ವಿಶ್ವೇಶ್ವರಯ್ಯ ತಾಂತ್ರಿಕ ವಿಶ್ವವಿದ್ಯಾಲಯ, ಬೆಳಗಾವಿ
VISVESVARAYA TECHNOLOGICAL UNIVERSITY - BELAGAVI

BANGALORE INSTITUTE OF TECHNOLOGY

K.R.ROAD, V.V. PURA, BENGALURU -560 004

Department of Computer Science and Engineering

BCSL404

Analysis and Design of Algorithms Laboratory Record

IV- Semester

Prepared By:

Prof. Suma L
Prof. Nikhitha K S
Prof. Ashwini T N

Bangalore Institute of Technology

K. R. Road, V. V. Pura, Bengaluru- 560004

Department of Computer Science and Engineering

VISION:

To be a center of excellence in computer engineering education, empowering graduates as highly skilled professionals.

MISSION:

M1	To provide a platform for effective learning with emphasis on technical excellence.
M2	To train the students to meet current industrial standards and adapt to emerging technologies.
M3	To instill the drive for higher learning and research initiatives.
M4	To inculcate the qualities of leadership and Entrepreneurship.

PROGRAM EDUCATIONAL OBJECTIVES (PEO)

PEO-1	Graduates will apply fundamental and advanced concepts of Computer Science and Engineering for solving real world problems.
PEO-2	Graduates will build successful professional careers in various sectors to facilitate societal needs.
PEO-3	Graduates will have the ability to strengthen the level of expertise through higher studies and research.
PEO-4	Graduates will adhere to professional ethics and exhibit leadership qualities to become an Entrepreneur.

PROGRAM SPECIFIC OUTCOMES (PSOs)

PSO-1	The graduates of the program will have the ability to build software products by applying theoretical concepts and programming skills.
PSO-2	The graduates of the program will have the ability to pursue higher studies and Research in the modern computing era.

PROGRAM OUTCOMES (POs)

Engineering Graduates will be able to:

1. **Engineering knowledge**: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
2. **Problem analysis**: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
3. **Design/development of solutions**: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
4. **Conduct investigations of complex problems**: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
5. **Modern tool usage**: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
6. **The engineer and society**: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
7. **Environment and sustainability**: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
8. **Ethics**: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
9. **Individual and team work**: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
10. **Communication**: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
11. **Project management and finance**: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
12. **Life-long learning**: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

Bangalore Institute of Technology

K R Road, VV pura, Bengaluru 560004

Department of Computer Science and Engineering

Course Learning Objectives (CLO):

This course will enable students to:

1. Explain the methods of analyzing the algorithms and to analyze performance of algorithms.
2. State algorithm's efficiencies using asymptotic notations.
3. Solve problems using algorithm design methods such as the brute force method, greedy method, divide and conquer, decrease and conquer, transform and conquer, dynamic programming, backtracking and branch and bound.
4. Choose the appropriate data structure and algorithm design method for a specified application.
5. Introduce P and NP classes.

Course Outcomes (COs):

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. Demonstrate and present the development of program, its execution and running time(s) and record the results/inferences.

	PO 1	PO 2	PO 3	PO 4	PO 5	PO 6	PO 7	PO 8	PO 9	PO 10	PO 11	PO 12	PSO1	PSO2
CO1	3	2	3		1								2	
CO2	3	3	3		1	2				2		1	2	
CO3	2				3						1	1	2	
CO4			2							2			1	

ANALYSIS AND DESIGN OF ALGORITHMS LABORATORY

Sl No.	Name of Experiment
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. 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

ANALYSIS AND DESIGN OF ALGORITHMS LABORATORY

Subject Code: BCSL404 **CIE Marks : 50**

Hours/Week : 0:0:2 **Total Hours : 24**

EVALUATION CRITERIA

Program Write-up and Execution rubrics (Max: 07 Marks)

		Good	Average
a.	**Understanding of problem and approach to solve.** **(3 Marks)**	Demonstrate good knowledge of design concepts of given problem/algorithm and implementation in Java. (3)	Moderate understanding of the algorithm and its implementation in Java. (2)
b.	**Execution and Testing** **(2 Marks)**	Program hands all possible conditions while execution with proper results. (2)	Average conditions are defined and verified during execution. (1)
c.	**Results and Documentation** **(2 Marks)**	Meticulous documentation of results obtained in data sheets and manual.(2)	Acceptable documentation shown in data sheets and manual.(1)

Viva Rubrics (Max: 03 Marks)

		Good	Average
a.	**Conceptual Understanding** **(2 Marks)**	Explains different steps of algorithm/problem and related concepts involved.(2)	Adequately provides explanation on algorithm/problem.(1)
b.	**Formulation of program/Program Explanation.** **(1 Marks)**	Different strategies and concepts to derive solutions to specified program or Tracing of the program. (1)	Intuitive use of strategies and conceptsto derive solutions to specified program. (0.5)

ANALYSIS AND DESIGN OF ALGORITHMS LABORATORY

Subject Code: BCSL404 **CIE Marks : 50**

Hours/Week : 0:0:2 **Total Hours : 24**

Schedule of Experiments

Sl. No	Name of Experiment	To be Completed
1	Sample programs	Week1
2	Selection sort	Week2
3	Quick Sort with time complexity	Week3
4	Merge sort with time complexity.	Week4
5	Topological ordering	Week5
6	Prim's , Kruskal's Algorithm to find MST	Week6
7	Dijkstra's Algorithm to find shortest path to other vertices	Week7
8	**Lab Test -1**	Week8
9	Knapsack Using Greedy Method	Week9
10	0/1 Knapsack Using Dynamic programing	Week10
11	Floyd's Algorithm to implement All Pair Shortest Path Problem and Warshal's Algorithm to compute Transitive Closure	Week11
12	Sum of Subset Problem, N-Queens Problem	Week12
13	**Final Lab Test**	Week13

ANALYSIS AND DESIGN OF ALGORITHMS LABORATORY

Subject Code: BCSL404 **CIE Marks : 50**

Hours/Week : 0:0:2 **Total Hours : 24**

PROCEDURE FOR PROGRAM EXECUTION

ENVIRONMENT: **Eclipse tool on Linux platform**

1. **Click on Applications → Programming → Eclipse**
2. **New → Project → C Project**
3. **Project name → Enter USN**
4. **Project type → Empty Project**
5. **Toolchains → Linux GCC**
6. **Click on next →**
   i. **Select configurations → select both Debug and Release or click on select all**
   optionon the right end
7. **Finish**

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

Aim : To apply Kruskal's Algorithm for computing the minimum spanning tree is directly based on the generic MST algorithm. It builds the MST in forest.

Definition: Kruskal's algorithm is an algorithm in graph theory that finds a minimum spanning tree for a connected weighted graph. This mean it finds a subset of the edges that forms a tree that includes every vertex, where the total weight of all the edges in the tree is minimized. If the graph is not connected, then it finds a minimum spanning forest .It is an example of a greedy algorithm**.**

Efficiency:With an efficient sorting algorithm, the time efficiency of Kruskal's algorithm will be in O(E log E)

Algorithm

Start with an empty set A, and select at every stage the shortest edge that has not been chosen or rejected, regardless of where this edge is situated in graph.

- Initially, each vertex is in its own tree in forest.
- Then, algorithm consider each edge in turn, order by increasing weight.

- If an edge (u, v) connects two different trees, then (u, v) is added to the set of edges of the MST, and two trees connected by an edge (u, v) are merged into a single tree.

- On the other hand, if an edge (u, v) connects two vertices in the same tree, then edge (u, v) is discarded.

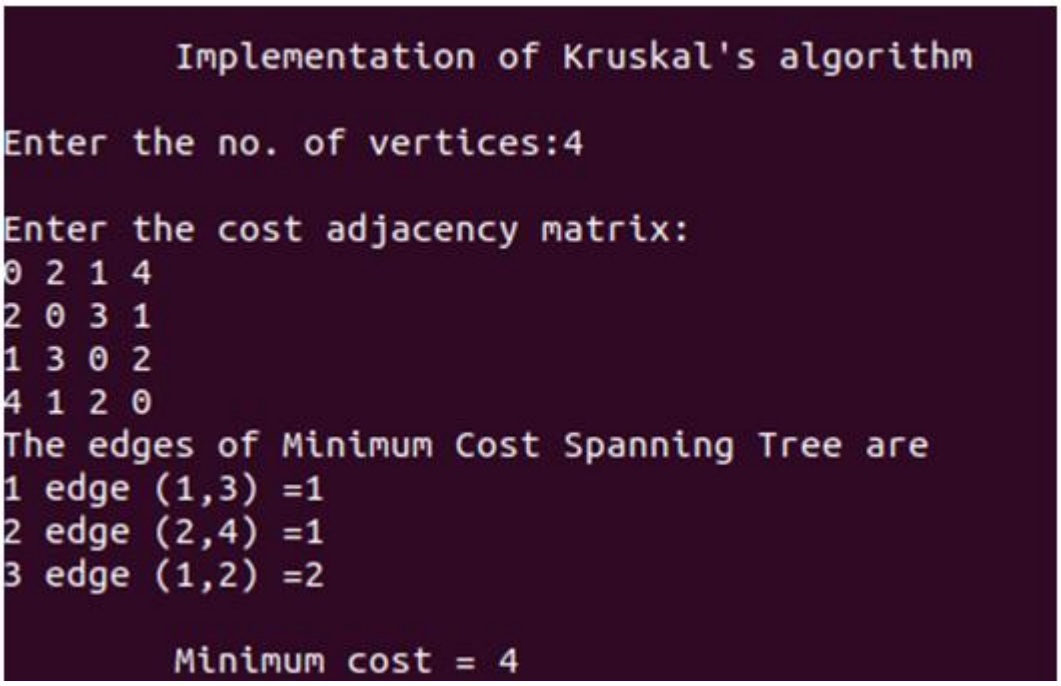
Kruskals algorithm can be implemented using **disjoint set** data structure or **priority queue** data structure.

MST_KRUSKAL (G, w)

1. A ← {} // A will ultimately contains the edges of the MST
2. for each vertex v in V[G]
3. do Make_Set (v)
4. Sort edge of E by nondecreasing weights w
5. for each edge (u, v) in E
6. do if FIND_SET (u) ≠ FIND_SET (v)
7. then A = AU{(u, v)}
8. UNION (u, v)
9. Return A

```
#include<stdio.h>
int i,j,k,a,b,u,v,n,ne=1;
int min,mincost=0,cost[9][9],parent[9];
int find(int);
int uni(int,int);

void main()
{
printf("\n\t Implementation of  Kruskal's
algorithm\n");printf("\nEnter the no. of vertices:");
scanf("%d",&n);
printf("\nEnter the cost 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;
    }
}
printf("The edges of Minimum Cost Spanning Tree are\n");
while(ne < n)
{
    for(i=1,min=999;i<=n;i++)
{
            for(j=1;j <= n;j++)
            {
                    if(cost[i][j] < min)
                    {

    min=cost[i][j];

    a=u=i;
                                    b=v=j;
                    }
            }
}
u=find(u);
v=find(v);
```

```
        if(uni(u,v))

    {
                printf("%d edge (%d,%d) =%d\n",ne++,a,b,min);
                mincost +=min;
        }
        cost[a][b]=cost[b][a]=999;
    }
    printf("\n\tMinimum cost = %d\n",mincost);
}
    int find(int i)
    {
                while(parent[i])
                        i=parent[i];
                return i;
    }
    int uni(int i,int j)
    {
                if(i!=j)
                {
                        parent[j]=i;
                        return 1;
                }
                return 0;}
```



Input/Output:

Enter the n value:5
Enter the cost adjacency matrix:
0 10 15 9 999
10 0 999 17 15
15 999 0 20 999
9 17 20 0 18
999 15 999 18 0

The edges of minimum cost spanning tree are:
1 edge (1 ,4)
2 edge (1 ,2)
3 edge (1 ,3)
4 edge (2,5)
Minimum cost =49

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

Aim: To find minimum spanning tree of a given graph using prim's algorithm

Definition: Prim's is an algorithm that finds a minimum spanning tree for a connected weighted undirected graph. This means it finds a subset of the edges that forms a tree that includes every vertex, where the total weight of all edges in the tree is minimized. Prim's algorithm is an example of a greedy algorithm.

Algorithm

MST_PRIM (G, *w*, *v*)

// Prim's algorithm for constructing a minimum spanning tree
// Input : A weighted connected graph G=(V,E)
// Output : The set of edges composing a minimum spanning tree of G

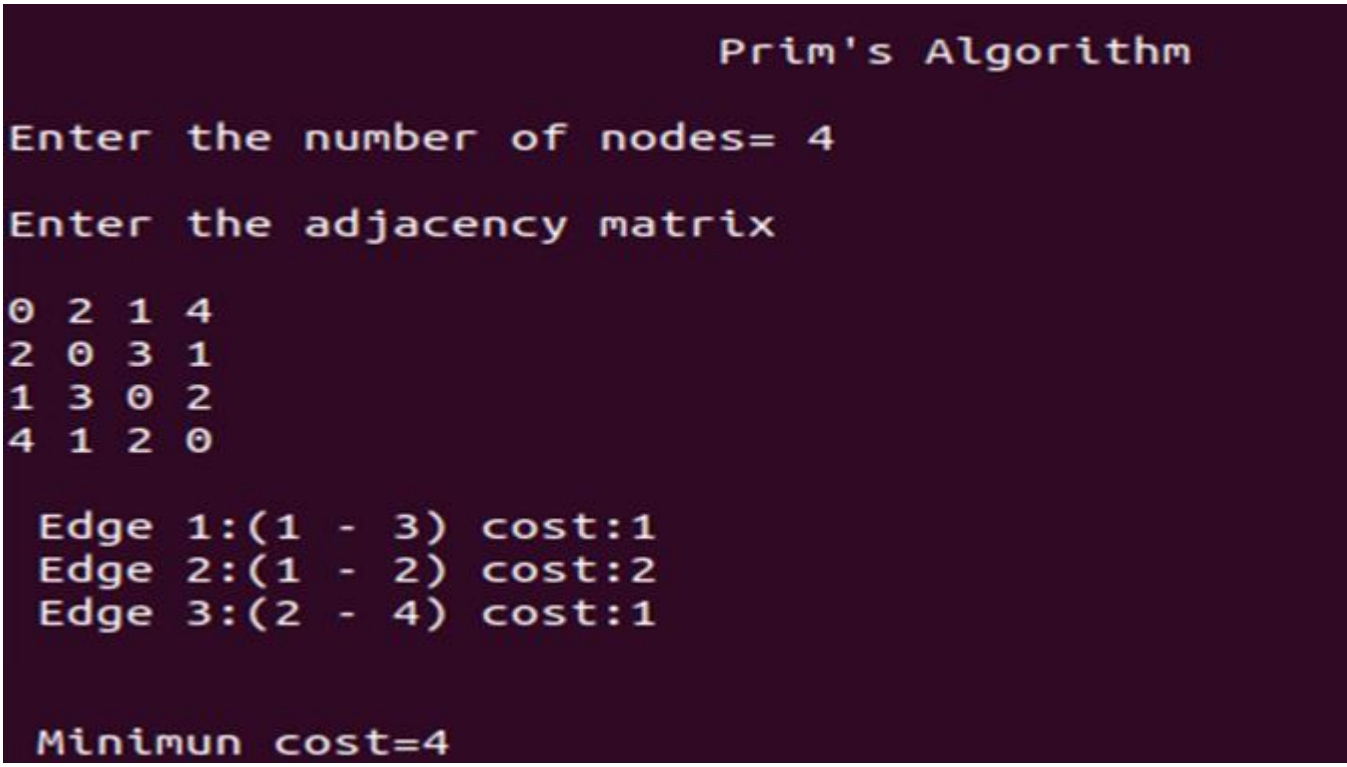
1. Q ← V[G]
2. for each *u* in Q do
3. key [*u*] ← ∞
4. key [*r*] ← 0
5. π[*r*] ← NIl
6. while queue is not empty do
7. *u* ← EXTRACT_MIN (Q)
8. for each *v* in Adj[*u*] do
9. if v is in Q and *w*(*u*, *v*) < key [*v*]
10. then π[*v*] ← *w*(*u*, *v*)
11. key [*v*] ← *w*(*u*, *v*)

Program:

```
#include<stdio.h>

int visited[10]={0}, cost[10][10], min, mincost=0;
int i,j,ne=1, a, b, u, v;

int main()
{
        int num;
        printf("\n\t\t\tPrim's Algorithm");
        printf("\n\nEnter the number of nodes= ");
        scanf("%d", &num);
        printf("\nEnter the adjacency matrix\n\n");

        for(i=1; i<=num; i++)
        {
```

```
        for(j=1; j<=num; j++)
        {
                scanf("%d", &cost[i][j]);
                if(cost[i][j]==0)
                        cost[i][j]=999;
        }
}
visited[1]=1;   while(ne < num)
{
        for(i=1,min=999;i<=num;i++)
        for(j=1;j<=num;j++)
                if(cost[i][j]< min)
                if(visited[i]!=0)
                {
                        min=cost[i][j];
                        a=u=i;
                        b=v=j;
                }
                printf("\n Edge %d:(%d - %d) cost:%d",ne++,a,b,min);
                                mincost=mincost+min;
                visited[b]=1;

cost[a][b]=cost[b][a]=999;
}
printf("\n\n\n Minimun cost=%d",mincost);
return 0; }
```

1)



2) Input/output:
Enter n value:3
Enter the graph data:
0 10 1
10 0 6
1 6 0
Enter the souce node:1
1 -> 3 cost=1
3 -> 2 cost=7
minimum Cost=7

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

Definition: The **Floyd algorithm** is a graph analysis algorithm for finding shortest paths in a weighted graph (with positive or negative edge weights). A single execution of the algorithm will find the lengths (summed weights) of the shortest paths between *all* pairs of vertices though it does not return details of the paths themselves. The algorithm is an example of dynamic programming.

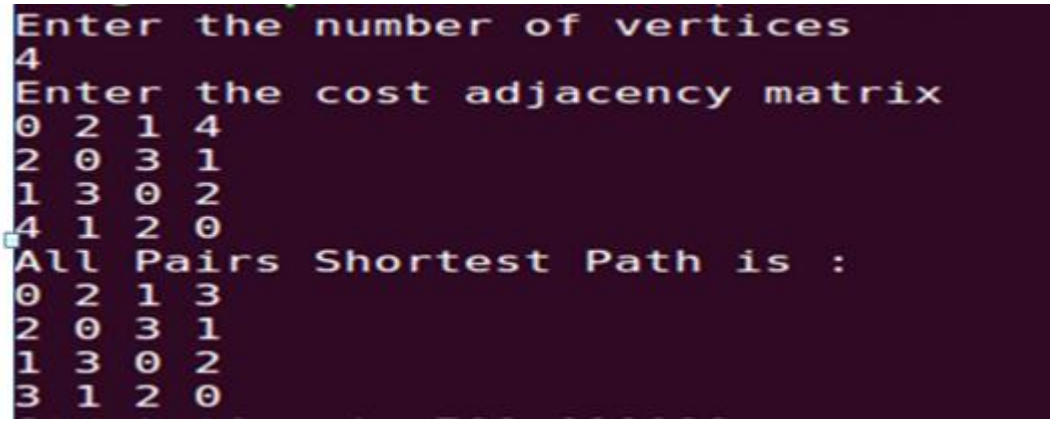
Algorithm:
Floyd's Algorithm
Accept no .of vertices
Call graph function to read weighted graph // w(i,j)

```
Set D[ ] <- weighted graph matrix // get D{d(i,j)} for k=0

// If there is a cycle in graph, abort. How to find?
        Repeat for k = 1 to n
                Repeat for i = 1 to n
                    Repeat for j = 1 to n
                        D[i,j] = min {D[i,j], D[i,k] + D[k,j]}
Print D
```

```
#include<stdio.h>
#define INF 999

int min(int a,int b)
{
        return(a<b)?a:b;
}

void floyd(int p[][10],int n)
{
        int i,j,k;
        for(k=1;k<=n;k++)
        for(i=1;i<=n;i++)
        for(j=1;j<=n;j++)
        p[i][j]=min(p[i][j],p[i][k]+p[k][j]);
}

void main()
{
int a[10][10],n,i,j;
        printf("\nEnter the n value:");
        scanf("%d",&n);
        printf("\nEnter the graph data:\n");
        for(i=1;i<=n;i++)
        for(j=1;j<=n;j++)
        scanf("%d",&a[i][j]);
        floyd(a,n);
        printf("\nShortest path matrix\n");
        for(i=1;i<=n;i++)
        {
                for(j=1;j<=n;j++)
                printf("%d ",a[i][j]);
                printf("\n");
        }
```

}

Output:



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

Definition: The Warshall algorithm is a graph analysis algorithm used to determine the transitive closure of a directed graph or all paths in a directed graph by using the adjacency matrix.. The algorithm is an example of Dynamic programming.

Algorithm

//Input: Adjacency matrix of digraph
//Output: R, transitive closure of digraph
Accept no .of vertices
Call graph function to read directed graph
Set R[ ] <- digraph matrix // get R {r(i,j)} for k=0
Print digraph
Repeat for k = 1 to n
Repeat for i = 1 to n
Repeat for j = 1 to n
R(i,j) = 1 if
{$r_{ij}^{(k-1)} = 1$ OR
$r_{ik}^{(k-1)} = 1$ and $r_{kj}^{(k-1)} = 1$}
Print R

Program b):

```
#include<stdio.h>
void warsh(int p[][10],int n)
{
        int i,j,k;
```

```
        for(k=1;k<=n;k++)
        for(i=1;i<=n;i++)
        for(j=1;j<=n;j++)
        p[i][j]=p[i][j] || p[i][k] && p[k][j];
}

int main()
{
        int a[10][10],n,i,j;
        printf("\nEnter the n value:");
        scanf("%d",&n);
        printf("\nEnter the graph data:\n");
        for(i=1;i<=n;i++)
        for(j=1;j<=n;j++)
        scanf("%d",&a[i][j]);
        warsh(a,n);
        printf("\nResultant path matrix\n");
        for(i=1;i<=n;i++)
        {
                for(j=1;j<=n;j++)
                printf("%d ",a[i][j]);
                printf("\n");
        }
        return 0;
}
```

Output:

```
Enter the n value:4
Enter the graph data:
0 1 0 0
0 0 0 1
0 0 0 0
1 0 1 0
Resultant path matrix
1 1 1 1
1 1 1 1
0 0 0 0
1 1 1 1
```

4) Design and implement C Program to find shortest paths from a given vertex in a weighted connected graph to other vertices using Dijkstra's algorithm.

Dijikstra's algorithm : For a given source vertex(node) in the graph, the algorithm finds the path with lowest cost between that vertex and every other vertex. It can also be used for finding cost of shortest paths from a single vertex to a single destination vertex by stopping the algorithm once the shortest path to the destination vertex has been determined.

Efficiency:
1)$\Theta(|V^2|)$ graph represented by weighted matrix and priority queue as unordered array

2)$O(E \log_2 V)$ graph represented by adjacency lists and priority queue as min-heap

Dijkstra(G,s)

```
// Dijkstra's algorithm for single-source shortest paths

// Input : A weighted connected graph G=(V,E) with nonnegative weights and
           its vertex s
// Output : The length dv of a shortest path from s to v and its penultimate vertex
            pv for every v in V.
Initialise(Q)                          // Initialise vertex priority queue to empty
for every vertex v in V do
   {
            dv←∞;
            pv←null
            Insert(Q,v,dv
  //Initialise vertex priority in the priority queue
    ds←0;
    Decrease(Q,s ds)                          //Update priority of s with ds
    Vt←Ø
     for i←0 to |v|-1 do
     u* ← DeleteMin(Q)                        //delete the minimum priority element
    Vt ←Vt U {u*}
    for every vertex u in V-VT that is adjacent to u* do
    if du* + w(u*,u)<du
    du←du*+w(u*,u);
    pu←u*
    Decrease(Q,u,du)
```

```
#include<stdio.h>
#define INF 999
void dijkstra(int c[10][10],int n,int s,int d[10])
{
    int v[10],min,u,i,j;
    for(i=1;i<=n;i++)
    {
        d[i]=c[s][i];
        v[i]=0;
    }
```

```
    v[s]=1;
    for(i=1;i<=n;i++)
    {
        min=INF;
        for(j=1;j<=n;j++)
        if(v[j]==0 && d[j]<min)
        {
                min=d[j];
                u=j;
        }
v[u]=1;
        for(j=1;j<=n;j++)
        if(v[j]==0 && (d[u]+c[u][j])<d[j])
        d[j]=d[u]+c[u][j];
    }
}

int main()
{
    int c[10][10],d[10],i,j,s,sum,n;
    printf("\nEnter n value:");
    scanf("%d",&n);
    printf("\nEnter the graph data:\n");
    for(i=1;i<=n;i++)
    for(j=1;j<=n;j++)
    scanf("%d",&c[i][j]);
    printf("\nEnter the souce node:");
    scanf("%d",&s);
    dijkstra(c,n,s,d);
    for(i=1;i<=n;i++)
    printf("\nShortest distance from %d to %d is %d",s,i,d[i]);
  return 0;
}
```

Input/Output

1) Enter n value:6
Enter the graph data:
0 15 10 999 45 999
999 0 15 999 20 999
20 999 0 20 999 999
999 10 999 0 35 999
999 999 999 30 0 999
999 999 999 4 999 0

Enter the souce node:2

Shortest distance from 2 to 1 is 35
Shortest distance from 2 to 2 is 0
Shortest distance from 2 to 3 is 15
Shortest distance from 2 to 4 is 35
Shortest distance from 2 to 5 is 20
Shortest distance from 2 to 6 is 999

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

Aim: To find topological ordering of given graph

Definition: Topological ordering is that for every edge in the graph, the vertex where the edge starts is listed before the edge where the edge ends.

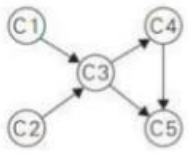
Algorithm:
1. Repeatedly identify in a remaining digraph a source which is a vertex with no incoming edges and delete it along with all edges outgoing from it
2. The order in which the vertices are deleted yields a solution to the topological sorting.

```
#include<stdio.h>
void topo_sort(int a[20][20], int n)
{
	int t[10],vis[10],stack[10],i,j,indeg[10],top=0,ele,k=1;
	for(i=1;i<=n;i++)
	{
		t[i]=0;
		vis[i]=0;
		indeg[i]=0;
	}
	for(i=1;i<=n;i++)
	{
		for(j=1;j<=n;j++)
		{
			if(a[i][j]==1)
			{
				indeg[j]=indeg[j]+1;
			}
		}
	}
	printf("Indegree Array:");
	for(i=1;i<=n;i++)
		printf("%d ",indeg[i]);
```

```
	for(i=1;i<=n;i++)
	{
		if(indeg[i]==0)
		{
			stack[++top]=i;
			vis[i]=1;
		}
	}
	while(top>0)
	{
		ele=stack[top--];
		t[k++]=ele;

		for(j=1;j<=n;j++)
		{
			if(a[ele][j]==1 && vis[j]==0)
			{
				indeg[j]=indeg[j]-1;
				if(indeg[j]==0)
				{
					stack[++top]=j;
					vis[j]=1;
				}
			}
		}
	}
	printf("\nTopological Ordering is:");
	for(i=1;i<=n;i++)
	printf("%d",t[i]);
}
int main()
{
	int n,a[20][20],i,j;
	printf("Enter the number of nodes\n");
	scanf("%d",&n);
	printf("Enter Adjacency matric\n");
	for(i=1;i<=n;i++)
	{
		for(j=1;j<=n;j++)
		{
			scanf("%d",&a[i][j]);
		}
	}
	topo_sort(a,n);
```

}

Output:




```
Enter the number of nodes
5
Enter Adjacency matric
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
Indegree Array:0 0 2 1 2
Topological Ordering is:21345
```

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

Aim: To implement 0/1 Knapsack problem using Dynamic programming

Definition: using **Dynamic programming**
It gives us a way to design custom algorithms which systematically search all possibilities (thus guaranteeing correctness) while storing results to avoid recomputing (thus providing efficiency).

We are given a set of *n* items from which we are to select some number of items to be carried in a knapsack(BAG). Each item has both a *weight* and a *profit*. The objective is to choose the set of items that fits in the knapsack and maximizes the profit.
Given a knapsack with maximum capacity *W*, and a set *S* consisting of *n* items , Each item *i* has some weight *wi* and benefit value *bi* (all *wi , bi* and *W* are integer values)
Problem: How to pack the knapsack to achieve maximum total value of packed items?

ALGORITHM

//(n items, W weight of sack) Input: n, $w_{i,,}$, v_i and W – all integers
//Output: V(n,W)
// Initialization of first column and first row elements
Repeat for i = 0 to n
set V(i,0) = 0
Repeat for j = 0 to W
Set V(0,j) = 0
//complete remaining entries row by row
Repeat for i = 1 to n
repeat for j = 1 to W
if (w_i <= j) V(i,j)) = max{ V(i-1,j), V(i-1,j-w_i) + v_i }
if (w_i > j) V(i,j) = V(i-1,j)

Print V(n,W)

PROGRAM:

```
#include<stdio.h>
int w[10],p[10],n;

int  max(int a,int b)
{
        return a>b?a:b;
}
int  knap(int i,int m)
{
        if(i==n)  return w[i]>m?0:p[i];

        if(w[i]>m)  return knap(i+1,m);

        return   max(knap(i+1,m),knap(i+1,m-w[i])+p[i]);
}

int  main()
{
        int m,i,max_profit;
        printf("\nEnter the no. of objects:");
        scanf("%d",&n);
        printf("\nEnter the knapsack capacity:");
        scanf("%d",&m);
        printf("\nEnter profit followed by weight:\n");
        for(i=1;i<=n;i++)
        scanf("%d %d",&p[i],&w[i]);
        max_profit=knap(1,m);
        printf("\nMax profit=%d",max_profit);
        return 0;
}
```

Input/Output:

```
Enter the no. of objects:4
Enter the knapsack capacity:6
Enter profit followed by weight:
78 2
45 3
92 4
71 5
```

Max profit=170

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

This program first calculates the profit-to-weight ratio for each item, then sorts the items based on this ratio in non-increasing order. It then fills the knapsack greedily by selecting items with the highest ratio until the knapsack is full. If there's space left in the knapsack after selecting whole items, it adds fractional parts of the next item. Finally, it prints the optimal solution and the solution vector.

Here's a simplified version of the C program to solve discrete Knapsack and continuous Knapsack problems using the greedy approximation method:

```
#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("\nThe result vector is:- ");
  for (i = 0; i < n; i++)
    printf("%f\t", x[i]);
```

```
  printf("\nMaximum profit is:- %f", tp);

}


int main()
{
  float weight[20], profit[20], capacity;
  int num, i, j;
  float ratio[20], temp;

  printf("\nEnter the no. of objects:- ");
  scanf("%d", &num);

  printf("\nEnter the wts and profits of each object:- ");
  for (i = 0; i < num; i++) {
    scanf("%f %f", &weight[i], &profit[i]);
  }

  printf("\nEnter the capacity of knapsack:- ");
  scanf("%f", &capacity);

  for (i = 0; i < num; i++) {
    ratio[i] = profit[i] / weight[i];
  }

  for (i = 0; i < num; i++) {
    for (j = i + 1; j < num; 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(num, weight, profit, capacity);
  return(0);
}
```

Input/Output:

```
1) Enter the no. of objects:- 4
Enter the wts and profits of each object:-  56 23 78 45 98 76 78 78
Enter the capacity of knapsack:- 100
The result vector is:-: 1.00000  0.000000   0.000000   0.000000
Maximum profit is:- 78.0

2) Enter the no. of objects:- 3
Enter the wts and profits of each object:-  20 30 25 40 10 35
Enter the capacity of knapsack:- 40
The result vector is:-: 1.00000  1.000000   0.25
Maximum profit is:- 82.5
```

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.**

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

Algorithm:

```
 SumOfSub (s, k, r)
//Values of x[ j ], 1 <= j < k, have been determined
//Node creation at level k taking place: also call for creation at level K+1 if possible
// s = sum of 1 to k-1 elements and r is sum of k to n elements
//generating left child that means including k in solution
Set x[k] = 1
If (s + s[k] = d) then subset found, print solution
If (s + s[k] + s[k+1] <=d)
        then SumOfSum (s + s[k],  k+1,  r – s[k])
//Generate right child i.e. element k absent
If (s + r - s[k] >=d) AND  (s + s[k+1] )<=d
THEN { x[k]=0;
       SumOfSub(s, k+1, r – s[k])
```

Program

```
#include<stdio.h>

#define MAX 10
```

```
int s[MAX],x[MAX],d;
void sumofsub(int p,int k,int r)
{
        int i;
        x[k]=1;
        if((p+s[k])==d)
        {
                for(i=1;i<=k;i++)
                if(x[i]==1)
                        printf("%d ",s[i]);
                printf("\n");
        }
        else
        if(p+s[k]+s[k+1]<=d)
                sumofsub(p+s[k],k+1,r-s[k]);
        if((p+r-s[k]>=d) && (p+s[k+1]<=d))
        {
                x[k]=0;
                sumofsub(p,k+1,r-s[k]);
}
}
int main()
{
        int i,n,sum=0;
        printf("\nEnter the n value:");
        scanf("%d",&n);
        printf("\nEnter the set in increasing order:");
```

```
        for(i=1;i<=n;i++)
                scanf("%d",&s[i]);
        printf("\nEnter the max subset value:");
        scanf("%d",&d);
        for(i=1;i<=n;i++)
                sum=sum+s[i];
        if(sum<d || s[1]>d)
                printf("\nNo subset possible");
        else
        sumofsub(0,1,sum);
        return 0;
}
```

Input/output:

```
Enter the n value:9
Enter the set in increasing order:1 2 3 4 5 6 7 8 9
Enter the max subset value:9
1 2 6
1 3 5
1 8
2 3 4
2 7
3 6
4 5
9
```

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.

Aim: Sort a given set of elements using Selection sort and determine the time required to sort 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.

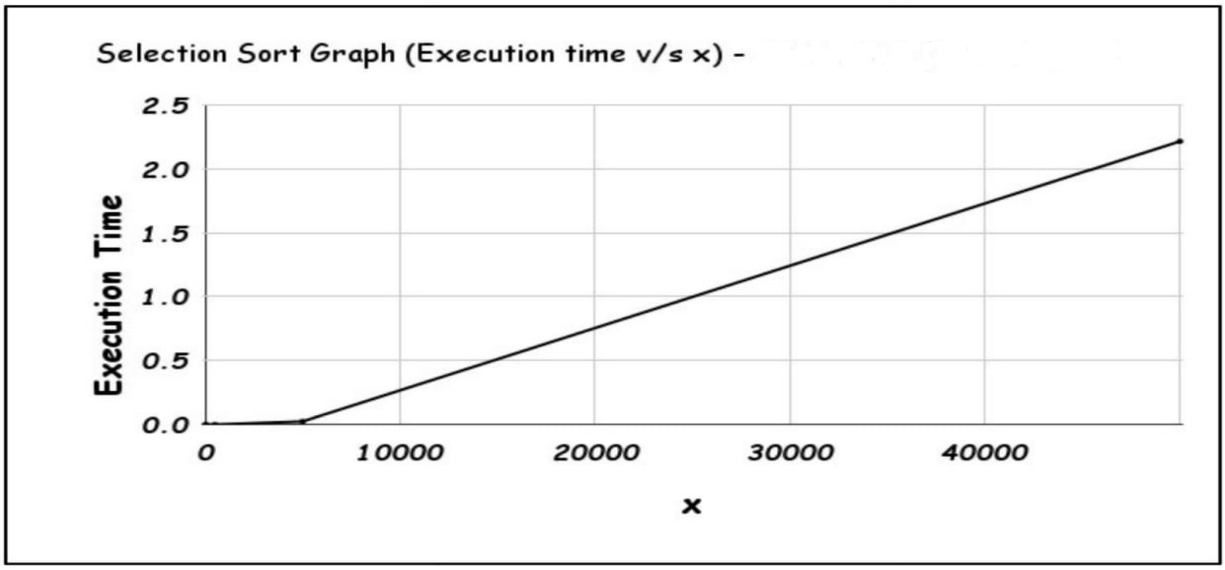
Definition: selection sort is a sorting routine that scans a list of items repeatedly and, on each pass, selects the item with the lowest value and places it in its final position. It is based on brute force approach. Sequential search is a $\Theta(n^2)$ algorithm on all inputs.

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


// Function to perform selection sort
voidselectionSort(intarr[], int n)
{
inti, j, min_idx,temp;

   // One by one move boundary of unsorted subarray
for (i = 0; i<=n-2; i++)
{
      // Find the minimum element in unsorted array
min_idx = i;
for (j = i+1; j <=n-1; j++)
{
if (arr[j] <arr[min_idx])
min_idx = j;
      }
      // Swap the found minimum element with the first element
temp=arr[i];
arr[i]=arr[min_idx];
arr[min_idx]=temp;
   }
}

int main()
 {
int n ;// Number of elements
printf("enter the value of n\n");
scanf("%d",&n);

intarr[n];
srand(time(NULL)); // Seed for random number generation

   // Generate random numbers and fill the array
printf("Original Array: ");
```

```
for (inti = 0; i< n; i++)
 {
arr[i] = rand() % 1000; // Generating random numbers between 0 and 999
printf("%d ", arr[i]);
   }
printf("\n");

   // Perform selection sort
clock_tstart_time = clock();
selectionSort(arr, n);
clock_tend_time = clock();
doubletotal_time = (double)(end_time - start_time) / CLOCKS_PER_SEC;

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

   // Print time complexity
printf("Time Complexity: O(n^2)\n");
printf("Execution Time: %f seconds\n", total_time);

return 0;
}
```

Output:

Note down the different values of n(eg: 5000,6000,7000,8000,9000,10000) and required time for all values of n respectively. Plot the graph for the same.



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.

Aim:

The aim of this program is to sort 'n' randomly generated elements using Quick sort and Plotting the graph of the time taken to sort n elements versus n.

Definition: Quicksort is based on the Divide and conquer approach. Quick sort divides array according to their value. Partition is the situation where all the elements before some position s are smaller than or equal to A[s] and all the elements after position s are greater than or equal to A[s].

Efficiency: Cbest(n)ЄΘ(nlog2n),Cworst(n)ЄΘ(n^2),Cavg(n)Є1.38nlog2n

Algorithm: Quicksort (A[l….r])

//Sorts a sub array by quick sort

//Input:Asub arrayA[l..r]ofA[0..n-1] ,defined by its left and right indices l

//andr

```
  //Output:The subarrayA[l..r] sorted in nondecreasing
          order

   ifl < r

              s=Partition(A[l..r])//sisasplitpositio
              n Quick sort (A[l …s-1])

              Quicksort (A[s+1…r])
```

ALGORITHM Partition (A[l…r])

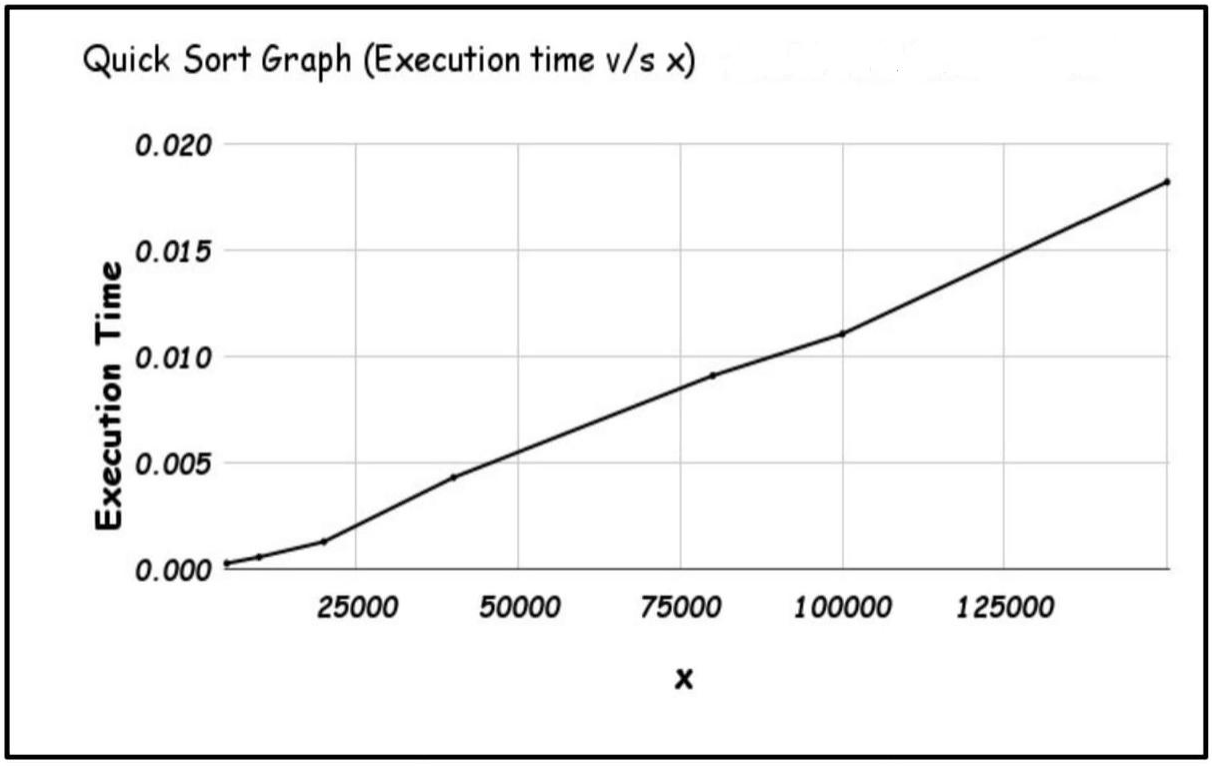
```
//Partitionasub arraybyusingits firstelement asa pivot

//Input :Asubarray A[l…r]ofA[0…n-1]definedby itsleft andright indicesl and// r(l

<r)

//Output:ApartitionofA[l…r],withthesplitpositionreturnedasthisfunction’svalue p=A[l]

i=l; j=r+1; repeat

   delay(500);

   repeati=i+1untilA[i]>
   =p  repeat  j=j-1  until
   A[J]<=p            Swap
   (A[i],A[j])

until I>=j

Swap(A[i],A[j])//UndolastSwapwheni>=j
Return j

Program:
#include <stdio.h>
#include<stdlib.h>
#include <time.h>


//Functionto partitionthearray
intpartition(inta[],intlow,inthigh){ in
   t pivot = a[low]; // Pivot element
   inti=low,temp;//Indexofsmallerelement int
   j=high+1;// Index of largest element

   while(i<=j)
   {
```

```
        doi++;while(pivot>=a[i]
        ); do j--
        ;while(pivot<a[j]);

            if(
            {

                temp=a[i]; a[i]=a[j]; a[j]=temp;


            }
}
    temp=a[low];
    a[low]=a[j];
    a[j]=temp;

    return j;
}


//Functiontoperform Quick Sort
void quickSort(inta[],intlow,int high)
{
            intk;
            if(low>high) return;

        //Partitioning index
        k=partition(a,low, high);

        //Separatelysortelements after partition
```

```
        quickSort(a, low, k - 1);// left part of pivot element
        quickSort(a,k+1,high);//rightpartofpivotelement
 }


   intmain()
{

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

    intn;
    printf("Enter the number of elements:");
     scanf("%d", &n);

    inta[n];
    printf("Randomly generated array:\n");
    for (int i = 0; i < n; i++) {
        a[i]=rand()%1000;//Generatingrandomnumbersbetween0and999
        printf("%d ", a[i]);
    } a[n]=9999;
    clock_tstart=clock();
    quickSort(a,0,n-1); clock_t end=clock();
    printf("\nSortedarray:\n");
    for (int i = 0; i < n; i++) {
        printf("%d",a[i]);
    }

    printf("\n");

    printf("Total      time    taken    to    sort    %d    elememts    is
                        %lf\n",n,((double)(end- start)/CLOCKS_PER_SEC));

    return 0;

 }
```

Output:

Note down the different values of n (eg: 5000,6000,7000,8000,9000,10000) and required time for all values of n respectively. Plot the graph for the same.



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

Aim:

The aim of this program is to sort 'n' randomly generated elements using Merge sort and Plotting the graph of the time taken to sort n elements versus n.

Definition: Merge sort is a sort algorithm based on divide and conquer technique. It divides the array element based on the position in the array. The concept is that we first break the list into two smaller lists of roughly the same size, and then use merge sort recursively on the sub problems, until they cannot subdivide anymore. Then, we can merge by stepping through the lists in linear time.

Its time efficiency is $\Theta(n \log n)$.

Algorithm: Merge sort (A[0…n-1]

```
// Sorts array A[0..n-1] by Recursive merge sort
// Input : An array A[0..n-1]  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[0…(n/2)-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;
     While i < p and j < q do
       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:

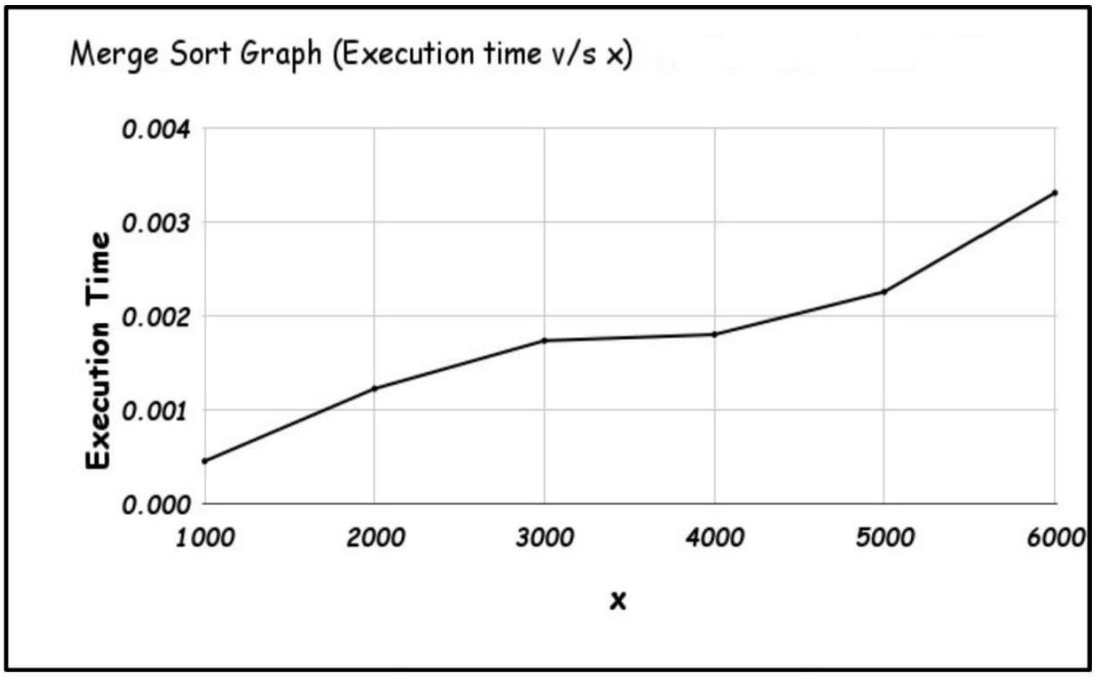
```
#include<stdlib.h>
#include<stdio.h>
#include<time.h>
void merge(int arr[], int l, int m, int r)
{
        int i, j, k;
        int n1 = m - l + 1;
        int n2 = r - m;
        // Create temp arrays
        int L[n1], R[n2];
```

```
        // Copy data to temp array
        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 temp arrays
        i = 0;
        j = 0;
        k = 1;
        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[]
        while (i < n1)
        {
                arr[k] = L[i];
                i++;
                k++;
        }

// Copy the remaining elements of R[]
        while (j < n2)
        {
                arr[k] = R[j];
                j++;
                k++;
        }
}
```

```
void mergeSort(int arr[], int l, int r)
{
	if (l < r)
	{
		// Finding mid element
		int m = l+(r-l)/2;
		// Recursively sorting both the halves
		mergeSort(arr, l, m);
		mergeSort(arr, m+1, r);
		merge(arr, l, m,r);
	}
}


void main()
{
	int n;
	printf("enter the value of n\n");
	scanf("%d",&n);

	int arr[n],i;
	srand(time(NULL));


	printf("the original array");
	for(i=0;i<n;i++)
	{
		arr[i]=rand()%1000;
		printf("%d  ",arr[i]);

	}
	printf("\n");
	clock_t start=clock();
	mergeSort(arr, 0,n);
	clock_t end=clock();


	printf("Sorted array:");
	for(i=0;i<n;i++)
	{
```

```
            printf("%d  ",arr[i]);
        }
        printf("\n");

        printf("Total time taken to sort %d elements is %lf\n",n,((double)(end-
start)/CLOCKS_PER_SEC));


}
```

Output:
Note down the different values of n (eg: 5000,6000,7000,8000,9000,10000) and required time for all values of n respectively. Plot the graph for the same.



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

Aim: To implement N Queens Problem using Back Tracking.

Definition:

The object is to place queens on a chess board in such a way as no queen can capture another one in a single

move.

Recall that a queen can move horizontal, vertical, or diagonally an infinite distance.This implies that no two queens can be on the same row, col, or diagonal We usually want to know how many different placements there are **Using Backtracking Techniques.**

Algorithm for new queen be placed	**All solutions to the n-queens problem**

Algorithm Place(k,i)
//Return true if a queen can be placed in kth row & ith column
//Other wise return false
{
for j:=1 to k-1 do
if(x[j]=i or Abs(x[j]-i)=Abs(j-k)))
then return false
return true
}
	Algorithm NQueens(k, n)
	// its prints all possible placements of n-queens on an n×n chessboard.
	{
	for i:=1 to n do{
	if Place(k,i) then
	{
	X[k]:=I;
	if(k==n) then write (x[1:n]);
	else NQueens(k+1, n);
	}
	}}

Program

```
#include<stdio.h>

#include<stdlib.h>

#include<math.h>


#define True 1

#define False 0


int x[10];                    // Initial solution vector

int n;

int count;                    //Number of solutions


void printsolution()

{

   char c[10][10];
```

```
    printf("Solution %d:\n\n",++count);
// No queen has placed initially
    for(int i=1;i<=n;i++)
    {
      for(int j=1;j<=n;j++)
      {
         c[i][j]='X';
      }
    }
// Place the queens on the chess board
    for(int i=1;i<=n;i++)
    {
      c[i][x[i]]='Q';
    }
// Place where the queens have been placed on the chess board
     for(int i=1;i<=n;i++)
      {
        for(int j=1;j<=n;j++)
        {
          printf("%c",c[i][j]);
        }
        printf("\n");
      }
 }
// Function to check whether  the queens can be placed successfully or not
```

```
int place(int k,int xk)
{
   for(int i=1;i<=k-1;i++)
   {
      //check whether two queens attach vertically or diagonally
     if((x[i]==xk)||(abs(i-k)==abs(x[i]-xk)))
     {
      return False;//Queen cannot be placed in the kth column
      }
   }
      return True;//kth queen can be successfully placed
}

void nqueen(int k)
{
   int j;
   for(int j=1;j<=n;j++)
   {
     if(place(k,j))//Try to place in various columns
     {
       x[k]=j;//Queen can be placed
       if(k==n)
       printsolution(),printf("\n");
       else
       nqueen(k+1);//Place the next queen in next row
```

```
        }
      }
    }
void main()
 {
   printf("enter the number of queens\n");
   scanf("%d",&n);
   nqueen(1);
 }
```

Input/Output:

Enter the no. of queens: 4
Solution 1
XQXX
XXXQ
QXXX
XXQX

Solution 2
XXQX
QXXX
XXXQ
XQXX