RNS INSTITUTE OF TECHNOLOGY

(AICTE Approved, VTU Affiliated and NAAC 'A' Accredited)

UG Programs - CSE, ECE, ISE, EIE and EEE have been Accredited by NBA up to 30.06.2025

DR. VISHNUVARDHAN ROAD, CHANNASANDRA, RR NAGAR POST, BENGALURU – 560 098

ANALYSIS & DESIGN OF ALGORITHMS LABORATORY MANUAL

For FOURTH Semester B.E [VTU/NEP, 2022 syllabus]

Subject Code – BCSL404

NAME:_____

BRANCH: _____

SECTION :_____

USN :_____

VISION AND MISSION OF INSTITUTION

Vision

Building RNSIT into a World Class Institution

Mission

To impart high quality education in Engineering, Technology and Management with a Difference, Enabling Students to Excel in their Career by

1. Attracting quality Students and preparing them with a strong foundation in fundamentals so as to achieve distinctions in various walks of life leading to outstanding contributions
2. Imparting value based, need based, choice based and skill based professional education to the aspiring youth and carving them into disciplined, World class Professionals with social responsibility
3. Promoting excellence in Teaching, Research and Consultancy that galvanizes academic consciousness among Faculty and Students
4. Exposing Students to emerging frontiers of knowledge in various domains and make them suitable for Industry, Entrepreneurship, Higher studies, and Research & Development
5. Providing freedom of action and choice for all the Stake holders with better visibility

VISION AND MISSION OF DEPARTMENT

Vision

Preparing Better Computer Professionals for a Real World

Mission

The Department of CSE will make every effort to promote an intellectual and ethical environment by

1. Imparting solid foundations and applied aspects in both Computer Science Theory and Programming practices
2. Providing training and encouraging R&D and Consultancy Services in frontier areas of Computer Science and Engineering with a Global outlook
3. Fostering the highest ideals of ethics, values and creating awareness of the role of Computing in Global Environment
4. Educating and preparing the graduates, highly sought after, productive, and well-respected for their work culture.
5. Supporting and inducing lifelong learning

ANALYSIS & DESIGN OF ALGORITHMS LABORATORY-BCSL404

INTERNAL EVALUATION SHEET

EVALUATION (MAX MARKS 40)			
TEST A	**REGULAR EVALUATION B**	**RECORD C**	**TOTAL MARKS A+B+C**
20	**10**	**20**	**50**

R1: REGULAR LAB EVALUATION WRITE UP RUBRIC (MAX MARKS 10)				
Sl. No.	**Parameters**	**Good**	**Average**	**Needs improvement**
a.	**Understanding of problem (3 marks)**	Clear understanding of problem statement while designing and implementing the program (3)	Problem statement is understood clearly but few mistakes while designing and implementing program (2)	Problem statement is not clearly understood while designing the program (1)
b.	**Writing program (4 marks)**	Program handles all possible conditions (4)	Average condition is defined and verified. (3)	Program does not handle possible conditions (1)
c.	**Result and documentation (3 marks)**	Meticulous documentation and all conditions are taken care (3)	Acceptable documentation shown (2)	Documentation does not take care all conditions (1)

R2: REGULAR LAB EVALUATION VIVA RUBRIC (MAX MARKS 10)					
Sl. No.	**Parameter**	**Excellent**	**Good**	**Average**	**Needs Improvement**
a.	**Conceptual understanding (10 marks)**	Answers 80% of the viva questions asked (10)	Answers 60% of the viva questions asked (7)	Answers 30% of the viva questions asked (4)	Unable to relate the concepts (1)

R3: REGULAR LAB PROGRAM EXECUTION RUBRIC (MAX MARKS 10)				
Sl. No.	**Parameters**	**Excellent**	**Good**	**Needs Improvement**
a.	**Design, implementation, and demonstration (5 marks)**	Program follows syntax and semantics of C programming language. Demonstrates the complete knowledge of the program written (5)	Program has few logical errors, moderately demonstrates all possible concepts implemented in programs (3)	Syntax and semantics of C programming is not clear (1)
b.	**Result and documentation (5 marks)**	All test cases are successful, all errors are debugged with own practical knowledge and clear documentation according to the guidelines (5)	Moderately debugs the programs , few test case are unsuccessful and Partial documentation (3)	Test cases are not taken care , unable to debug the errors and no proper documentation (1)

R4: RECORD EVALUATION RUBRIC (MAX MARKS 20)					
Sl. No.	**Parameter**	**Excellent**	**Good**	**Average**	**Needs Improvement**
a.	**Documentation (20 marks)**	Meticulous record writing including program, comments and test cases as per the guidelines mentioned (20)	Write up contains program and test cases, but comments are not included (18)	Write up contains only program (15)	Program written with few mistakes (10)

TEST /LAB INTERNALS MARKS (MAX MARKS 20)

TEST #	Write up 6	Execution 28	Viva 6	Sign	Total 40	Avg. 40	Final 20
TEST-1							
TEST-2						**40**	**20**

REGULAR LAB EVALUATION (MAX MARKS 10)

Lab program	Date of Execution	Additional programs	Write up (10)	Exen. (10)	Viva (10)	Total 30	Teacher Signature
1							
2							
3							
4							
5							
6							
7							
8							
9							
10							
11							
12							
Total Marks		**360**	**30**			**10**	

Final Marks obtained from **A. Test (20) +** **B. Regular Evaluation (10) +** **C. Record (20)**	50	**Lab in charge:** **HOD:**

PREFACE

We have developed this comprehensive laboratory manual on **Analysis & Design of Algorithms Lab** with two primary objectives: To make the students comfortable with various data structures for solving the problems and to train them in choosing efficient algorithmic techniques as an efficient C programmer by strengthening their programming abilities.

This material provides the students an exposure to problem solving approaches and solutions to prescribed problems using C programming language. The problems discussed in this manual comprise of a programming solution, viva questions and practicing programming problems constitute an indispensable part of this material.

Our profound and sincere efforts will be fruitful only when students acquire extensive knowledge by reading this manual and apply the concepts learnt apart from the requirements specified in Analysis & Design of Algorithms Laboratory as prescribed by VTU, Belagavi.

Department of CSE

TABLE OF CONTENTS

Analysis & Design of Algorithms Lab		Semester	4
Course Code	**BCSL404**	CIE Marks	50
Teaching Hours/Week (L:T:P: S)	0:0:2:0	SEE Marks	50
Credits	01	Exam Hours	2
Examination type (SEE)	Practical		

Course objectives:

- To design and implement various algorithms in C/C++ programming using suitable development tools to address different computational challenges.
- To apply diverse design strategies for effective problem-solving.
- To Measure and compare the performance of different algorithms to determine their efficiency and suitability for specific tasks.

Sl.No	Experiments
1	Design and implement C/C++ Program to find Minimum Cost Spanning Tree of a given connected undirected graph using Kruskal's algorithm.
2	Design and implement C/C++ Program to find Minimum Cost Spanning Tree of a given connected undirected graph using Prim's algorithm.
3	a. Design and implement C/C++ Program to solve All-Pairs Shortest Paths problem using Floyd's algorithm. b. Design and implement C/C++ Program to find the transitive closure using Warshal's algorithm.
4	Design and implement C/C++ Program to find shortest paths from a given vertex in a weighted connected graph to other vertices using Dijkstra's algorithm.
5	Design and implement C/C++ Program to obtain the Topological ordering of vertices in a given digraph.
6	Design and implement C/C++ Program to solve 0/1 Knapsack problem using Dynamic Programming method.
7	Design and implement C/C++ Program to solve discrete Knapsack and continuous Knapsack problems using greedy approximation method.
8	Design and implement C/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.
9	Design and implement C/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.
10	Design and implement C/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.
11	Design and implement C/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.
12	Design and implement C/C++ Program for N Queen's problem using Backtracking.

Course outcomes (Course Skill Set):

At the end of the course the student will be able to:

1. Develop programs to solve computational problems using suitable algorithm design strategy.
2. Compare algorithm design strategies by developing equivalent programs and observing running times for analysis (Empirical).
3. Make use of suitable integrated development tools to develop programs
4. Choose appropriate algorithm design techniques to develop solution to the computational and complex problems.
5. Demonstrate and present the development of program, its execution and running time(s) and record the results/inferences.

Assessment Details (both CIE and SEE)

The weightage of Continuous Internal Evaluation (CIE) is 50% and for Semester End Exam (SEE) is 50%. The minimum passing mark for the CIE is 40% of the maximum marks (20 marks out of 50) and for the SEE minimum passing mark is 35% of the maximum marks (18 out of 50 marks). A student shall be deemed to have satisfied the academic requirements and earned the credits allotted to each subject/ course if the student secures a minimum of 40% (40 marks out of 100) in the sum total of the CIE (Continuous Internal Evaluation) and SEE (Semester End Examination) taken together

Continuous Internal Evaluation (CIE):

CIE marks for the practical course are **50 Marks**.

The split-up of CIE marks for record/ journal and test are in the ratio **60:40**.

- Each experiment is to be evaluated for conduction with an observation sheet and record write-up. Rubrics for the evaluation of the journal/write-up for hardware/software experiments are designed by the faculty who is handling the laboratory session and are made known to students at the beginning of the practical session.
- Record should contain all the specified experiments in the syllabus and each experiment write-up will be evaluated for 10 marks.
- Total marks scored by the students are scaled down to **30 marks** (60% of maximum marks).
- Weightage to be given for neatness and submission of record/write-up on time.
- Department shall conduct a test of 100 marks after the completion of all the experiments listed in the syllabus.
- In a test, test write-up, conduction of experiment, acceptable result, and procedural knowledge will carry a weightage of 60% and the rest 40% for viva-voce.
- The suitable rubrics can be designed to evaluate each student's performance and learning ability.
- The marks scored shall be scaled down to **20 marks** (40% of the maximum marks).

The Sum of scaled-down marks scored in the report write-up/journal and marks of a test is the total CIE marks scored by the student.

Semester End Evaluation (SEE):

- SEE marks for the practical course are 50 Marks.

- SEE shall be conducted jointly by the two examiners of the same institute, examiners are appointed by the Head of the Institute.
- The examination schedule and names of examiners are informed to the university before the conduction of the examination. These practical examinations are to be conducted between the schedule mentioned in the academic calendar of the University.
- All laboratory experiments are to be included for practical examination.
- (Rubrics) Breakup of marks and the instructions printed on the cover page of the answer script to be strictly adhered to by the examiners. **OR** based on the course requirement evaluation rubrics shall be decided jointly by examiners.
- Students can pick one question (experiment) from the questions lot prepared by the examiners jointly.
- Evaluation of test write-up/ conduction procedure and result/viva will be conducted jointly by examiners.

General rubrics suggested for SEE are mentioned here, writeup-20%, Conduction procedure and result in -60%, Viva-voce 20% of maximum marks. SEE for practical shall be evaluated for 100 marks and scored marks shall be scaled down to 50 marks (however, based on course type, rubrics shall be decided by the examiners)

Change of experiment is allowed only once and 15% of Marks allotted to the procedure part are to be made zero.

The minimum duration of SEE is 02 hours

Suggested Learning Resources:

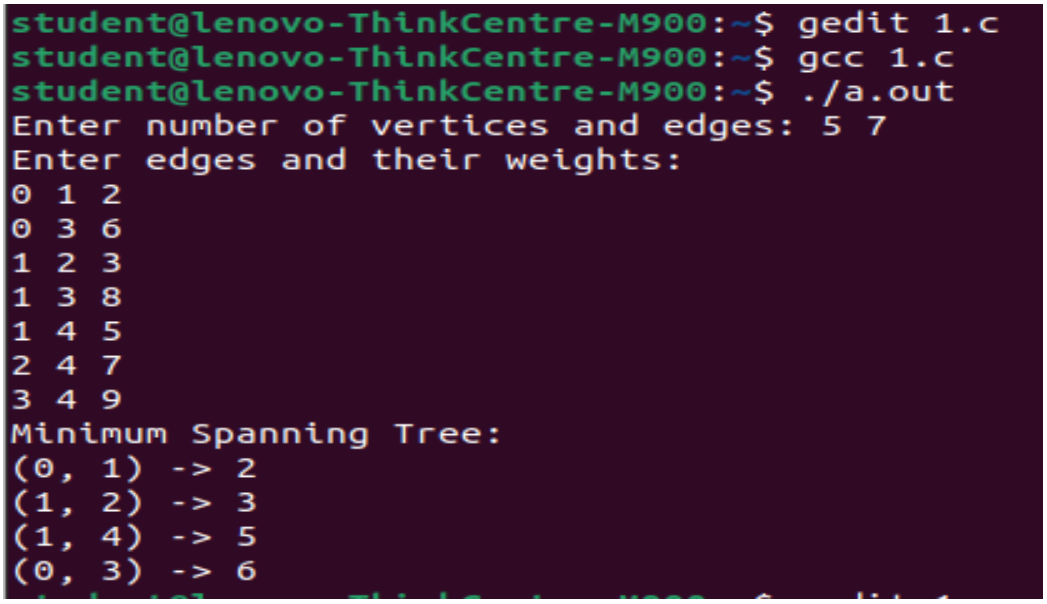
- Virtual Labs (CSE): http://cse01-iiith.vlabs.ac.in/

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

```
#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:


student@lenovo-ThinkCentre-M900:~$ gedit 1.c
student@lenovo-ThinkCentre-M900:~$ gcc 1.c
student@lenovo-ThinkCentre-M900:~$ ./a.out
Enter number of vertices and edges: 5 7
Enter edges and their weights:
0 1 2
0 3 6
1 2 3
1 3 8
1 4 5
2 4 7
3 4 9
Minimum Spanning Tree:
(0, 1) -> 2
(1, 2) -> 3
(1, 4) -> 5
(0, 3) -> 6

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

```
#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:


student@lenovo-ThinkCentre-M900:~$ gedit 2.c
student@lenovo-ThinkCentre-M900:~$ gcc 2.c
student@lenovo-ThinkCentre-M900:~$ ./a.out
Enter the number of vertices and edges: 5
7
Enter the source vertex, destination vertex, and weight for each edge:
0 1 2
0 3 6
1 2 3
1 3 8
1 4 5
2 4 7
3 4 9
Edge Weight
0 - 1 2
1 - 2 3
0 - 3 6
1 - 4 5

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

```
#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.

```
#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

```
#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:

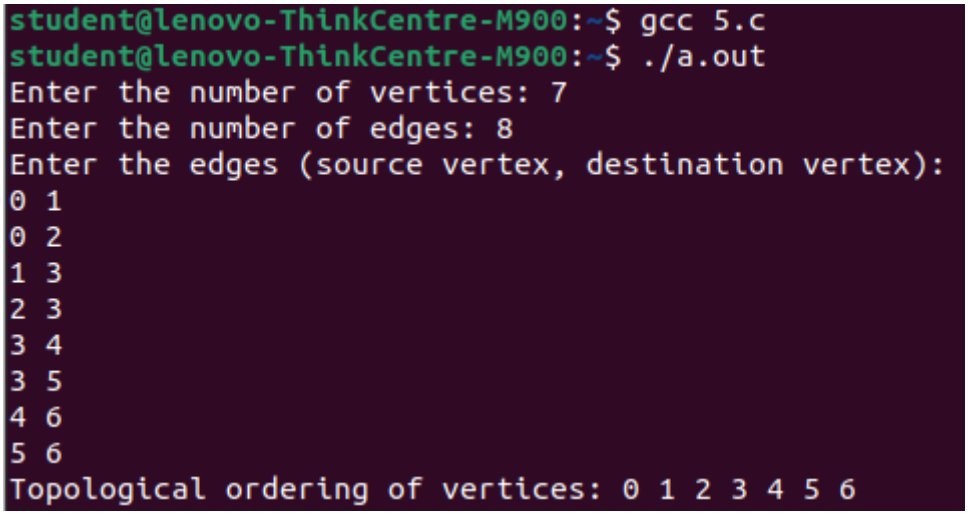
```
student@lenovo-ThinkCentre-M900:~$ gedit 4.c
student@lenovo-ThinkCentre-M900:~$ gcc 4.c
student@lenovo-ThinkCentre-M900:~$ ./a.out
Enter the number of vertices: 5
Enter the number of edges: 7
Enter the source vertex, destination vertex, and weight for each edge:
0 1 2
0 3 6
1 2 3
1 3 8
1 4 5
2 4 7
3 4 9
Vertex          Distance from Source
0               0
1               2
2               5
3               6
4               7
```

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

```
#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;
}
```

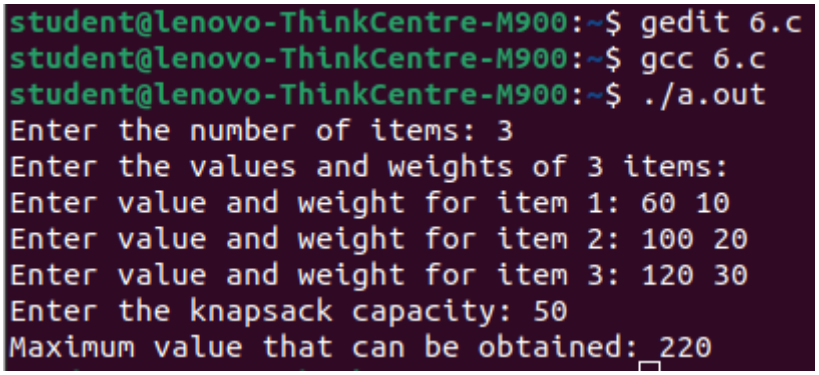
OUTPUT:



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

```
#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:

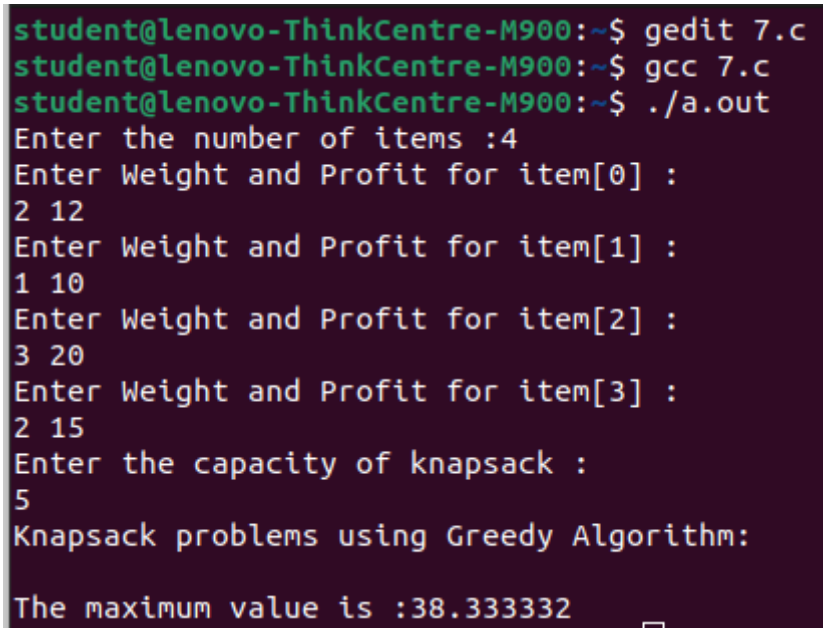

student@lenovo-ThinkCentre-M900:~$ gedit 6.c
student@lenovo-ThinkCentre-M900:~$ gcc 6.c
student@lenovo-ThinkCentre-M900:~$ ./a.out
Enter the number of items: 3
Enter the values and weights of 3 items:
Enter value and weight for item 1: 60 10
Enter value and weight for item 2: 100 20
Enter value and weight for item 3: 120 30
Enter the knapsack capacity: 50
Maximum value that can be obtained: 220

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

```
#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;
}
```

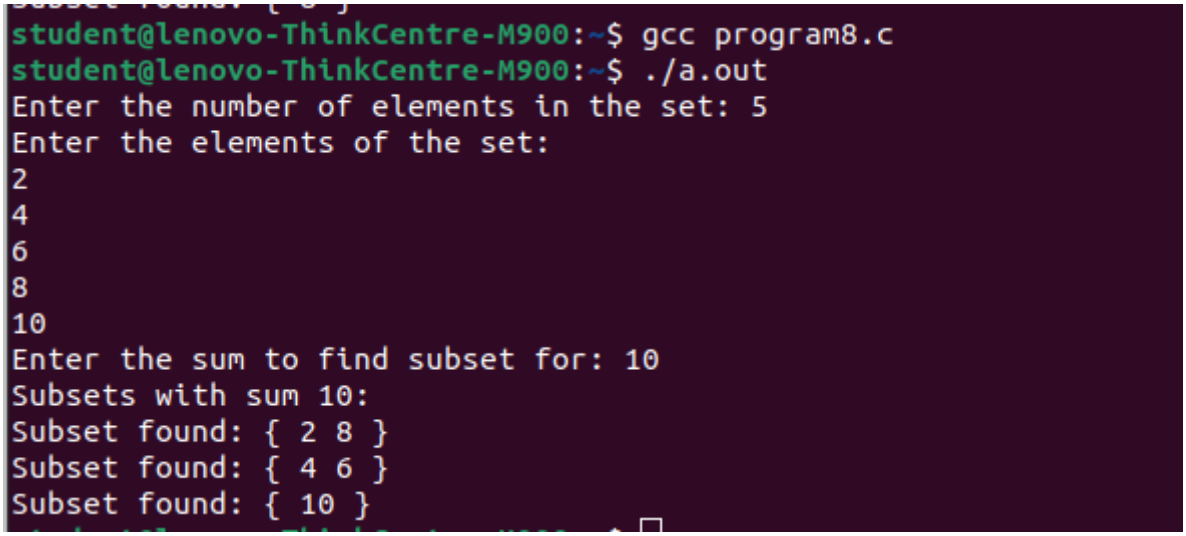
OUTPUT:



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.

```
#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:

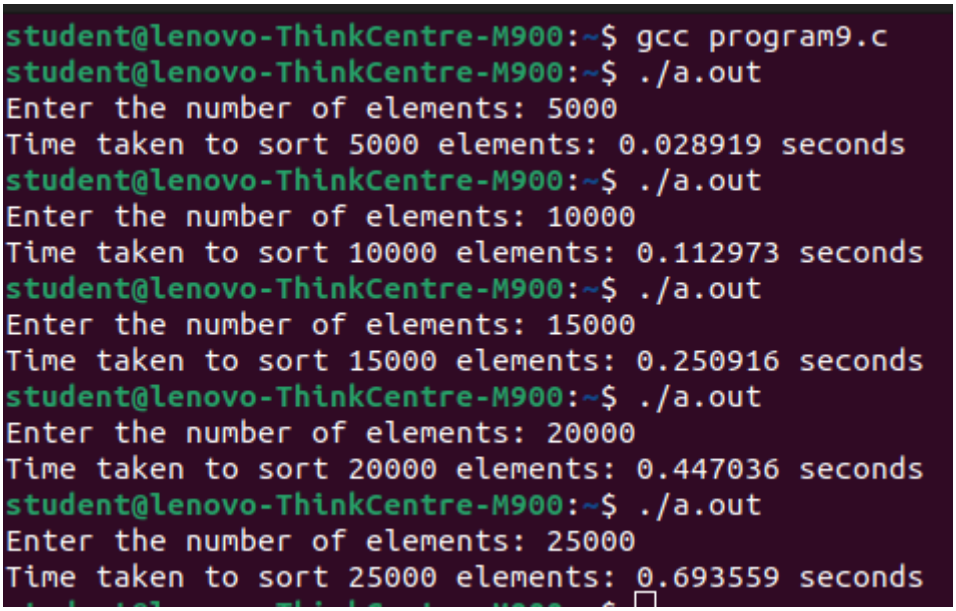

student@lenovo-ThinkCentre-M900:~$ gcc program8.c
student@lenovo-ThinkCentre-M900:~$ ./a.out
Enter the number of elements in the set: 5
Enter the elements of the set:
2
4
6
8
10
Enter the sum to find subset for: 10
Subsets with sum 10:
Subset found: { 2 8 }
Subset found: { 4 6 }
Subset found: { 10 }

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.

```
#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:

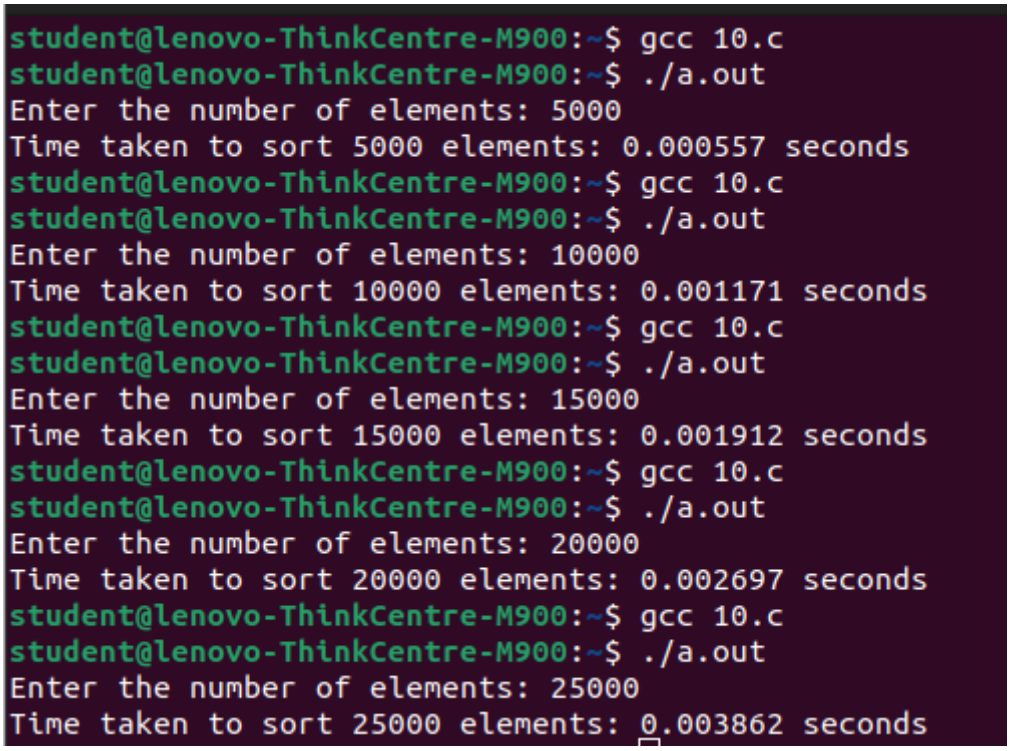


10.Design and implement C Program to sort a given set of n integer elements using Quick Sort method and compute its time complexity. Run the program for varied values of n> 5000 and record the time taken to sort. Plot a graph of the time taken versus n. The elements can be read from a file or can be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

```
#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:

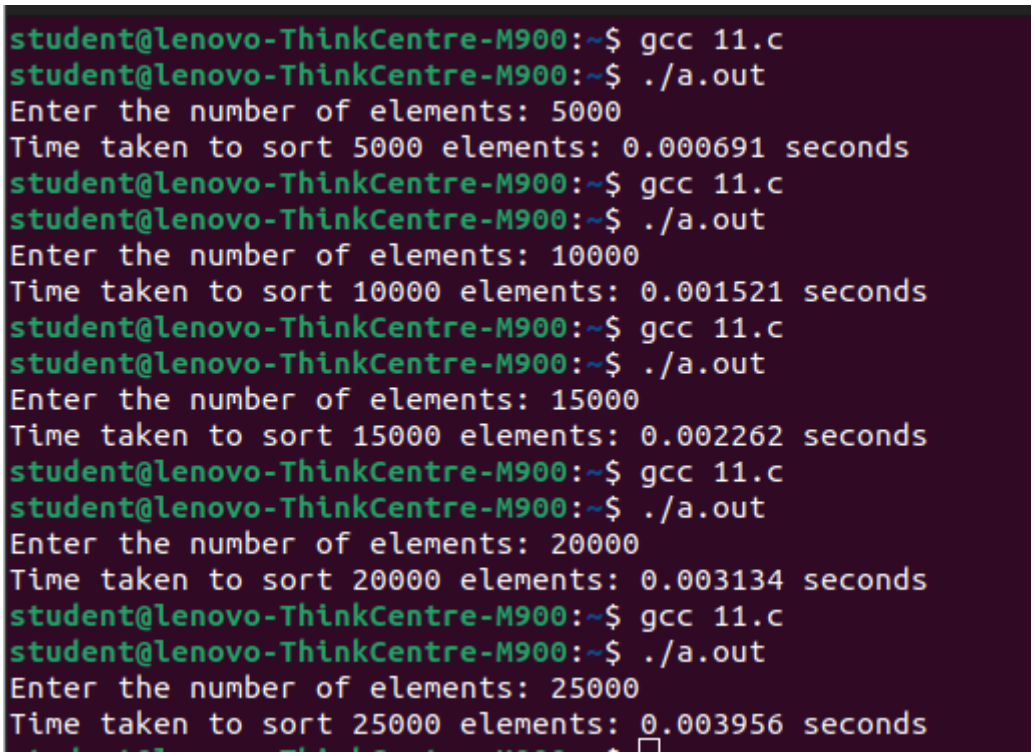


11.Design and implement C Program to sort a given set of n integer elements using Merge Sort method and compute its time complexity. Run the program for varied values of n> 5000, and record the time taken to sort. Plot a graph of the time taken versus n. The elements can be read from a file or can be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

```
#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:


student@lenovo-ThinkCentre-M900:~$ gcc 11.c
student@lenovo-ThinkCentre-M900:~$ ./a.out
Enter the number of elements: 5000
Time taken to sort 5000 elements: 0.000691 seconds
student@lenovo-ThinkCentre-M900:~$ gcc 11.c
student@lenovo-ThinkCentre-M900:~$ ./a.out
Enter the number of elements: 10000
Time taken to sort 10000 elements: 0.001521 seconds
student@lenovo-ThinkCentre-M900:~$ gcc 11.c
student@lenovo-ThinkCentre-M900:~$ ./a.out
Enter the number of elements: 15000
Time taken to sort 15000 elements: 0.002262 seconds
student@lenovo-ThinkCentre-M900:~$ gcc 11.c
student@lenovo-ThinkCentre-M900:~$ ./a.out
Enter the number of elements: 20000
Time taken to sort 20000 elements: 0.003134 seconds
student@lenovo-ThinkCentre-M900:~$ gcc 11.c
student@lenovo-ThinkCentre-M900:~$ ./a.out
Enter the number of elements: 25000
Time taken to sort 25000 elements: 0.003956 seconds

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

```
#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:

```
student@lenovo-ThinkCentre-M900:~$ gedit 12.c
student@lenovo-ThinkCentre-M900:~$ gcc 12.c
student@lenovo-ThinkCentre-M900:~$ ./a.out
 - N Queens Problem Using Backtracking -

Enter number of Queens:4


Solution 1:

        1       2       3       4

1       -       Q       -       -

2       -       -       -       Q

3       Q       -       -       -

4       -       -       Q       -

Solution 2:

        1       2       3       4

1       -       -       Q       -

2       Q       -       -       -

3       -       -       -       Q

4       -       Q       -       -student@lenov
```

Appendix-A

This section lists the Viva questions for each lab session:

SELECTION SORT

1. As the input size increases, the performance of selection sort decreases.(T/F)
2. What is the best case, worst case, average case time complexity for quick sort algorithm?
3. Explain working of selection sort

QUICK SORT

1. Which sorting technique is also called partition exchange sort?
2. Which is the technique used by quick sort?
3. What is the best case, worst case, average case time complexity for quick sort algorithm?
4. Explain divide and conquer technique.
5. Define in place sorting algorithm.
6. List different ways of selecting pivot element.
7. Is quick sort stable?

MERGE SORT

1. What is the best case, worst case, average case time complexity for merge sort algorithm?
2. What is the output of merge sort after the 1st pass given the following sequence of numbers?
   25 57 48 37 12 92 86 33
3. What is the running time of merge sort?
4. What technique is used to sort elements in merge sort?
5. Is merge sort in place sorting algorithm?
6. Define stable sort algorithm. Is merge sort stable?

WARSHALL'S ALGORITHM

1. Define transitive closure.
2. Define topological sequence.
3. What is the time complexity of Warshall's algorithm?

KNAPSACK PROBLEM

1. Define knapsack problem.
2. Define principle of optimality.
3. What is the optimal solution for knapsack problem?
4. What is the time complexity of knapsack problem?
5. Provide a use case where greedy knapsack cannot yield optimal solution.

SHORTEST PATHS ALGORITHM

1. What is the time complexity of Dijkstra's algorithm?
2. Define cost matrix.
3. Define directed graph.
4. Define connected graph.

MINIMUM COST SPANNING TREE

1. What is the time complexity of Kruskal's algorithm?
2. Define spanning tree.
3. Define minimum cost spanning tree.

GRAPH TRAVERSALS

1. Define graph, connected graph.
2. List the different graph traversals.
3. Explain DFS traversal.
4. Explain BFS traversal.
5. What are the time complexities of BFS and DFS algorithms?
6. What are the data structures used to implement DFS and BFS traversals?

SUM OF SUBSETS PROBLEM

1. Define is Back-Tracking.
2. Explain Sum of subset problem.
3. What is time complexity of sum of subset problem?
4. Let S= {1,2,3,4,5} and d= 6. Obtain the subsets whose sum leads to d

TRAVELLING SALESPERSON PROBLEM

1. Define Optimal Solution.
2. Explain Travelling Salesman's Problem.
3. What is the time complexity of Travelling Salesman's Problem?
4. For n- cities_____solutions are possible.
5. Deterministic algorithms exist for TSP(True/False)
6. Is TSP problem is NP-Hard or NP-Complete? Explain

MINIMUM COST SPANNING TREE

1. What is Minimum Cost spanning Tree.
2. Explain Prim's algorithm.
3. What is the time complexity of Prim's algorithm?
4. Explain the concept of Kruskal's algorithm
5. Derive the time complexity for Kruskal's algorithm.

SINGLE SOURCE SHORTEST PATH

1. Define Single Source Shortest Path problem.
2. Explain positive and negative weight cycle along with some examples
3. Estimate the time complexity of the Dijkstra's algorithm.

ALL PAIRS SHORTEST PATH

1. Define All Pair Shortest Path Floyd's algorithm(APSP)
2. What is the time complexity of Floyd's algorithm?
3. Define Distance Matrix.

BACKTRACKING

1. Define n- Queens's problem?
2. 3- Queen problem has a solution (T/ F)
3. DFS can be solved using backtracking (T/F).
4. What is promising and non-promising node?
5. Define Hamiltonian cycle.