ATME COLLEGE OF ENGINEERING

13th KM Stone, Mysuru, Kanakapura-Bangalore Road - 570 028

DEPARTMENT OF COMPUTER SCIENCE & DESIGN

LABORATORY MANUAL OF

DESIGN AND ANALYSIS OF ALGORITHM LABORATORY

ACADEMIC YEAR 2024-25

SUBJECT CODE: BCSL404

[As per Choice Based Credit System (CBCS) scheme]

SEMESTER–IV

Prepared by	Verified by	Approved by
Mr. MAHADEVASWAMY D.M	**Ms. DARSHINI Y**	**Dr. NASREEN FATHIMA**
PROGRAMMER	**FACULTY COORDINATOR**	**HOD, CS&DESIGN**

INSTITUTIONAL MISSION AND VISION

Objectives

- To provide quality education and groom top-notch professionals, entrepreneurs and leaders for different fields of engineering, technology and management.
- To open a Training-R & D-Design-Consultancy cell in each department, gradually introduce doctoral and postdoctoral programs, encourage basic & applied research in areas of socialrelevance, and develop the institute as a center ofexcellence.
- To develop academic, professional and financial alliances with the industry as well as the academia at national and transnational levels.
- To cultivate strong community relationships and involve the students and the staff in Local community service.
- To constantly enhance the value of the educational inputs with the participation of students, faculty, parents and industry.

Vision

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

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.

.

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

CONTENTS

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 indeveloping new algorithms for required applications

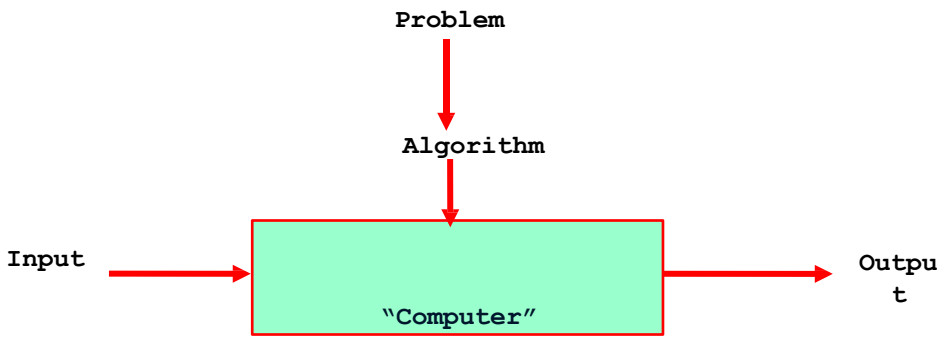
Algorithm

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

In addition, all algorithms must satisfy the following criteria:

1. **Input**: Each algorithm should have zero or more inputs. The range of input 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, in a 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**

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

Ex: travelling salesman problem, graph coloring problem etc

- **Geometric problems**

This problem dealwithgeometric objects such as points, curves, lines, polygon etc.

Ex: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 ofdata.

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 ofdata items ina certain order.

2. Graphs

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

- **Undirected graph**: A 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 graphthat 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 analgorithm's efficiencywith respect to resources:

• **Running time (time efficiency)**

It indicates how fast an algorithm in question runs

• **Memory space (space efficiency)**

It deals withthe 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 operationwill 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:

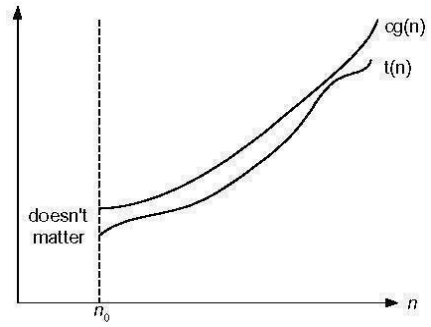
Average time taken(number of times the basic operationwill be executed) **to solve all the possible instances (random) of the input.**

Asymptotic Notations

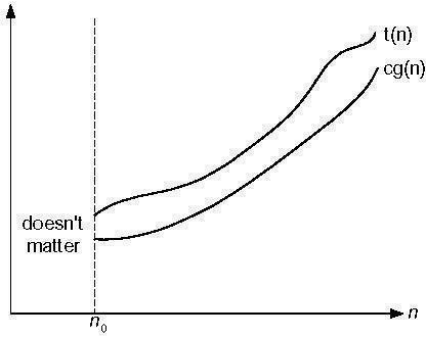
- Asymptotic notation is a way of comparing functions that ignores constant factors and small input sizes.
- Three notations used to compare orders of growth of an algorithm's basic operation are: **O, Ω, θ notations.**

Big-oh notation:

A function t(n) is said to be in O(g(n)),denoted t(n)€O(g(n)),if t(n) is bounded above by some constant multiple of g(n) for all large n i.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 notationt(n)€O(g(n))



Big-omega notation t(n)€Ω,(g(n))

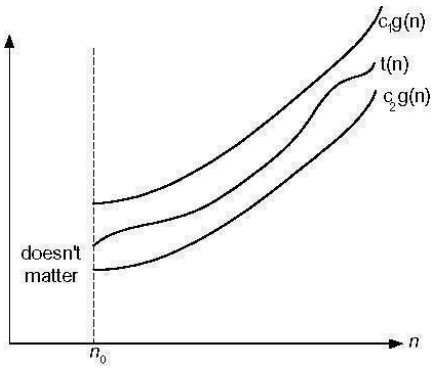
Big-omega notation:

A function t(n) is said to be inΩ(g(n)),denoted t(n)€ Ω(g(n)),ift(n) is bounded below by some constant multiple of g(n) for all large n i.e., if there exist some positive constant c and Some nonnegative integer n◌ 0 suchthat

t(n)>=cg(n) for all n>=n_0Big-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 notation

t(n)€θ(g(n))Basic Asymptotic Efficiency classes:

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.

```
#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

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

```
#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
```

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

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

#define V 4

void floydWarshall(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
```

3B Design and implement C/C++ Program to find the transitive closure 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

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.

```
#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
```

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

```
#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 findindegree(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 matirx 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
```

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

```
#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 solution 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

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

```
#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 continuousKnapsack(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) {
        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;
}
```

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
```

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.

```
#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

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.

```
#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
    selectionSort(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 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

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.

```
#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

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.

```
#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

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

```
#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;  solve4Queens();
    return 0;
}
```

Output:

```
Solution  for  4  Queens  problem :
-    -  Q    -
-    Q    -    -    -
-    -    -   Q
-    Q   -    -
```

VIVA QUESTIONS

1. What is an Algorithm?

Algorithm is a Step by step procedure to Solve a given problem for a finite number of input producing finitenumber of output with desired output.

2. What is a Flow Chart?

Flow chart is a Graphical Representation of a solution to the Problem.

3. What is the difference between Algorithm, Flow Chart, Program?

- Algorithm specifies the different things to be followed for solving a Problem.
- Flow Chart is a Graphical Representation of a Solution to the Problem. Both Algorithm and Flow Chartare Machine Independent.
- Program is a Set of Instructions which is used as a tool to communicate to the machine to get our workdone,Program is Machine Dependent for particular Machine.

4. What is the Aim of DAA lab or why we need to study DAA Lab

DAA is a discipline, where we are dealing with designing or writing the algorithm keeping in Consideration of Space and Time Complexity, Such that Our Algorithm should execute in a very minimum amount of time by Minimum Space or RAM.

5. Define Space and Time Complexity? Among this which one is more prioritized?

Space Complexeity is a measure of Amount of Space taken by a Program to finish its Execution.

Time Complexeity is a measure of amount of time taken by a program to complte its Execution.Depending Upon Application it is considered,EX:For Mobile or Handheld Devices,We give Prefernce for both Spaceand time.

For a Huge and Inter active Systems like Web Applications we give more Preferences to time Complexeity.

6. What is Design and what is Analysis of a Program?

Design is a Process of Writing an algorithm to a given Problem so that it should accept finite number ofinput and finite number of output with a definite output and Should Exit appropriately.

Analysis:Analysis is a next Phase of Writing an Algorithm ,in this phase we calculate the Efficiency of anAlgorithm i.e time and space needed by an algorithm.

7. Write the general plan for analyzing the recursive algorithms.

- Identify the inputs.
- Identify the output is depended only on number of inputs.
- Identify the Basic Operation in Algorithm.
- Form or write the Recursive Relation to the Algorithm.

8. What are the various notations used to write an algorithm?

(i) Pseudocode (ii)Natural Language and etc..

9. What is a Pseudocode?

It's a notation which is having the combination of Programming Constructs and English like Statements.

10. What is the Time Complexeity of Bubble Sort,Selection Sort ,Merge Sort,Quick Sort?(L3)

Bubble Sort-$n^{2,}$ Selection Sort- n^2 Merge Sort-nlog.n Quick Sort -nLogn, Worst case for Quick Sort- n2

11. Which sorting agorithm is more Efficient and why?

Quick Sorting is More Efficient ,because this algorithm is instable algorithm andinplace.

12. What do you mean by the term Instable Algorithms?

The Instable Algorithms are one, which divides the array as certainly depending upon pivot or key elementand hence i index precedes index j

13. Which algorithms are faster?

Instable Algorithms are much Faster compared to Stable Algorithms.

14. For what type of instance Merge sort do better than Quick Sort?

For a Larger input and a sorted input values.

15. For what type of instance Quick sort do better than Merge Sort?

For Smaller Set of input numbers.

16. What are Inplace Algorithms?

Inplace Algorithms are the one which doesn't occupies Extra Space.

17. Write the order of growth terms as per the time Execution in Ascending Order.

logn,n,nlogn,n^2,n^3,.....n^n,2^n,n!

18. What is Brute Force Technique? When We Should Use?

Brute Force is a straight Forward Technique to solve a problem, We used to solve a Problem throughthis approach when we don't have sufficient data to solve a problem in Efficient Way.

19. What is the difference between Divide and Conquer, Decrease and Conquer?

Divide and Conquer can be solved to solve a problem with a larger data set and when there is no dependency between any of the data sets.

- Divide and Solve as Small as Small sets.

Conquer or Merge it get one final resultant data set.

Decrease and Conquer is almost similar to Divide and Conquer but we are finding a solutions to the problem in a different variations,EX:Decrease by Constant (Usually by One),Decrease by Constant factorwhich is almost similar to Divide and Conquer Technique(Usually by two),Decrease by Variable(The Dividing Criteria changes for each iteration depends upon the data set.

20. Define Greedy Technique.

Greedy Technique is always applied for the problem of the type optimization type, which reduces loss and increases profit.

21. Define Optimal and Feasible Solution.

Optimal Solution is a solution which is best among N Feasible Solution. Feasible Solution is a solution which Satisfies a Problem Constraints/conditions.

22. Can A Problem solved by all the algorithmic Techniques.

Yes,but some problems will give better results with some Algorithmic Technique and it may give worstresult when it is applied with other technique.

23. State and Explain Knapsack Problem.

Filling the Maximum number of items to the Knapsack (Container) Which Increases the profit anddecreases the Loss.

24. Which one is Most Admired algorithmic Technique?

Dynamic Programming.

25. What is Spanning tree and Minimum Spanning tree?

A tree Without Cycles are called as Spanning tree . A Minimum Spanning Tree is a spanning tree which yeilds the very less Cost when all the edges costsummed up.

26. How Many Spanning Tree can a Tree can have?

A tree can have 1 to many number of Possible ways of Spanning Tree.

27. Differentiate between Prims and Kruskals Algorithm for finding MST.

In Prims We consider any one vertex in the graph as Source and We compute the distance from that source to other vertices ,after computing the vertices which has minimum value among (n-1) vertices is added to tree vertices and that respective edges added to tree Edges Set.The above mentioned Process continues till we reach (n-1) vertices.

In Kruskals we first arrange the edges in Ascending Order and then we start to form the tree which wont formcycles,if adding that edges forms cycles then that edges is dropped from adding to tree edges.The above saidprocess is continues till we reach the count of (n-1) Vertices.

28. What is the Application of Prims and Kruskals Algorithm?

In Networks to remove the Cyclicity of the Network.

29. Explain Job Sequencing With Deadlines?

Placing or scheduling the maximum number of Jobs to a machine without violating the deadlines constraintof any of the Jobs in Sequence.

30. Why the Name Bubble Sort named?Because in first Pass the first highest data will bubbles up,so since the largest element bubbles up in the first and second largest element bubbles up in the Second pass and so on, so hence the name bubble sort.

31. Why the Name Selection Sort?(L3)

The Selection sort is named because we initially first select an arrays first element as minimum and will compare with other elements ,so in pass one first least element goes to the first position and so on so forth for 2nd,3rd and so on. Selecting

32. What is the difference between Brute force strings matching to Horspool String Matching Method? (L2) In brute Force we compare each and every element of the text to the pattern by shiftingthe text positionby one and in Horspool method we shift it by number of shift positions recorded in theshift table.

33. Explain Merge Sort?

In Merge Sort will divide the entire input set by 2 until we reach low<high and later will find a solution to each item by comparing half of the array data set to the other half array data set and finally we mergeit to form a sinle array(conquer)

34. What is the Basic Operations in Merge sort and Quick sort?

In Merge Sort the Basic Operations is Comparisions and in Quick sort basic Operations is Partitioning andhence also known as partitioning sort.

35. Why the Insertion Sort?

We are Inserting an element to its suitable place by comparing n elements for each pass.

36. What is the Use of DFS and BFS?

DFS and BFS both used to check the Connectivity of a graph,Cyclicity in a graph,Spanning tree of agraph.

37. Differentiate between DFS and BFS.

DFS and BFS are both the Graph Traversing Technique,in which DFS Traverse the Graph in a depthwise(Vertical) and BFS Traverse the Graph from left to right(Horizontal)

38. Which Data structures used in BFS and DFS.

BFS USes Queue as its data structure and DFS uses as stack its Data structure.

39. What are back edges in DFS and Cross Edges in BFS.

Back Edges and Cross edges are the Edges which already visited by a ancestor node.

40. What is Topological Sorting?

Topological Sorting is a Sorting Technique used for sorting Vertices in the Graph.

41. What is the Conditions necessary for a Topological Sorting?

For a Topological Sorting the Graph Should be DAG(Directed Acyclic Graph)

42. What are the Different methods used to solve a topological Sorting ?

1. Source Removal Method
2. Using DFS based Scheme.

43. What is the Use of Topological Sorting?

Use of Topological Ordering is in the field of Operating System for Scheduling and in Networks,Automation and Robotics.

44. What is Dijikstra's Algorithm?

Dijikstra's Algorithm is Used to find the Single shortest Path from source to the other vertex.

45. What is a graph?

Graph is a component which is having a set of Edges and vertices G={V,E}

46. What are the different ways that can be represents a graph?

Adjaceny Matrix and Adjacency List.

47. What is Adjacency Matrix?

Is a Matrix which illustrates the Graph in the form of Matrix,if it is a weights Graph then we initialize the value of the cost in that position(i,j) or else simply we write 1 to mention ther exist an edge between (i,j) OR else we use 0 or 9999 to mention non connectivity of a graph.

48. What is the limitations of Algorithms?

Algorithm can't find the better the soltions when we come across the tight lower bound,So we can find thebetter solutins which is not possible with Algorithmic way.To find the Tight lower bound we use DecisionTrees.

49. What is Tight lower Bound?

It is a Lower bound which is a best lower bound for an problem,beyond that no algorithm will produce better results.

50. What are Decision Trees?

Decision trees are also known as Comparision Trees used to find the tight lower bound for a particular Problem EX:Tight Lower Bound For Sorting is n.logn

and tight lower bound for Searching is logn which is not possible to get the better result.

51. What is a polynomial problem (P-type)

P-type problem are decision problems in which we can find the solutions in a polynomial time and is oftype deterministic.

52. What is NP-problem?

NP-Problem belongs to decision problem and these problems are Non Deterministic Polynomial i.e forwhich the problem doesn't have deterministic solutions and Can be solved in Polynomial time

There are 2 phases in solving a problem.

(i) Guessing(Non-Deterministic stage) Producing N number of Candidate Outputs.

(ii) Verification(Deterministic Stage) Verifying The correctness of N Number of Candidate Outputs.

53. What is NP-Complete Problems?

Np_Complete Problems belongs to Decision problems and NP type Problems . These problems can be find the solutions by converting or reducing to the problem which we know theSolutions.

54. What is a trivial lower bound?

Trivial bound can be derived by formulating the number of inputs that has to be given and number ofoutputs that has to be generated.

55. Explain Bactracking W.r.t

(I)Subset Problem (ii)N-Queens Problem

56. Explain Subset Problem.

In a given Set S ,find the Subset,in which the sum of all subset elements is equal to the sum d which is predefined in a problem.

57. Explain N-Queens Problem.(

N-Queens Problem is of Placing a N-Queens in a N*N Chess board such that No 2-Queens Should be placed in the same Row,Column and same diagnol(N=Should consider both principal diagonal elements)

58. What is Hamiltonian Circuit?(L2)

Hamiltonian circuit is a problem in which that circuit starts from a source vertex and has other vertex in any order without repeating and Should end with the Source vertex only i.e source and Destination vertexshould be same.

59. Explain the Problem of TSP.(L2)

Travelling Sales Person Problem is a problem in which he should Visit N number of cities with a minimumnumber of Cost by visiting every city Exactly one and the city what he is started should end with same city.

60. What is the Concept of Dynamic Programming?(L3)

Deriving a Solution to the basic Condition and Extracting the solutions for the rest of the other data sets by Previously drawnd Solution.Computer to other algorithmic Technique Dynamic Programming because it avoids lot of reworking on the same Solution what we have solved in the earlier phases of deriving the solution to the problem.

61. What is the goal of Warshalls Algorithm?(L3)

Warshall's algorithm is to find the shortest distance between a node to all the other nodes in the graph.It Uses the Property of Transitive Closure i.e if there exist a path between (i,k) and (k,j) then there surely exist a path between (i,j)

(i,k) &(k,j)---□(I,J)What is the use of Floyds algorithm?(L3)

It is use to find the All pairs shortest Path of an Graph.