DAYANANDA SAGAR COLLEGE OF ENGINEERING

(An Autonomous Institution Affiliated to VTU, Belagavi)

SHAVIGE MALLESHWARA HILLS, K.S.LAYOUT, BANGALORE-560078

Department of Computer Science and Engineering (Cyber Security)

2023-2024

IV Semester

Analysis and Design of Algorithms Laboratory Manual

Compiled by,
Dr. Deepthi V.S.

ANALYSIS & DESIGN OF ALGORITHMS LABORATORY			
Course Code:22CYL45	**Credits: 01**	**L: P: T: S: 0: 0: 2: 0**	**Total Hours: 12**
CIE Marks: 50	**SEE Marks: 50**		**Exam Hours: 03**

Course objectives:

1. Apply using algorithms and measure its performance.
2. Measure algorithm’s efficiencies using asymptotic notations.
3. Implement various algorithm design methods for a given problem.

Sl. No.	**Programs**
1	Write and execute a program to sort a given set of elements using the Quick sort method and determine the time required to sort the elements. Repeat the experiment for different values of n, the number of elements in the list to be sorted and plot a graph of the time taken versus n. The elements should be read from a file/can be generated using the random number generator
2	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 non graph sheet. The elements can be read from a file or can be generated using the random number generator. Demonstrate using C how the divide-and-conquer method works along with its time complexity analysis: worst case, average case and best case.
3	Write and execute a C program to print all the nodes reachable from a given starting node in a graph using BFS and DFS methods.
4	Write and execute a C program to arrange nodes in topological order using source removal technique.
5	Implement in C, the Knapsack problem using Greedy method.
6	Find Minimum Cost Spanning Tree of a given connected undirected graph using Kruskal's algorithm.
7	Find Minimum Cost Spanning Tree of a given connected undirected graph using Prim's algorithm.
8	Write and execute a program to find shortest path to all other nodes in weighted graph using Dijkstra’s Strategy.
9	Given two sequences X = <x1; x2; : : : ; xm>, Y = <y1; y2; : : : ; yn>and required to find a longest-common-subsequence , of X and Y using dynamic programming.
10	Write and execute a program to find solution to n- queens problem.

Course Outcomes: After completion of the course, the graduates will be able to:

1. Analyze time complexity of algorithms using asymptotic notations
2. Apply various design techniques for the given problem
3. Implement various searching, sorting and graph traversal algorithms.

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.

Semester-End Examination:

- 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.
- All laboratory experiments are to be included for practical examination.
- 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.
- 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

Continuous Internal Evaluation: for the practical component

On completion of every experiment/program in the laboratory, the students shall be evaluated and marks shall, be awarded on the same day. The 30 marks are for conducting the experiment and preparation of the laboratory record is for 10 marks, the other 10 marks shall be for the test conducted at the end of the semester.

Typical Evaluation pattern for PCC course is shown in Table 1.

Component		**Max Marks**	**Total Marks**	**Min Marks**
CIE Lab	Practical(Conduction)	30	50	12
	Practical(Record)	10		-
	Practical(Test)	10		8
		Total CIE Practical	**50**	**20**
SEE	Semester End Examination	50	50	18
		CIE+SEE	**100**	**40**

1. **Write and execute a program to sort a given set of elements using the Quick sort method and determine the time required to sort the elements. Repeat the experiment for different values of n, the number of elements in the list to be sorted and plot a graph of the time taken versus n. The elements should be read from a file/can be generated using the random number generatorALGORITHM Quicksort(A[l..r])**

```
//Sorts a subarray by quicksort
//Input: Subarray of array A[0..n − 1], defined by its left and right indices l and r
//Output: Subarray A[l..r] sorted in nondecreasing order
if l< r
s ← Partition(A[l..r])
Quicksort(A[l..s-1])
Quicksort(A[s+1..r])
```

ALGORITHM Partition(A[l..r])

```
//Partitions a subarray by Hoare's algorithm, using the first element as a pivot
//Input: Subarray of array A[0..n − 1], defined by its left and right indices l and r (l < r)
//Output: 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 ] ≤ p
    swap(A[i], A[j ])
    until i ≥ j
    swap(A[i], A[j ]) //undo last swap when i ≥ j
    swap(A[l], A[j ])
    return j
```

PROGRAM

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

```
#include<stdlib.h>
#include<time.h>
int partition(int a[],int low,int high)
{
	int i,j,temp,key;
	key=a[low];
	i=low+1;
	j=high;
	while(1)
	{
		while(i<high && key>=a[i])
			i++;
		while(key<a[j])
			j--;
		if(i<j)
		{
			temp=a[i];
			a[i]=a[j];
			a[j]=temp;
		}
		else
		{
			temp=a[low];
			a[low]=a[j];
			a[j]=temp;
		}
		return j;
	}
}
void quicksort(int a[],int low,int high)
```

```
{
        int j;
        if(low<high)
        {
                j=partition (a,low,high);
                quicksort (a,low,j-1);
                quicksort(a,j+1,high);
        }
}
int main()
{
        int i ,n,a[100];
        time_t start,end;
        printf("enter the number of elements\n");
        scanf("%d",&n);
        for(i=0;i<n;i++)
   {
                a[i]=rand();
                printf("%d\t",a[i]) ;
        }
start=clock();
quicksort(a,0,n-1);
        end=clock();
printf("\nthe sorted array is \n");
        for(i=0;i<n;i++)
        {
                printf("%d\t",a[i]);
        }
        double tc=(difftime(end,start)/CLOCKS_PER_SEC);
printf("\ntime taken is %f",tc);
```

```
        return 0;
}
```

Output:

```
enter the number of elements
10
41      18467   6334    26500   19169   15724   11478   29358   26962   24464
the sorted array is
41      6334    11478   15724   18467   19169   26500   24464   26962   29358
time taken is 0.0002
```

2. **Write and execute a program to sort a given set of elements using the Merge sort method and determine the time required to sort the elements. Repeat the experiment for different values of n, the number of elements in the list to be sorted and plot a graph of the time taken versus n. The elements should be read from a file/can be generated using the random number generator**

```
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 copy A[0.. n/2 − 1] to B[0..n/2 − 1]
        copy A[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)
```

```
ALGORITHM Merge(B[0..p − 1], C[0..q − 1], A[0..p + q − 1])
//Merges two sorted arrays into one sorted array
//Input: Arrays B[0..p − 1] and C[0..q − 1] both sorted
//Output: Sorted array A[0..p + q − 1] of the elements of B and C
i ← 0; j ← 0; k ← 0
whilei< p and j < q
if B[i] ≤ C[j ]
A[k]← B[i]; i ← i + 1
else A[k]← C[j ]; j ← j + 1
k ← k + 1
if i = p
copy C[j..q − 1] to A[k..p + q − 1]
else copy B[i..p − 1] to A[k..p + q − 1]
```

PROGRAM

```
#include<stdio.h>
#include<stdlib.h>
#include<dos.h>
#include<time.h>
void merge(int arr[], int l, int m, int r)
{
        int i, j, k;
        int n1 = m - l + 1;
        int n2 = r - m;
        int L[n1], R[n2];
        for (i = 0; i < n1; i++)
        l[i] = arr[l + i];
        for (j = 0; j < n2; j++)
        r[j] = arr[m + 1+ j];
        i = 0;
        j = 0;
        k = l;
        while (i < n1 && j < n2)
        {
                if (L[i] <= R[j])
                {
                        arr[k] = L[i];
                        i++;
                }
              else
              {
                        arr[k] = R[j];
                        j++;
           }
```

```
                k++;
    }
    while (i < n1)
    {
                arr[k] = L[i];
                i++;
                k++;
    }
    while (j < n2)
    {
          arr[k] = R[j];
        j++;
        k++;
    }
}
void mergesort(int arr[], int l, int r)
{
    if (l < r)
    {
          int m = l+(r-l)/2;
          mergesort(arr, l, m);
          mergesort(arr, m+1, r);
          merge(arr, l, m, r);
    }
}
int main()
{
          int  i,n,a[100];
          time_t start,end;
          printf("enter the number of elements\n");
          scanf("%d",&n);
          for(i=0;i<n;i++)
{
                a[i]=rand();
                printf("%d\t",a[i]) ;
          }
          start=clock();
          mergesort(a,0,n-1);
          end=clock();
          printf("\nsorted array elements are\n");
          for(i=0;i<n;i++)
          {
                printf("%d\t",a[i]);
       }
       double tc=(difftime(end,start)/CLOCKS_PER_SEC);
       printf("\ntime taken is %f",tc);
       return 0;
}
```

Output:

```
enter the number of elements
10
41      18467   6334    26500   19169   15724   11478   29358   26962   24464
sorted array elements are
41      6334    11478   15724   18467   19169   24464   26500   26962   29358
time taken is 0.0002
```

3. **Write and execute a C program to print all the nodes reachable from a given starting node in a graph using BFS and DFS methods.**

DFS:

ALGORITHM dfs(G)

```
//Implements a depth-first search traversal of a given graph
//Input: Graph G = V,E
//Output: Graph G with its vertices marked with consecutive integers in the order they are first encountered
    by the DFS traversal mark each vertex in V with 0 as a mark of being "unvisited"
count ← 0
for each vertex v in V do
if v is marked with 0
dfs(v)
dfs(v)
//visits recursively all the unvisited vertices connected to vertex v by a path and numbers them in the order
     they
//are encountered via global variable count
count ← count + 1;
mark v with count
for each vertex w in V adjacent to v do
if w is marked with 0
dfs(w)
```

PROGRAM

```
#include<stdio.h>
int a[20][20],reach[20],n;
void dfs(int v)
{
        int i;
        reach[v]=1;
        for (i=1;i<=n;i++)
        if(a[v][i] && !reach[i])
        {
                printf("\n %d->%d",v,i);
                dfs(i);
        }
}
int main()
{
        int i,j,count=0,s;
        printf("\n Enter number of vertices:");
        scanf("%d",&n);
        for (i=1;i<=n;i++)
        {
                reach[i]=0;
         }
        printf("\n Enter the adjacency matrix:\n");
        for (i=1;i<=n;i++)
         for (j=1;j<=n;j++)
         scanf("%d",&a[i][j]);
printf("Enter the source vertex\n");
        scanf("%d",&s);
        dfs(s);
        printf("\n");
        for (i=1;i<=n;i++)
        {
        if(reach[i])
         count++;
        }
        if(count==n)
        printf("\n Graph is connected"); else
        printf("\n Graph is not connected");
```

```
return 0;
}
```

Output:

```
Enter number of vertices:4
Enter the adjacency matrix:
1 1 1 1
0 1 0 0
0 0 1 0
0 0 0 1
Enter the source vertex: 1
 1->2
 1->3
 1->4
Graph is connected
```

BFS:

ALGORITHM bfs(G)

```
//Implements a breadth-first search traversal of a given graph
//Input: Graph G = V,E
//Output: Graph G with its vertices marked with consecutive integers
// in the order they are visited by the bfs traversal mark each vertex in V with 0 as a mark of being "unvisited"
count ← 0
for each vertex v in V do
if v is marked with 0
bfs(v)
bfs(v)
count ← count + 1; mark v with count and initialize a queue with v
while the queue is not empty do
for each vertex w in V adjacent to the front vertex do
if w is marked with 0
count ← count + 1; mark w with count
add w to the queue
remove the front vertex from the queue
```

PROGRAM

```
#include<stdio.h>
int v[10];
void bfs(int n,int a[][10],int s)
{
        int i,q[10],u;
        int f=0,r=-1;
        v[s]=1;
        q[++r]=s;
        while(f<=r)
        {
                u=q[f++];
                for(i=1;i<=n;i++)
                        if(a[u][i]==1&&v[i]==0)
                        {
                                q[++r]=i;
                                v[i]=1;
                        }
        }
}
int  main()
{
        int n,a[10][10],i,j,s;
        printf("enter the no of node\n");
        scanf("%d",&n);
        printf("enter the adjacency matrix\n");
        for(i=1;i<=n;i++)
        {
                for(j=1;j<=n;j++)
                {
                        scanf("%d",&a[i][j]);
                }
        }
        printf("enter the source\n");
        scanf("%d",&s);
        for(i=1;i<=n;i++)
        {
                v[i]=0;
        }
        bfs(n,a,s);
```

```
	for(i=1;i<=n;i++)
	{
		if(v[i]==0)
			printf("\n the node %d is not reachable\n",i);
		else
			printf("\n the node %d is reachable\n",i);
	}
	return 0;
}
```

Output:

```
enter the no of node
4
enter the adjacency matrix
1 1 1 1
0 1 0 0
0 0 1 0
0 0 0 1
enter the source
1
the node 1 is reachable
the node 2 is reachable
the node 3 is reachable
the node 4 is reachable
```

4. Write and execute a C program to arrange nodes in topological order using source removal technique.

ALGORITHM: Steps involved in finding the topological ordering of a DAG:

```
Step-1: Compute in-degree (number of incoming edges) for each of the vertex present in the DAG and
initialize the count of visited nodes as 0.
Step-2: Pick all the vertices with in-degree as 0 and add them into a queue (Enqueue operation)
Step-3: Remove a vertex from the queue (Dequeue operation) and then.
        Increment the count of visited nodes by 1.
        Decrease in-degree by 1 for all its neighboring nodes.
        If the in-degree of neighboring nodes is reduced to zero, then add it to the queue.
Step 4: Repeat Step 3 until the queue is empty.
Step 5: If the count of visited nodes is not equal to the number of nodes in the graph then the topological
sort is not possible for the given graph.
```

PROGRAM

```
#include <stdio.h>
#include <stdbool.h>
#define MAX_NODES 100
typedef struct Node
{
        int id;
        int dependencies[MAX_NODES];
        int numDependencies;
        bool visited;
} Node;
Node nodes[MAX_NODES];
int numNodes;
void initializeNodes()
{
        int i;
        for (i = 0; i < MAX_NODES; i++)
        {
                nodes[i].id = i;
                nodes[i].numDependencies = 0;
                nodes[i].visited = false;
         }
        numNodes = 0;
}
void addDependency(int from, int to)
{
        nodes[to].dependencies[nodes[to].numDependencies] = from;
        nodes[to].numDependencies++;
}
void topologicalSort()
{
        int i, j;
        int numVisited = 0;
        bool noDependencies;
        while (numVisited < numNodes)
        {
                noDependencies = true;
                for (i = 0; i < numNodes; i++)
                {
                        if (!nodes[i].visited)
                        {
                        for (j = 0; j < nodes[i].numDependencies; j++)
                                {
                                if (!nodes[nodes[i].dependencies[j]].visited)
                                        {
                                        noDependencies = false;
                                        break;
                                }
```

```
                    }
                    if (noDependencies)
                        {
                        printf("%d ", nodes[i].id);
                        nodes[i].visited = true;
                        numVisited++;
                        break;
                     }
                    }
                }
            }
}

int main()
{
        initializeNodes();
        addDependency(1, 3);
        addDependency(2, 3);
        addDependency(3, 4);
        addDependency(3, 5);
        addDependency(4, 6);
        addDependency(5, 6);
        numNodes = 7;
        printf("Topological Order: ");
        topologicalSort();
        return 0;
}
```

OUTPUT:

```
Topological Order: 0 1 2 3 4 5 6
```

5. **Implement in C, the Knapsack problem using Greedy method.**

KNAPSACK PROBLEM

Given a set of items, each with a weight and a value, determine a subset of items to include in a collection so that the total weight is less than or equal to a given limit and the total value is as large as possible.

The knapsack problem is in combinatorial optimization problem. It appears as a subproblem in many, more complex mathematical models of real-world problems. One general approach to difficult problems is to identify the most restrictive constraint, ignore the others, solve a knapsack problem, and somehow adjust the solution to satisfy the ignored constraints.

APPLICATIONS

In many cases of resource allocation along with some constraint, the problem can be derived in a similar way of Knapsack problem. Following is a set of example.

- Finding the least wasteful way to cut raw materials
- portfolio optimization
- Cutting stock problems

FRACTIONAL KNAPSACK

In this case, items can be broken into smaller pieces, hence the thief can select fractions of items.

Steps

- Calculate value per weight for each item (we can call this value density)
- Sort the items as per the value density in descending order
- Take as much item as possible not already taken in the knapsack

SOURCE CODE

```
# include<stdio.h>
void knapsack(int n,float weight[],float profit[],float capacity)
{
float x[20],tp=0;
int i, j, u;
```

```
 u=capacity;
for(i=0;i<n;i++)
     x[i]=0.0;
for(i=0;i<n;i++)
{
if(weight[i]>u)
break;
else
{
     x[i]=1.0;
     tp=tp+profit[i];
     u=u-weight[i];
}
}

if(i<n)
       x[i]=u/weight[i];
tp=tp+(x[i]*profit[i]);
printf("\n The result vector is:- ");
for(i=0;i<n;i++)
printf("%ft",x[i]);
printf("m Maximum profit is:- %f",tp);
}
int main()
{
float weight[20], profit[20], capacity;
int n, i ,j;
float ratio[20], temp;
printf("n Enter the no. of objects:- ");
scanf("%d",&n);
printf("n Enter the wts and profits of each object:- ");
```

```
for(i=0; i<n; i++)
{
scanf("%f %f",&weight[i],&profit[i]);
}
printf("n enter the capacity of knapsack:- ");
scanf("%f",&capacity);
for(i=0; i<n; i++)
{
 ratio[i]=profit[i]/weight[i];
}
for(i=0; i<n; i++)
{
for(j=i+1;j< n;j++)
{
if(ratio[i]<ratio[j])
{
      temp= ratio[j];
      ratio[j]= ratio[i];
      ratio[i]= temp;

     temp= weight[j];
     weight[j]= weight[i];
     weight[i]= temp;

     temp= profit[j];
     profit[j]= profit[i];
     profit[i]= temp;
}
}
}
```

```
knapsack(n, weight, profit, capacity);
}
```

OUTPUT

```
Enter the no. of objects:-7
Enter the wts and profits of each object:-
2  10
3  5
5  15
7  7
1  6
4  18
1  3
Enter the capacity of knapsack:-15

The result vector is:-1.000000   1.000000    1.000000    1.000000
1.000000      0.666667       0.000000

Maximum profit is:-55.333332
```

6. Find Minimum Cost Spanning Tree of a given connected undirected graph using Kruskal's algorithm.

```
//Kruskal's algorithm for constructing a minimum spanning tree
//Input: A weighted connected graph G = V,E
//Output: $E_T$ , the set of edges composing a minimum spanning tree of G sort E in non-decreasing order of the edge weights $w(e_{i1}) \leq \ldots \leq w(e_{i|E|})$
$E_T \leftarrow \emptyset$; ecounter $\leftarrow 0$ //initialize the set of tree edges and its size
k $\leftarrow 0$ //initialize the number of processed edges
while ecounter $< |V| - 1$ do
k $\leftarrow$ k + 1
if ET $\cup \{e_{ik}\}$ is acyclic
$E_T \leftarrow E_T \cup \{eik\}$; ecounter $\leftarrow$ ecounter + 1
Return $E_T$
```

PROGRAM

```
#include <stdio.h>
#include <stdlib.h>
int comparator(const void* p1, const void* p2)
{
        const int(*x)[3] = p1;
        const int(*y)[3] = p2;
        return (*x)[2] - (*y)[2];
}
void makeSet(int parent[], int rank[], int n)
{
        for (int i = 0; i < n; i++)
        {
                parent[i] = i;
                rank[i] = 0;
        }
}
int findParent(int parent[], int component)
{
        if (parent[component] == component)
                return component;
        return parent[component]= findParent(parent, parent[component]);
}
void unionSet(int u, int v, int parent[], int rank[], int n)
{
        u = findParent(parent, u);
        v = findParent(parent, v);
        if (rank[u] < rank[v])
        {
                parent[u] = v;
        }
        else if (rank[u] > rank[v])
        {
                parent[v] = u;
        }
        else
        {
                parent[v] = u;
                rank[u]++;
        }
}
void kruskalAlgo(int n, int edge[n][3])
{
        qsort(edge, n, sizeof(edge[0]), comparator);
        int parent[n];
        int rank[n];
        makeSet(parent, rank, n);
        int minCost = 0;
        printf("Following are the edges in the constructed MST\n");
        for (int i = 0; i < n; i++)
        {
                int v1 = findParent(parent, edge[i][0]);
                int v2 = findParent(parent, edge[i][1]);
                int wt = edge[i][2];
                if (v1 != v2)
                {
                        unionSet(v1, v2, parent, rank, n);
                        minCost += wt;
                        printf("%d -- %d = %d\n", edge[i][0], edge[i][1], wt);
                }
        }
        printf("Minimum Cost Spanning Tree: %d\n", minCost);
}
int main()
{
        int edge[5][3] = { { 0, 1, 10 },
                        { 0, 2, 6 },
                        { 0, 3, 5 },
                        { 1, 3, 15 },
                        { 2, 3, 4 } };
        kruskalAlgo(5, edge);
        return 0;
}
```

OUTPUT:

```
Following are the edges in the constructed MST
```

```
2 -- 3 = 4

0 -- 3 = 5

0 -- 1 = 10

Minimum Cost Spanning Tree: 19
```

7. Find Minimum Cost Spanning Tree of a given connected undirected graph using Prim's algorithm.

```
//Prim’s algorithm for constructing a minimum spanning tree
//Input: A weighted connected graph G = V,E
//Output: $E_T$ , the set of edges composing a minimum spanning tree of G
$V_T \leftarrow \{v0\}$ //the set of tree vertices can be initialized with any vertex
$E_T \leftarrow \emptyset$
for i ← 1 to |V | − 1 do
find a minimum-weight edge $e* = (v*, u*)$ among all the edges (v, u) such that v is in $V_T$ and u is in $V - V_T$
        $V_T \leftarrow V_T \cup \{u*\}$
        $E_T \leftarrow E_T \cup \{e*\}$
return$E_T$
```

PROGRAM

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

#define MAX_V 100 // Define a maximum number of vertices

// Function to find the vertex with the minimum key value, from the set of vertices not yet included in the MST
int minKey(int key[], bool mstSet[], int V) {
   int min = INT_MAX, min_index;
   for (int v = 0; v < V; v++)
      if (mstSet[v] == false && key[v] < min)
         min = key[v], min_index = v;
   return min_index;
}
```

```
// Function to print the constructed MST stored in parent[]
void printMST(int parent[], int graph[MAX_V][MAX_V], int V) {
    printf("Edge \tWeight\n");
    for (int i = 1; i < V; i++)
        printf("%d - %d \t%d \n", parent[i], i, graph[i][parent[i]]);
}

// Function to construct and print MST for a graph represented using adjacency matrix representation
void primMST(int graph[MAX_V][MAX_V], int V) {
    int parent[MAX_V]; // Array to store constructed MST
    int key[MAX_V]; // Key values used to pick minimum weight edge in cut
    bool mstSet[MAX_V]; // To represent set of vertices included in MST

    // Initialize all keys as INFINITE
    for (int i = 0; i < V; i++)
        key[i] = INT_MAX, mstSet[i] = false;

    // Always include the first vertex in MST.
    key[0] = 0; // Make key 0 so that this vertex is picked as first vertex
    parent[0] = -1; // First node is always root of MST

    // The MST will have V vertices
    for (int count = 0; count < V - 1; count++) {
        // Pick the minimum key vertex from the set of vertices not yet included in MST
        int u = minKey(key, mstSet, V);

        // Add the picked vertex to the MST Set
        mstSet[u] = true;
        // Update key value and parent index of the adjacent vertices of the picked vertex.
        // Consider only those vertices which are not yet included in MST
        for (int v = 0; v < V; v++)
            // graph[u][v] is non-zero only for adjacent vertices of u
            // mstSet[v] is false for vertices not yet included in MST
            // Update the key only if graph[u][v] is smaller than key[v]
            if (graph[u][v] && mstSet[v] == false && graph[u][v] < key[v])
                parent[v] = u, key[v] = graph[u][v];
    }

    // Print the constructed MST
    printMST(parent, graph, V);
}

int main() {
    int V;
    int graph[MAX_V][MAX_V];

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

    printf("Enter the adjacency matrix:\n");
    for (int i = 0; i < V; i++) {
        for (int j = 0; j < V; j++) {
            scanf("%d", &graph[i][j]);
        }
    }

    // Print the solution
    primMST(graph, V);
    return 0;
}
```

OUTPUT:

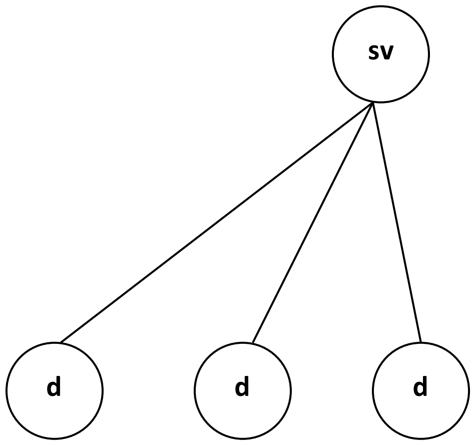
```
Enter the number of vertices: 6
Enter the adjacency matrix:
0 4 4 999 999 999
4 0 2 999 999 999
4 2 0 3 2 4
999 999 3 0 999 3
999 999 2 999 0 3
999 999 4 3 3 0
Minimum Spanning Tree:
Edge    Weight
0 - 1   4
1 - 2   2
2 - 3   3
2 - 4   2
4 - 5   3
```

8. **Write and execute a program to find shortest path to all other nodes in weighted graph using Dijkstra's Strategy.**

Consider you have a graph. Display the edges which constitutes shortest path (i,j) from the given single source to all the vertices/nodes. Here i is source whose value is constant (Given by user) and j is destination whose value ranges from 1 to n.

DIJKSTRA'S ALGORITHM

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.

ALGORITHM

Algorithm Dijkstra(n,Cost,Source,d)

//To compute the shortest distance from source to destination.

```
//Input:   n, cost, source vertex.
//Output: shortest distance from source to all other nodes.
S1. Initialization.
For(i=0 to n-1) do
            D[i]=cost[source][i];
S2. Add source to visited array (S)
       S[source]=1
S3. Find the shortest distance
For(i=0 to n-1) do
 Find u and d[u] such that d[u] is minimum and u € (V-S)
   Add u to S
For(every v€ V-S ) do
If(d[u]+cost[u][v]<d[v])
           d[v]=d[u]+cost[u][v]
endif
endfor
endfor
S4. return
```

DIJKSTRA'S ALGORITHM COMPLEXITY

Time Complexity: O(E Log V)

where, E is the number of edges and V is the number of vertices.

Space Complexity: O(V)

DIJKSTRA'S ALGORITHM APPLICATIONS

- To find the shortest path
- In social networking applications
- In a telephone network

- To find the locations in the map

PROGRAM

```
#include<stdio.h>
int minimum(int,int);
int main()
{
        int cost[20][20],s[20],d[20];
        int source,n,mini,u;
        int i,j,v;
        printf("\n\n\t Dijkstra's Algorithm \nEnter the number of verticies : ");
        scanf("%d",&n);
        printf("Enter the weights of the graph\n");
        printf("If no connection enter 999 and for self loop enter 0\n");
        for(i=1;i<=n;i++)
                for(j=1;j<=n;j++)
                        scanf("%d",&cost[i][j]);
        printf("Enter the source node : ");
        scanf("%d",&source);
for(i=1;i<=n;i++)
{
        s[i]=0;
        d[i]=cost[source][i];
}
s[source]=1;
for(i=1;i<=n-1;i++)
{
        mini=999;
        u=0;
        for(j=1;j<=n;j++)
                if(d[j]<mini && s[j]==0)
                 {
                        mini=d[j];
                        u=j;
         }
        s[u] = 1;
        for(v=1;v<=n;v++)
                if(s[v]==0)
                        d[v]=minimum(d[v],d[u]+cost[u][v]);
}
for(i=1;i<=n;i++)
        printf("Shortest Path From %d to %d is = %d\n",source,i,d[i]);
}
int minimum(inta,int b)
{
return((a<b)?a:b);
}
```

OUTPUT

```
     Dijkstra's Algorithm


Enter the number of verticies : 4

Enter the weights of the graph

If no connection enter 999 and for self loop enter 0

1 0  9  3

0 0 999 2

0 999 3 4

999 1 2 3

Enter the source node : 1

Shortest Path From 1 to 1 is = 1

Shortest Path From 1 to 2 is = 0

Shortest Path From 1 to 3 is = 4

Shortest Path From 1 to 4 is = 2
```

9. **Given two sequences X = <x1; x2; : : : ; xm>, Y = <y1; y2; : : : ; yn>and required to find a longest-common-subsequence , of X and Y using dynamic programming.**

ALGORITHM:

```
X and Y be two given sequences
Initialize a table LCS of dimension X.length * Y.length
X.label = X
Y.label = Y
LCS[0][] = 0
LCS[][0] = 0
Start from LCS[1][1]
Compare X[i] and Y[j]
if X[i] = Y[j]
  LCS[i][j] = 1 + LCS[i-1, j-1]
 Point an arrow to LCS[i][j]
else
LCS[i][j] = max(LCS[i-1][j], LCS[i][j-1])
 Point an arrow to max(LCS[i-1][j], LCS[i][j-1])
```

PROGRAM

```
#include <stdio.h>
#include <string.h>
int i, j, m, n, LCS_table[20][20];
char S1[20] = "ACADB", S2[20] = "CBDA", b[20][20];
void lcsAlgo()
{
        m = strlen(S1);
```

```
n = strlen(S2);
for (i = 0; i <= m; i++)
LCS_table[i][0] = 0;
for (i = 0; i <= n; i++)
LCS_table[0][i] = 0;
for (i = 1; i <= m; i++)
        for (j = 1; j <= n; j++)
        {
        if (S1[i - 1] == S2[j - 1])
        {
        LCS_table[i][j] = LCS_table[i - 1][j - 1] + 1;
        }
        else if (LCS_table[i - 1][j] >= LCS_table[i][j - 1])
        {
                LCS_table[i][j] = LCS_table[i - 1][j];
        }
        else
        {
        LCS_table[i][j] = LCS_table[i][j - 1];
        }
        }
int index = LCS_table[m][n];
char lcsAlgo[index + 1];
lcsAlgo[index] = '\0';
int i = m, j = n;
while (i > 0 && j > 0)
{
if (S1[i - 1] == S2[j - 1])
```

```
                    {
                    lcsAlgo[index - 1] = S1[i - 1];
                    i--;
                    j--;
                    index--;
                    }
                    else if (LCS_table[i - 1][j] > LCS_table[i][j - 1])
          i--;
          else
          j--;
          }
    printf("S1 : %s \nS2 : %s \n", S1, S2);
    printf("LCS: %s", lcsAlgo);
}
int main()
{
lcsAlgo();
printf("\n");
}
```

OUTPUT:

```
S1 : ACADB
S2 : CBDA
LCS: CB
```

10. Write and execute a program to find solution to n- queens problem.

ALGORITHM

```
function solveNQueens(board, col, n):
    if col >= n:
        print board
        return true
    for row from 0 to n-1:
        if isSafe(board, row, col, n):
            board[row][col] = 1
        if solveNQueens(board, col+1, n):
            return true
        board[row][col] = 0
        return false
function isSafe(board, row, col, n):
    for i from 0 to col-1:
        if board[row][i] == 1:
            return false
    for i,j from row-1, col-1 to 0, 0 by -1:
         if board[i][j] == 1:
            return false
    for i,j from row+1, col-1 to n-1, 0 by 1, -1:
        if board[i][j] == 1:
            return false
        return true
board = empty NxN chessboard
solveNQueens(board, 0, N)
```

ALGORITHM Backtrack(X[1..i])

```
//Gives a template of a generic backtracking algorithm
//Input: X[1..i] specifies first i promising components of a solution
//Output: All the tuples representing the problem's solutions
if X[1..i] is a solution
```

```
        write X[1..i]
else
        for each element x ∈ Si+1 consistent with X[1..i] and the constraints do
        X[i + 1]← x
        Backtrack(X[1..i + 1])
```

PROGRAM

```
#include<stdio.h>
#include<math.h>
int a[30],count=0;
int place(int pos)
{
        int i;
        for (i=1;i<pos;i++)
        {
                if((a[i]==a[pos])||((abs(a[i]-a[pos])==abs(i-pos))))
                        return 0;
        }
        return 1;
}
void print_sol(int n)
{
        int i,j;
        count++;
        printf("\n\nSolution #%d:\n",count);
        for (i=1;i<=n;i++)
        {
                for (j=1;j<=n;j++)
                {
                        if(a[i]==j)
                                printf("Q\t"); else
                                printf("*\t");
                }
                printf("\n");
        }
}
void queen(int n)
{
        int k=1;
        a[k]=0;
        while(k!=0)
        {
                a[k]=a[k]+1;
                while((a[k]<=n)&&!place(k))
                        a[k]++;
                if(a[k]<=n)
                {
                        if(k==n)
                           print_sol(n);
                        else
                        {
                                k++;
                                a[k]=0;
                        }
                }
                else
                k--;
        }
}
int main()
{
        int n;
        printf("Enter the number of Queens\n");
        scanf("%d",&n);
        queen(n);
        printf("\nTotal solutions=%d",count);
        return 0;
}
```

OUTPUT:

```
Enter the number of Queens
4
Solution #1:
*    Q    *    *
*    *    *    Q
Q    *    *    *
*    *    Q    *
Solution #2:
*    *    Q    *
Q    *    *    *
*    *    *    Q
*    Q    *    *
Total solutions=2
```

VIVA QUESTIONS

1. What is an 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 of time

2. What are important problem types?

1. Sorting 2. Searching 3. Numerical problems 4. Geometric problems5. Combinatorial Problems 6. Graph Problems 7. String processing Problems

3. Name some basic Efficiency classes

1. Constant 2. Logarithmic 3. Linear 4. nlogn5. Quadratic 6. Cubic 7. Exponential 8. Factorial

4. What are algorithm design techniques?

Algorithm design techniques (or strategies or paradigms) are general approaches to solving problems algorithmically, applicable to a variety of problems from different areas of computing. General design techniques are:
(i) Brute force (ii) divide and conquer (iii) decrease and conquer (iv) transform and concquer (v) greedy technique (vi) dynamic programming(vii) backtracking (viii) branch and bound

5. **How is an algorithm's time efficiency measured?**

Time efficiency indicates how fast the algorithm runs. An algorithm's time efficiency is measured as a function of its input size by counting the number of times its basic operation (running time) is executed. Basic operation is the most time consuming operation in the algorithm's innermost loop.

6. How is the efficiency of the algorithm defined?

The efficiency of an algorithm is defined with the components.(i) Time efficiency -indicates how fast the algorithm runs(ii) Space efficiency -indicates how much extra memory the algorithm needs

7. What are the characteristics of an algorithm?

Every algorithm should have the following five characteristics (i) Input(ii) Output(iii) Definiteness (iv) Effectiveness(v) Termination.Therefore, an algorithm can be defined as a sequence of definite and effective instructions, which terminates with the production of correct output from the given input.

8. Write general plan for analyzing non-recursive algorithms.
i. Decide on parameter indicating an input's size.ii. Identify the algorithm's basic operationiii. Checking the no.of times basic operation executed depends on size of input.iv. Set up sum expressing the no.of times the basic operation is executed. Depends on some additional property,then best,worst,avg cases need to be investigated (establishing order of growth)

9. Define the terms: pseudocode, flow chart
A pseudocode is a mixture of a natural language and programming language like constructs. A pseudocode is usually more precise than natural language. A flowchart is a method of expressing an algorithm by a collection of connected geometric shapes containing descriptions of the algorithm's steps.

10. Write general plan for analyzing recursive algorithms.
i. Decide on parameter indicating an input's size.ii. Identify the algorithm's basic operationiii. Checking the no.of times basic operation executed depends on size of input.if it depends on some additional property,then best,worst, avg cases need to be investigatedof times the basic operation is executed.iv. Set up the recurrence relation,with an appropriate initial condition,for the numberv. Solve recurrence (establishing order of growth)

11. Define the divide and conquer method.
Given a function to compute on 'n' inputs the divide-and-conquer strategy suggests splitting the inputs in to'k' distinct subsets, 1<k <n, yielding 'k' sub problems. The sub problems must be solved recursively, and then a method must be found to combine sub solutions into a solution of the whole.

12. What is Merge sort?

Merge sort is divide and conquer strategy that works by dividing an input array in to two halves,sorting them recursively and then merging the two sorted halves to get the original array sorted.

13. What is general divide and conquer recurrence?

Time efficiency T(n)of many divide and conquer algorithms satisfies the equation $T(n)=a*T(n/b)+f(n)$.This is the general recurrence relation.

14. Explain the greedy method.

Greedy method is the most important design technique, which makes a choice that looks best at that moment. A given 'n' inputs are required us to obtain a subset that satisfies some constraints that is the feasible solution. A greedy method suggests that one can device an algorithm that works in stages considering one input at a time.

15. Define feasible and optimal solution.

Given n inputs and we are required to form a subset such that it satisfies some given constraints then such a subset is called feasible solution.A feasible solution either maximizes or minimizes the given objective function is called as optimal solution

16. What are the constraints of knapsack problem?

To maximize $\sum p_i x_i$The constraint is : $\sum w_i x_i \geq m$ and $0 \leq xi \leq 1$ $1\leq i \leq n$

Where m is the bag capacity, n is the number of objects and for each object i $w_i$and p_i are the weight and profit of object respectively.

17. Specify the algorithms used for constructing Minimum cost spanning tree.

a) Prim's Algorithm

b) Kruskal's Algorithm

18. **State single source shortest path algorithm (Dijkstra's algorithm).**

For a given vertex called the source in a weighted connected graph,find shortest paths to all its other vertices.Dijikstra's algorithm applies to graph with non-negative weights only.

19. State efficiency of prim's algorithm.

O($|v|^2$) (weight matrix and priority queue as unordered array)

O(|E| LOG|V|) (adjacency list and priority queue as min-heap)

20. State Kruskal Algorithm.

The algorithm looks at a MST for a weighted connected graph as an acyclic sub graph with |v|-1 edges for which the sum of edge weights is the smallest.

21. State efficiency of Dijkstra's algorithm.

O($|v|^2$) (weight matrix and priority queue as unordered array)

O(|e| log|v|) (adjacency list and priority queue as min-heap)

22. Define multistage graph

A multistage graph G =(V,E) is a directed graph in which the vertices are partitioned into K>=2 disjoint sets Vi,1<=i<=k.The multi stage graph problem is to find a minimum cost paths from s(source) to t(sink)Two approach(forward and backward)

23. Define all pair shortest path problem.

Given a weighted connected graph, all pair shortest path problem asks to find the lengths of shortest paths from each vertex to all other vertices.

24.Define Floyd's algorithm

To find all pair shortest path

25. State the time efficiency of Floyd's algorithm

O(n^3) It is cubic

26. What is the time complexity of linear search?

$\Theta(n)$

27. What is the time complexity of binary search?

$\Theta(\log_2 n)$

28. What is the major requirement for binary search?

The given list should be sorted.

29. What is binary search?

It is an efficient method of finding out a required item from a given list, provided the list is in order.

The process is:1. First the middle item of the sorted list is found.2. Compare the item with this element. 3. If they are equal search is complete.4. If the middle element is greater than the item being searched, this process is repeated in the upper half of the list.5. If the middle element is lesser than the item being searched, this process is repeated in the lower half of the list.

30. What is parental dominance?

The key at each node is greater than or equal to the keys at its children.

31. What is heap?

A **heap** can be defined as a binary tree with keys assigned to its nodes (one key per node) provided the following two conditions are met:1 The tree's shape requirement – The binary tree is essentially complete (or simply complete), that is, all its levels are full except possibly the last level, where only some rightmost leaves may be missing. 2. The parental dominance requirement – The key at each node is greater than or equal to the keys at its children

32. What is height of Binary tree?

It is the longest path from the root to any leaf.

33. What is the time complexity of heap sort?

$\Theta(n\log n)$

34. What is Merge Sort?

Merge sort is an O (n log n) comparison-based sorting algorithm. Where the given array is divided into two equal parts. The left part of the array as well as the right part of the array is sorted recursively. Later, both the left sorted part and right sorted part are merged into a single sorted array.

35. Who invented Merge Sort?

John Von Neumann in 1945.

36. On which paradigm is it based?

Merge-sort is based on the divide-and-conquer paradigm.

37. What is the time complexity of merge sort?

n log n.

38. What is the space requirement of merge sort?

Space requirement: $\Theta(n)$ (not in-place).

39. When do you say an algorithm is stable and in place?

Stable – if it retains the relative ordering.

In– place if it does not use extra memory.

40. Who invented Dijkstra's Algorithm?

Edsger Dijkstra invented this algorithm.

41. What is the other name for Dijkstra's Algorithm?

Single Source Shortest Path Algorithm.

42. Which technique the Dijkstra's algorithm is based on?

Greedy Technique.

43. What is the purpose of Dijkstra’s Algorithm?

To find the shortest path from source vertex to all other remaining vertices

44. Name the different algorithms based on Greedy technique.

Prim’s Algorithm, Kruskal’s algorithm

45. What is the constraint on Dijkstra’s Algorithm?

This algorithm is applicable to graphs with non-negative weights only.

46. What is a Weighted Graph?

A weighted graph is a graph with numbers assigned to its edges. These numbers are called weights orcosts.

47. What is a Connected Graph?

A graph is said to be connected if for every pair of its vertices u and v there is a path from u to v.

48.What is the time complexity of Dijkstra’s algorithm?

For adjacency matrix, It is O(IVI*IVI)For adjacency list with edges stored as min heap, it is O(|E|logIVI)

49. Differentiate b/w Traveling Salesman Problem (TSP) and Dijkstra’s **Algorithm.**

In TSP, given n cities with known distances b/w each pair, find the shortest tour that passes through all the cities exactly once before returning to the starting city.In Dijkstra’s Algorithm, find the shortest path from source vertex to all other remaining vertices

50. Differentiate b/w Prim’s Algorithm and Dijkstra’s Algorithm?

Prim’s algorithm is to find minimum cost spanning tree.Dijkstra’s algorithm is to find the shortest path from source vertex to all other remaining vertices.

51.What are the applications of Dijkstra’s Algorithm?

It finds its application even in our day to day life while travelling.They are helpful in routing applications.

52. Who was the inventor of the Quicksort?
C.A.R.HOARE, a British Scientist.

53. Which design strategy does Quicksort uses?
Divide and Conquer

54. What is Divide and Conquer Technique?
(I) Divide the instance of a problem into two or more smaller instances (II) Solve the smaller instances recursively (III) Obtain solution to original instances by combining these solutions

55. What is another name for Quicksort?
Partition Exchange Sort

56.Is Quicksort Stable as well as In place?
Not Stable but In place.

57. When do you say that a Quick sort having best case complexity?
When the pivot exactly divides the array into equal half

58.When do you say that Quick sort having worst case complexity?
When any one of the subarray is empty

59. How many more comparisons are needed for average case compared to best case?
38% more

60.When do you say that the pivot element is in its final position?
When all the elements towards the left of pivot are <= pivot and all the elements towards the right

of pivot are >= pivot.

61.What technique does kruskal's algorithm follow?
Greedy technique

62. What is the time complexity of Kruskalalgorithm.
With an efficient sorting algorithm, the time efficiency of an kruskal's algorithm will be in O(|E|log|E|).

63.Who is the inventor of kruskal's algorithm?
Joseph Kruskal.

64. Which technique is used to solve BFS & DFS problems?
Decrease and Conquer

65. What is DFS traversal?
Consider an arbitrary vertex v and mark it as visited on each iteration, proceed to an unvisited vertex w adjacent to v. we can explore this new vertex depending upon its adjacent information. This process continues until dead end is encountered.(no adjacent unvisited vertex is available). At the dead end the algorithm backs up by one edge and continues the process of visiting unvisited vertices. This process continues until we reach the starting vertex.

66. What are the various applications of BFS & DFS tree traversal technique?
Application ofBFS
Checking connectivity and checking acyclicity of a graph.
Check whether there is only one root in the BFS forest or not.
If there is only one root in the BFS forest, then it is connected graph otherwise disconnected graph.
For checking cycle presence, we can take advantage of the graph's representation in the form of a BFS forest, if the latter vertex does not have cross edge, then the graph is acyclic.
Application ofDFS
To check whether given graph is connected or not. To check whether the graph is cyclic or

not.To find the spanning tree.
Topological sorting.

67. Which data structures are used in BFS & DFS tree traversal technique?
We use Stack data structure to traverse the graph in DFS traversal.
We use queue data structure to traverse the graph in BFS traversal.

68. What is Dynamic Programming?
It is a technique for solving problems with overlapping sub problems.

69. What does Dynamic Programming have in common with Divide and Conquer?
Both solve the problems by dividing the problem into sub problems. Using the solutions of sub problems, the solutions for larger instance of the problem can be obtained.

70. What is a principle difference between the two techniques?
Only one instance of the sub problem is computed & stored. If the same instance of sub problem is encountered, the solution is retrieved from the table and never recomputed. Very precisely re - computations are not preformed.

71. What is a spanning tree?
A spanning tree of a weighted connected graph is a connected acyclic (no cycles) sub graph (i.e. a tree)that contains all the vertices of a graph and number of edges is one less than number of vertices.

72. What is a minimum spanning tree?
Minimum spanning tree of a connected graph is its spanning tree of smallestweight.

73. Prim’s algorithm is based on which technique.
Greedy technique

74. Name some of the application greedy technique

They are helpful in routing applications: In the case of network where large number of computers are connected, to pass the data from one node to another node we can find the shortest distance easily by this algorithm. Communication Networks: Suppose there are 5 different cities and we want to establish computer connectivity among them, then we take into the account of cost factor for communication links. Building a road network that joins n cities with minimum cost.

75.Does Prim's Algorithm always yield a minimum spanning tree ?

Yes

76.What is thepurpose of Floyd's algorithm?

To find the all pair shortest path.

77. What is the time efficiency of Floyd's algorithm?

It is same as warshall's algorithm that is $\theta(n^3)$.

78. What is shortest path?

It is the path which is having shortest length among all possible paths.

79.Warshall's algorithm is optimization problem or non-optimization problem ?

Non-optimization problem

80.For what purpose we use Warshall's algorithm?

Warshall's algorithm for computing the transitive closure of a directed graph

81.Warshall's algorithm depends on which property?

Transitive property

82.What is the time complexity of Warshall's algorithm?

Time complexity of warshall's algorithm is $\theta(n^3)$.

83.What is state space tree?

Constructing a tree of choices being made and processing is known as state-spacetree. Root represents an initial state before the search for solution begins.

84.What are promising and non-promising state-space-tree?

Node which may leads to solution is called promising. Node which doesn’t leads to solution is called non-promising.

85. What are the requirements that are needed for performing Backtracking?

Complex set of constraints. They are: i. Explicit constraints. ii. Implicit constraints.