**VISVESVARAYA TECHNOLOGICAL UNIVERSITY**

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

****

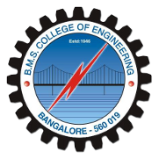
**LAB REPORT**

**on**

**Analysis and design of Algorithms**

***Submitted by***

**TANMAY AGARWAL (1WA23CS010)**

****

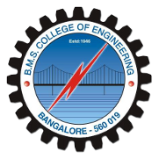
B. M. S. College of Engineering,

Bull Temple Road, Bangalore 560019

(Affiliated To Visvesvaraya Technological University, Belgaum) Department of Computer Science and Engineering

B. M. S. College of Engineering,

Bull Temple Road, Bangalore 560019



CERTIFICATE

This is to certify that the Lab work entitled “Analysis and design of algorithm– **23CS4PCADA**” carried out by **TANMAY AGARWAL(1WA23CS010),** 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 year 2024-2025. The Lab report has been approved as it satisfies the academic requirements in respect of Analysis and design of Algorithms - (23CS4PCADA) work prescribed for the said degree.

**Vikranth B.M Dr. Kavitha Sooda**

Assistant Professor Professor and HOD

Department of CSE Department of CSE

BMSCE, Bengaluru BMSCE, Bengaluru

**Index Sheet**

|  |  |  |
| --- | --- | --- |
| **Sl.**  **No.** | **Experiment Title** | **Page No.** |
| 1. | Write program to obtain the Topological ordering of vertices in a given digraph. | 4-6 |
| 2. | Sort a given set of N integer elements using Merge Sort technique and compute its time taken. | 7-9 |
| 3. | Sort a given set of N integer elements using Quick Sort technique and compute its time taken. | 10-12 |
| 4. | 1. Find Minimum Cost Spanning Tree of a given undirected graph using Prim’s algorithm. 2. Find Minimum Cost Spanning Tree of a given undirected graph using Kruskal’s algorithm. | 13-20 |
| 5. | From a given vertex in a weighted connected graph, find shortest paths to other vertices using Dijkstra’s algorithm. | 21-23 |
| 6. | Implement Johnson Trotter algorithm to generate permutations. | 24-27 |
| 7. | Implement fractional Knapsack problem using Greedy technique. | 28-30 |
| 8. | Implement 0/1 Knapsack problem using dynamic programming. | 31-33 |
| 9. | Sort a given set of N integer elements using Heap Sort technique and compute its time taken. | 34-36 |
| 10. | Implement All Pair Shortest paths problem using Floyd’s algorithm. | 37-40 |
| 11. | Implement “N-Queens Problem” using Backtracking. | 41-43 |

1. Write program to obtain the Topological ordering of vertices in a given digraph. #include<stdio.h>

int a[10][10],n,t[10],indegree[10]; int stack[10],top=-1;

void computeIndegree(int,int [][10]);

void tps\_SourceRemoval(int,int [][10]); int main(){

printf("Enter the no. of nodes: "); scanf("%d",&n);

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

}

}

computeIndegree(n,a); tps\_SourceRemoval(n,a); printf("Solution:"); for(i=0;i<n;i++){ printf("%d ",t[i]);

}

return 0;

}

void computeIndegree(int n,int a[][10]){ int i,j,sum=0;

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

for(j=0;j<n;j++){ sum=sum+a[j][i];

}

indegree[i]=sum;

}

}

void tps\_SourceRemoval(int n,int a[][10]){ int i,j,v;

for(i=0;i<n;i++){ if(indegree[i]==0){

stack[++top]=i;

}

}

int k=0; while(top!=-1){

v=stack[top--]; t[k++]=v; for(i=0;i<n;i++){

if(a[v][i]!=0){ indegree[i]=indegree[i]-1; if(indegree[i]==0){

stack[++top]=i;

}

}

}

}

}

Output –



1. Sort a given set of N integer elements using Merge Sort technique and compute its time taken. Code-

#include <stdio.h> #include<time.h> int a[20],n;

void simple\_sort(int [],int,int,int); void merge\_sort(int[],int,int);

int main() { int i;

clock\_t start, end; double time\_taken;

printf("Enter the no. of elements:"); scanf("%d", &n);

printf("Enter the array elements:"); for (i = 0; i < n; i++) { scanf("%d", &a[i]);

}

start = clock(); merge\_sort(a, 0, n - 1); end = clock();

time\_taken = (double)(end - start) / CLOCKS\_PER\_SEC; printf("Sorted array:");

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

}

printf("\n");

printf("Time taken to sort: %f seconds\n", time\_taken);

return 0;

}

void merge\_sort(int a[],int low, int high){ if(low<high){

int mid=(low+high)/2; merge\_sort(a,low,mid); merge\_sort(a,mid+1,high); simple\_sort(a,low,mid,high);

}

}

void simple\_sort(int a[],int low, int mid, int high){ int i=low,j=mid+1,k=low;

int c[n];

while(i<=mid && j<=high){ if(a[i]<a[j]){

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

}else{ c[k++]=a[j]; j++;

}

}

while(i<=mid){ c[k++]=a[i]; i++;

}

while(j<=high){

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

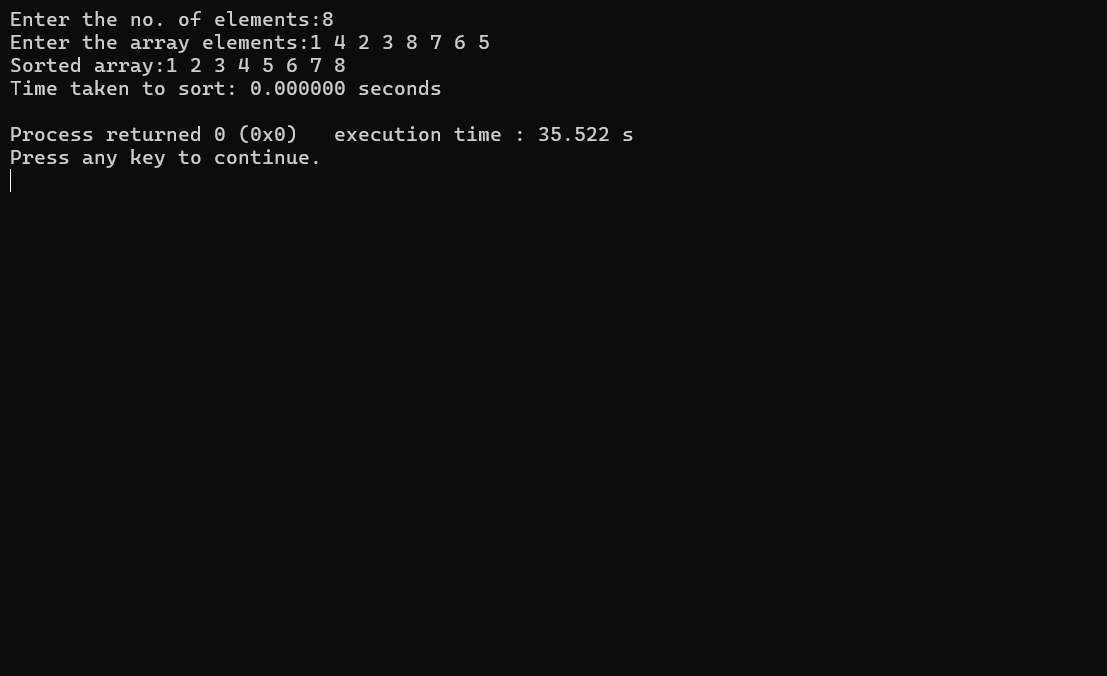
}

for(i=low;i<=high;i++){ a[i]=c[i];

}

}

Code-



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

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

int a[20],n;

int partition(int [],int, int); void quick\_sort(int [],int,int); void swap(int\*,int\*);

int main() { int i;

clock\_t start, end; double time\_taken;

printf("Enter the no. of elements:"); scanf("%d", &n);

printf("Enter the array elements:"); for (i = 0; i < n; i++) { scanf("%d", &a[i]);

}

start = clock(); quick\_sort(a, 0, n - 1); end = clock();

time\_taken = (double)(end - start) / CLOCKS\_PER\_SEC; printf("Sorted array:");

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

}

printf("\n");

printf("Time taken to sort: %f seconds\n", time\_taken); return 0;

}

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

\*a=\*b;

\*b=temp;

}

void quick\_sort(int a[],int low,int high){ if(low<high){

int mid=partition(a,low,high); quick\_sort(a,low,mid-1); quick\_sort(a,mid+1,high);

}

}

int partition(int a[],int low,int high){ int pivot=a[low];

int i=low;

int j=high+1; while(i<=j){

do{

i=i+1;

}while(a[i]<pivot && i<=high); do{

j=j-1;

} while(a[j]>pivot && j>=low); if(i<j){

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

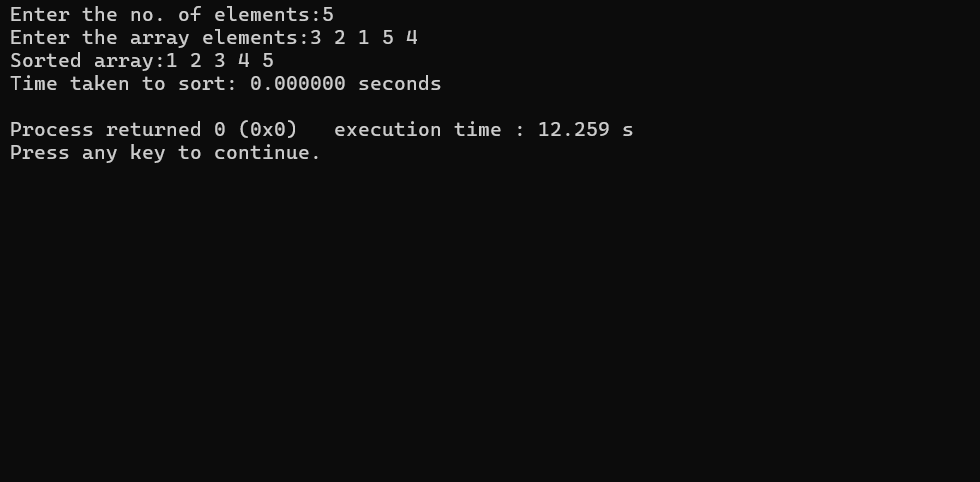
}

}

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

}

Output -



4 a. Find Minimum Cost Spanning Tree of a given undirected graph using Prim’s algorithm. Code -

#include <stdio.h>

int cost[10][10], n, t[10][2], sum; void prims(int cost[10][10], int n); int main() {

int i, j;

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

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

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

}

}

prims(cost, n);

printf("Edges of the minimal spanning tree:\n"); for (i = 0; i < n - 1; i++) {

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

}

printf("\nSum of minimal spanning tree: %d\n", sum); return 0;

}

void prims(int cost[10][10], int n) { int i, j, u, v;

int min, source;

int p[10], d[10], s[10];

min = 999;

source = 0;

// Initialize arrays

for (i = 0; i < n; i++) { d[i] = cost[source][i]; s[i] = 0;

p[i] = source;

}

s[source] = 1;

sum = 0; int k = 0;

// Find MST

for (i = 0; i < n - 1; i++) { min = 999;

u = -1;

// Find the vertex with minimum distance to the MST for (j = 0; j < n; j++) {

if (s[j] == 0 && d[j] < min) { min = d[j];

u = j;

}

}

if (u != -1) {

// Add edge to MST t[k][0] = u;

t[k][1] = p[u]; k++;

sum += cost[u][p[u]]; s[u] = 1;

// Update distances

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

if (s[v] == 0 && cost[u][v] < d[v]) { d[v] = cost[u][v];

p[v] = u;

}

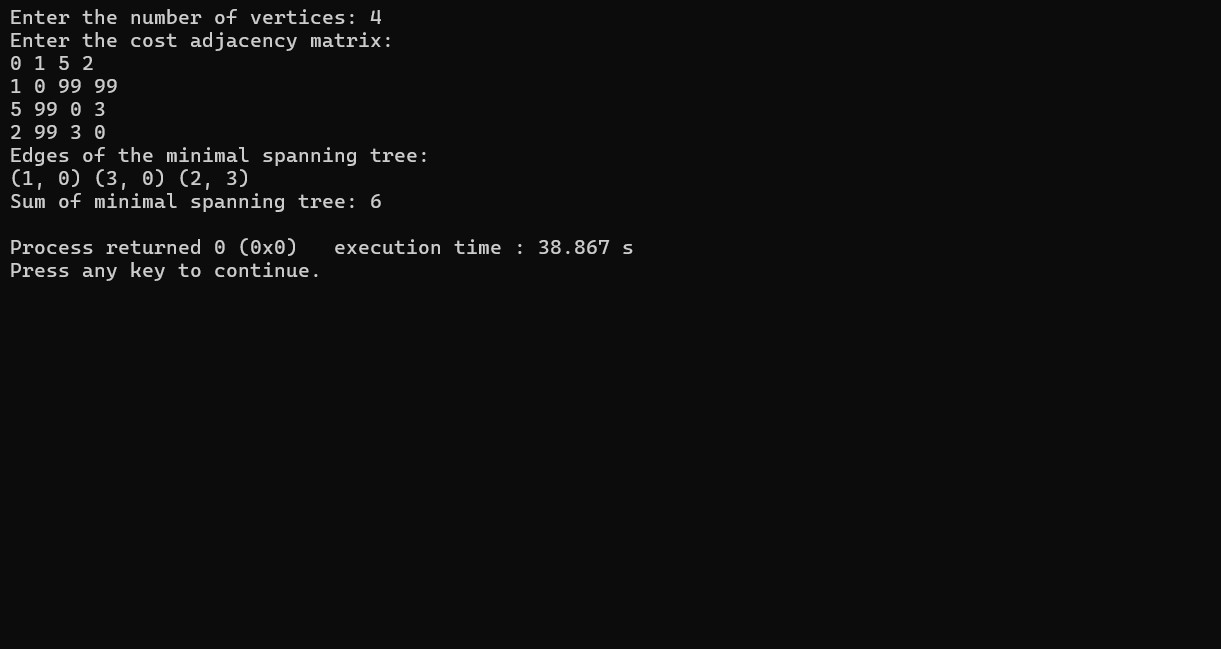
}

}

}

}

Output-



4 b. Find Minimum Cost Spanning Tree of a given undirected graph using Kruskal’s algorithm. Code-

#include <stdio.h>

int cost[10][10], n, t[10][2], sum; void kruskal(int cost[10][10], int n); int find(int parent[10], int i);

int main() { int i, j;

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

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

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

}

}

kruskal(cost, n);

printf("Edges of the minimal spanning tree:\n"); for (i = 0; i < n - 1; i++) {

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

}

printf("\nSum of minimal spanning tree: %d\n", sum); return 0;

}

void kruskal(int cost[10][10], int n) { int min, u, v, count, k;

int parent[10];

k = 0;

sum = 0;

// Initialize parent array for Union-Find for (int i = 0; i < n; i++) {

parent[i] = i;

}

count = 0;

while (count < n - 1) { min = 999;

u = -1;

v = -1;

// Find the minimum edge for (int i = 0; i < n; i++) {

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

if (find(parent, i) != find(parent, j) && cost[i][j] < min) { min = cost[i][j];

u = i; v = j;

}

}

}

// Perform Union operation int root\_u = find(parent, u); int root\_v = find(parent, v); if (root\_u != root\_v) {

parent[root\_u] = root\_v; t[k][0] = u;

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

count++;

}

}

}

int find(int parent[10], int i) { while (parent[i] != i) {

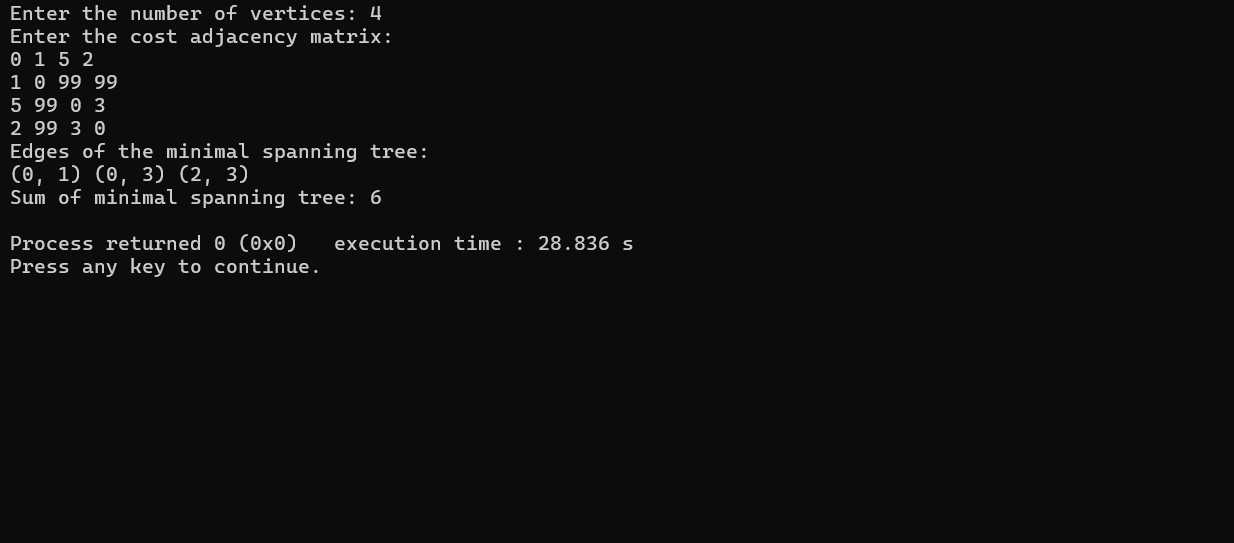
i = parent[i];

}

return i;

}

Output-



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

// C program to implement Dijkstra's algorithm #include <stdio.h>

int cost[10][10], n, result[10][2], weight[10]; void dijkstras(int [][10], int );

void main() { int i, j, s;

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

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

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

}

}

printf("Enter the source vertex: "); scanf("%d", &s);

dijkstras(cost, s); printf("Path:\n");

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

printf("(%d, %d) with weight %d ", result[i][0], result[i][1], weight[result[i][1]]);

}

}

void dijkstras(int cost[][10], int s){ int d[10], p[10], visited[10];

int i, j, min, u, v, k; for(i = 0; i < 10; i++){

d[i] = 999;

visited[i] = 0; p[i] = s;

}

d[s] = 0;

visited[s] = 1;

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

u = 0;

for(j = 0; j < n; j++){ if(visited[j] == 0){

if(d[j] < min){ min = d[j]; u = j;

}

}

}

visited[u] = 1;

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

if(visited[v] == 0 && (d[u] + cost[u][v] < d[v])){ d[v] = d[u] + cost[u][v];

p[v] = u;

}

}

}

for(i = 0; i < n; i++){ result[i][0] = p[i];

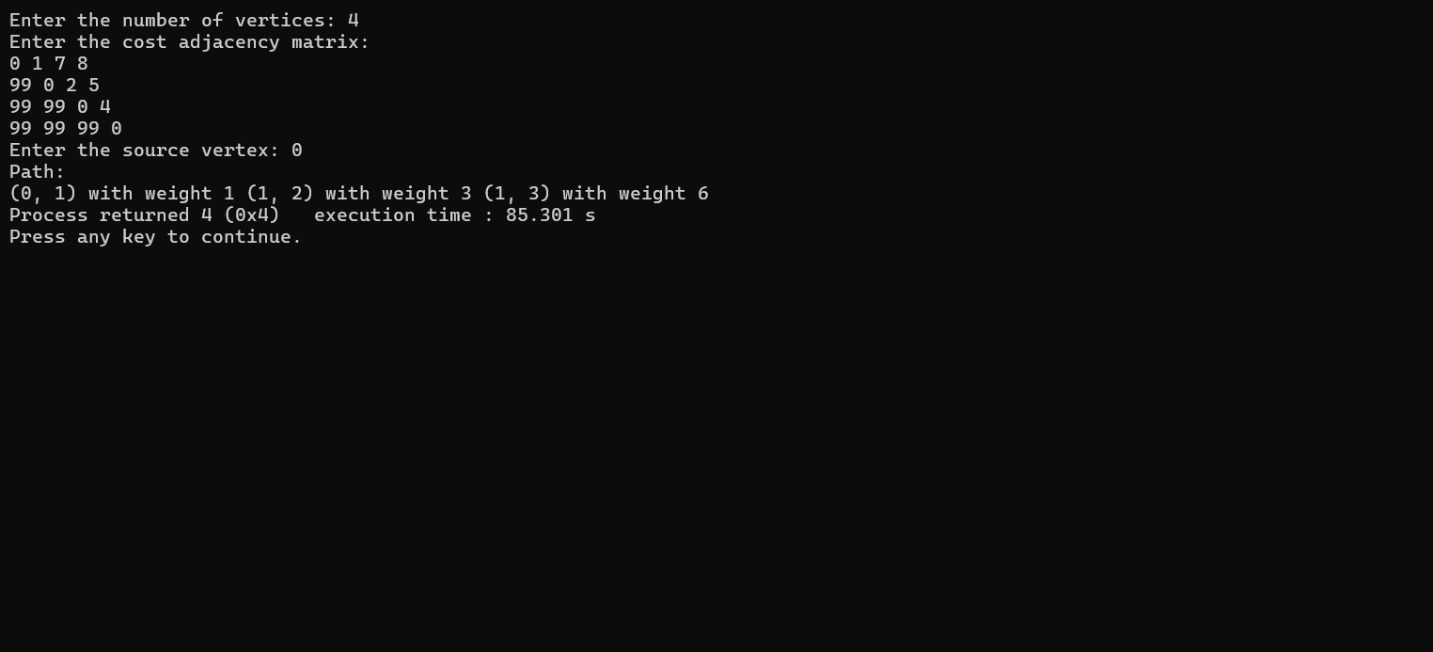
result[i][1] = i;

weight[i] = d[i];

}

}

Output –



6. Implement Johnson Trotter algorithm to generate permutations. #include <stdio.h>

#include <stdlib.h> #define LEFT -1

#define RIGHT 1

int getLargestMobile(int\* arr, int\* dir, int n) { int mobile = 0;

for (int i = 0; i < n; i++) { int neighbor = i + dir[i];

if (neighbor >= 0 && neighbor < n && arr[i] > arr[neighbor]) { if (arr[i] > mobile) {

mobile = arr[i];

}

}

}

return mobile;

}

int findIndex(int\* arr, int n, int val) { for (int i = 0; i < n; i++) {

if (arr[i] == val) return i;

}

return -1;

}

void printPermutation(int\* arr, int n) { for (int i = 0; i < n; i++)

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

}

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

for (int i = 2; i <= n; i++)

f \*= i; return f;

}

void johnsonTrotter(int\* arr, int n) { int dir[n];

for (int i = 0; i < n; i++) dir[i] = LEFT;

printPermutation(arr, n); // print first permutation int total = factorial(n);

for (int step = 1; step < total; step++) {

int mobile = getLargestMobile(arr, dir, n);

if (mobile == 0) break; // no more mobile integers int pos = findIndex(arr, n, mobile);

int moveTo = pos + dir[pos];

if (moveTo < 0 || moveTo >= n) { break;

}

int temp = arr[pos]; arr[pos] = arr[moveTo]; arr[moveTo] = temp; int dtemp = dir[pos]; dir[pos] = dir[moveTo]; dir[moveTo] = dtemp; pos = moveTo;

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

}

}

printPermutation(arr, n);

}

}

int main() { int n;

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

if (n <= 0) {

printf("Invalid input.\n"); return 1;

}

int arr[n];

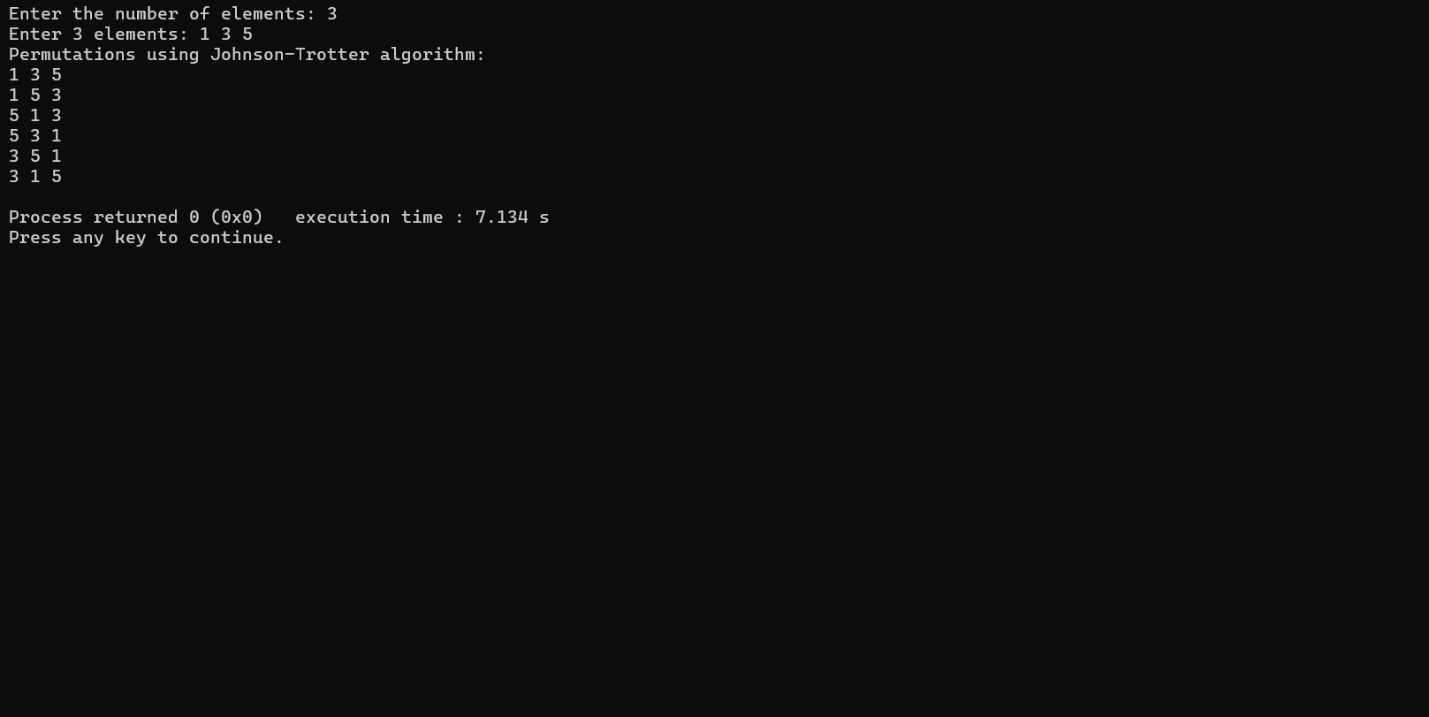
printf("Enter %d elements: ", n); for (int i = 0; i < n; i++)

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

printf("Permutations using Johnson-Trotter algorithm:\n"); johnsonTrotter(arr, n);

}

Output-



9. Implement fractional Knapsack problem using Greedy technique. Code-

#include <stdio.h>

void knapsack(int n, int p[], int w[], int W) { int used[n];

for (int i = 0; i < n; ++i) used[i] = 0;

int cur\_w = W; float tot\_v = 0.0; int i, maxi;

while (cur\_w > 0) { maxi = -1;

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

if ((used[i] == 0) && ((maxi == -1) || ((float)w[i]/p[i] > (float)w[maxi]/p[maxi]))) maxi = i;

used[maxi] = 1;

if (w[maxi] <= cur\_w) { cur\_w -= w[maxi]; tot\_v += p[maxi];

printf("Added object %d (%d, %d) completely in the bag. Space left: %d.\n", maxi + 1, w[maxi], p[maxi], cur\_w);

} else {

int taken = cur\_w; cur\_w = 0;

tot\_v += (float)taken/p[maxi] \* p[maxi];

printf("Added %d%% (%d, %d) of object %d in the bag.\n", (int)((float)taken/w[maxi] \* 100), w[maxi], p[maxi], maxi + 1);

}

}

printf("Filled the bag with objects worth %.2f.\n", tot\_v);

}

int main(){ int n, W;

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

int p[n], w[n];

printf("Enter the profits of the objects: "); for(int i = 0; i < n; i++){

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

}

printf("Enter the weights of the objects: "); for(int i = 0; i < n; i++){

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

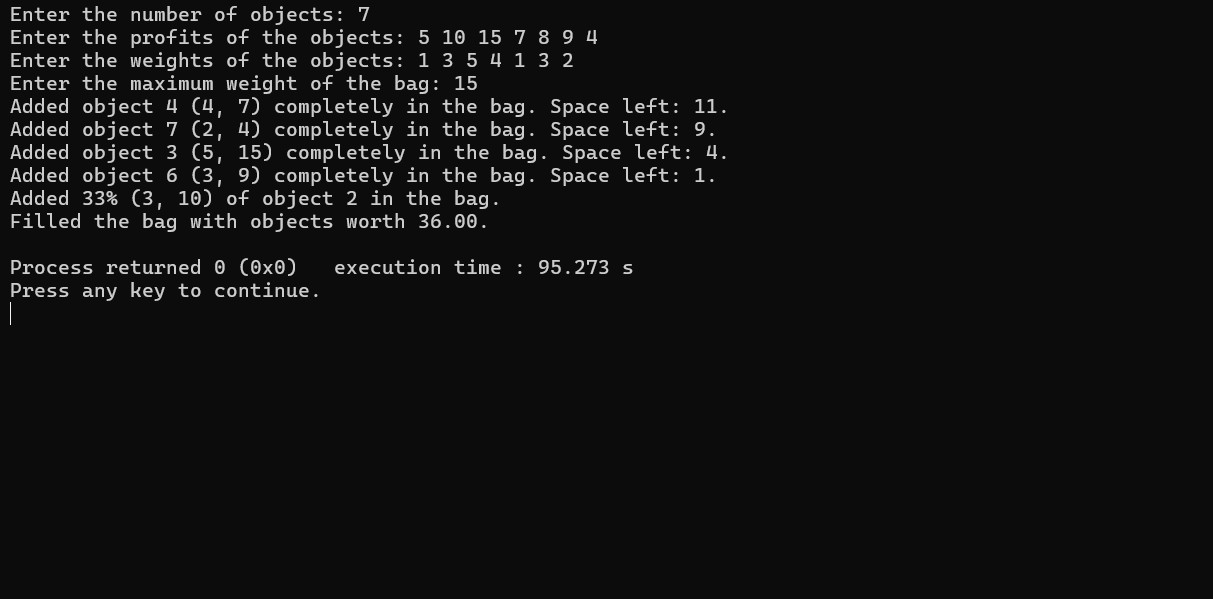
}

printf("Enter the maximum weight of the bag: "); scanf("%d", &W);

knapsack(n, p, w, W); return 0;

}

Output –



1. Implement 0/1 Knapsack problem using dynamic programming. Code-

#include <stdio.h>

int n,m,w[10],p[10],v[10][10];

void knapsack(int,int,int[],int[]); int max(int,int);

int main()

{

int i,j;

printf("Enter the no. of items:"); scanf("%d",&n);

printf("Enter the capacity of knapsack:"); scanf("%d",&m);

printf("Enter weights:"); for(i=0;i<n;i++){ scanf("%d",&w[i]);

}

printf("Enter profits:"); for(i=0;i<n;i++){ scanf("%d",&p[i]);

}

knapsack(n,m,w,p); printf("Optimal Solution:\n"); for(i=0;i<n;i++){ for(j=0;j<n;j++){

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

}

printf("\n");

}

return 0;

}

void knapsack(int n, int m, int w[],int p[]){ int i,j;

for(i=0;i<n;i++){ for(j=0;j<m;j++){ if(i==0 || j==0){ v[i][j]=0;

}else if(w[i]>j){ v[i][j]=v[i-1][j];

}else{

v[i][j]=max(v[i-1][j],((v[i-1][j-w[i]])+p[i]));

}

}

}

}

int max(int a,int b){ if(a>b){

return a;

}else{ return b;

}

}

Code-



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

#include <stdio.h>

int a[10][10],D[10][10],n;

void floyd(int [][10],int); int min(int,int);

int main(){

printf("Enter the no. of vertices:"); scanf("%d",&n);

printf("Enter the cost adjacency matrix:\n"); int i,j;

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

}

}

floyd(a,n);

printf("Distance Matrix:\n"); for(i=0;i<n;i++){ for(j=0;j<n;j++){

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

}

printf("\n");

}

return 0;

}

void floyd(int a[][10],int n){ int i,j,k;

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

for(j=0;j<n;j++){ D[i][j]=a[i][j];

}

}

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

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

}

}

}

}

int min(int a,int b){ if(a<b){

return a;

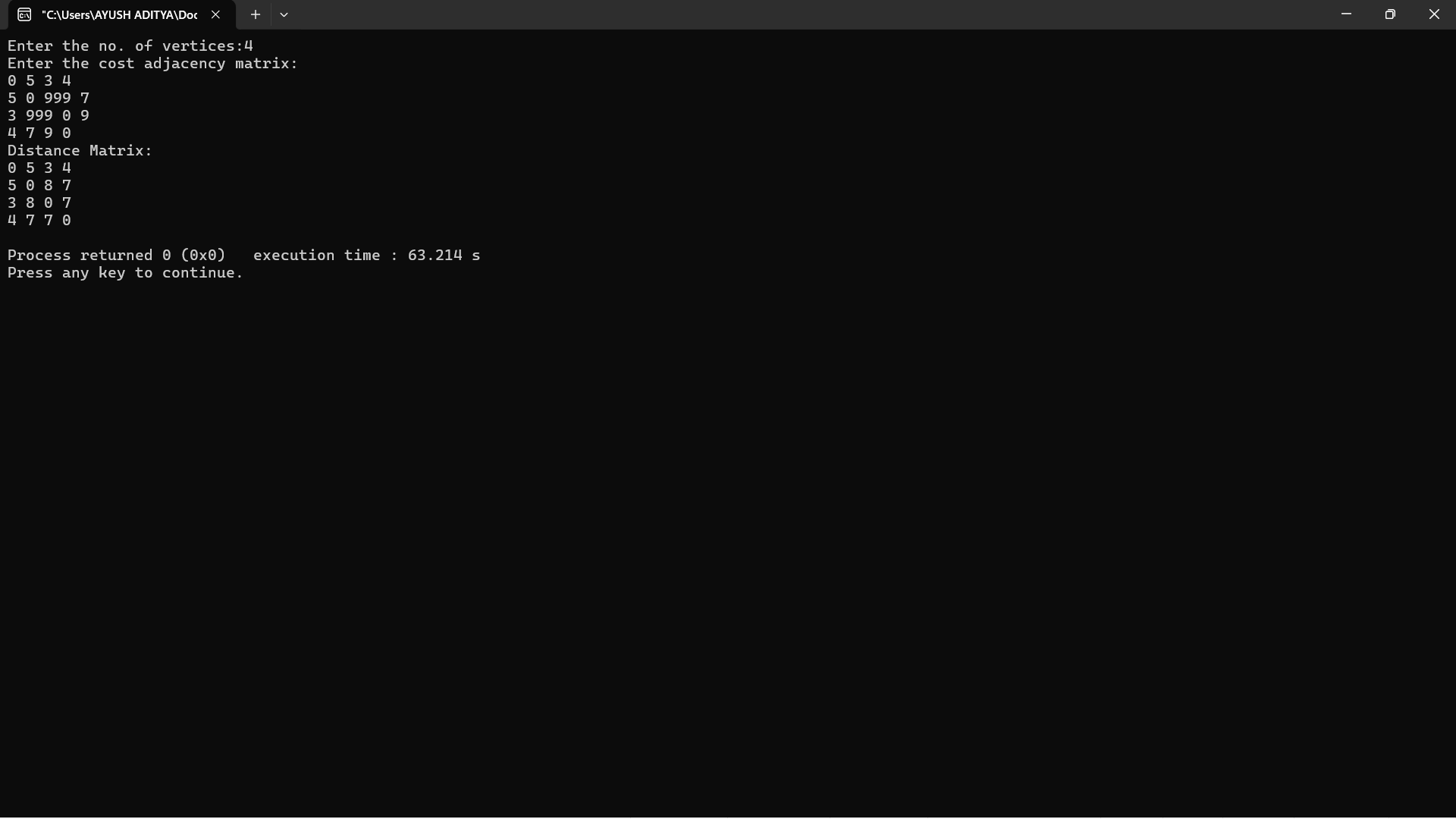
}else{

return b;

}

}

Output -



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

Code-

#include <stdio.h> #include<time.h> #define MAX 100 int a[MAX], n;

void heapify(int a[], int n, int i); void heapSort(int a[], int n); void swap(int \*x, int \*y);

int main() {

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

if (n > MAX) {

printf("Array size exceeds maximum limit.\n"); return 1;

}

printf("Enter array elements:\n"); for (int i = 0; i < n; i++) {

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

}

clock\_t start, end; start = clock(); heapSort(a, n); end = clock(); float total = 0;

printf("Sorted array using Heap Sort:\n"); for (int i = 0; i < n; i++) {

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

}

printf("\n");

total = (float)(end-start)/CLOCKS\_PER\_SEC; return 0;

}

// Function to perform heap sort void heapSort(int a[], int n) {

// Build max heap (heapify non-leaf nodes from bottom up) for (int i = n / 2 - 1; i >= 0; i--) {

heapify(a, n, i);

}

// One by one extract elements from heap for (int i = n - 1; i > 0; i--) {

// Move current root (max) to end swap(&a[0], &a[i]);

// Heapify reduced heap heapify(a, i, 0);

}

}

// To heapify a subtree rooted at index i in array of size n void heapify(int a[], 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 exists and is greater than root if (left < n && a[left] > a[largest])

largest = left;

// If right child exists and is greater than current largest if (right < n && a[right] > a[largest])

largest = right;

// If largest is not root, swap and continue heapifying if (largest != i) {

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

}

}

// Utility function to swap two elements void swap(int \*x, int \*y) {

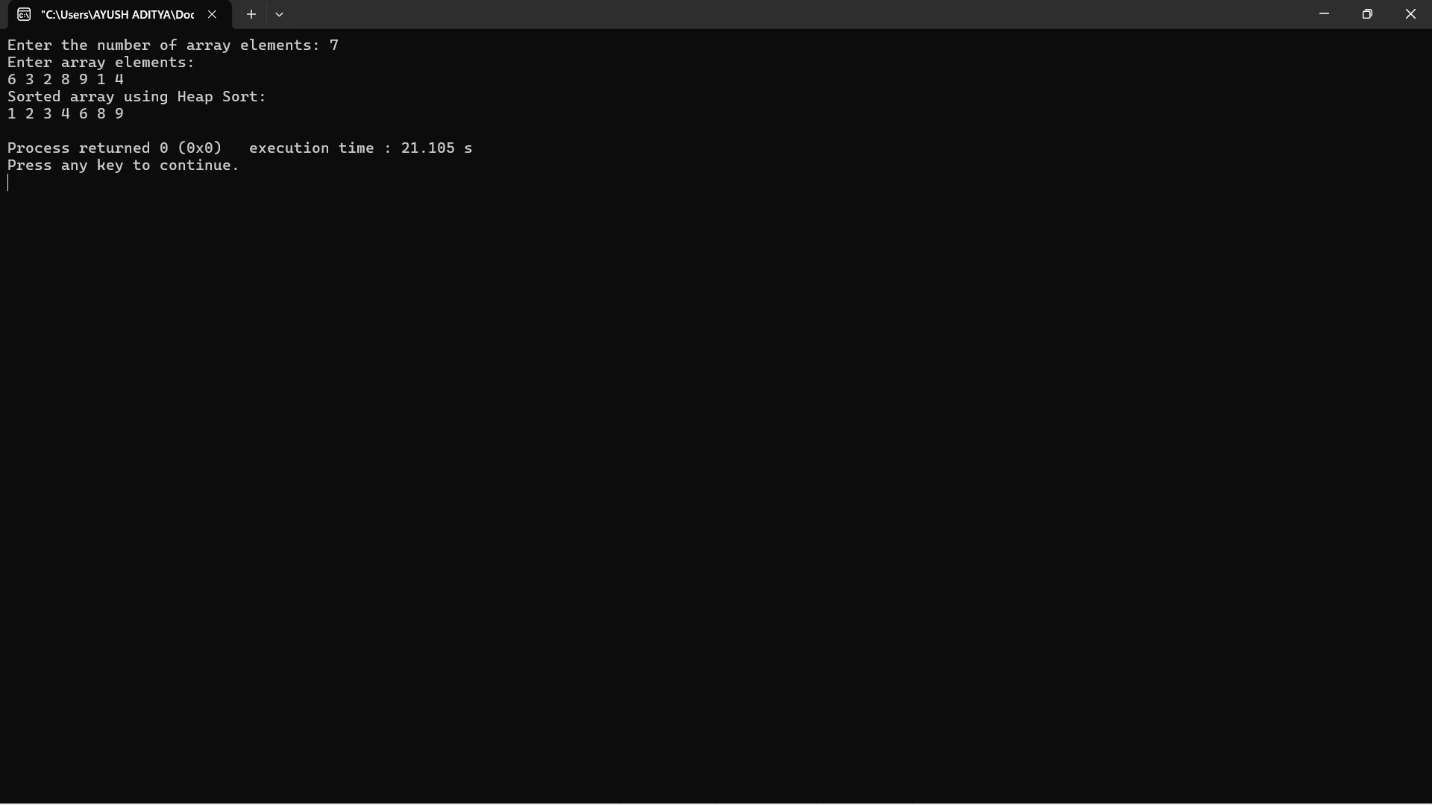
int temp = \*x;

\*x = \*y;

\*y = temp;

}

Output –



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

#include <stdio.h> #include <stdbool.h> bool place(int[], int);

void printSolution(int[], int); void nQueens(int);

void main() { int n;

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

nQueens(n);

}

void nQueens(int n){ int x[10];

int count=0; int k=1; while(k!=0){

x[k]=x[k]+1;

while(x[k]<=n && !place(x,k)){ x[k]=x[k]+1;

}

if(x[k]<=n){ if(k==n){

printSolution(x, n); printf("Solution found\n"); count++;

}else{

k++;

x[k]=0;

}

}else{

k--;

}

}

printf("Total solutions: %d\n", count);

}

bool place(int x[10], int k){ int i;

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

if((x[i]==x[k])||(i-x[i]==k-x[k])||(i+x[i]==k+x[k])){ return false;

}

}

return true;

}

void printSolution(int x[10], int n){ int i;

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

}

printf("\n");

}

Output-

