ATME College of Engineering

13thKM Stone, Bannur Road, Mysore - 560 028

Department of CSE–Artificial Intelligence & Machine Learning

(Academic Year 2025-26)

LABORATORY MANUAL

SUBJECT: ANALYSIS & DESIGN OF ALGORITHMS

SUB CODE: BCSL404

SEMESTER: IV

SCHEME: 2022

Prepared By	Verified By	Approved by
Ms. GEETHA.B Instructor	Mrs. VANITHA G NAIK Faculties Co_Ordinators Dept. of CSE-AI &ML	Dr. M S SUNITHA PATEL Assoc. Professor & Head Dept. of CS-AI &ML

Institute Vision

- Development of academically excellent, culturally vibrant, socially responsible and globally competent human resources.

Institute Mission

- To keep pace with advancements in knowledge and make the students competitive and capable at the global level.
- To create an environment for the students to acquire the right physical, intellectual, emotional and moral foundations and shine as torch bearers of tomorrow's society.
- To strive to attain ever-higher benchmarks of educational excellence.

Department Vision

To impart technical education in the field of Artificial intelligence and machine learning of topnotch quality with a high level of professional competence, social obligation, and global cognizance among the students.

Department Mission

- To impart technical education that is up to date, relevant and makes students to compete at global level
- Fostering an ambiance where students can adopt the suitable moral, intellectual, emotional, and physical attributes to shine as the leaders of tomorrow's society.
- To strive to meet ever higher educational standard.

Program Outcomes (PO's)

1. Engineering knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

2. Problem analysis: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

3. Design/development of solutions: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

4. Conduct investigations of complex problems: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

5. Modern tool usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.

6. The engineer and society: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

7. Environment and sustainability: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

8. Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

9. Individual and team work: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

10. Communication: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and

design documentation, make effective presentations, and give and receive clear instructions.

11. Project management and finance: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

12. Life-long learning: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change

Program Educational Objectives (PEO's):

PEO1: Graduates will be able to hone their problem-solving abilities and capacity to offer solutions to challenges that arise in the actual world.

PEO2: Able to design and develop AI based solutions to real-world problems in a business, research, or social environment.

PEO3: Graduates shall acquire and inculcate corporate culture, core attributes, and leadership qualities as well as professional etiquette's and lifelong learning.

Program Specific Outcomes (PSO's)

PSO1: Ability to design and develop artificial intelligent based solutions by applying optimal algorithms to solve real world issues.

PSO2: Ability to apply suitable AI tools and techniques to offer solutions in the various domains of engineering.

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
Examinationtype(SEE)	Practical		

Course objectives:

- To design and implement various algorithms in C/C++programming using suitabledevelopment 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 tofind Minimum Cost Spanning Tree of agiven connected undirected graph using Prim's algorithm.
3	a. Design and implement C/C++ Program tosolve 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 toother 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 positiveintegers whosesum 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 thecourse the student will be able to:

1. Develop programs tosolve computational problems using suitable algorithm design strategy.
2. Comparealgorithm design strategies by developing equivalent programs and observing running times for analysis (Empirical).
3. Makeuse of suitable integrated development tools todevelop 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 whois handling the laboratory session and aremade known tostudents 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.
- Totalmarks scored by the students are scaled down to **30 marks** (60% of maximum marks).
- Weightageto be given for neatness and submission of record/write-up on time.
- Department shall conduct attest of 100 marks after the completion of all the experiments listed inthe syllabus.
- In atest, test write-up, conduction of experiment, acceptable result, and procedural knowledge will carry a weightage of 60% and the rest 40% for viva-voce.
- Thesuitable rubrics can be designed to evaluate each student's performance and learning ability.
- Themarks 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):

- SEEmarks 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 theacademic 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.

Theminimum duration of SEE is 02 hours

CONTENTS

Sl. No.	Particulars	Page No
	Introduction	**1**
1	Find Minimum Cost Spanning Tree of a given connect Undirected graph using Kruskal's algorithm.	**7**
2	Find Minimum Cost Spanning Tree of a given connected un directedgraph using **Prim's algorithm.**	**12**
3A	C/C++ Program to solve All-Pairs Shortest Paths problem using Floyd's.	**15**
3B	C/C++ Program to fin the transitive closure using Warshal's.	**18**
4	From a given vertex in a weighted connected graph, find shortest paths to other vertices using Dijkstra's algorithm. Write the program in Java.	**20**
5	Find Topological ordering using Digraph	**24**
6	C/C++ Program to solve 0/1 Knapsack problem using DynamicProgramming method.	**28**
7	C/C++ Program to solve discrete Knapsack and continuous Knapsackproblems using greedy approximation method.	**31**
8	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.	**35**
9	Design and implement C/C++ Program to sort a given set of n integer elements using selection sort	**39**
10	Design and implement C/C++ Program to sort a given set of n integer elements using quick sort	**43**
11	Design and implement C/C++ Program to sort a given set of n integer elements using merge sort	**49**
12	Design and implement C/C++ Program for N Queen's problem usingBacktracking.	**54**

@# 16032024

CHAPTER 1

INTRODUCTION

Need for studying algorithms

Theoretical importance

- The study of algorithms is the concepts of computer science.
- It is a standard set of important algorithms, they further our analytical skills& help us in developing new algorithms for required applications

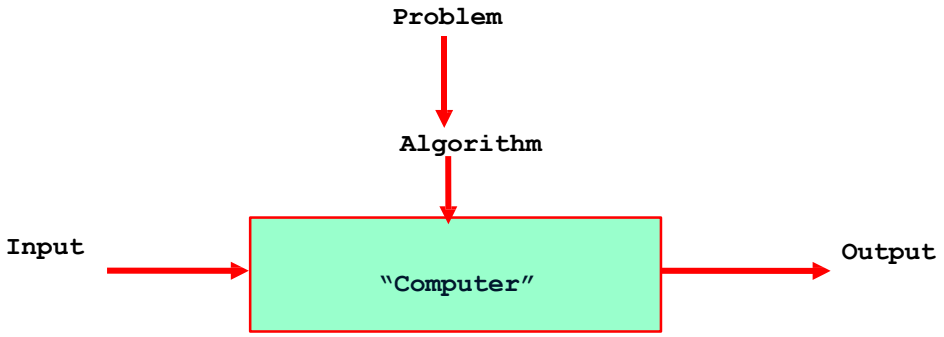
Algorithm

An algorithm is a sequence of unambiguous instructions for solving a problem, i.e., for obtaining a required output for anylegitimate input in a finite amount of time.

In addition, all algorithms must satisfy the following criteria:

1. **Input**: Each algorithm should have zero or more inputs. The range ofinput for which algorithms works should be specified carefully.
2. **Output**: The algorithm should produce correct results. At least one quantity has to be produced.
3. **Definiteness**: Each instruction should be clear and unambiguous.
4. **Effectiveness: Every** instruction should be simple and should transform the given Input to the desired output. so that it can be carried out, in
5. **Finiteness**: If we trace out the instruction of an algorithm, then for all cases, the algorithm must terminate after a finite numb2er of steps.

Notion of algorithm



Fundamentals of Algorithmic problem solving

- Understanding the problem
- Ascertain the capabilities of the computational device
- Exact /approximate solution.
- Decide on the appropriate data structure
- Algorithm design techniques
- Methods of specifying an algorithm
- Proving an algorithms correctness
- Analyzing an algorithm

Important Problem Types of different categories

- **Sorting**

It refers to the problem of re-arranging the items of a given list in ascending or descending order. The various algorithms that can be used for sorting are bubble sort, selection sort, insertion sort, quick sort, merge sort, heap sort etc.

- **Searching**
  - This problem deals with finding a value, called a search key, ina given set.
  - The various algorithms that can be used for searching are binary search, linear search, hashing, interpolation search etc.
- **String processing**
  - This problem deals with manipulation of characters or strings, string matching, search and replace, deleting a string in a text etc.
  - String processing algorithm such as string matching algorithms is used in the design of assemblers and compliers.
- **Graph problems**
  - The graph is a collection of vertices and edges.
  - Ex: graph traversal problems, topological sorting etc

- **Combinatorial problems**

  Theseproblems are used to find combinatorial object such as permutation and combinations.

  Ex: travelling salesman problem, graph coloring problem etc.

- **Geometric problems**

  This problem deal with geometric objects such as points, curves, lines, polygonetc.: closest-pair problem, convex hull problem

- **Numerical problems**

  These problems involving mathematical manipulations solving equations, computing differentiations and integrations, evaluating various types of functions etc.

  Ex: Gauss elimination method, Newton-Rap son method

Fundamentals of data Structures

Since most of the algorithms operate on the data, particular ways of arranging the data play a critical role in the design & analysis of algorithms.

A data structure can be defined as a particular way of arrangement of data.

The commonly used data structures are:

1. Linear data structures
2. Graphs
3. Trees.
4. Sets and dictionaries

1. Linear data structures

The most common linear data structures are arrays and linked lists.

Arrays: Is a collection of homogeneous items. An item's place within the collection is called an index.

Linked list: finite sequence of data itemsi.e.it is a collection of data items in a certain order.

2. Graphs

A data structure that consists ofa set of nodes and a set of edges that relate the nodes to each other is called a graph.

- **Undirected graph**: Graph in which the edges have no direction
- **Directed graph (Digraph)**: A graph in which each edge is directed from one vertex to another (or the same) vertex.

3. Tree

A tree is a connected acyclic graph that has no cycle.

4. Sets and dictionaries

* A set is defined as an unordered collection of distinct items called an element of the set
* Dictionary is a data structure that implements searching, adding of objects

Analysis of algorithms

Analysis of algorithms means to investigate an algorithm's efficiency with respect to resources:

- **Running time (time efficiency)**
  It indicates how fast an algorithm in question runs
- **Memory space (space efficiency)**
  It deals with the extra space the algorithm requires

Theoretical analysis of time efficiency

Algorithm efficiency depends on the **input size n.** And for some algorithms efficiency depends on **type of input**.

We have best, worst & average case efficiencies

Worst-case efficiency:

Efficiency (number of times the basic operation will be executed) **for the worst case input of size n.** *i.e.* The algorithm runs the longest among all possible inputs of size n.

Best-case efficiency:

Efficiency (number of times the basic operation will be executed **for the best case input of size n.** ***i.e.*** The algorithm runs the fastest among all possible inputs of size n.

Average-case efficiency:

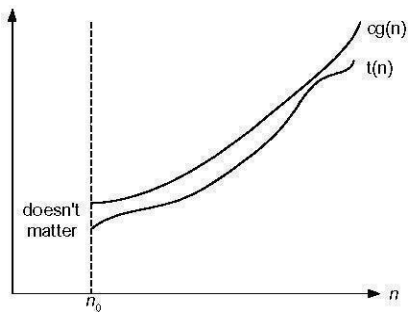
Average time taken (number oftimes the basic operation willbe executed) **to solve all the possible instances (random) of the input.**

Asymptotic Notations

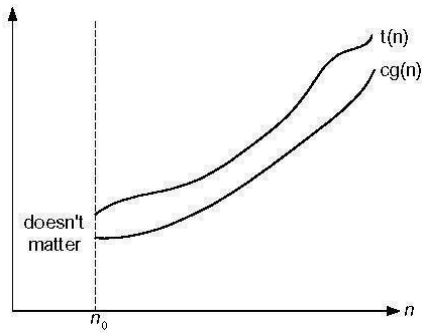
- Asymptotic notation is a way of comparing functions that ignores constant factors and small input sizes.
- Three notations used to compare orders of growth of an algorithm's basic operation are: **O, Ω, θ notations.**

Big-oh notation:

Afunction t(n) is said tobe inO(g(n)), denoted t(n)€O(g(n)), if t(n) is bounded above by someconstant multiple of g(n) for all large ni.e., if there exist some positive constant c and some nonnegative integer n$_0$ such that **t (n)<=cg(n) for all n>= n**0



Big-oh notation t(n)€O(g(n))



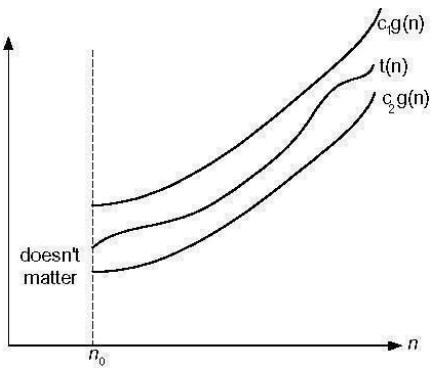
Big-omega notation t(n)€Ω, (g(n))

Big-omega notation:

A function t(n) is said to be in Ω(g(n)), denoted t(n)€ Ω(g(n)), if t(n) is bounded belowby some constant multiple of g(n) for all large ni.e., if there exist some positive constant c and Some nonnegative integer n$_0$ 0 such that t(n)>=cg(n) for all n>=n$_0$Big-theta notation:

A function t(n) is said to be in θ(g(n)), denoted t(n)€ θ(g(n)), if t(n) is bounded both above and below by some positive constant multiple of g(n) for all large n i.e., if there exist some positive constant c

c2g (n) <=t (n) <=c1g (n) for all n>= n_0



Big-theta notationt(n)€θ(g(n))

<u>Basic Asymptotic Efficiency classes</u>:

1	**constant**
log *n*	**logarithmic**
N	**linear**
n* log *n	***n* log *n***
n^2	**quadratic**
n^3	**cubic**
2^n	**exponential**
n!	**factorial**

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

Objectives: This Program enable students to:

- Learn Kruskal's algorithm to find shortest paths.
- Find Minimum Cost Spanning Tree of a givenundirected graphusing **Kruskal's algorithm**

Kruskal's algorithm looks at a minimum spanning three for a weighted connected graph G=(V,E) as an acyclic sub graph with V-1 edges for which the sum of the edge weights is the smallest.

The algorithm begins by sorting the graph's edges in increasing order of their weights. Then starting with the empty sub graph, it scans this sorted list adding the next edge on the list to the current sub graph if such an inclusion does not create a cycle and simply skipping the edge otherwise.

When you select a minimum weight path, it will have source and destination vertices. These two vertices should not have same ancestor or parent. If both the vertices have the same origin, t h e n it is a cyclic graph.

Aim:

Kruskal's Algorithm for computing the minimum spanning tree is directly based on the generic MST algorithm. It builds the MST in forest. Prim's algorithm is based ona generic MSTalgorithm. It uses greedy approach.

Algorithm:

Kruskal's Algorithm

Start withanempty set A, and select at everystage the shortest edge that has not been chosen or rejected, regardless ofwhere this edge is situated in graph.

- Initially, each vertex is in its own tree in forest.
- Then, algorithm considers each edge inturn, order by increasing weight.
- If an edge(u,v) connects two different trees, then(u,v) is added to the set of edgesof the MST,and two trees connected by an edge(u,v)are merged into a single tree.
- On the other hand, if an edge (u, v) connects two vertices in the same tree, then
- edge (u, v) is discarded.

```
#include<stdio.h>

int ne=1, min_cost=0;

void main ( )
{
int n,i,j, min,a,u,b,v,cost[20][20],parent[20];
printf("Enter the no. of vertices:");
scanf("%d",&n);
printf("\nEnter the cost matrix:\n");
for(i=1;i<=n;i++)
for(j=1;j<=n;j++)
scanf("%d",&cost[i][j]);
for(i=1;i<=n;i++)
parent[i]=0;
printf("\nThe edges of spanning tree are\n");
  while(ne<n)
{
min=999;
for(i=1;i<=n;i++)
{
for(j=1;j<=n;j++)
{
if(cost[i][j]<min)
{
min=cost[i][j];
  a=u=i;
b=v=j;
}}}
while(parent[u])
u=parent[u];
while(parent[v])
v=parent[v];
if(u!=v)
```

```
    {
     printf("Edge %d\t(%d->%d)=%d\n",ne++,a,b,min);
     min_cost=min_cost+min;
     parent[v]=u;
}
    cost[a][b]=cost[a][b]=999;
}
    printf("\nMinimum cost=%d\n",min_cost);
    }
```

Output:

```
Enter the no. of vertices:
6

Enter the cost matrix:
999    3    999    999    6    5
 3     999  1      999    999  4
999    1    999    6      999  4
999    999  6      999    8    5
 6     999  999    8      999  2
 5     4    4      5      2    999


The edges of spanning tree are

Edge 1    (2->3)=1
Edge 2    (5->6)=2
Edge 3    (1->2)=3
Edge 4    (2->6)=4
Edge 5    (4->6)=5

Minimum cost=15
```

Outcomes: On completion of this Program, the students are able to :

- Solve Kruskal's algorithm to find shortest paths.
- Solve Minimum Cost Spanning Tree of a given undirected graph using kruskal algorithm

Viva questions

1 Whichare the data structures is used to represent the graph.
2 What is tree?
3 What is spanning tree?
4 What is MST (Minimum-cost Spanning Tree)?
5 Difference betweenGreedy algorithm and Dynamic algorithm method.

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

Objectives: This Program enable students to:

- Learn **Prim's** algorithm to find shortest paths.
- Find Minimum Cost Spanning Tree ofa given undirected graph using **Prim's algorithm**

Choose a node and build a tree from there selecting at every stage the shortest available edge that can extend the tree to an additional node.

- Prim'salgorithmhastheprepertythattheedgesinthesetAalwaysformasingletree.
- We begin with some vertex in a given graph G= (V, E), defining theinitial set of vertices A.
- In each iteration, we choose a minimum-weight edge(u,v),connecting a vertexv in the set A to the vertex u outside of set A.
- Thevertex$u$isbroughtintoA.Thisprocessisrepeateduntilaspanningtreeisformed.
- Like Kruskal's algorithm, here too, the import ant fact about MST sis weal ways choose the smallest-weight edge joining a vertex inside set A to the one outside the set A.
- The implication of this fact is that It adds on ly edges that are safe for A;
- Therefore, when the algorithm terminates, the edges inset A form a MST

```
#include<stdio.h>
int a,b,u,v,n,i,j,ne=1;
int visited [10]={0}, min,mincost=0,cost[10][10];
void main ( )
{
printf("\n Enter the number of nodes:");
scanf("%d",&n);
printf("\n Enter the adjacency matrix:\n");
for(i=1;i<=n;i++)
for(j=1;j<=n;j++)
{
scanf("%d",&cost[i][j]);
if(cost[i][j]==0)
cost[i][j]=999;
}
visited[1]=1;
printf("\n");
while(ne<n)
{
   for(i=1,min=999;i<=n;i++
) for(j=1;j<=n;j++)
if(cost[i][j]<min)
if(visited[i]!=0)
{
min=cost[i][j];
  a=u=i;
b=v=j;
}
if(visited[u]==0 || visited[v]==0)
{
printf("\n Edge %d:(%d %d) cost:%d",ne++,a,b,min);
mincost+=min;
visited[b]=1;
}
cost[a][b]=cost[b][a]=999;
}
```

```
printf("\n Minimun cost=%d",mincost);
}
```

Output

```
Enter the number of nodes:4

Enter the adjacency matrix:
999   1    5    2
1     999  999  999
5     999  999  3
2     999  3    999


Edge 1:(1 2) cost:1
Edge 2:(1 4) cost:2
Edge 3:(4 3) cost:3
Minimun cost=6
```

Outcomes: On completion of this Program, the students are able to :

- Solve Prim's algorithm to find shortest paths.
- Solve Minimum Cost Spanning Tree of a given undirected graph using Prim's algorithm

Viva questions

- What is the difference between Prims and Kruskals algorithm?
- Canwe use kruskals to find maximum cost spanning tree?
- What is minimum cost spanning tree.

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

Objectives: This Program enable students to :

- Learn Floyd's algorithm to find All-Pairs Shortest paths.
- FindMinimumCostSpanningTreeofagivenundirectedgraphusingFloyd's**algorithm**

Floyds algorithm is a classic example for the application of dynamic programming. It is used to compute the shortest distance between every two pair of nodes in a given graph and hence it is also called "**all-pair shortest path**"algorithm

The following formula can be used to find each element in the distance matrix.

$$d_{ij}^{(k)} \leftarrow min\{d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\} \quad \text{where } k>=1, d_{ij}^{(0)} \leftarrow w_{ij}$$

ALGORITHM

Algorithm Floyd (W[1..n, 1..n])

//Implements Floyd'salgorithm

// Input: Weight matrix W

// Output: Distance matrix of shortest paths'

```
length D(0)←w
    fork←1 to ndo
      fori←1 to ndo
            forj←1 to ndo
  D(k)←min{d(k-1)[i,j], d(k-1)[i,k]+d(k-1)[k,j]}
     ReturnD(n)
```

```
#include <stdio.h>
#include <limits.h>

#define V 4

void Floyd Warshall(int graph[V][V])
{
 int dist[V][V];

 for (int i = 0; i < V; i++)
 for (int j = 0; j < V; j++)
        dist[i][j] = graph[i][j];

for (int k = 0; k < V; k++)
 for (int i = 0; i < V; i++)
   for (int j = 0; j < V; j++)
   if (dist[i][k] != INT_MAX && dist[k][j] != INT_MAX && dist[i][k] + dist[k][j] <
  dist[i][j])dist[i][j] = dist[i][k] + dist[k][j]
  ("Shortest distances between every pair of vertices:\n");
  for (int i = 0; i < V; i++) {
 for (int j = 0; j < V; j++) {
    if (dist[i][j] == INT_MAX)
        printf("INF\t");
    else
        printf("%d\t", dist[i][j]);
 }
 printf("\n");
}
}

int main() {
  int graph[V][V] = {{0, INT_MAX, 3, INT_MAX},
                {2, 0, INT_MAX, INT_MAX},
                {INT_MAX, 7, 0, 1},{6, INT_MAX, INT_MAX, 0}};
```

```
    floydWarshall(graph);

    return 0;
}
```

Output:

```
Shortest distances between every pair of vertices:
0    10    3    4
2    0     5    6
7    7     0    1
6    16    9    0
```

Outcomes: On Completion of these Program enable students to:

- Solve Floyd's algorithm to find All-Pairs Shortest paths.
- Solve &Find Minimum Cost Spanning Tree of a given undirected graph using Floyd's **algorithm**

Viva questions

- Floyd's algorithm is an example of which type of algorithm.
- Difference between Floyds and Dijkstra'salgorithm.
- What is the time complexity of Floyd'salgorithm?
- What is the best, worst and average case of Floyd's algorithm?

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

Objectives: This Program enable students to:

- Learn Warshallalgorithm to find transitive closure.
- Find path matrix graph using Warshal's **algorithm**

```
# include <stdio.h>
int n,a[10][10],p[10][10];
void path()
{
int i,j,k;
for(i=0;i<n;i++)
for(j=0;j<n;j++)
p[i][j]=a[i][j];
for(k=0;k<n;k++)
for(i=0;i<n;i++)
for(j=0;j<n;j++)
if(p[i][k]==1&&p[k][j]==1)
p[i][j]=1;
}
void main ( )
{
int i,j;
printf("Enter the number of nodes:");
scanf("%d",&n);
printf("\nEnter the adjacency matrix:\n");
for(i=0;i<n;i++)
for(j=0;j<n;j++)
scanf("%d",&a[i][j]);
path();
printf("\nThe path matrix is shown below\n");
for(i=0;i<n;i++)
{
```

```
for(j=0;j<n;j++)
printf("%d ",p[i][j]);
printf("\n");
}
}
```

Output:

Enter the number of nodes:4

Enter the adjacency matrix:
0 1 0 0
0 0 1 0
0 0 0 1
1 0 0 0

The path matrix is shown below
1 1 1 1
1 1 1 1
1 1 1 1
1 1 1 1

Outcomes: After completion of Programenable students to :

- Solve Warshallalgorithm to find transitive closure.
- Solve path matrix graph using Warshal's **algorithm**

Viva questions

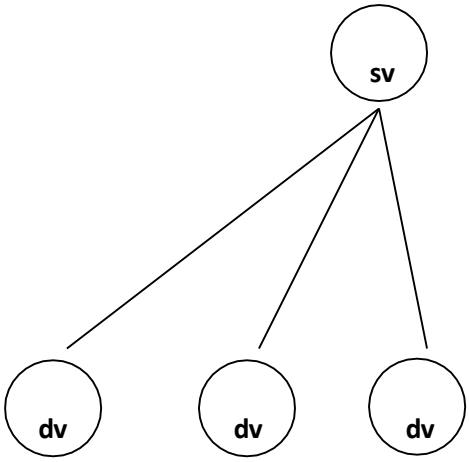
- Warshall algorithm is an example of which type of algorithm.
- Difference between Floyds and warshall Algorithm
- What is the time complexity of warshalls algorithm?
- What is the best, worst and average case of warshall's algorithm?

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.

Objectives: This Program enable students to :

- LearnDijkstra's sorting and searching techniq ues to find shortest paths.
- Getexposed toDijkstra's algorithm design technique

Dijkstra's algorithm finds shortest paths to a graph's vertices in order of their distance from a given source. First, it finds the shortest path from the source to a vertex nearest to it, then to a second nearest and so on.



Sv: starting vertex. Dv: destination vertex.

Aim:

Algorithm:

// Input: A weighted connected graph G = {V, E}, source s

// Output dv:the distance-vertex matrix

- Read number of vertices of graph G
- Read weighted graph G
- Print weighted graph
- Initializedistancefromsourceforallverticesasweightbetweensourcenodeand other vertices, i, and none in tree

// initialcondition

- For all other vertices,

  dv[i]=wt_graph[s,i],TV[i]=0,prev[i]=0dv[s]=0,prev[s]=s// source vertex

- Repeat for y= 1 ton

  v= next vertex with minimum dv value, by calling Find Next Near() Add v to tree.

  For all the adjacent u of v and u is not added to the tree, if dv[u]> dv[v]+wt_graph[v,u] then dv[u]=dv[v]+wt_graph[v,u]andprev[u]= v. □ findNextNear

//Input:graph, dv matrix

//Output: jthe next nearest vertex

- min m = 9999 o For k =1 to n ifkvertexisnotselectedintree

and ifdv[k] <minm

{ minm = dv [k]

j=k

}

```
#include<stdio.h>
void dij(int, int [20][20], int [20], int [20], int);
void main() {
    int i, j, n, visited[20], source, cost[20][20], d[20];
    printf("Enter no. of vertices: ");
    scanf("%d", &n);
    printf("Enter the cost adjacency matrix\n");
    for (i = 1; i <= n; i++) {
        for (j = 1; j <= n; j++)
        { scanf("%d", &cost[i][j]);
        }
    }
    printf("\nEnter the source node: ");
    scanf("%d", &source);
    dij(source, cost, visited, d, n);  for (i
    = 1; i <= n; i++) {
        if (i != source)
            printf("\nShortest path from %d to %d is %d", source, i, d[i]);
    }
}
void dij(int source, int cost[20][20], int visited[20], int d[20], int n)
{  int i, j, min, u, w;
    for (i = 1; i <= n; i++) {
        visited[i] = 0;
        d[i] = cost[source][i];
    }
    visited[source] = 1;
    d[source] = 0;
    for (j = 2; j <= n; j++)
    {  min = 999;
        for (i = 1; i <= n; i++)
        { if (!visited[i])  {
                if (d[i] < min)
                { min = d[i];
                    u = i;
```

```
            }
        }
    }
    visited[u] = 1;
    for (w = 1; w <= n; w++) {
        if (cost[u][w] != 999 && visited[w] == 0)
        { if (d[w] > cost[u][w] + d[u])
            d[w] = cost[u][w] + d[u];}}}}
```

Output:

```
Enter no. of vertices: 6
Enter the cost adjacency matrix
999 3   999  999 6   5
3   999  1   999 999 4
999 1   999  6   999 4
999 999 6   999  8   5
6   999 999  8   999 2
5   4   4   5   2   999

Enter the source node: 1

Shortest path from 1 to 2 is 3
Shortest path from 1 to 3 is 4
Shortest path from 1 to 4 is 10
Shortest path from 1 to 5 is 6
Shortest path from 1 to 6 is 5
```

Outcomes: On completion of this Program, the students are able to:

- Solve Dijkstra's sorting and searching techniques to find shortest paths.
- Develop a program that can be solved to Dijkstra's algorithm design techniques.

Viva questions

1. How do you represent the graph?
2. Application of Dijkstras Algorithm.
3. Difference between Floyds and Dijkstras algorithm.
4. Dijkstras algorithm can be solved byusing which type algorithm method.

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

Objectives: This Program enable students to :

- Learn Topological ordering of vertices in digraph.
- Source Removal Method design technique

Method: Source removal method. Each vertex may have an incoming edge or may not have incoming edge. Vertex with no incoming edges deleted along with all the edges outgoing from it. (If there are several sources, break tie arbitrarily. If there none, stop because the problem cannot be solved). The order in which the vertices are deleted is to be displayed. Here While numbering the vertices of the graph, remember to assign 1 to vertex with no incoming edge.

A situation can be modeled by a graph in which vertices represent courses and directed edges indicate prerequisite requirements. In terms of this digraph, the question is whether we can list its vertices in such an order that for every edge in graph, the vertex where the edge starts is listed before the vertex where the edge ends. This problem is called topological sorting. If a directed graph has a directed cycle, then sorting is not possible. For topological sorting to be possible, a digraph must be a dag (Directed Acyclic graph). It turns out that being a dag is not only necessary but also sufficient for topological sorting to be possible. i.e if a diagraph has no cycles, the topological sorting problem for it has a solution. There are two efficient algorithms that both verify whether a digraph is a dag and if it is, produce an ordering vertices that solves the topological sorting problem.

First algorithm is DFS. 2nd algorithm is Source removal method. Second method is based on decrease and conquer technique: Repeatedly identifying in a remaining digraph. **A source, which is a vertex with no incoming edges** and delete it along with all the edges outgoing from it. **(If there are several sources, break tie arbitrarily. If there none, stop because the problem cannot be solved)**. The order in which the vertices are deleted yields a solution to the topological sorting problem. The solution obtained by the source removal algorithm is different from the one obtained by the DFS based algorithm. Both of them are correct, of course. The topological sorting problem may have several alternative solutions.

```
#include<stdio.h>
void findindegree(int [10][10],int[10],int);
void topological(int,int [10][10]);
void main ()
{
int a[10][10],i,j,n;
printf("Enter the number of nodes:"); scanf("%d",&n);
printf("\nEnter the adjacency matrix\n");
for(i=1;i<=n;i++)
for(j=1;j<=n;j++)
scanf("%d",&a[i][j]);
printf("\nThe adjacency matirx is:\n"); for(i=1;i<=n;i++)
{
for(j=1;j<=n;j++)
{
printf("%d\t",a[i][j]);
}
printf("\n");
}
topological(n,a);
}
void find indegree(int a[10][10],int indegree[10],int n)
{
int i,j,sum; for(j=1;j<=n;j++)
{
sum=0;
for(i=1;i<=n;i++)
{
sum=sum+a[i][j];
}
indegree[j]=sum;
}
}
void topological(int n,int a[10][10])
{
int k,top,t[100],i,stack[20],u,v,indegree[20];
  k=1;
top=-1;
```

```
findindegree(a,indegree,n);
for(i=1;i<=n;i++)

{
if(indegree[i]==0)
{
stack[++top]=i;
}
}
while(top!=-1)
{
u=stack[top--]; t[k++]=u;
for(v=1;v<=n;v++)
{
if(a[u][v]==1)
{
indegree[v]--;
if(indegree[v]==0)
{
stack[++top]=v;
}}
}}
printf("\nTopological sequence is\n");
for(i=1;i<=n;i++)
printf("%d\t",t[i]);
}
```

Output:

Enter the number of nodes:5

Enter the adjacency matrix

```
0  0  1  0  0
0  0  1  0  0
0  0  0  1  1
0  0  0  0  1
0  0  0  0  0
```

The adjacency matrix is:

```
0    0    1    0    0
0    0    1    0    0
0    0    0    1    1
0    0    0    0    1
0    0    0    0    0
```

Topological sequence is

```
2    1    3    4    5
```

Outcomes: on completion student will be able to

- Solve Topological ordering of verticesindigraph.
- Source Removal Method design technique

Viva questions

- What is topological sorting?
- Find the topological order of vertices for the given graph.
- What is the time complexity of topological sorting?
  Which are the two algorithm is used to solve topological

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

Objectives: This Program enable students to :

Learn the **0/1 Knapsack** problemusing Dynamic Programming method

ALGORITHM:

Knapsack (i, j)

//**Input:** A nonnegative integer i indicating the number ofthe first items being considered and a non-negative integer j indicating the Knapsack's capacity.

//**Output**: The value of an optimal feasible subset of the first i items.

//**Note**: Uses as global variables input arrays Weights [1...n], Values [1… n] and table V [0…n, 0…W] whose entries are initialized with -1's except for row 0 and column 0 initialized with 0's.

```
ifV [i, j] < 0

        if j< Weights[i]
then value = Knapsack
( i-1, j )

else
    value = max ( Knapsack ( i-1, j),
        values[I] + Knapsack ( i-1, j-
            Weights[i] ) ) V[i j] = value

returnV [ i, j]
```

```
#include<stdio.h>
#define MAX 50
int p[MAX],w[MAX],n; int
knapsack(int,int);
int max(int,int); void
main()
{
int m,i,optsoln;

printf("Enter no. of objects: ");
scanf("%d",&n); printf("\nEnter the
weights:\n"); for(i=1;i<=n;i++)
scanf("%d",&w[i]); printf("\nEnter
the profits:\n"); for(i=1;i<=n;i++)
scanf("%d",&p[i]);
printf("\nEnter the knapsack capacity:");
scanf("%d",&m); optsoln=knapsack(1,m);
printf("\nThe optimal soluntion is:%d",optsoln);
}
int knapsack(int i,int m)
{
if(i==n)
return (w[n]>m) ? 0 : p[n];
if(w[i]>m)
return knapsack(i+1,m);
return  max(knapsack(i+1,m),knapsack(i+1,m-w[i])+p[i]);
}
int max(int a,int b)
{
if(a>b)
return a; else
return b;
}
```

Output:

Enter no. of objects: 3

Enter the weights:
100 10 14

Enter the profits:
20 18 15

Enter the knapsack capacity:116

The optimal solution is:38

Outcomes: On completion of this Program, the students are able to :

- Solve **0/1 Knapsack** problem using Greedy method to implement in java

Viva questions :

1. What is dynamic program?
2. Knapsack problem can be solved byusing which method.
3. Whatisthedifferencebetweendynamicmethodandgreedymethodprograming

Techniques?

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

Objectives: This Program enable students to:

Learn the **0/1 Knapsack** problem using Greedy method to implement using c/c++

ALGORITHM:

Knapsack (i, j)

//**Input:** A nonnegative integer i indicating the number ofthe first items being considered and a non-negative integer j indicating the Knapsack's capacity.

//**Output**: The value of an optimal feasible subset of the first i items.

//**Note**: Uses as global variables input arrays Weights [1...n], Values [1… n] and table V [0…n, 0…W] whose entries are initialized with -1's except for row 0 and column 0 initialized with 0's.

```
ifV [i, j] < 0

        if j< Weights[i]
then value = Knapsack
( i-1, j )

else
    value = max ( Knapsack ( i-1, j),
        values[I] + Knapsack ( i-1, j-
            Weights[i] ) ) V[i j] = value

returnV [ i, j]
```

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;
// Structure to represent an item
struct Item {
    int weight; int
    value;
};
// Function to solve discrete knapsack using greedy approach
int discreteKnapsack(vector<Item>& items, int capacity) {
    // Sort items based on their value per unit weight
     sort(items.begin(), items.end(), [](const Item& a, const Item& b) {
        return (double)a.value / a.weight > (double)b.value / b.weight;
    });
    int totalValue = 0;
    int currentWeight = 0;
    // Fill the knapsack with items for
    (const Item& item : items) {
        if (currentWeight + item.weight <= capacity) {
            currentWeight += item.weight;
            totalValue += item.value;
        }
    }
    return totalValue;
}
// Function to solve continuous knapsack using greedy approach
double continuous Knapsack(vector<Item>& items, int capacity) {
    // Sort items based on their value per unit weight sort (items. Begin (),items.
    End (), [](const Item& a, const Item& b) {
        return (double)a.value / a. weight > (double)b.value / b. weight;
    });
    double totalValue = 0.0; int
    currentWeight = 0;
    // Fill the knapsack with items fractionally for
    (const Item& item: items) {
```

BCSL404

```
        if (currentWeight + item.weight <= capacity) {
            currentWeight += item.weight;
            totalValue += item.value;
        } else {
            int remainingCapacity = capacity - currentWeight;
            totalValue += (double)item.value / item.weight * remainingCapacity;
            break;

        } }
    return totalValue;
}
int main() { vector<Item>
    items; int n, capacity;
    // Input number of items and capacity of knapsack
    cout << "Enter the number of items: ";
    cin >> n;
    cout << "Enter the capacity of knapsack: "; cin
    >> capacity;
    // Input the weight and value of each item
    cout << "Enter the weight and value of each item:" << endl; for
    (int i = 0; i < n; i++) {
        Item item;
        cout << "Item " << i + 1 << ": "; cin
        >> item.weight >> item.value;
        items.push_back(item);
    }
    // Solve discrete knapsack problem
    int discreteResult = discreteKnapsack(items, capacity);
    cout << "Maximum value for discrete knapsack: " << discreteResult << endl;
    // Solve continuous knapsack problem
    double continuousResult = continuousKnapsack(items, capacity);
    cout << "Maximum value for continuous knapsack: " << continuousResult << endl; return
    0;
}
```

BCSL404

Output:

Enter the number of items :4
Enter the capacity of knapsack :10
Enter the weight and value of each item :
Item 1: 2 10
Item 2: 3 5
Item 3: 5 15
Item 4: 7 7
Maximum value for discrete knapsack :30
Maximum value for continuous knapsack :30

Outcomes : Programenable students to :

the **0/1 Knapsack** problem using Greedy method to implement using c/c++

Viva questions :

1. What is dynamic program?
2. Knapsack problem can be solved byusing which method.
3. Whatisthedifferencebetweendynamicmethodandgreedymethodprograming

Techniques?

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.

Objectives: This Program enable students to :

- LearnSubset problem.
- Find SUM is equal to a givenpositive integer

Aim:

An instance ofthe Subset Sum problem Is a pair(S,t),where S={x1,x2,..... ,xn} is a set of Positive integers and t (the target) is a positive integer. The decision problem asks for asubset of S whose sum is as large as possible, but not larger than t.

This problem is NP-complete.

This problem arises in practical applications. Similar to the knapsack problem we may have a truck that can carry at most t pounds and we have n different boxes to ship and the ith box weighs xipounds. The naive approach of computing the sum of the elements of every subset of S and then selecting the best requires exponential time. Below we present an exponential time exact algorithm.

```
#include<stdio.h>
void subset(int,int,int);
int x[10],w[10],d,count=0;
void main()
{
int i,n,sum=0;

printf("Enter the no. of elements: ");
scanf("%d",&n);
printf("\nEnter the elements in ascending order:\n");
for(i=0;i<n;i++)
scanf("%d",&w[i]);
printf("\nEnter the sum: ");
scanf("%d",&d);
for(i=0;i<n;i++)
sum=sum+w[i];
if(sum<d)
{
printf("No solution\n");
return;
}
subset(0,0,sum);
if(count==0)
{
printf("No solution\n");
return;
}
}
void subset(int cs,int k,int r)
{
int i; x[k]=1;
if(cs+w[k]==d)
{
printf("\n\nSubset %d\n",++count);
for(i=0;i<=k;i++)
```

```
if(x[i]==1)
 printf("%d\t",w[i]);
}
else

if(cs+w[k]+w[k+1]<=d)
subset(cs+w[k],k+1,r-w[k]); if(cs+r-
w[k]>=d && cs+w[k]<=d)
{ x[k]=0;
subset(cs, k+1,r-w[k]);
}
}
```

Output:

Enter the no. of elements: 5

Enter the elements in ascending order:
1
2
5
6
8

Enter the sum: 9

Subset 1
1 2 6

Subset 2
1 8

Outcomes: This Program enable students to:

- Study Subset problem.
- StudySUM is equal to a given positive integer

Viva questions

- Solve the given problem for subset sum int [] A= {3, 2, 7, 1}, S =6
- Give the time complexity equation for subset sum.
- Subset problem can be solved by using which type of algorithm.
- What is the difference between Backtracking and Branch and bound algorithm?

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.

Objectives: This Program enables students to:

- Learn Selection sorting and searching techniques.
- Get exposed to Selection Sort algorithm designtechniques.

Algorithm:

Selection sort is a simple sorting algorithm. This sorting algorithm is an in-place comparison-based algorithm in whichthe list is divided into two parts, the sorted part at the left end and the unsorted part

At the right ends. Initially, the sorted part is empty and the unsorted part is the entire list.

The smallest element is selected from the unsorted array and swapped with the leftmost element, and that element becomes a part of the sorted array. This process continues moving unsorted array boundary by one element to the right.

Steps:

1) class Selection Sort(arr,n) is created
2) Step 1: Repeat 2
   and 3 For i=0
   to n-1.
3) Step 2: call smallest(arr,I,n,pos)
4) Step 3: swap arr[i] with arr[pos]
5) END OF LOOP
6) Step 4:exit
7) Smallest (arr,I,n,pos)
8) Step 1: (INITIALIZE) SET SMALL = arr[i]

```
#include  <stdio.h>
#include <stdlib.h>
#include <time.h>
// Function to perform selection sort
void selectionSort(int arr[], int n)
 {
    int i, j,  minIndex,  temp;
     for (i = 0; i < n - 1; i++) {
        minIndex = i;
           for (j = i + 1; j < n; j++) {
            if (arr[j] < arr[minIndex])
            {  minIndex = j;
            }
        }
        // Swap the found minimum element with the first element  temp
        = arr[minIndex];
        arr[minIndex] = arr[i];
        arr[i] = temp;
    }
}
// Function to generate random numbers between 0 and 999
int generateRandomNumber() {
    return rand() % 1000;
}
int main() {
    // Set n value
    int n = 6000;

    // Allocate memory for the array
    int* arr = (int*)malloc(n * sizeof(int));
```

```
    // Generate random elements for the array
    srand(time(NULL));
    printf("Random numbers for n = %d:\n", n); for
    (int i = 0; i < n; i++) {
        arr[i] = generateRandomNumber();
        printf("%d ", arr[i]);
    }
    printf("\n");

    // Record the start time
    clock_t start = clock();

    // Perform selection sort
    selection Sort(arr, n);
    // Record the end time
    clock_t end = clock();
    // Calculate the time taken for sorting
    double time_taken = ((double)(end - start)) / CLOCKS_PER_SEC;
    // Output the time taken to sort for the current value of n
    printf("\nTime taken to sort for n = %d: %lf seconds\n\n", n, time_taken);
    // Display sorted numbers
    printf("Sorted numbers for n = %d:\n", n); for
    (int i = 0; i < n; i++) {
          printf("%d ", arr[i]);
    }
    printf("\n\n");
    // Free the dynamically allocated memory free(arr);
    return 0;
}
```

Output:

Random numbers for n=6000:
243 112 599 677 912 413 721 547 640 822 … (more numbers) .....394.....

Time taken to sort for n=6000: 1.058000 seconds

Sorted numbers for n=6000:
0 0 1 2 2 3 3 3 4 5 …..(more numbers) ….. 995 996 996 997 998 999 999

Outcomes: Oncompletion of this Program, the students are able to:
- Solve problems by applying Selection Sort algorithms.

Viva Questions:-

1. How Selection sort works
2. What is the worst-case behavior for Selection sort?
3. What is the Best case behavior for Selection sort?

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.

Objectives: This Program enable students to:

- Learn quick sorting and searching techniques.
- Get exposed to Quick Sort algorithm design techniques.

Quick sort is one of the sorting algorithms working on the Divide and Conquer principle. Quick sort works by finding an element, called pivot, in the givenarray and partitions the array into three sub arrays such that

The left sub arraycontains all elements which are less than or equal to the pivot The middle sub arrays contains pivot
The right sub array contains all elements which are greater than or equal to the pivot.

Quick sort is well-known sorting algorithms based onthe divide and conquer technique.

Quicksort divide s its input elements according to the ir value .Specifically, it rearrange s elements of a given array A[n-1]to achieve its **partition**, as it uation where all the elements before some position s are smaller than or equal to A[s] and all the elements after some position s are greater than or equal to A[s].We call this element the pivot.

A[0]…A[s-1] A[s] A[s+1]…A[n-1]

Allare<=A[s] all are>=A[s]

After a partition has been achieved, A[s] will be in its final position in the sorted array, and wecan continue sorting the two sub arrays of the elements preceding and following A[s] independently.

ALGORITHM

Quicksort(A[*l*...*r*])

// Sorts a subarray by quicksort
// Input: A subarray A [*l*...*r*] of A [0...n-1], defined by its left andright indices l and r
// Output: The subarray A [*l*...*r*] sorted in non-decreasing order if l< r

s← partition(A[*l*...*r*]) //***s* in a split position**
Quicksort (A[*l*...*s*-1])
Quicksort (A[s+1...*r*])

ALGORITHM

Partition(A[*l*...*r*])

// Partitions a subarray by using int first element as a pivot

// Input: A sub-array A[*l*...*r*] of A[0...n-1], defined by its left and right Indices *l* and *r* (*l*<*r*)
// Output: A partition of A [*l*... *r*], with the split position returned as this function's value

P ← A[*l*]
i← *l* ; j← *r* + 1

repeat
repeat i← i+ 1 **until** A[i] >= p
repeat j← j– 1 **until** A[j] <=pswap (A[i] , A[j])
until i> = j

Swap (A [i] , A [j]) // **undo the last swap when i > = j**
Swap (A[*l*] , A[j])
return j

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

// Function to swap two elements 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 perform 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);
    }
```

```
}

// Function to generate random numbers between 0 and 999 int
generateRandomNumber() {
    return rand() % 1000;
}

int main() {
    // Set n value int n
    = 6000;

    // Allocate memory for the array
    int* arr = (int*)malloc(n * sizeof(int));

    // Generate random elements for the array
    srand(time(NULL));
    printf("Random numbers for n = %d:\n", n); for
    (int i = 0; i < n; i++) {
        arr[i] = generateRandomNumber();
        printf("%d ", arr[i]);
    }
    printf("\n");

    // Record the start time
    clock_t start = clock();

    // Perform quick sort
    quickSort(arr, 0, n - 1);

    // Record the end time
    clock_t end = clock();

    // Calculate the time taken for sorting
    double time_taken = ((double)(end - start)) / CLOCKS_PER_SEC;

    // Output the time taken to sort for the current value of n
    printf("\nTime taken to sort for n = %d: %lf seconds\n\n", n, time_taken);
```

```
    // Display sorted numbers
    printf("Sorted numbers for n = %d:\n", n);  for
    (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n\n");

    // Free the dynamically allocated memory free(arr);

    return 0;
}
```

Output:

Random numbers for n=6000:
243 112 599 677 912 413 721 547 640 822 … (more numbers) .....394....

Time taken to sprt for n=6000 : 1.058000 seconds

Sorted numbers for n=6000:
0 0 1 2 2 3 3 3 4 5 …..(more numbers) ….. 995 996 996 997 998 999 999

Outcomes: Oncompletion of this Program, the students are able to :

- Solve problems byapplying Quick Sort algorithms.

Viva questions

1. How quick sort works
2. What is the worst-case behavior (number of comparisons) for quick sort?
3. Explain Divide and conquer method. Give an example

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.

Objectives: This Program enable students to:

- Learn quick sorting and searching techniques.
- Get exposed to merge Sort algorithm designtechniques.

The basic operation in merge sort is comparison and swapping. Merge sort algorithm calls it self recursively. Merge sort divides the array into sub arrays based on the position of the elements whereas quick sort divides the array into sub arrays based on the value of the elements.

Merge sort requires an auxiliary array to do the merging. The merging of two subarrays, which are already sorted, into an auxiliary array can be done in O(n) where n is the total number of elements in both the sub arrays. This is possible because both the sub arrays are sorted.

ALGORITHM

```
Mergesort(A[0...n-1])
        //Sorts array A [0...n-1] by recursive mergesort.
        //Input an array A [0...n-1] of orderable elements.
        // Output Array A [0...n-1] sorted in non-decreasing order.

 If n>1
        CopyA[0...(n/2)-1] to B[0...(n/2)-1]
        CopyA[(n/2)...n-1], to C[0...(n/2)-1]
        Mergesort(B[0...(n/2)-1])
        Mergesort(C[0...(n/2)-
        1]) Merge(B,C,A)
```

```
#include  <stdio.h>
#include <stdlib.h>
#include <time.h>
// Merge two subarrays of arr[]
// First subarray is arr[l..m]
// Second subarray is 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 there are any
     while (i <  n1) {
        arr[k] = L[i];
        i++;
        k++;
```

```
    }


    // Copy the remaining elements of R[], if there are 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);
    }
}

// Function to generate random numbers between 0 and 999
int generateRandomNumber() {
    return rand() % 1000;
}

int main() {
    // Set n value
     int n = 6000;

    // Allocate memory for the array
    int* arr = (int*)malloc(n * sizeof(int));
```

```
    // Generate random elements for the array
    srand(time(NULL));
    printf("Random numbers for n = %d:\n", n);
    for (int i = 0; i < n; i++) {
        arr[i] = generateRandomNumber();
        printf("%d ", arr[i]);
    }
    printf("\n");

    // Record the start time

    clock_t start = clock();
    // Perform merge sort
    mergeSort(arr, 0, n - 1);
    // Record the end time clock_t
    end = clock();
    // Calculate the time taken for sorting
    double time_taken = ((double)(end - start)) / CLOCKS_PER_SEC;

    // Output the time taken to sort for the current value of n
    printf("\nTime taken to sort for n = %d: %lf seconds\n\n", n, time_taken);
    // Display sorted numbers
    printf("Sorted numbers for n = %d:\n", n); for
    (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n\n");
    // Free the dynamically allocated memory
    free(arr);
    return 0;
}
```

Output:

Random numbers for n=6000:
243 112 599 677 912 413 721 547 640 822 … (more numbers).....394.....

Time taken to sprt for n=6000 : 1.058000 seconds

Sorted numbers for n=6000:
0 0 1 2 2 3 3 3 4 5 …..(more numbers) ….. 995 996 996 997 998 999 999

Outcomes: On completion of this Program, the students are able to:

- Solve problems by applying Merge Sort algorithms

Viva questions

1. What is Mergesort
2. Explain Decrease/transfer and conquer method. Give an example.
3. Difference between Divide and conquer method
4. . Sort the given elements using Merge sort 67 24 9 5 25 66 88 4990

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

objectives: This Program enable students to:

To find Hamiltonian Cycle inan undirected GraphG of*n* vertices.

Algorithm:

Input:

A 2D array graph[V][V] where V is the number of vertices in graph and graph[V][V] is adjacency matrix representation of the graph. A value graph[i][j] is 1 if there is a direct edge from i to j, otherwise graph[i][j] is 0.

Output:

An array path[V] that should contain the Hamiltonian Path. path[i] should represent the ith vertexintheHamiltonianPath.Thecodeshouldalsoreturnfalseifthereis no Hamiltonian Cycle in the graph.

Backtracking Algorithm

Create an empty path array and add vertex 0 to it. Add other vertices, starting from the vertex 1. Before adding a vertex, check for whether it Is adjacent to the previously added vertex and not already added. If we find such a vertex, we add the vertex as part of the solution. If we do not find a vertex, then we return false.

```
#include <iostream>
#include <vector> using
namespace std;
// Function to print the solution
void printSolution(const vector<vector<char>>& board) { for
    (const auto& row : board) {
            for (char cell : row)
             cout << " " << cell << " ";
        cout << endl;
    }
}
// Function to check if a queen can be placed on board[row][col] bool
isSafe(const vector<vector<char>>& board,  int row, int col) {
    int i, j;
    int n = board.size();

    // Check the row on the left side  for
    (i = 0; i < col; i++)
        if (board[row][i] == 'Q')
            return false;
    // Check upper diagonal on the left side
    for (i = row, j = col;  i >= 0 && j >= 0; i--, j--)  if
        (board[i][j] == 'Q')
            return false;
    // Check lower diagonal on the left side
    for (i = row, j = col; j >= 0 && i < n; i++, j--) if
        (board[i][j] == 'Q')
            return false;  return
    true;
}
// Recursive function to solve N Queens problem
bool solveNQUtil(vector<vector<char>>&  board,  int  col)  {  int
    n = board.size();// If all queens are placed, return true
if (col >= n)
return true;
```

```
    // Consider this column and try placing this queen in all rows one by one for
    (int i = 0; i < n; i++) {
        // Check if the queen can be placed on board[i][col] if
        (isSafe(board, i, col)) {
            // Place this queen in board[i][col] board[i][col]
            = 'Q';
            // Recur to place rest of the queens if
            (solveNQUtil(board, col + 1))
                return true;
            // If placing queen in board[i][col] doesn't lead to a solution, then remove queen from
board[i][col]
            board[i][col] = '-';
        }
    }
    // If the queen cannot be placed in any row in this column, then return false
    return false;
}
// Function to solve N Queens problem for 4 queens
void solve4Queens() {
    int n = 4;
    vector<vector<char>> board(n, vector<char>(n, '-'));
      if (solveNQUtil(board, 0) == false) {
        cout << "Solution does not exist" << endl; return;
    }
    printSolution(board);
}
// Driver function
 int main() {
    cout << "Solution for 4 Queens problem:" << endl;
    solve4 Queens();
    return 0;
}
```

Output:

Solution for 4 Queens problem:

```
-   -   Q   -
-   Q   -   -   -
-   -   -   Q
-   Q   -   -
```

Outcomes: On Completion of these Program enable students to:Solve

Hamiltonian Cycle in an undirected GraphG of *n* vertices.

Viva questions

- What is N-Queens problem?
- Difference between 'Branch and bound' and 'Backtracking 'algorithm.
- What is the time complexity ofN-queen's problem?