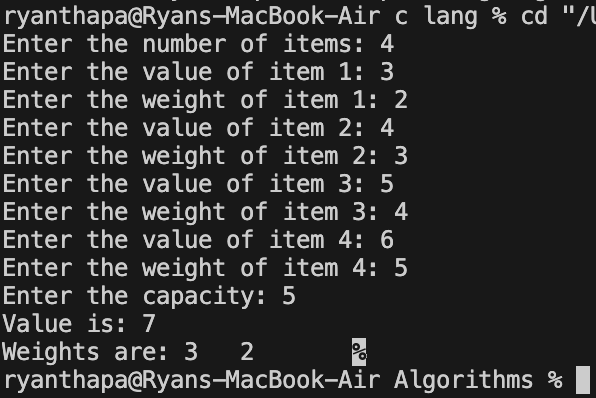
    scanf("%d",&W);

    printknapSack(W, wt, val, n);

    return 0;

}

**OUTPUT:**

****

# PROGRAM 8

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

## CODE:

#include<stdio.h>

int min(int, int);

int main(){

    int n;

    printf("Enter the size of the adjacency matrix\n");

    scanf("%d",&n);

    int a[n][n], p[n][n];

    for(int i=0;i<n;i++){

        for(int j=0;j<n;j++){

        scanf("%d",&a[i][j]);

        p[i][j] = a[i][j];

        }

    }

    for(int k=0; k<n; k++){

        for(int i=0; i<n; i++){

            for(int j=0; j<n; j++){

                if(p[i][j] > 0){

                    p[i][j] = min(p[i][j], p[i][k]+p[k][j]);

                }

            }

        }

    }

    printf("\nThe original matrix is:\n");

    for(int i=0;i<n;i++){

        for(int j=0;j<n;j++){

        printf("%d\t",a[i][j]);

        }

        printf("\n");

    }

    printf("\nThe distance matrix is:\n");

    for(int i=0;i<n;i++){

        for(int j=0;j<n;j++){

        printf("%d\t",p[i][j]);

        }

        printf("\n");

    }

    return 0;

}

int min(int a, int b){

    if(a<=b)

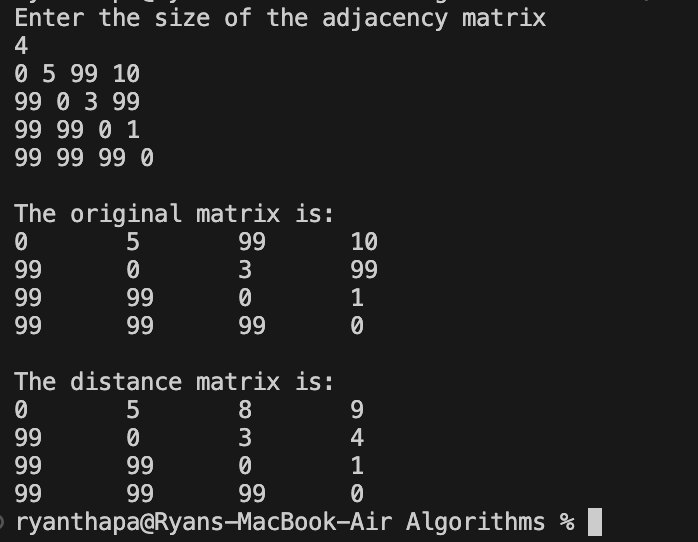
        return a;

    else

        return b;

}

**OUTPUT:**



# PROGRAM 9

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

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

## KRUSKALS ALGORITHM CODE:

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

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

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

int find(int); int uni(int, int);

void main()

{

printf("Kruskal's algorithm in C\n"); printf("========================\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("The 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("%d edge (%d,%d) =%d\n", ne++, a, b, min); mincost += min;

}

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

}

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

}

int find(int i)

{

while (parent[i]) i = parent[i]; return i;

}

int uni(int i, int j)

{

if (i != j)

{

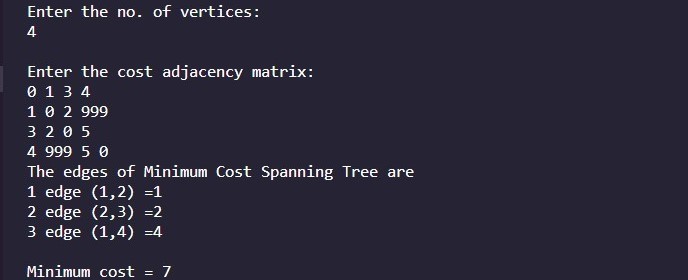
parent[j] = i; return 1;

}

return 0;

}

## OUTPUT:



**PRIMS ALGORITHM CODE:**

#include<stdio.h> #include<conio.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); getch();

}

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)

{

}

}

j--;

}

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

v=m;

}

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

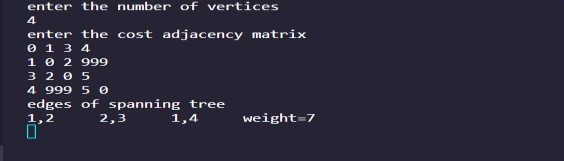
et[s][1]=v; e++;

vis[v]=1; sum=sum+min;

}

}

**OUTPUT:**

v 

# PROGRAM 10

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

## CODE:

#include <stdio.h> #define INFINITY 999

#define MAX 10

void Dijkstra(int Graph[MAX][MAX], int n, int start);

void Dijkstra(int Graph[MAX][MAX], int n, int start) { int cost[MAX][MAX], distance[MAX], pred[MAX]; int visited[MAX], count, mindistance, nextnode, i, j;

// Creating cost matrix for (i = 0; i < n; i++) for (j = 0; j < n; j++)

if (Graph[i][j] == 0) cost[i][j] = INFINITY; else

cost[i][j] = Graph[i][j];

for (i = 0; i < n; i++) { distance[i] = cost[start][i]; pred[i] = start;

visited[i] = 0;

}

distance[start] = 0;

visited[start] = 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++;

}

// Printing the distance for (i = 0; i < n; i++) if (i != start) {

printf("\nDistance from source to %d: %d", i, distance[i]);

}

}

int main() {

int Graph[MAX][MAX], i, j, n, u;

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

printf("Enter the adjacency matrix of the graph \n"); for (i = 0; i < n; i++) {

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

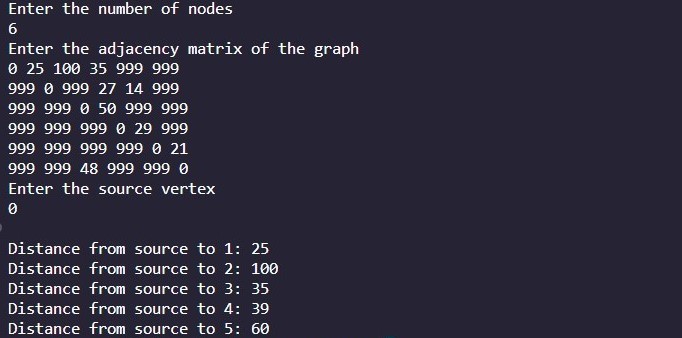
}

}

printf("Enter the source vertex \n"); scanf("%d", &u);

Dijkstra(Graph, n, u); return 0;

**OUTPUT:**



# PROGRAM 11

Implement “N-Queens Problem” using Backtracking.

## CODE:

#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;

}

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

}

}

}

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; //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:

