1.Design and implement C Program to find Minimum Cost Spanning Tree of a given connected undirected graph using Kruskal's algorithm

PROGRAM:

#include <stdio.h>

#include <stdlib.h>

#define MAX\_EDGES 1000

typedef struct Edge {

int src, dest, weight;

} Edge;

typedef struct Graph {

int V, E;

Edge edges[MAX\_EDGES];

} Graph;

typedef struct Subset {

int parent, rank;

} Subset;

Graph\* createGraph(int V, int E) {

Graph\* graph = (Graph\*) malloc(sizeof(Graph));

graph->V = V;

graph->E = E;

return graph;

}

int find(Subset subsets[], int i) {

if (subsets[i].parent != i) {

subsets[i].parent = find(subsets, subsets[i].parent);

}

return subsets[i].parent;

}

void Union(Subset subsets[], int x, int y) {

int xroot = find(subsets, x);

int yroot = find(subsets, y);

if (subsets[xroot].rank < subsets[yroot].rank) {

subsets[xroot].parent = yroot;

} else if (subsets[xroot].rank > subsets[yroot].rank) {

subsets[yroot].parent = xroot;

} else {

subsets[yroot].parent = xroot;

subsets[xroot].rank++;

}

}

int compare(const void\* a, const void\* b) {

Edge\* a\_edge = (Edge\*) a;

Edge\* b\_edge = (Edge\*) b;

return a\_edge->weight - b\_edge->weight;

}

void kruskalMST(Graph\* graph) {

Edge mst[graph->V];

int e = 0, i = 0;

qsort(graph->edges, graph->E, sizeof(Edge), compare);

Subset\* subsets = (Subset\*) malloc(graph->V \* sizeof(Subset));

for (int v = 0; v < graph->V; ++v) {

subsets[v].parent = v;

subsets[v].rank = 0;

}

while (e < graph->V - 1 && i < graph->E) {

Edge next\_edge = graph->edges[i++];

int x = find(subsets, next\_edge.src);

int y = find(subsets, next\_edge.dest);

if (x != y) {

mst[e++] = next\_edge;

Union(subsets, x, y);

}

}

printf("Minimum Spanning Tree:\n");

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

printf("(%d, %d) -> %d\n", mst[i].src, mst[i].dest, mst[i].weight);

}

}

int main() {

int V, E;

printf("Enter number of vertices and edges: ");

scanf("%d %d", &V, &E);

Graph\* graph = createGraph(V, E);

printf("Enter edges and their weights:\n");

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

scanf("%d %d %d", &graph->edges[i].src, &graph->edges[i].dest, &graph->edges[i].weight);

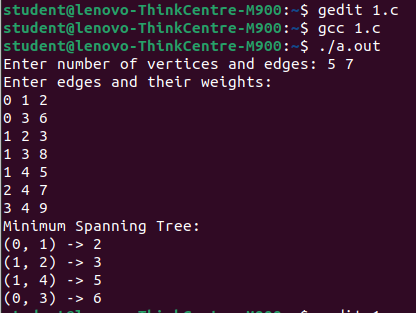
}

kruskalMST(graph);

return 0;

}

OUTPUT:



2.Design and implement C Program to find Minimum Cost Spanning Tree of a given connected undirected graph using Prim's algorithm

PROGRAM:

#include <stdio.h>

#include <limits.h>

#define V\_MAX 100 // Maximum number of vertices

// Function to find the vertex with the minimum key value, from the set of vertices not yet included in the MST

int minKey(int key[], int mstSet[], int V) {

int min = INT\_MAX, min\_index;

for (int v = 0; v < V; v++)

if (mstSet[v] == 0 && key[v] < min)

min = key[v], min\_index = v;

return min\_index;

}

// Function to print the constructed MST stored in parent[]

void printMST(int parent[], int n, int graph[V\_MAX][V\_MAX], int V) {

printf("Edge Weight\n");

for (int i = 1; i < V; i++)

printf("%d - %d %d \n", parent[i], i, graph[i][parent[i]]);

}

// Function to construct and print MST for a graph represented using adjacency matrix representation

void primMST(int graph[][V\_MAX], int V) {

int parent[V\_MAX]; // Array to store constructed MST

int key[V\_MAX]; // Key values used to pick minimum weight edge in cut

int mstSet[V\_MAX]; // To represent set of vertices not yet included in MST

// Initialize all keys as INFINITE, mstSet[] as 0

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

key[i] = INT\_MAX, mstSet[i] = 0;

// Always include first 1st vertex in MST. Make key 0 so that this vertex is picked as the first vertex

key[0] = 0;

parent[0] = -1; // First node is always the root of MST

// The MST will have V vertices

for (int count = 0; count < V - 1; count++) {

// Pick the minimum key vertex from the set of vertices not yet included in MST

int u = minKey(key, mstSet, V);

// Add the picked vertex to the MST set

mstSet[u] = 1;

// Update key value and parent index of the adjacent vertices of the picked vertex

// Consider only those vertices which are not yet included in the MST

for (int v = 0; v < V; v++)

if (graph[u][v] && mstSet[v] == 0 && graph[u][v] < key[v])

parent[v] = u, key[v] = graph[u][v];

}

// Print the constructed MST

printMST(parent, V, graph, V);

}

int main() {

int V, E;

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

scanf("%d %d", &V, &E);

// Create the graph as an adjacency matrix

int graph[V\_MAX][V\_MAX];

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

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

graph[i][j] = 0; // Initialize the graph with 0s

}

}

// Prompt the user to enter the source vertex, destination vertex, and weight for each edge

printf("Enter the source vertex, destination vertex, and weight for each edge:\n");

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

int source, dest, weight;

scanf("%d %d %d", &source, &dest, &weight);

graph[source][dest] = weight;

graph[dest][source] = weight; // Since the graph is undirected

}

// Print the MST using Prim's algorithm

primMST(graph, V);

return 0;

}

OUTPUT:



3.a. Design and implement C Program to solve All-Pairs Shortest Paths problem using Floyd's

algorithm.

PROGRAM:

#include<stdio.h>

int min(int,int);

void floyds(int p[10][10],int n) {

int i,j,k;

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

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

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

if(i==j)

p[i][j]=0; else

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

}

int min(int a,int b) {

if(a<b)

return(a); else

return(b);

}

void main() {

int p[10][10],w,n,e,u,v,i,j;

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

scanf("%d",&n);

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

scanf("%d",&e);

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

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

p[i][j]=999;

}

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

printf("\n Enter the end vertices of edge%d with its weight \n",i);

scanf("%d%d%d",&u,&v,&w);

p[u][v]=w;

}

printf("\n Matrix of input data:\n");

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

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

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

printf("\n");

}

floyds(p,n);

printf("\n Transitive closure:\n");

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

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

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

printf("\n");

}

printf("\n The shortest paths are:\n");

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

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

if(i!=j)

printf("\n <%d,%d>=%d",i,j,p[i][j]);

}

}

OUTPUT:

student@lenovo-ThinkCentre-M900:~$ gcc 3a.c

student@lenovo-ThinkCentre-M900:~$ ./a.out

Enter the number of vertices:4

Enter the number of edges:

5

Enter the end vertices of edge1 with its weight

1 3 3

Enter the end vertices of edge2 with its weight

2 1 2

Enter the end vertices of edge3 with its weight

3 2 7

Enter the end vertices of edge4 with its weight

3 4 1

Enter the end vertices of edge5 with its weight

4 1 6

Matrix of input data:

999 999 3 999

2 999 999 999

999 7 999 1

6 999 999 999

Transitive closure:

0 10 3 4

2 0 5 6

7 7 0 1

6 16 9 0

The shortest paths are:

<1,2>=10

<1,3>=3

<1,4>=4

<2,1>=2

<2,3>=5

<2,4>=6

<3,1>=7

<3,2>=7

<3,4>=1

<4,1>=6

<4,2>=16

3b.Design and implement C Program to find the transitive closure using Warshal's algorithm.

PROGRAM:

#include<stdio.h>

#include<math.h>

int max(int, int);

void warshal(int p[10][10], int n) {

int i, j, k;

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

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

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

p[i][j] = max(p[i][j], p[i][k] && p[k][j]);

}

int max(int a, int b) {

;

if (a > b)

return (a);

else

return (b);

}

void main() {

int p[10][10] = { 0 }, n, e, u, v, i, j;

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

scanf("%d", &n);

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

scanf("%d", &e);

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

printf("\n Enter the end vertices of edge %d:", i);

scanf("%d%d", &u, &v);

p[u][v] = 1;

}

printf("\n Matrix of input data: \n");

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

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

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

printf("\n");

}

warshal(p, n);

printf("\n Transitive closure: \n");

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

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

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

printf("\n");

}

}

OUTPUT:

student@lenovo-ThinkCentre-M900:~$ gedit 3b.c

student@lenovo-ThinkCentre-M900:~$ gcc 3b.c

student@lenovo-ThinkCentre-M900:~$ ./a.out

Enter the number of vertices:5

Enter the number of edges:11

Enter the end vertices of edge 1:1 1

Enter the end vertices of edge 2:1 4

Enter the end vertices of edge 3:3 2

Enter the end vertices of edge 4:3 3

Enter the end vertices of edge 5:3 4

Enter the end vertices of edge 6:4 2

Enter the end vertices of edge 7:4 4

Enter the end vertices of edge 8:5 2

Enter the end vertices of edge 9:5 3

Enter the end vertices of edge 10:5 4

Enter the end vertices of edge 11:5 5

Matrix of input data:

1 0 0 1 0

0 0 0 0 0

0 1 1 1 0

0 1 0 1 0

0 1 1 1 1

Transitive closure:

1 1 0 1 0

0 0 0 0 0

0 1 1 1 0

0 1 0 1 0

0 1 1 1 1

4.Design and implement C Program to find shortest paths from a given vertex in a weighted

connected graph to other vertices using Dijkstra's algorithm

PROGRAM:

#include <stdio.h>

#include <stdbool.h>

#include <limits.h>

#define MAX\_VERTICES 10 // Maximum number of vertices

#define INF INT\_MAX

// A function to find the vertex with the minimum distance value, from the set of vertices not yet included in the shortest path tree

int minDistance(int dist[], bool sptSet[], int V) {

int min = INF, min\_index;

for (int v = 0; v < V; v++)

if (sptSet[v] == false && dist[v] <= min)

min = dist[v], min\_index = v;

return min\_index;

}

// A utility function to print the constructed distance array

void printSolution(int dist[], int V) {

printf("Vertex \t\t Distance from Source\n");

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

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

}

// Dijkstra's algorithm for adjacency matrix representation of the graph

void dijkstra(int graph[MAX\_VERTICES][MAX\_VERTICES], int src, int V) {

int dist[MAX\_VERTICES]; // The output array. dist[i] will hold the shortest distance from src to i

bool sptSet[MAX\_VERTICES]; // sptSet[i] will be true if vertex i is included in the shortest path tree

// Initialize all distances as INFINITE and sptSet[] as false

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

dist[i] = INF, sptSet[i] = false;

dist[src] = 0;

// Find shortest path for all vertices

for (int count = 0; count < V - 1; count++) {

int u = minDistance(dist, sptSet, V);

sptSet[u] = true;

for (int v = 0; v < V; v++)

if (!sptSet[v] && graph[u][v] && dist[u] != INF && dist[u] + graph[u][v] < dist[v])

dist[v] = dist[u] + graph[u][v];

}

printSolution(dist, V);

}

// Driver code

int main() {

int V, E;

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

scanf("%d", &V);

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

scanf("%d", &E);

int graph[MAX\_VERTICES][MAX\_VERTICES] = {{0}};

printf("Enter the source vertex, destination vertex, and weight for each edge:\n");

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

int source, dest, weight;

scanf("%d %d %d", &source, &dest, &weight);

graph[source][dest] = weight;

graph[dest][source] = weight; // Assuming undirected graph

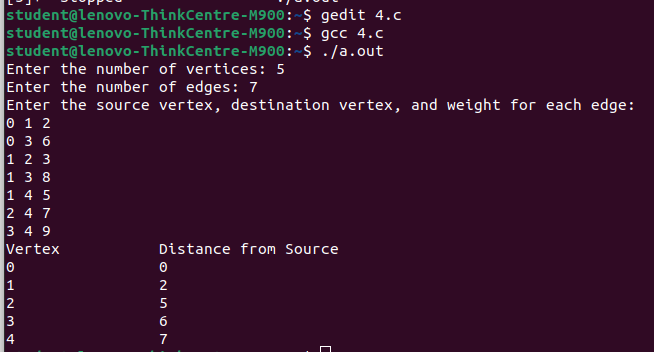
}

dijkstra(graph, 0, V);

return 0;

}

OUTPUT:



5.Design and implement C Program to obtain the Topological ordering of vertices in a given

digraph.

PROGRAM:

#include <stdio.h>

#include <stdlib.h>

#define MAX\_VERTICES 100

// Structure to represent a graph

typedef struct {

int V;

int\*\* adjMatrix;

} Graph;

// Function to create a new graph

Graph\* createGraph(int V) {

Graph\* graph = (Graph\*)malloc(sizeof(Graph));

graph->V = V;

graph->adjMatrix = (int\*\*)calloc(V, sizeof(int\*));

for (int i = 0; i < V; i++) graph->adjMatrix[i] = (int\*)calloc(V, sizeof(int));

return graph;

}

// Function to add an edge to the graph

void addEdge(Graph\* graph, int src, int dest) {

graph->adjMatrix[src][dest] = 1;

}

// Function to perform topological sorting

void topologicalSort(Graph\* graph) {

int V = graph->V, inDegree[MAX\_VERTICES] = {0}, queue[MAX\_VERTICES], front = 0, rear = -1;

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

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

if (graph->adjMatrix[i][j] == 1) inDegree[j]++;

for (int i = 0; i < V; i++) if (inDegree[i] == 0) queue[++rear] = i;

printf("Topological ordering of vertices: ");

while (front <= rear) {

int vertex = queue[front++];

printf("%d ", vertex);

for (int i = 0; i < V; i++) if (graph->adjMatrix[vertex][i] == 1 && --inDegree[i] == 0) queue[++rear] = i;

}

printf("\n");

}

// Driver code

int main() {

int V, E;

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

scanf("%d", &V);

Graph\* graph = createGraph(V);

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

scanf("%d", &E);

printf("Enter the edges (source vertex, destination vertex):\n");

for (int i = 0, src, dest; i < E; i++) {

scanf("%d %d", &src, &dest);

addEdge(graph, src, dest);

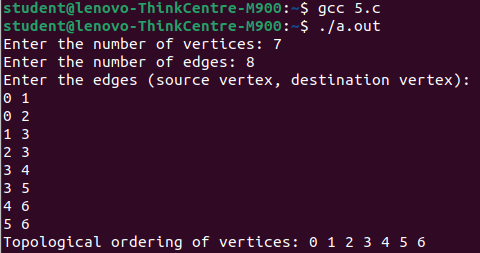
}

topologicalSort(graph);

return 0;

}

OUTPUT:



6.Design and implement C Program to solve 0/1 Knapsack problem using Dynamic Programming method.

PROGRAM:

#include <stdio.h>

// Function to find maximum of two integers

int max(int a, int b) {

return (a > b) ? a : b;

}

// Function to solve 0/1 Knapsack problem

int knapsack(int W, int wt[], int val[], int n) {

int i, w;

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

// Build table K[][] in bottom-up manner

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];

}

}

// K[n][W] contains the maximum value that can be put in a knapsack of capacity W

return K[n][W];

}

int main() {

int val[100], wt[100]; // Arrays to store values and weights

int W, n; // Knapsack capacity and number of items

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

scanf("%d", &n);

printf("Enter the values and weights of %d items:\n", n);

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

printf("Enter value and weight for item %d: ", i + 1);

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

}

printf("Enter the knapsack capacity: ");

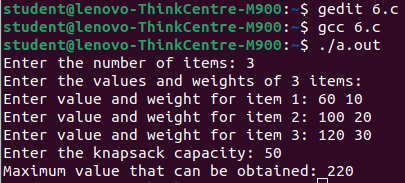
scanf("%d", &W);

printf("Maximum value that can be obtained: %d\n", knapsack(W, wt, val, n));

return 0;

}

OUTPUT:



7.Design and implement C Program to solve discrete Knapsack and continuous Knapsack problems using greedy approximation method.

PROGRAM:

#include<stdio.h>

int main()

{

float weight[50],profit[50],ratio[50],Totalvalue,temp,capacity,amount;

int n,i,j;

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

scanf("%d",&n);

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

{

printf("Enter Weight and Profit for item[%d] :\n",i);

scanf("%f %f", &weight[i], &profit[i]);

}

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

scanf("%f",&capacity);

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

ratio[i]=profit[i]/weight[i];

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

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

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

{

temp = ratio[j];

ratio[j] = ratio[i];

ratio[i] = temp;

temp = weight[j];

weight[j] = weight[i];

weight[i] = temp;

temp = profit[j];

profit[j] = profit[i];

profit[i] = temp;

}

printf("Knapsack problems using Greedy Algorithm:\n");

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

{

if (weight[i] > capacity)

break;

else

{

Totalvalue = Totalvalue + profit[i];

capacity = capacity - weight[i];

}

}

if (i < n)

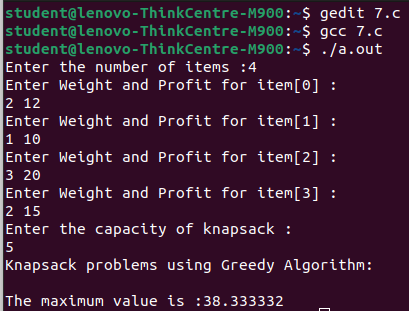
Totalvalue = Totalvalue + (ratio[i]\*capacity);

printf("\nThe maximum value is :%f\n",Totalvalue);

return 0;

}

OUTPUT:



8.Design and implement C Program to find a subset of a given set S = {sl , s2,.....,sn} of n positive integers whose sum is equal to a given positive integer d.

PROGRAM:

#include <stdio.h>

#include <stdbool.h>

#define MAX\_SIZE 100

// Function to find subset with given sum

void subsetSum(int set[], int subset[], int n, int subSize, int total, int nodeCount, int sum) {

if (total == sum) {

// Print the subset

printf("Subset found: { ");

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

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

}

printf("}\n");

return;

} else {

// Check the sum of the remaining elements

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

subset[subSize] = set[i];

subsetSum(set, subset, n, subSize + 1, total + set[i], i + 1, sum);

}

}

}

int main() {

int set[MAX\_SIZE];

int subset[MAX\_SIZE];

int n, sum;

// Input the number of elements in the set

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

scanf("%d", &n);

// Input the elements of the set

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

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

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

}

// Input the target sum

printf("Enter the sum to find subset for: ");

scanf("%d", &sum);

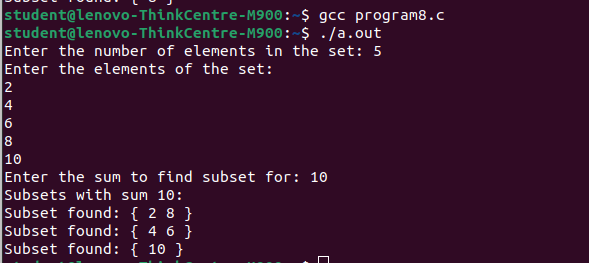
printf("Subsets with sum %d:\n", sum);

subsetSum(set, subset, n, 0, 0, 0, sum);

return 0;

}

OUTPUT:



9.Design and implement C Program to sort a given set of n integer elements using Selection Sort method and compute its time complexity. Run the program for varied values of n> 5000 and record the time taken to sort. Plot a graph of the time taken versus n. The elements can be read from a file or can be generated using the random number generator.

PROGRAM:

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

// Function to swap two integers

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

int temp = \*a;

\*a = \*b;

\*b = temp;

}

// Function to perform Selection Sort

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

int i, j, min\_idx;

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

min\_idx = i;

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

if (arr[j] < arr[min\_idx])

min\_idx = j;

}

swap(&arr[min\_idx], &arr[i]);

}

}

int main() {

int n;

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

scanf("%d", &n);

int arr[n];

srand(time(0)); // Seed for random number generation

// Generating random numbers for elements

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

arr[i] = rand() % 10000; // Generating random numbers between 0 and 9999

}

clock\_t start, end;

double cpu\_time\_used;

start = clock();

selectionSort(arr, n);

end = clock();

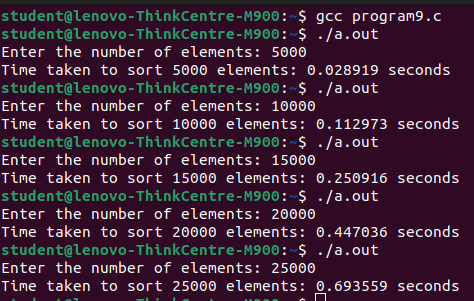
cpu\_time\_used = ((double) (end - start)) / CLOCKS\_PER\_SEC;

printf("Time taken to sort %d elements: %f seconds\n", n, cpu\_time\_used);

return 0;

}

OUTPUT:



10.Design and implement C Program to sort a given set of n integer elements using Quick Sort method and compute its time complexity. Run the program for varied values of n> 5000 and record the time taken to sort. Plot a graph of the time taken versus n. The elements can be read from a file or can be generated using the random number generator.

PROGRAM:

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

// Function to swap two integers

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

int temp = \*a;

\*a = \*b;

\*b = temp;

}

// Function to partition the array and return the pivot index

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

int pivot = arr[high];

int i = low - 1;

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

if (arr[j] < pivot) {

i++;

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

}

}

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

return i + 1;

}

// Function to implement Quick Sort

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

if (low < high) {

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

quickSort(arr, low, pi - 1);

quickSort(arr, pi + 1, high);

}

}

int main() {

FILE \*fp;

fp = fopen("numbers.txt", "w");

int n;

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

scanf("%d", &n);

srand(time(NULL));

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

int num = rand() % 10000;

fprintf(fp, "%d ", num);

}

fclose(fp);

int arr[n];

fp = fopen("numbers.txt", "r");

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

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

}

fclose(fp);

clock\_t start, end;

double cpu\_time\_used;

start = clock();

quickSort(arr, 0, n - 1);

end = clock();

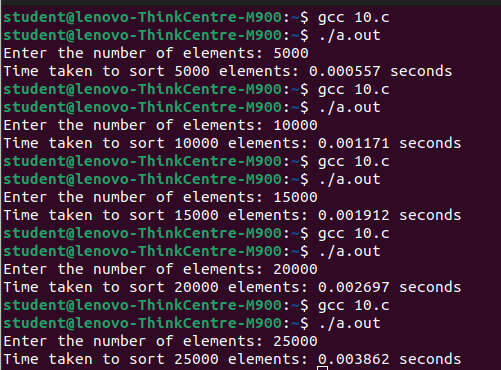
cpu\_time\_used = ((double)(end - start)) / CLOCKS\_PER\_SEC;

printf("Time taken to sort %d elements: %f seconds\n", n, cpu\_time\_used);

return 0;

}

OUTPUT:



11.Design and implement C Program to sort a given set of n integer elements using Merge Sort method and compute its time complexity. Run the program for varied values of n> 5000, and record the time taken to sort. Plot a graph of the time taken versus n. The elements can be read from a file or can be generated using the random number generator.

PROGRAM:

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

// Function to merge two subarrays arr[l..m] and arr[m+1..r]

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

int i, j, k;

int n1 = m - l + 1;

int n2 = r - m;

// Create temporary arrays

int L[n1], R[n2];

// Copy data to temporary arrays L[] and R[]

for (i = 0; i < n1; i++)

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

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

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

// Merge the temporary arrays back into arr[l..r]

i = 0; // Initial index of first subarray

j = 0; // Initial index of second subarray

k = l; // Initial index of merged subarray

while (i < n1 && j < n2) {

if (L[i] <= R[j]) {

arr[k] = L[i];

i++;

} else {

arr[k] = R[j];

j++;

}

k++;

}

// Copy the remaining elements of L[], if any

while (i < n1) {

arr[k] = L[i];

i++;

k++;

}

// Copy the remaining elements of R[], if any

while (j < n2) {

arr[k] = R[j];

j++;

k++;

}

}

// Merge Sort function

void mergeSort(int arr[], int l, int r) {

if (l < r) {

// Same as (l+r)/2, but avoids overflow for large l and r

int m = l + (r - l) / 2;

// Sort first and second halves

mergeSort(arr, l, m);

mergeSort(arr, m + 1, r);

// Merge the sorted halves

merge(arr, l, m, r);

}

}

int main() {

FILE \*fp;

fp = fopen("numbers.txt", "w");

int n;

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

scanf("%d", &n);

srand(time(NULL));

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

int num = rand() % 10000;

fprintf(fp, "%d ", num);

}

fclose(fp);

int arr[n];

fp = fopen("numbers.txt", "r");

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

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

}

fclose(fp);

clock\_t start, end;

double cpu\_time\_used;

start = clock();

mergeSort(arr, 0, n - 1);

end = clock();

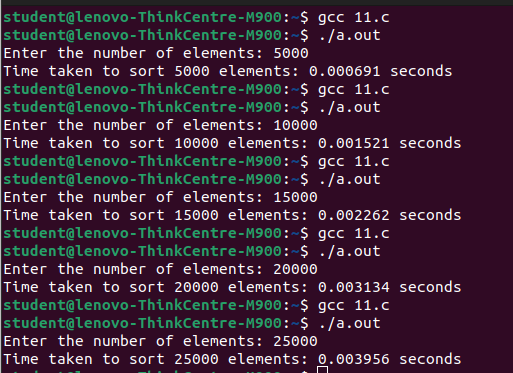
cpu\_time\_used = ((double)(end - start)) / CLOCKS\_PER\_SEC;

printf("Time taken to sort %d elements: %f seconds\n", n, cpu\_time\_used);

return 0;

}

OUTPUT:



12.Design and implement C Program for N Queen's problem using Backtracking

PROGRAM:

#include<stdio.h>

#include<math.h>

#include<stdlib.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:

