| | |
|---|---|
| | **HKBK COLLEGE OF ENGINEERING**<br>(Affiliated to VTU, Belgaum and Approved by AICTE)<br>BANGALORE -560045<br>**DEPARTMENT OF ARTIFICAL INTELLIGENCE & MACHINE LEARNING** |

**ANALYSIS AND DESIGN OF ALGORITHMS LABORATORY**

**BCSL404**

As per VTU - Choice Based Credit System - 22 Scheme

(Effective from the academic year of 2023 -2024)

IV Semester CSE

# LABORATORY MANUAL

**Prepared By**

Prof. Vani Valsaraj

**Verified By**

| | |
|---|---|
| | |
| **DQAC** | **HOD** |

**HKBK COLLEGE OF ENGINEERING**
(Affiliated to VTU, Belgaum and Approved by AICTE)
BANGALORE -560045
**DEPARTMENT OF ARTIFICAL INTELLIGENCE & MACHINE LEARNING**

## Vision and Mission of the Institution

### Vision

To empower students through wholesome education and enable the students to develop into highly qualified and trained professionals with ethics and emerge as responsible citizen with broad outlook to build a vibrant nation.

### Mission

- To achieve academic excellence in science, engineering and technology through dedication to duty, innovation in teaching and faith in human values.
- To enable our students to develop into outstanding professional with high ethical standards to face the challenges of 21st century.
- To provide educational opportunities to the deprived and weaker section of the society to uplift their socioeconomic status.

## Vision and Mission of the Department

### Vision

To advance the intellectual capacity of the nation and the international community by imparting knowledge to graduates who are globally recognized as innovators, entrepreneur and competent professionals.

### Mission

- To provide excellent technical knowledge and computing skills to make the graduates globally competitive with professional ethics.
- To involve in research activities and be committed to lifelong learning to make positive contributions to the society

# HKBK COLLEGE OF ENGINEERING

(Affiliated to VTU, Belgaum and Approved by AICTE)
BANGALORE -560045

**DEPARTMENT OF ARTIFICAL INTELLIGENCE & MACHINE LEARNING**

| Programme Outcomes | |
|---|---|
| **PO1** | **Engineering Knowledge:** Apply knowledge of mathematics, science, engineering fundamentals and an engineering specialization to the solution of complexengineering problems. |
| **PO2** | **Problem Analysis:** Identif**y,** formulate, research literature and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences and engineering sciences |
| **PO3** | **Design/Development of Solutions**: Design solutions for complex engineering problems and design system components or processes that meet specified needswith appropriate consideration for public health and safety, cultural, societal and environmental considerations. |
| **PO4** | **Conduct investigations of complex problems:** Using research-based knowledge and research methods including design of experiments, analysis and interpretation of data and synthesis of information to provide valid conclusions. |
| **PO5** | **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 under- standing ofthe limitations. |
| **PO6** | **The Engineer and Society:** Apply reasoning informed by contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to professional engineering practice. |
| **PO7** | **Environment and Sustainability**: Understand the impact of professional engineering solutions in societal and environmental contexts and demonstrate knowledge of and need for sustainable development. |
| **PO8** | **Ethics**: Apply ethical principles and commit to professional ethics andresponsibilities and norms of engineering practice. |
| **PO9** | **Individual and Team Work**: Function effectively as an individual, and as amember or leader in diverse teams and in multi-disciplinary settings. |
| **PO10** | **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**.** |
| **PO11** | **Project Management and Finance**: Demonstrate knowledge and understanding of 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. |
| **PO12** | **Life-long Learning**: Recognize the need for and have the preparation andability to engage in independent and life- long learning in the broadest context of technological change. |

| | |
|---|---|
| | **HKBK COLLEGE OF ENGINEERING**<br>(Affiliated to VTU, Belgaum and Approved by AICTE)<br>BANGALORE -560045<br>**DEPARTMENT OF ARTIFICAL INTELLIGENCE & MACHINE LEARNING** |

| **Programme Specific Outcomes** | |
|---|---|
| **PSO1** | **Problem-Solving Skills:** An ability to investigate and solve a problem by analysis, interpretation of data, design and implementation through appropriate techniques,tools and skills. |
| **PSO2** | **Professional Skills**: An ability to apply algorithmic principles, computing skills and computer science theory in the modelling and design of computer-based systems. |
| **PSO3** | **Entrepreneurial Ability:** An ability to apply design, development principles and management skills in the construction of software product of varying complexity to become an entrepreneur |

## Programme Educational Objectives

| | |
|---|---|
| **PEO-1** | To provide students with a strong foundation in engineering fundamentals a n d in the computer science and engineering to work in the global scenario. |
| **PEO-2** | To provide sound knowledge of programming and computing techniques and good communication and interpersonal skills so that they will be capable of analyzing, designing and building innovative software systems. |
| **PEO-3** | To equip students in the chosen field of engineering and related fields to enable him to work in multidisciplinary teams. |
| **PEO-4** | To inculcate in students professional, personal and ethical attitude to relate engineering issues to broader social context and become responsible citizen. |
| **PEO-5** | To provide students with an environment for life-long learning which allow them to successfully adapt to the evolving technologies throughout their professional carrier and face the global challenges. |

## COURSE OUTCOMES

| | Course Outcomes | Level |
|---|---|---|
| CO1 | Develop programs to solve computational problems using suitable algorithm design strategy. | L3 |
| CO2 | Compare algorithm design strategies by developing equivalent programs and observing running times for analysis (Empirical). | L3 |
| CO3 | Make use of suitable integrated development tools to develop programs | L3 |
| CO4 | Choose appropriate algorithm design techniques to develop solution to the computational and complex problems. | L3 |
| CO5 | Demonstrate and present the development of program, its execution and running time(s) and record the results/inferences. | L3 |

## Mapping of COs to POs to Cos

**Mapping of Course outcome to Program Outcomes Based on Components Identified**

| | PO1 | PO2 | PO3 | PO4 | PO5 | PO6 | PO7 | PO8 | PO9 | PO10 | PO11 | PO12 | PSO1 | PSO2 | PSO3 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **CO1** | 3 | 3 | 3 | - | 3 | - | - | - | - | - | - | - | 3 | - | - |
| **CO2** | 3 | 3 | 3 | 3 | 3 | - | - | - | - | - | - | - | 3 | - | - |
| **CO3** | 3 | 3 | 3 | - | 3 | - | - | - | 3 | - | - | 3 | 3 | - | - |
| **CO4** | 3 | 3 | 3 | 3 | 3 | - | - | - | 3 | - | 3 | - | 3 | - | - |
| **CO5** | 3 | 3 | 3 | - | 3 | - | - | - | 3 | - | - | 3 | 3 | - | - |
| **Avg** | **3.00** | **3.00** | **3.00** | **3.00** | **3.00** | **-** | **-** | **-** | **3.00** | **-** | **3.00** | **3.00** | **3.00** | **-** | **-** |

**Rubrics:**

| | | | | | |
|---|---|---|---|---|---|
| 3 | High | 2 | Moderate | 1 | Low |

## Syllabus

| ANALYSIS AND DESIGN OF ALGORITHMS LABORATORY<br>(Effective from the academic year 2023 -2024)<br>SEMESTER – IV | | | |
|---|---|---|---|
| Course Code | BCSL404 | CIE Marks | 50 |
| Number of Contact Hours/Week (L:T:P: S) | 0:0:2:0 | SEE Marks | 50 |
| Credits | 01 | Exam Hours | 2 |
| Examination type (SEE) | Practical | | |

**Course Learning Objectives:** This course (BCSL404) will enable students to:

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

**Programs List:**

| | |
|---|---|
| 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. | Design and implement C/C++ Program to solve All-Pairs Shortest Paths problem using Floyd's algorithm. b. Design and implement C/C++ Program to find the transitive closure using Warshal's algorithm. |
| 4. | Design and implement C/C++ Program to find shortest paths from a given vertex in a weighted connected graph to other vertices using Dijkstra's algorithm. |
| 5. | Design and implement C/C++ Program to obtain the Topological ordering of vertices in a given digraph. |
| 6. | Design and implement C/C++ Program to solve 0/1 Knapsack problem using Dynamic Programming method. |

| | |
|---|---|
| 7 | Design and implement C/C++ Program to solve discrete Knapsack and continuous Knapsack problems using greedy approximation method. |
| 8. | Design and implement C/C++ Program to find a subset of a given set S = {sl , s2,......,sn} of n positive integers whose sum is equal to a given positive integer d. |
| 9. | Design and implement C/C++ Program to sort a given set of n integer elements using Selection Sort method and compute its time complexity. Run the program for varied values of n> 5000 and record the time taken to sort. Plot a graph of the time taken versus n. The elements can be read from a file or can be generated using the random number generator. |
| 10. | Design and implement C/C++ Program to sort a given set of n integer elements using Quick Sort method and compute its time complexity. Run the program for varied values of n> 5000 and record the time taken to sort. Plot a graph of the time taken versus n. The elements can be read from a file or can be generated using the random number generator. |
| 11. | Design and implement C/C++ Program to sort a given set of n integer elements using Merge Sort method and compute its time complexity. Run the program for varied values of n> 5000, and record the time taken to sort. Plot a graph of the time taken versus n. The elements can be read from a file or can be generated using the random number generator. |
| 12. | Design and implement C/C++ Program for N Queen's problem using Backtracking. |

**Course outcomes (Course Skill Set):**

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

1. Develop programs to solve computational problems using suitable algorithm design strategy.

2. Compare algorithm design strategies by developing equivalent programs and observing running times for analysis (Empirical).

3. Make use of suitable integrated development tools to develop programs

4. Choose appropriate algorithm design techniques to develop solution to the computational and complex problems.

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

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

**Kruskal's algorithm:**

Kruskal’s algorithm finds the minimum spanning tree for a weighted connected graph G=(V,E) to get an acyclic subgraph with |V|-1 edges for which the sum of edge weights is the smallest. Consequently the algorithm constructs the minimum spanning tree as an expanding sequence of subgraphs, which are always acyclic but are not necessarily connected on the intermediate stages of algorithm. The algorithm begins by sorting the graph’s edges in non decreasing order of their weights. Then starting with the empty subgraph, it scans the sorted list adding the next edge on the list to the current subgraph if such an inclusion does not create a cycle and simply skipping the edge otherwise.

```
Algorithm : Kruskal(G)
// 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($e_{i1}$)<=……>= w($e_{i\,|E|}$)
        $E_T \leftarrow \varnothing$   ; ecounter ← 0 //Initialize the set of tree edges and its size
        k ← 0 //initialize the number of processed edges
        while ecounter <|V|-1 do
        {
                        k ←       k+1
                        if $E_T$U {$e_{ik}$} is acyclic
                                        $E_T \leftarrow E_T$ U {$e_{ik}$}; ecounter ← ecounter+1
        }
        return $E_T$
}
```

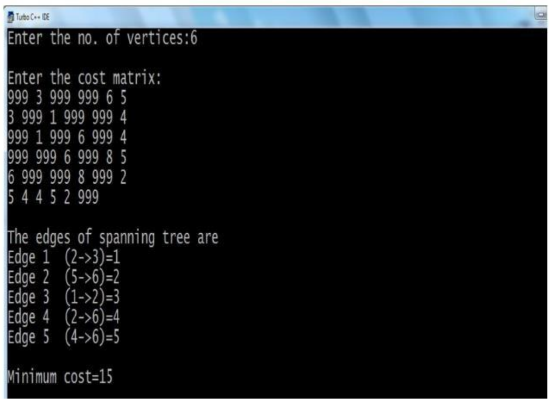
Complexity: With an efficient sorting algorithm, the time efficiency of kruskal’s algorithm will be in O(|E| log |E|).

**Program:**

```
#include<stdio.h>
int ne=1,min_cost=0;

void main()
{
        int n,i,j,min,a,u,b,v,cost[20][20],parent[20];
        clrscr();

        printf("Enter the no. of vertices:");
        scanf("%d",&n);

        printf("\nEnter the cost matrix:\n");
        for(i=1;i<=n;i++)
                for(j=1;j<=n;j++)
```

```
                    scanf("%d",&cost[i][j]);

    for(i=1;i<=n;i++)
            parent[i]=0;

    printf("\nThe edges of spanning tree are\n");
    while(ne<n)
    {
            min=999;
            for(i=1;i<=n;i++)
            {
                    for(j=1;j<=n;j++)
                    {
                            if(cost[i][j]<min)
                            {
                                    min=cost[i][j];
                                    a=u=i;
                                    b=v=j;
                            }
                    }
            }
            while(parent[u])
                    u=parent[u];

            while(parent[v])
                    v=parent[v];

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

**OUTPUT:**



```
Enter the no. of vertices:6

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

The edges of spanning tree are
Edge 1  (2->3)=1
Edge 2  (5->6)=2
Edge 3  (1->2)=3
Edge 4  (2->6)=4
Edge 5  (4->6)=5

Minimum cost=15
```

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

**Prim’s Algorithm:**

Prim’s algorithm finds the minimum spanning tree for a weighted connected graph G=(V,E) to get an acyclic subgraph with |V|-1 edges for which the sum of edge weights is the smallest. Consequently the algorithm constructs the minimum spanning tree as expanding sub-trees. The initial subtree in such a sequence consists of a single vertex selected arbitrarily from the set V of the graph’s vertices. On each iteration, we expand the current tree in the greedy manner by simply attaching to it the nearest vertex not in that tree. The algorithm stops after all the graph’s vertices have been included in the tree being constructed.

```
Algorithm : Prim(G)
// 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
{
    $V_T \leftarrow \{v_0\}$   //the set of tree vertices can be initialized with any vertex
    $E_T \leftarrow \varnothing$

    for i ← 0 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 $E_T$
}
```

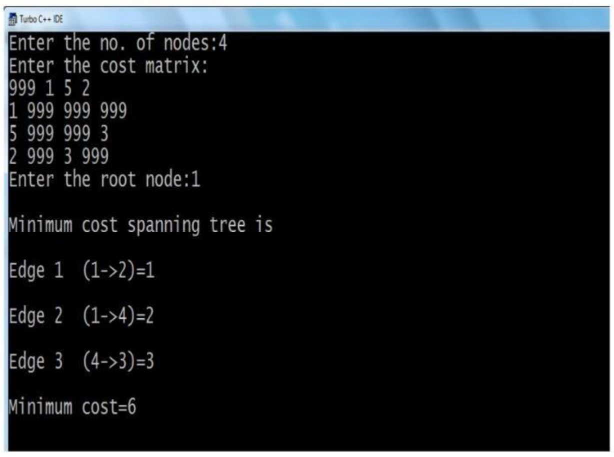
**Complexity:** The time efficiency of prim’s algorithm will be in O(|E| log |V|).

**Program:**

```
#include<stdio.h>

int ne=1,min_cost=0;

void main()
{
        int n,i,j,min,cost[20][20],a,u,b,v,source,visited[20];
        clrscr();

        printf("Enter the no. of nodes:");
        scanf("%d",&n);

        printf("Enter the cost matrix:\n");
        for(i=1;i<=n;i++)
        {
                for(j=1;j<=n;j++)
                {
                        scanf("%d",&cost[i][j]);
                }
        }
```

```
    for(i=1;i<=n;i++)
            visited[i]=0;

    printf("Enter the root node:");
    scanf("%d",&source);
    visited[source]=1;

    printf("\nMinimum cost spanning tree is\n");
    while(ne<n)
    {
            min=999;
            for(i=1;i<=n;i++)
            {
                    for(j=1;j<=n;j++)
                    {
                            if(cost[i][j]<min)
                                    if(visited[i]==0)
                                            continue;
                            else
                            {
                                    min=cost[i][j];
                                    a=u=i;
                                    b=v=j;
                            }
                    }
            }

  if(visited[u]==0||visited[v]==0)
  {

                            printf("\nEdge %d\t(%d->%d)=%d\n",ne++,a,b,min);
                            min_cost=min_cost+min;
                            visited[b]=1;
                    }
                    cost[a][b]=cost[b][a]=999;
            }
            printf("\nMinimum cost=%d\n",min_cost);
            getch();
    }
```

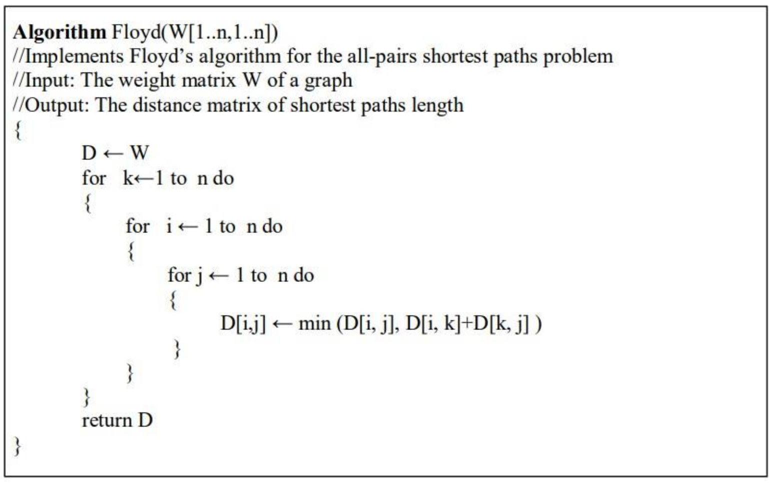
**OUTPUT:**



```
Enter the no. of nodes:4
Enter the cost matrix:
999 1 5 2
1 999 999 999
5 999 999 3
2 999 3 999
Enter the root node:1

Minimum cost spanning tree is

Edge 1  (1->2)=1

Edge 2  (1->4)=2

Edge 3  (4->3)=3

Minimum cost=6
```

**3. a. Design and implement C/C++ Program to solve All-Pairs Shortest Paths problem using Floyd's algorithm.**
**b. Design and implement C/C++ Program to find the transitive closure using Warshal's algorithm.**

**Floyd's Algorithm**:
Floyd's algorithm is applicable to both directed and undirected graphs provided that they do not contain a cycle. It is convenient to record the lengths of shortest path in an n- by- n matrix D called the distance matrix. The element dij in the ith row and jth column of matrix indicates the shortest path from the ith vertex to jth vertex (1<=i, j<=n). The element in the ith row and jth column of the current matrix D(k-1) is replaced by the sum of elements in the same row i and kth column and in the same column j and the kth column if and only if the latter sum is smaller than its current value.



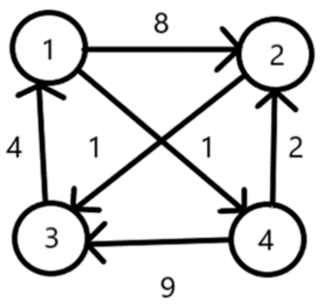
```
Algorithm Floyd(W[1..n,1..n])
//Implements Floyd's algorithm for the all-pairs shortest paths problem
//Input: The weight matrix W of a graph
//Output: The distance matrix of shortest paths length
{
        D ← W
        for  k←1 to  n do
        {
            for  i ← 1 to  n do
            {
                for j ← 1 to  n do
                {
                        D[i,j] ← min (D[i, j], D[i, k]+D[k, j] )
                }
            }
        }
        return D
}
```

**Complexity**: The time efficiency of Floyd's algorithm is cubic i.e. $\Theta$ ($n^3$)

```
#include<stdio.h> int

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

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

int main()
{
        int n,  cost[10][10];
        printf("Enter no. of Vertices: ");
        scanf("%d",&n);
        printf("Enter the cost matrix\n");
        for(int i=1;i<=n;i++)
                 for(int j=1;j<=n;j++)
              scanf("%d",&cost[i][j]);
        floyd(cost,n);

        printf("All pair shortest path\n");
        for(int i=1;i<=n;i++)
        {
                    for(int j=1;j<=n;j++)
              printf("%d  ",cost[i][j]);
           printf("\n");
        }
}
```

Sample Input and Output:

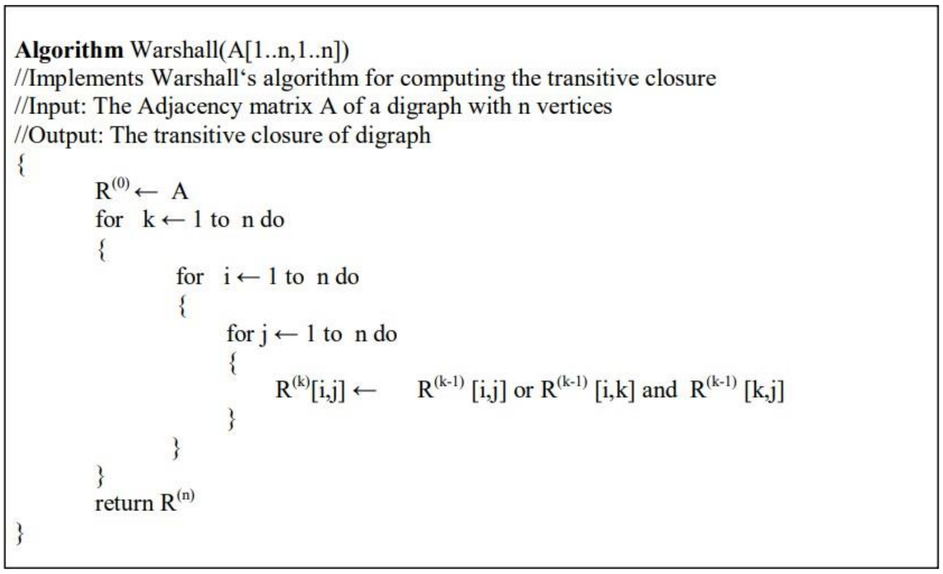


```
Enter no. of Vertices: 4
Enter the cost matrix 999 8
4 999
999 999 1     999
4     999 999 999
999 2     9 999
All pair shortest path
8   8   4 999
5  13  1  999
4  12  8  999
```

```
7  2  3  999
```

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

**Warshall's algorithm:**
The transitive closure of a directed graph with n vertices can be defined as the n-by-n boolean matrix T={tij}, in which the element in the ith row(1<=i<=n) and jth column(1<=j<=n) is 1 if there exists a non trivial directed path from ith vertex to jth vertex, otherwise, tij is 0. Warshall's algorithm constructs the transitive closure of a given digraph with n vertices through a series of n-by-n boolean matrices: R(0) ,….,R(k-1) , R(k) ,….,R(n) where, R (0) is the adjacency matrix of digraph and R(1) contains the information about paths that use the first vertex as intermediate. In general, each subsequent matrix in series has one more vertex to use as intermediate for its path than its predecessor. The last matrix in the series R(n) reflects paths that can use all n vertices of the digraph as intermediate and finally transitive closure is obtained. The central point of the algorithm is that we compute all the elements of each matrix R(k) from its immediate predecessor R (k-1) in series.



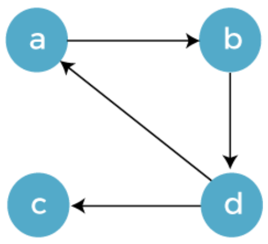
```
Algorithm Warshall(A[1..n,1..n])
//Implements Warshall's algorithm for computing the transitive closure
//Input: The Adjacency matrix A of a digraph with n vertices
//Output: The transitive closure of digraph
{
        $R^{(0)} \leftarrow A$
        for  k ← 1 to  n do
        {
                for  i ← 1 to  n do
                {
                      for j ← 1 to  n do
                      {
                            $R^{(k)}[i,j] \leftarrow R^{(k-1)}[i,j]$ or $R^{(k-1)}[i,k]$ and $R^{(k-1)}[k,j]$
                      }
                }
        }
        return $R^{(n)}$
}
```

Complexity: The time efficiency of Warshall's algorithm is in Θ (n3 )

**Program:**
```
#include<stdio.h>

void warshal(int A[][10],int n)
{
          for(int k=1;k<=n;k++)
              for(int i=1;i<=n;i++)
                  for(int j=1;j<=n;j++)
                      A[i][j]=A[i][j] || (A[i][k] && A[k][j]);
}

void main()
{
          int n, adj[10][10];
          printf("Enter no. of Vertices: ");
```

```
        scanf("%d",&n);
        printf("Enter the adjacency matrix\n");
        for(int i=1;i<=n;i++)
            for(int j=1;j<=n;j++)
                scanf("%d",&adj[i][j]);
        warshal(adj,n);

        printf("Transitive closure of the given graph is\n");
        for(int i=1;i<=n;i++)
        {
                for(int j=1;j<=n;j++)
                printf("%d ",adj[i][j]);
            printf("\n");
        }
}
```

## Sample Input and Output:



```
Enter no. of Vertices: 4 Enter
the adjacency matrix0 1 0 0
0 0 0 1
0 0 0 0
1 0 1 0
Transitive closure of the given graph is
     1  1  1  1
     1  1  1  1
     0  0  0  0
     1  1  1  1
```

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

**Single Source Shortest Paths Problem:**
For a given vertex called the source in a weighted connected graph, find the shortest paths to all its other vertices. Dijkstra's algorithm is the best known algorithm for the single source shortest paths problem. This algorithm is applicable to graphs with nonnegative weights only and finds the shortest paths to a graph's vertices in order of their distance from a given source. It finds the shortest path from the source to a vertex nearest to it, then to a second nearest, and so on. It is applicable to both undirected and directed graphs.

```
Algorithm : 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←œ; pv←null
                Insert(Q,v,dv)   //Initialise vertex priority queue 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)
                        }
                }
        }
}
```

Complexity: The Time efficiency for graphs represented by their weight matrix and the priority queue implemented as an unordered array and for graphs represented by their adjacency lists and the priority queue implemented as a min-heap, it is O(|E| log |V|).

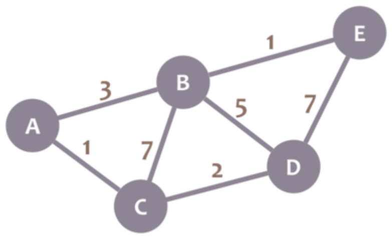
**Program:**

```
#include<stdio.h>

int cost[10][10],n,dist[10];


int minm(int m, int n)
{
        return(( m < n) ? m: n);
}
```

```
void dijkstra(int source)
{
        int s[10]={0};
        int min, w=0;

        for(int i=0;i<n;i++)
              dist[i]=cost[source][i];

        //Initialize dist from source to source as 0
        dist[source] = 0;

        //mark source vertex - estimated for its shortest path
        s[source] = 1;
        for(int i=0; i < n-1; i++)
        {
           //Find the nearest neighbour vertex
           min = 999;
           for(int j = 0; j < n; j++)
           {
                                 if ((s[j] == 0 ) && (min > dist[j]))
              {
                 min = dist[j];
                 w = j;
              }
           }
           s[w]=1;
           //Update the shortest path of neighbour of w
           for(int v=0;v<n;v++)
           {
                                 if(s[v]==0 && cost[w][v]!=999)
              {
                          dist[v]= minm(dist[v],dist[w]+cost[w][v]);
              }
           }
        }
}

int main()
{
    int source;

        printf("Enter the no.of vertices:");
        scanf("%d",&n);
        printf("Enter the cost matrix\n");
        for(int i=0;i<n;i++)
           for(int j=0;j<n;j++)
              scanf("%d",&cost[i][j]);

        printf("Enter the source vertex:");
        scanf("%d",&source);
        dijkstra(source);
```

```
            printf("the shortest distance is...");
            for(int i=0; i<n; i++)
                printf("Cost from %d to %d is %d\n",source,i,dist[i]);
}
```

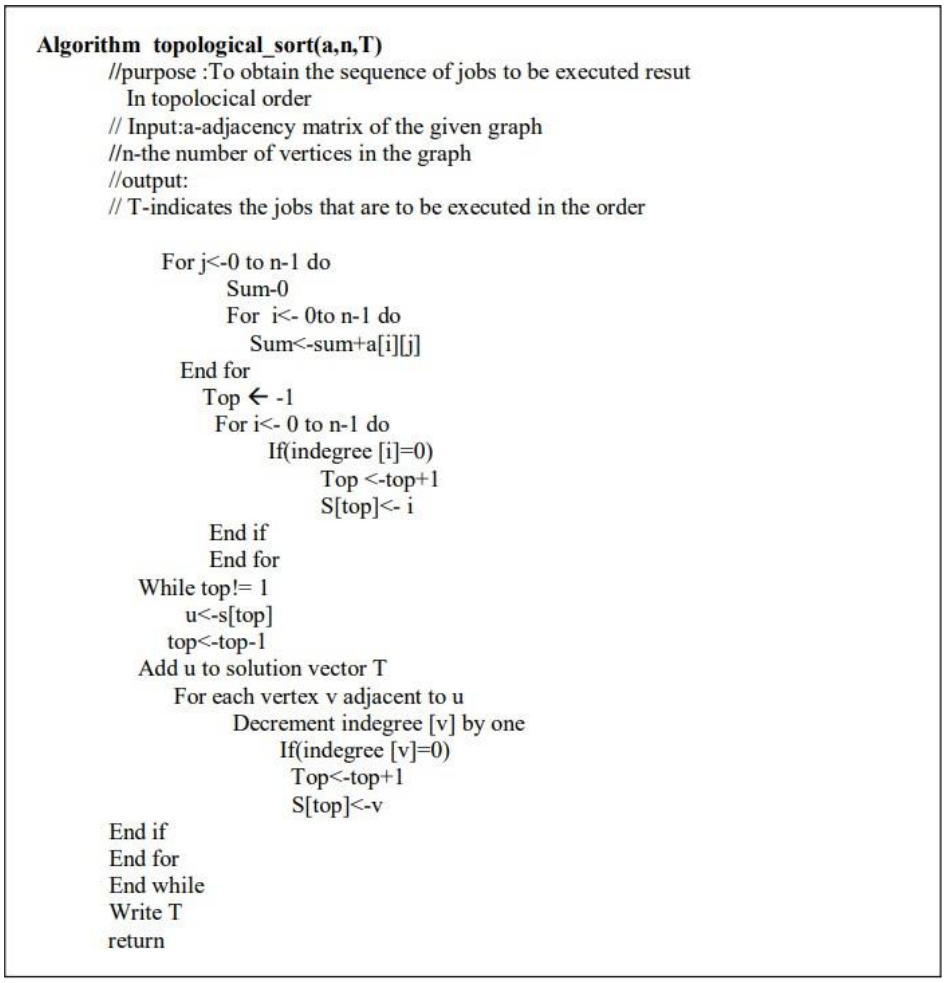
## Sample Input and Output:



```
Enter the no.of vertices:5
Enter the cost matrix
0 3 1 999 999
3 0 7 5 1
1 7 0 2 999
999 5 2 0 7
999 1 999 7 0
Enter the source vertex:0
the shortest distance is...Cost from 0 to 0 is 0
Cost from 0 to 1 is 3
Cost from 0 to 2 is 1
Cost from 0 to 3 is 3
Cost from 0 to 4 is 4
```

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

**Topological Ordering:**
This method is based on decrease and conquer technique a vertex with no incoming edges is selected and deleted along with no incoming edges is selected and deleted along with the outgoing edges. If there are several vertices with no incoming edges arbitrarily a vertex is selected.
The order in which the vertices are visited and deleted one by one results in topological sorting.
1) Find the vertices whose indegree is zero and place them on the stack
2) Pop a vertex u and it is the task to be done
3) Add the vertex u to the solution vector
4) Find the vertices v adjacent to vertex u. The vertices v represents the jobs which depend on job u
5) Decrement indegree[v] gives the number of depended jobs of v are reduced by one.



```
Algorithm  topological_sort(a,n,T)
      //purpose :To obtain the sequence of jobs to be executed resut
        In topolocical order
      // Input:a-adjacency matrix of the given graph
      //n-the number of vertices in the graph
      //output:
      // T-indicates the jobs that are to be executed in the order

            For j<-0 to n-1 do
                  Sum-0
                  For  i<- 0to n-1 do
                     Sum<-sum+a[i][j]
              End for
                Top ← -1
                 For i<- 0 to n-1 do
                      If(indegree [i]=0)
                           Top <-top+1
                           S[top]<- i
                 End if
                 End for
         While top!= 1
              u<-s[top]
           top<-top-1
         Add u to solution vector T
              For each vertex v adjacent to u
                    Decrement indegree [v] by one
                        If(indegree [v]=0)
                         Top<-top+1
                         S[top]<-v
      End if
      End for
      End while
      Write T
      return
```

Complexity: Complexity of topological sort is given by o[V*V]

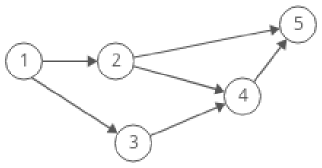
**Program:**

**//Solved using Source removal Method**

```
#include<stdio.h>

int cost[10][10],n,colsum[10];void

cal_colsum()
{
```

```
                for(int j=0;j<n;j++)
                {
                        colsum[j]=0;
                        for(int i=0;i<n;i++)
                                colsum[j]+=cost[i][j];
                }
}

void source_removal()
{
                int i,j,k,select[10]={0};
                printf("Topological ordering is:");
                for(i=0;i<n;i++)
                {
                        //Calculate the outdegree for each vertices
                        cal_colsum();
                        for(j=0;j<n;j++)
                        {
                                if(select[j]==0 && colsum[j]==0)//source vertex
                                        break;
                        }
                        printf("%d ",j);
                        select[j]=1;
                        //Remove source vertex j from cost matrix
                        for(k=0;k<n;k++)
                                cost[j][k]=0;
                }
}

void main()
{
          printf("Enter no. of Vertices: ");
          scanf("%d",&n);
          printf("Enter the cost matrix\n");
          for(int i=0;i<n;i++)
             for(int j=0;j<n;j++)
                scanf("%d",&cost[i][j]);
          source_removal();
}
```

## Sample Input and Output:



```
Enter no. of Vertices: 5
Enter the cost matrix
0 1 1 0 0
0 0 0 1 1
0 0 0 1 0
```

```
0 0 0 0 1
0 0 0 0 0
Topological ordering is:0 1 2 3 4
```

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

**0/1 Knapsack problem:**

Given: A set S of n items, with each item i having

- $b_i$ - a positive benefit
- $w_i$ - a positive weight

Goal: Choose items with maximum total benefit but with weight at most W. i.e.

- Objective: maximize $\sum_{i \in T} b_i$
- Constraint: $\sum_{i \in T} w_i \le W$

```
Algorithm: 0/1Knapsack(S, W)
//Input: set S of items with benefit b_i and weight w_i; max. weight W
//Output: benefit of best subset with weight at most W
// Sk: Set of items numbered 1 to k.
//Define B[k,w] = best selection from Sk with weight exactly equal to w
{
    for w ← 0 to n-1 do
            B[w] ← 0
    for k ← 1 to n do
    {
        for w ← W downto w_k do
        {
            if B[w-w_k]+b_k > B[w] then
                B[w] ← B[w-w_k]+b_k
        }
    }
}
```

**Complexity:** The Time efficiency and Space efficiency of 0/1 Knapsack algorithm is $\Theta(nW)$.

**Program:**

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

int max(int a, int b)
{
        return(a>b?a:b);
}

void knapsack_DP()
{
        int V[10][10],i,j;
        for(i=0;i<=n;i++)
           for(j=0;j<=m;j++)
              if(i==0 || j==0)
                 V[i][j]=0;
              else if(j<w[i])//weight of the item is larger than capacity
                 V[i][j]=V[i-1][j];
              else
```

```
                V[i][j]=max(V[i-1][j],p[i]+V[i-1][j-w[i]]);//maximization

        for(i=0;i<=n;i++)
        {
           for(j=0;j<=m;j++)
              printf("%d ",V[i][j]);
           printf("\n");
        }
        /* tracking back the optimal solution vector */
        printf("Items included are:");
        while(n > 0)
        {
           if(V[n][m] != V[n-1][m])
           {
              printf("%d ",n);
              m = m - w[n];
           }
           n--;
        }
}

int main()
{
        int i;
        printf("Enter the no. of items: ");
        scanf("%d",&n);
        printf("Enter the weights of n items: ");
        for(i=1;i<=n;i++)
           scanf("%d",&w[i]);
        printf("Enter the prices of n items: ");
        for(i=1;i<=n;i++)
           scanf("%d",&p[i]);
        printf("Enter the capacity of Knapsack: ");
        scanf("%d",&m);
        knapsack_DP();
}
```

Sample Input and Output:

```
Enter the no. of items: 4
Enter the weights of n items: 7 3 4 5
Enter the prices of n items: 42 12 40 25
Enter the capacity of Knapsack: 10
    0  0  0  0   0   0   0   0   0   0   0
    0  0  0  0   0   0   0  42  42  42   0
    0  0  0 12  12  12  12  42  42  42   0
    0  0  0 12  40  40  40  52  52  52   0
    0  0  0 12  40  40  40  52  52  65  65
Items included are: 4  3
```

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

```
#include<stdio.h>

int n,m,p[10],w[10];
void greedy_knapsack()
{
        float max, profit=0;
        int k=0,i,j;
        printf("item included is :");
        for(i=0;i<n;i++)
        {
           max=0;
           //choose the item which has highest price to weight ratio
           for(j=0;j<n;j++)
           {
              if(((float)p[j])/w[j] > max)
              {
                 k=j;
                 max=((float)p[j])/w[j];
              }
           }
           //kth element has highest price to weight ratio
           if(w[k] <= m )
           {
              printf("%d",k);
              m = m - w[k];
              profit=profit+p[k];
              p[k]=0;
           }
           else
              break;//unable fit item k into knapsack
        }
        printf("Discrete Knapsack profit = %f\n",profit);
        printf("Continuous Knapsack also includes item %d with portion: %f\n", k, (float)m)/w[k]);
        profit = profit + ((float)m)/w[k] * p[k];
        printf("Continuous Knapsack profit = %f\n",profit);
}

int main()
{
        int i;
        printf("Enter the no. of items: ");
        scanf("%d",&n);
        printf("Enter the weights of n items: ");
        for(i=0;i<n;i++)
           scanf("%d",&w[i]);
        printf("Enter the prices of n items: ");
        for(i=0;i<n;i++)
           scanf("%d",&p[i]);
        printf("Enter the capacity of Knapsack: ");
        scanf("%d",&m);
        greedy_knapsack();
}
```

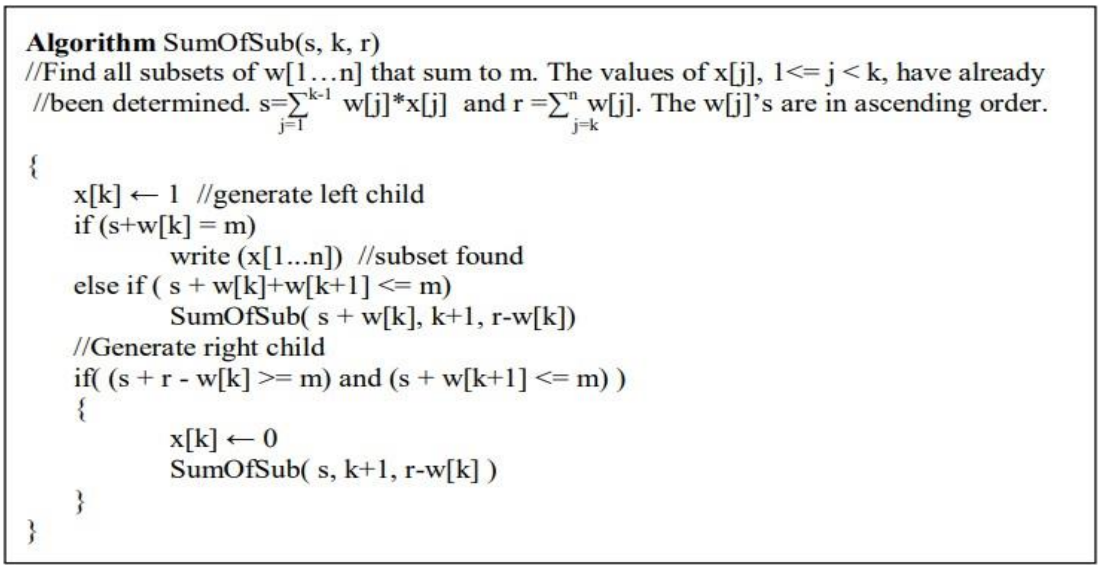
Sample Input and Output:

```
Enter the no. of items: 4
Enter the weights of n items: 2 1 3 2
Enter the prices of n items: 12 10 20 15
Enter the capacity of Knapsack: 5
item included is :1 3
Discrete Knapsack profit = 25.000000
Continuous Knapsack also includes item 2 with portion: 0.666667
Continuous Knapsack profit = 38.333332
```

**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.**

**Sum of Subsets**

Subset-Sum Problem is to find a subset of a given set S= {s1, s2… $s_n$} of n positive integers whose sum is equal to a given positive integer d. It is assumed that the set's elements are sorted in increasing order. The state-space tree can then be constructed as a binary tree and applying backtracking algorithm, the solutions could be obtained. Some instances of the problem may have no solutions



```
Algorithm SumOfSub(s, k, r)
//Find all subsets of w[1…n] that sum to m. The values of x[j], 1<= j < k, have already
//been determined. s=$\sum_{j=1}^{k-1}$ w[j]*x[j]  and r =$\sum_{j=k}^{n}$ w[j]. The w[j]'s are in ascending order.

{
    x[k] ← 1  //generate left child
    if (s+w[k] = m)
            write (x[1...n])  //subset found
    else if ( s + w[k]+w[k+1] <= m)
            SumOfSub( s + w[k], k+1, r-w[k])
    //Generate right child
    if( (s + r - w[k] >= m) and (s + w[k+1] <= m) )
    {
            x[k] ← 0
            SumOfSub( s, k+1, r-w[k] )
    }
}
```

Complexity: Subset sum problem solved using backtracking generates at each step maximal two new subtrees, and the running time of the bounding functions is linear, so the running time is O(2n ).

**Program:**

```
#include<stdio.h>

int x[10], w[10], count, d;
void sum_of_subsets(int s, int k, int rem)
{
        x[k] = 1;
        if( s + w[k] == d)
        {
           //if subset found
           printf("subset = %d\n", ++count);
           for(int i=0 ; i <= k ; i++)
               if ( x[i] == 1)
                   printf("%d ",w[i]);
           printf("\n");
        }
        else if ( s + w[k] + w[k+1] <= d )//left tree evaluation
           sum_of_subsets(s+w[k], k+1, rem-w[k]);

        if( ( s+rem-w[k] >= d) && ( s + w[k+1]) <= d)//right tree evaluation
        {
           x[k] = 0;
           sum_of_subsets(s,k+1,rem-w[k]);
        }
}

int main()
{
        int sum = 0,n;
        printf("enter no of elements:");
        scanf("%d",&n);
        printf("enter the elements in increasing order:");
        for( int i = 0; i < n ; i++)
        {
           scanf("%d",&w[i]);
              sum=sum+w[i];
        }
        printf("eneter the sum:");
        scanf("%d",&d);

        if ( ( sum < d ) || ( w[0] > d ) )
           printf("No subset possible\n");
        else
           sum_of_subsets(0,0,sum);
}
```

Sample Input and Output:

```
enter no of elements:5
enter the elements in increasing order:1 2 3 4 5
eneter the sum:10
subset = 1
```

```
1 2  3  4
subset =
 2 1 4  5
subset = 3
     2    3  5
```

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 valuesof n> 5000 and record the time taken to sort. Plot a graph of the time taken versus n. The elements can be read from a file or can be generated using the random number generator.**

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

int a[10000],n,count;void

selection_sort()
{
		for(int i=0;i<n-1;i++)
		{
			int min = i;
			for(int j=i+1;j<n;j++)
			{
				count++;
				if(a[j]<a[min])
					min=j;
			}
			int temp=a[i];
			a[i]=a[min];
			a[min]=temp;
		}
}

int main()
{
		printf("Enter the number of elements in an array:");
		scanf("%d",&n);
		printf("All the elements:");
		srand(time(0));
		for(int i=0;i<n;i++)
		{
			a[i]=rand();
			printf("%d ",a[i]);
		}
		selection_sort();
		printf("\nAfter sorting\n");
		for(int i=0;i<n;i++)
			printf("%d  ", a[i]);
		printf("\nNumber of basic operations = %d\n",count);
```

}

### Sample Input and Output:

```
Enter the number of elements in an array:5All
the elements:
24152 32742 28304 4804 22274
After sorting
4804 22274  24152  28304  32742
```

```
Number of basic operations = 10

Enter the number of elements in an array:10All
the elements:
24243 6017 4212 23217 16170 24802 1085 24280 9847 6392
After sorting
1085 4212  6017  6392  9847  16170  23217  24243  24280  24802
Number of basic operations = 45
```

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

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

int count=0;
int partition(int a[], int low,int high)
{
        int pivot=a[low],temp,i=low+1,j=high;
        while(1)
        {
           //Traverse i from left to right, segregating element of left group
           while(i<=high && a[i]<=pivot)//a[i]<=pivot used for avoiding multiple duplicates
           {
              i++; count++;
           }
           //Traverse j from right to left, segregating element of right group
           while(j>0 && a[j]>pivot)
           {
              j--; count++;
           }
           count+=2;
           //If grouping is incomplete
           if(i<j)
           {
              temp = a[i];
              a[i] = a[j];
              a[j] =temp;
           }
           else if(i>j)//If grouping is completed
           {
              temp = a[low];
              a[low] = a[j];
              a[j] = temp;
              return j;
           }
           else //Duplicate of Pivot found
              return j;
        }
}

void quicksort(int a[],int low, int high)
{
        int s;
        if(low<high)
        {
           //partition to place pivot element in between left and right group
           s = partition(a,low,high);
```

```
            quicksort(a,low,s-1);
            quicksort(a,s+1,high);
        }
}
int main()
{
                int a[10000],n;
                printf("Enter the number of elements in an array:");
                scanf("%d",&n);
                printf("All the elements:");
                srand(time(0));
                for(int i=0;i<n;i++)
                {
                        a[i]=rand();
                        printf("%d ",a[i]);
                }
                quicksort(a,0,n-1);
                printf("\nAfter sorting\n");
                for(int i=0;i<n;i++)
                        printf("%d  ", a[i]);
                printf("\nNumber of basic operations = %d\n",count);
}
```

Sample Input and Output:

```
Enter the number of elements in an array:5All
the elements:
24442 6310 12583 16519 22767
After sorting
6310 12583  16519  22767  24442
Number of basic operations = 18

Enter the number of elements in an array:10All
the elements:
24530 1605 3396 10868 6349 9906 12836 28823 21075 22418
After sorting
1605 3396  6349  9906  10868  12836  21075  22418  24530  28823
Number of basic operations = 44
```

**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 valuesof n> 5000, and record the time taken to sort. Plot a graph of the time taken versus n. The elements can be read from a file or can be generated using the random number generator.**

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

int count=0;
void merge(int a[], int low,int mid,int high)
{
        int i,j,k,c[10000];

        i=low, j=mid+1, k=0;
        while((i<=mid) && (j<=high))
        {
                count++;
                //choose the least element and store in Temporary array 'C'
                if(a[i]<a[j])
                        c[k++]=a[i++];
                else
                        c[k++]=a[j++];
        }

        //Copy the remaining array elements from any one of sub-array
        while(i<=mid)
                c[k++]=a[i++];
        while(j<=high)
                c[k++]=a[j++];
        for(i=low,j=0;j<k;i++, j++)
                a[i]=c[j];
}

void merge_sort(int a[], int low, int high)
{
        int mid;
        if(low < high)
        {
                //Divide the given array into 2 parts
                mid=(low+high)/2;
                merge_sort(a,low,mid);
                merge_sort(a,mid+1,high);
                merge(a,low,mid,high);
        }
}

int main()
{
        int a[10000],n,i;
        printf("Enter the number of elements in an array:");
        scanf("%d",&n);
        printf("All the elements:");
```

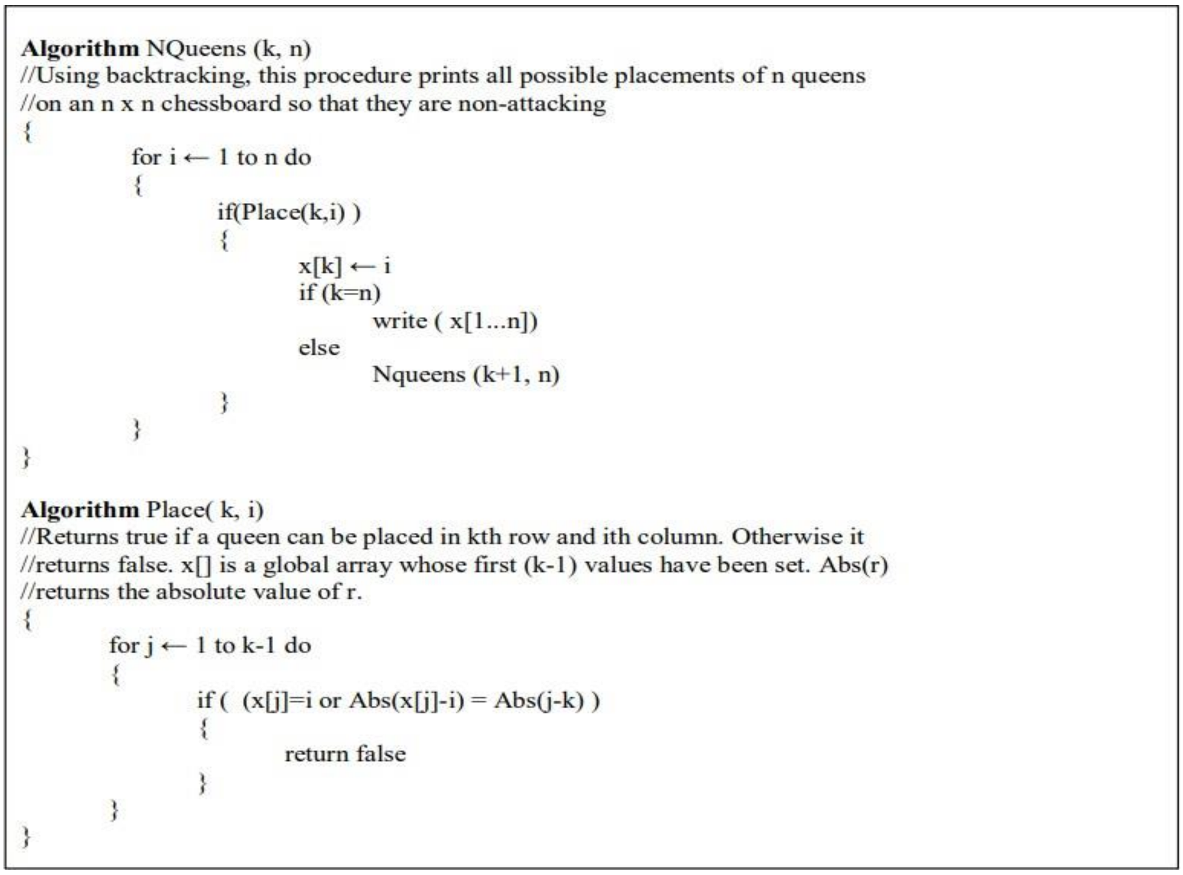
```
                srand(time(0));
                for(i=0;i<n;i++)
                {
                        a[i]=rand();
                        printf("%d ",a[i]);
                }
                merge_sort(a,0,n-1);
                printf("\nAfter sorting\n");
                for(i=0;i<n;i++)
                        printf("%d  ", a[i]);
                printf("\nNumber of basic operations = %d\n",count);
}
```

### Sample Input and Output:

```
Enter the number of elements in an array:5All
the elements:
24759 329 8704 24132 7473
After sorting
329  7473  8704  24132  24759
Number of basic operations = 8

Enter the number of elements in an array:10All
the elements:
24854 17121 2477 1072 11684 5437 26057 1167 17322 3583
After sorting
1072 1167  2477  3583  5437  11684  17121  17322  24854  26057
Number of basic operations = 22
```

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

**N Queen's problem:**
The n-queens problem consists of placing n queens on an n x n checker board in such a way that they do not threaten each other, according to the rules of the game of chess. Every queen on a checker square can reach the other squares that are located on the same horizontal, vertical, and diagonal line. So there can be at most one queen at each horizontal line, at most one queen at each vertical line, and at most one queen at each of the 4n-2 diagonal lines. Furthermore, since we want to place as many queens as possible, namely exactly n queens, there must be exactly one queen at each horizontal line and at each vertical line. The concept behind backtracking algorithm which is used to solve this problem is to successively place the queens in columns. When it is impossible to place a queen in a column (it is on the same diagonal, row, or column as another token), the algorithm backtracks and adjusts a preceding queen.



```
Algorithm NQueens (k, n)
//Using backtracking, this procedure prints all possible placements of n queens
//on an n x n chessboard so that they are non-attacking
{
        for i ← 1 to n do
        {
                if(Place(k,i) )
                {
                        x[k] ← i
                        if (k=n)
                                write ( x[1...n])
                        else
                                Nqueens (k+1, n)
                }
        }
}

Algorithm Place( k, i)
//Returns true if a queen can be placed in kth row and ith column. Otherwise it
//returns false. x[] is a global array whose first (k-1) values have been set. Abs(r)
//returns the absolute value of r.
{
        for j ← 1 to k-1 do
        {
                if (  (x[j]=i or Abs(x[j]-i) = Abs(j-k) )
                {
                        return false
                }
        }
}
```

Complexity: The power of the set of all possible solutions of the n queen’s problem is n! and the bounding function takes a linear amount of time to calculate, therefore the running time of the n queens problem is O (n!).

**Program:**

```
#include<stdio.h>
#include<math.h>          //for abs() function

int place(int x[],int k)
{
        for(int i=1;i<k;i++)
        {
                if( (x[i] == x[k]) || ( abs(x[i]-x[k]) == abs(i-k)) )
                        return 0;
        }
        return 1; //feasible
}
```

```
int nqueens(int n)
{
            int x[10], k, count=0;

            k=1;// select the first queen
            x[k]=0; //no positions allocated
            while(k != 0)  // until all queens are present
            {
                    x[k]++;      // place the kth queen in next column
                    while((x[k] <= n)  && (!place(x,k)))
                            x[k]++;     // check for the next column to place queen

                    if(x[k] <= n)
                    {
                            if(k == n)  // all queens are placed
                            {
                                    printf("\nSolution %d\n",++count);
                                    for(int i=1;i <= n;i++)
                                    {
                                       for(int j=1;j <= n;j++)
                                            printf("%c ",j==x[i]?'Q':'X');
                                            printf("\n");
                                    }
                            }
                            else
                            {

                                    ++k;                //select the next queen
                            x[k]=0;    // start from the next column
                            }
                    }
                    else
                            k--;             // backtrack
            }
```

```
        return count;
}

void main()
{
        int n;
        printf("Enter the size of chessboard: ");
        scanf("%d",&n);
        printf("\nThe number of possibilities are %d",nqueens(n));
}
```

### Sample Input and Output:

1. Enter the size of chessboard: 4

```
Solution 1
X Q X X
X X X Q
Q X X X
X X Q X

Solution 2
X X Q X
Q X X X
X X X Q
X Q X X

The number of possibilities are 2
```

2. Enter the size of chessboard: 3

```
The number of possibilities are 0
```

3. Enter the size of chessboard: 1

```
Solution 1
Q

The number of possibilities are 1
```