# PRACTICAL - 1

**Implement Linear search and Binary search and analyse their time complexity.**

```
import java.util.*;
class Timelinear {
    public static void main(String[] fgh) {
        double startTime = System.nanoTime();
        int[] arr = {3, 8, 11, 67, 33, 89, 32, 51, 44, 68};
        int size = arr.length;
        for (int i = 0; i < size; i++) {
            System.out.print(arr[i] + "\t");
        }
        System.out.println("");
        System.out.print("Enter target value: ");
        Scanner scv = new Scanner(System.in);
        int target = scv.nextInt();
        for (int i = 0; i < arr.length; i++) {
            if (target == arr[i]) {
                System.out.println("target found at index: " + i);
            }       }
        double endTime = System.nanoTime();
        double duration = (endTime - startTime) / 1000000;
        System.out.println("Execution time: " + duration + " miliseconds");
    }         }
```

**<u>OUTPUT: -</u>**

```
PS C:\Users\91708\Desktop\java folder> cd "c:\Users\91708\Desktop\java folder\D
3       8       11      67      33      89      32      51      44      68
Enter target value: 89

target found at index: 5
Execution time: 2357.8429 miliseconds
PS C:\Users\91708\Desktop\java folder\DAA>
```

# PRACTICAL - 2

**Implement following algorithm using array as a data structure and analyze its time complexity.**

**a. Bubble sort**

```
class BubbleTime {
    public static void main(String[] fgh) {
        double startTime = System.nanoTime();
        int[] arr = {3, 8, 22, 9, 1, 5, 67, 34, 90, 23};
        int size = arr.length;
        System.out.println("Array before sorting");
        for (int i = 0; i < size; i++) {
            System.out.print(arr[i] + "\t");
        }
        System.out.println("");
        System.out.println("Array After sorting");
        for (int i = 0; i < size; i++) {
            for (int j = 1; j < size - i; j++) {
                if (arr[j] < arr[j - 1]) {
                    int temp = arr[j];
                    arr[j] = arr[j - 1];
                    arr[j - 1] = temp;
                }
            }
        }
        for (int i = 0; i < size; i++) {
            System.out.print(arr[i] + "\t");
        }
        double endTime = System.nanoTime();
        double duration = (endTime - startTime) / 1000000;
```

```
            System.out.println("");
            System.out.println("Execution time: " + duration + " miliseconds");
        }
    }
```

**OUTPUT: -**

```
PS C:\Users\91708\Desktop\java folder> cd "c:\Users\91708\Desktop\java folder\DA
Array before sorting
3       8       22      9       1       5       67      34      90      23
Array After sorting
1       3       5       8       9       22      23      34      67      90
Execution time: 11.8999 miliseconds
PS C:\Users\91708\Desktop\java folder\DAA>
```

**b. Selection Sort**

```
public class SelectionTime {
    public static void main(String[] args) {
        double startTime = System.nanoTime();
        int[] array = {64, 25, 12, 22, 56, 89, 12, 90, 11};
        int n = array.length;
        System.out.println("Original array");
        for (int element : array) {
            System.out.print(element + "\t");
        }
        System.out.println("");
        for (int i = 0; i < n - 1; i++) {
            int minIndex = i;
            for (int j = i + 1; j < n; j++) {
                if (array[j] < array[minIndex]) {
                    minIndex = j;
                }
            }
```

```
            int temp = array[minIndex];
            array[minIndex] = array[i];
            array[i] = temp;
        }
        System.out.println("Sorted array");
        for (int element : array) {
            System.out.print(element + "\t");
        }
        System.out.println();
        double endTime = System.nanoTime();
        double duration = (endTime - startTime) / 1000000;
        System.out.println("");
        System.out.println("Execution time: " + duration + " miliseconds");
    }
}
```

**OUTPUT: -**

```
PS C:\Users\91708\Desktop\java folder> cd "c:\Users\91708\Desktop\java
Original array
64      25      12      22      56      89      12      90      11
Sorted array
11      12      12      22      25      56      64      89      90

Execution time: 22.0294 miliseconds
PS C:\Users\91708\Desktop\java folder\DAA>
```

**c. Insertion Sort**

```
class InsertionTime {
    public static void main(String[] fgygh) {
        double startTime = System.nanoTime();
        int[] array = {13, 23, 56, 12, 69, 41, 35, 89, 54};
        int n = array.length;
        System.out.println("Original array");
```

```
        printArray(array);
        for (int i = 1; i < n; ++i) {
            int key = array[i];
            int j = i - 1;
            while (j >= 0 && array[j] > key) {
                array[j + 1] = array[j];
                j = j - 1;
            }
            array[j + 1] = key;
        }
        System.out.println("Sorted array:");
        printArray(array);
        double endTime = System.nanoTime();
        double duration = (endTime - startTime) / 1000000;
        System.out.println("");
        System.out.println("Execution time: " + duration + " miliseconds");
    }
    public static void printArray(int[] array) {
        for (int element : array) {
            System.out.print(element + "\t");
        }
        System.out.println();
    }
}
```

**OUTPUT: -**

```
PS C:\Users\91708\Desktop\java folder> cd "c:\Users\91708\Desktop\jav
Original array
13      23      56      12      69      41      35      89      54
Sorted array:
12      13      23      35      41      54      56      69      89

Execution time: 23.641 miliseconds
PS C:\Users\91708\Desktop\java folder\DAA>
```

# PRACTICAL - 3

**Implement Randomized Binary Search.**

```
import java.util.Random;
public class RandomiseBinary {
    public static void main(String[] args) {
        int[] arr = {11, 24, 39, 43, 45, 56, 72, 88, 89, 110};
        int target = 72;

        System.out.println("Input Sorted Array");
        for (int element : arr) {
            System.out.print(element + "\t");
        }
        System.out.println("");

        int result = randomizedBinarySearch(arr, 0, arr.length - 1, target);

        if (result != -1) {
            System.out.println("Element found at index: " + result);
        } else {
            System.out.println("Element not found");
        }
    }
    public static int randomizedBinarySearch(int[] arr, int left, int right, int target) {
        if (right >= left) {
            Random rand = new Random();
            int randomPivot = left + rand.nextInt(right - left + 1);
```

```
            if (arr[randomPivot] == target) {
                return randomPivot;
            }
            if (arr[randomPivot] > target) {
                return randomizedBinarySearch(arr, left, randomPivot - 1, target);
            }
            return randomizedBinarySearch(arr, randomPivot + 1, right, target);
        }
        return -1;
    }
}
```

**OUTPUT: -**

```
PS C:\Users\91708\Desktop\java folder> cd "c:\Users\91708\Desktop\java folder\
Input Sorted Array
11      24      39      43      45      56      72      88      89      110
Element found at index: 6
PS C:\Users\91708\Desktop\java folder\DAA>
```

# PRACTICAL - 4

**Implement Strassen's matrix multiplication algorithm.**

```
public class StrassenMulti {
    public static int[][] add(int[][] A, int[][] B) {
        int n = A.length;
        int[][] result = new int[n][n];
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                result[i][j] = A[i][j] + B[i][j];
            }
        }
        return result;
    }
    public static int[][] subtract(int[][] A, int[][] B) {
        int n = A.length;
        int[][] result = new int[n][n];
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                result[i][j] = A[i][j] - B[i][j];
            }
        }
        return result;
    }
    public static int[][] strassenMultiply(int[][] A, int[][] B) {
        int n = A.length;
        if (n == 1) {
            // Base case: single element multiplication
            int[][] result = new int[1][1];
```

```
    result[0][0] = A[0][0] * B[0][0];
    return result;
}
// Splitting matrices into 4 sub-matrices each
int newSize = n / 2;
int[][] A11 = new int[newSize][newSize];
int[][] A12 = new int[newSize][newSize];
int[][] A21 = new int[newSize][newSize];
int[][] A22 = new int[newSize][newSize];
int[][] B11 = new int[newSize][newSize];
int[][] B12 = new int[newSize][newSize];
int[][] B21 = new int[newSize][newSize];
int[][] B22 = new int[newSize][newSize];

// Dividing matrix A into sub-matrices
for (int i = 0; i < newSize; i++) {
    for (int j = 0; j < newSize; j++) {
        A11[i][j] = A[i][j];
        A12[i][j] = A[i][j + newSize];
        A21[i][j] = A[i + newSize][j];
        A22[i][j] = A[i + newSize][j + newSize];
        B11[i][j] = B[i][j];
        B12[i][j] = B[i][j + newSize];
        B21[i][j] = B[i + newSize][j];
        B22[i][j] = B[i + newSize][j + newSize];
    }
}
int[][] M1 = strassenMultiply(add(A11, A22), add(B11, B22));
int[][] M2 = strassenMultiply(add(A21, A22), B11);
```

```java
    int[][] M3 = strassenMultiply(A11, subtract(B12, B22));
    int[][] M4 = strassenMultiply(A22, subtract(B21, B11));
    int[][] M5 = strassenMultiply(add(A11, A12), B22);
    int[][] M6 = strassenMultiply(subtract(A21, A11), add(B11, B12));
    int[][] M7 = strassenMultiply(subtract(A12, A22), add(B21, B22));

    // Calculating the final sub-matrices of C
    int[][] C11 = add(subtract(add(M1, M4), M5), M7);
    int[][] C12 = add(M3, M5);
    int[][] C21 = add(M2, M4);
    int[][] C22 = add(subtract(add(M1, M3), M2), M6);

    // Combining the 4 sub-matrices into a single matrix
    int[][] C = new int[n][n];
    for (int i = 0; i < newSize; i++) {
        for (int j = 0; j < newSize; j++) {
            C[i][j] = C11[i][j];
            C[i][j + newSize] = C12[i][j];
            C[i + newSize][j] = C21[i][j];
            C[i + newSize][j + newSize] = C22[i][j];
        }
    }
    return C;
}
public static void printMatrix(int[][] matrix) {
    for (int[] row : matrix) {
        for (int elem : row) {
            System.out.print(elem + " ");
        }
```

```
            System.out.println();
        }
    }
    public static void main(String[] args) {
        int[][] A = {{1, 3, 5, 7}, {2, 4, 6, 8}, {9, 11, 13, 15}, {10, 12, 14, 16}};
        System.err.println("First Matrix");
        printMatrix(A);
        System.err.println("");

        int[][] B = {{16, 14, 12, 10}, {15, 13, 11, 9}, {8, 6, 4, 2}, {7, 5, 3, 1}};
        System.err.println("Second Matrix");
        printMatrix(B);
        System.err.println("");

        int[][] C = strassenMultiply(A, B);
        System.out.println("Result of Strassen's Matrix Multiplication:");
        printMatrix(C);
    }
}
```

**OUTPUT: -**

```
PS C:\Users\91708\Desktop\java folder> cd "c:\Users\9170
First Matrix
1 3 5 7
2 4 6 8
9 11 13 15
10 12 14 16

Second Matrix
16 14 12 10
15 13 11 9
8 6 4 2
7 5 3 1

Result of Strassen's Matrix Multiplication:
150 118 86 54
196 156 116 76
518 422 326 230
564 460 356 252
PS C:\Users\91708\Desktop\java folder\DAA>
```

## PRACTICAL - 5

**Implement Radix sort and analyse its complexity.**

```
import java.util.Arrays;

public class RadixSort {

  static int getMax(int arr[], int n) {
    int max = arr[0];

    for (int i = 1; i < n; i++) {
      if (arr[i] > max) {
        max = arr[i];
      }
    }
    return max;
  }

  static void countSort(int arr[], int n, int exp) {
    int output[] = new int[n];
    int count[] = new int[10];
    Arrays.fill(count, 0);

    for (int i = 0; i < n; i++) {
      count[(arr[i] / exp) % 10]++;
    }
    for (int i = 1; i < 10; i++) {
      count[i] += count[i - 1];
    }
```

```
    for (int i = n - 1; i >= 0; i--) {
        int index = (arr[i] / exp) % 10;
        output[count[index] - 1] = arr[i];
        count[index]--;
    }
    System.arraycopy(output, 0, arr, 0, n);
}

static void radixSort(int arr[], int n) {
    int max = getMax(arr, n);

    for (int exp = 1; max / exp > 0; exp *= 10) {
        countSort(arr, n, exp);
    }
}

static void printArray(int arr[], int n) {
    for (int i = 0; i < n; i++) {
        System.out.print(arr[i] + " ");
    }
    System.out.println();
}

public static void main(String[] args) {
    double startTime = System.nanoTime();

    int arr[] = {170, 45, 75, 90, 802, 24, 52, 66, 89};
    int n = arr.length;
```

```
        System.out.println("Original array");
        printArray(arr, n);
        System.out.println("");

        radixSort(arr, n);

        System.out.println("Sorted array");
        printArray(arr, n);

        double endTime = System.nanoTime();
        double duration = (endTime - startTime) / 1000000;
        System.out.println("");
        System.out.println("Execution time: " + duration + " miliseconds");
    }
}
```

**OUTPUT: -**

```
PS C:\Users\91708\Desktop\java folder> cd "c:\Users\91708\Des
Original array
170 45 75 90 802 24 52 66 89

Sorted array
24 45 52 66 75 89 90 170 802

Execution time: 11.8695 miliseconds
PS C:\Users\91708\Desktop\java folder\DAA>
```

# PRACTICAL - 6

**Write a program to find minimum cost spanning tree using Prim's Algorithm.**

```
import java.util.*;
public class Prims {
    private static final int INF = Integer.MAX_VALUE;

    // Function to find the vertex with the minimum key value
    private int minKey(int[] key, boolean[] mstSet, int vertices) {
        int min = INF, minIndex = -1;

        for (int v = 0; v < vertices; v++) {
            if (!mstSet[v] && key[v] < min) {
                min = key[v];
                minIndex = v;
            }
        }
        return minIndex;
    }

    private void printMST(int[] parent, int[][] graph, int vertices) {
        System.out.println("Edge \tWeight");
        int totalWeight = 0;
        for (int i = 1; i < vertices; i++) {
            System.out.println(parent[i] + " - " + i + "\t" + graph[i][parent[i]]);
            totalWeight += graph[i][parent[i]];
        }
        System.out.println("Total Weight of MST: " + totalWeight);
    }
```

```
    // Function to construct and print MST using Prim's algorithm
    public void primMST(int[][] graph, int vertices) {
        int[] parent = new int[vertices];             // Array to store the MST
        int[] key = new int[vertices];                // Array to store minimum edge weight
        boolean[] mstSet = new boolean[vertices];    // Boolean array to represent the set of
vertices included in MST

        Arrays.fill(key, INF);
        Arrays.fill(mstSet, false);

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

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

            // Update key value and parent index of the adjacent vertices
            for (int v = 0; v < vertices; v++) {
                if (graph[u][v] != 0 && !mstSet[v] && graph[u][v] < key[v]) {
                    parent[v] = u;
                    key[v] = graph[u][v];
                }
            }
        }
        printMST(parent, graph, vertices);
    }
```

```
        public static void main(String[] args) {
            Prims mst = new Prims();

            int[][] graph = {
                {0, 3, 0, 0, 0, 1, 0, 0},
                {3, 0, 5, 0, 0, 4, 7, 0},
                {0, 5, 0, 6, 0, 0, 8, 0},
                {0, 0, 6, 0, 9, 0, 0, 4},
                {0, 0, 0, 9, 0, 2, 0, 5},
                {1, 4, 0, 0, 2, 0, 6, 0},
                {0, 7, 8, 0, 0, 6, 0, 3},
                {0, 0, 0, 4, 5, 0, 3, 0}
            };

            int vertices = graph.length;
            mst.primMST(graph, vertices);
        }
    }
```

**OUTPUT: -**

```
PS C:\Users\91708\Desktop\java folder> cd "c:\Use
Edge    Weight
0 - 1   3
1 - 2   5
7 - 3   4
5 - 4   2
0 - 5   1
7 - 6   3
4 - 7   5
Total Weight of MST: 23
```

# PRACTICAL - 7

**Write a program to find solution for knapsack problem using greedy method.**

```
import java.util.Arrays;
import java.util.Comparator;
class Item {
    int weight;
    int value;
    public Item(int weight, int value) {
        this.weight = weight;
        this.value = value;
    }
}
class Knapsack {
    // Function to calculate the maximum value achievable with given capacity
    public static double getMaxValue(Item[] items, int capacity) {
        // Sort items by value-to-weight ratio in descending order
        Arrays.sort(items, new Comparator<Item>() {
            public int compare(Item a, Item b) {
                double r1 = (double) a.value / a.weight;
                double r2 = (double) b.value / b.weight;
                return Double.compare(r2, r1);
            }
        });
        double totalValue = 0.0;
        for (Item item : items) {
            // If item can be added in whole
            if (item.weight <= capacity) {
                capacity -= item.weight;
```

```
                totalValue += item.value;
            }
            // Otherwise, add fraction of the item to maximize value
            else {
                double fraction = (double) capacity / item.weight;
                totalValue += item.value * fraction;
                break;
            }
        }
        return totalValue;
    }
    public static void main(String[] args) {
        // Define the weight capacity of the knapsack
        int capacity = 50;
        // Define the items (weight, value)
        Item[] items = {
            new Item(10, 60),
            new Item(20, 100),
            new Item(30, 120)
        };

        double maxValue = getMaxValue(items, capacity);
        System.out.println("Maximum value in the knapsack: " + maxValue);
    }
}
```

**OUTPUT: -**

```
PS C:\Users\91708\Desktop\java folder> cd "c:\Users\
Maximum value in the knapsack: 240.0

PS C:\Users\91708\Desktop\java folder\DAA>
```

## PRACTICAL - 8

**Implement All-Pairs Shortest Paths Problem using Floyd's algorithm.**

```
public class FloydWarshall {
    private static final int INF = 99999;

    public void floydWarshall(int[][] graph) {
        int vertices = graph.length;
        int[][] dist = new int[vertices][vertices];

        // Initialize distance matrix with the given graph values
        for (int i = 0; i < vertices; i++) {
            for (int j = 0; j < vertices; j++) {
                dist[i][j] = graph[i][j];
            }
        }

        // Run the Floyd-Warshall algorithm
        for (int k = 0; k < vertices; k++) {
            for (int i = 0; i < vertices; i++) {
                for (int j = 0; j < vertices; j++) {
                    // Check if the vertex k is on the shortest path between i and j
                    if (dist[i][k] != INF && dist[k][j] != INF && dist[i][k] + dist[k][j] <
dist[i][j]) {
                        dist[i][j] = dist[i][k] + dist[k][j];
                    }
                }
            }
        }
```

```
    printSolution(dist);
}

private void printSolution(int[][] dist) {
    System.out.println("Shortest distances between every pair of vertices:");
    for (int i = 0; i < dist.length; i++) {
        for (int j = 0; j < dist.length; j++) {
            if (dist[i][j] == INF) {
                System.out.print("INF ");
            } else {
                System.out.print(dist[i][j] + "   ");
            }
        }
        System.out.println();
    }
}

public static void main(String[] args) {
    FloydWarshall fw = new FloydWarshall();

    int[][] graph = {
        {0, 7, INF, INF, 3, 10},
        {7, 0, 4, 10, 2, 6},
        {INF, 4, 0, 2, INF, INF},
        {INF, 10, 2, 0, 11, 9},
        {3, 2, INF, 11, 0, 1},
        {10, 6, INF, 9, 1, 0}
    };
```

```
        fw.floydWarshall(graph);
    }
}
```

**OUTPUT: -**

```
PS C:\Users\91708\Desktop\java folder> cd "c:\User
Shortest distances between every pair of vertices:
0   5   9   11   3   4
5   0   4   6   2   3
9   4   0   2   6   7
11   6   2   0   8   9
3   2   6   8   0   1
4   3   7   9   1   0
PS C:\Users\91708\Desktop\java folder\DAA>
```

## PRACTICAL - 9

**Write a program to find minimum cost spanning tree using Kruskal's Algorithm.**

```
import java.util.*;
class Edge implements Comparable<Edge> {
   int src, dest, weight;
   public Edge(int src, int dest, int weight) {
      this.src = src;
      this.dest = dest;
      this.weight = weight;
   }
   // Compare edges by their weight
   public int compareTo(Edge other) {
      return this.weight - other.weight;
   }
}
class Subset {
   int parent, rank;
}
public class Krushkal {
   private int vertices;
   private List<Edge> edges;

   public Krushkal(int vertices) {
      this.vertices = vertices;
      this.edges = new ArrayList<>();
   }
   // Add an edge to the graph
   public void addEdge(int src, int dest, int weight) {
```

```
        edges.add(new Edge(src, dest, weight));
    }
    // Find set of an element i (uses path compression technique)
    private int find(Subset[] subsets, int i) {
        if (subsets[i].parent != i) {
            subsets[i].parent = find(subsets, subsets[i].parent);
        }
        return subsets[i].parent;
    }
    // Union of two sets (uses union by rank)
    private void union(Subset[] subsets, int x, int y) {
        int rootX = find(subsets, x);
        int rootY = find(subsets, y);

        if (subsets[rootX].rank < subsets[rootY].rank) {
            subsets[rootX].parent = rootY;
        } else if (subsets[rootX].rank > subsets[rootY].rank) {
            subsets[rootY].parent = rootX;
        } else {
            subsets[rootY].parent = rootX;
            subsets[rootX].rank++;
        }
    }
    // Kruskal's algorithm to find the Minimum Spanning Tree
    public void kruskalMST() {
        List<Edge> mst = new ArrayList<>();
        Collections.sort(edges);

        // Create subsets for union-find
```

```
        Subset[] subsets = new Subset[vertices];
        for (int i = 0; i < vertices; i++) {
            subsets[i] = new Subset();
            subsets[i].parent = i;
            subsets[i].rank = 0;
        }

        for (Edge edge : edges) {
            int x = find(subsets, edge.src);
            int y = find(subsets, edge.dest);

            // If including this edge does not cause a cycle, include it in the result
            if (x != y) {
                mst.add(edge);
                union(subsets, x, y);
            }
        }

        System.out.println("Edges in the Minimum Spanning Tree:");
        int totalWeight = 0;
        for (Edge edge : mst) {
            System.out.println(edge.src + " - " + edge.dest + " \tWeight: " + edge.weight);
            totalWeight += edge.weight;
        }
        System.out.println("Total weight of MST: " + totalWeight);
    }

    public static void main(String[] args) {
        int vertices = 8;
```

```java
        Krushkal graph = new Krushkal(vertices);

        // Example graph: add edges (src, dest, weight)
        graph.addEdge(0, 1, 3);
        graph.addEdge(0, 5, 1);
        graph.addEdge(1, 2, 5);
        graph.addEdge(1, 5, 4);
        graph.addEdge(1, 6, 7);
        graph.addEdge(2, 3, 6);
        graph.addEdge(2, 6, 8);
        graph.addEdge(3, 4, 9);
        graph.addEdge(3, 7, 4);
        graph.addEdge(4, 5, 2);
        graph.addEdge(4, 7, 5);
        graph.addEdge(5, 6, 6);
        graph.addEdge(6, 7, 3);

        graph.kruskalMST();
    }
}
```

**OUTPUT: -**

```
PS C:\Users\91708\Desktop\java folder> cd "c:\Users\9
Edges in the Minimum Spanning Tree:
0 - 5   Weight: 1
4 - 5   Weight: 2
0 - 1   Weight: 3
6 - 7   Weight: 3
3 - 7   Weight: 4
1 - 2   Weight: 5
4 - 7   Weight: 5
Total weight of MST: 23
```

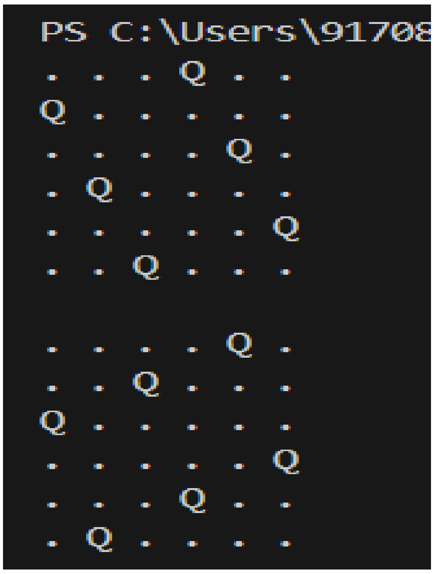
# PRACTICAL - 10

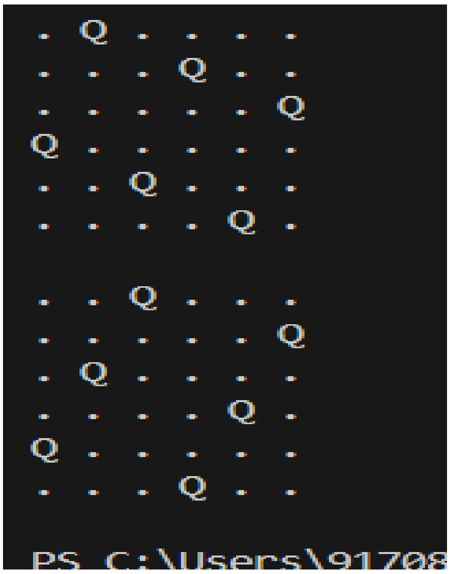
**Write a program to solve N-QUEENS problem.**

```
public class NQueens {

    private int N;
    private int[][] board;

    public NQueens(int N) {
        this.N = N;
        board = new int[N][N];
    }

    private void printSolution() {
        for (int i = 0; i < N; i++) {
            for (int j = 0; j < N; j++) {
                System.out.print(board[i][j] == 1 ? "Q " : ". ");

            }
            System.out.println();
        }
        System.out.println();
    }
    // Helper function to check if a queen can be placed at board[row][col]
    private boolean isSafe(int row, int col) {
        // Check left side of the current row
        for (int i = 0; i < col; i++) {
            if (board[row][i] == 1) {
                return false;
            }
        }
        // Check upper diagonal on the left side
        for (int i = row, j = col; i >= 0 && j >= 0; i--, j--) {
            if (board[i][j] == 1) {
                return false;
            }
        }
        // Check lower diagonal on the left side
        for (int i = row, j = col; j >= 0 && i < N; i++, j--) {
            if (board[i][j] == 1) {
                return false;
            }
        }

        return true;
    }
    private boolean solveNQueens(int col) {
        if (col >= N) {
```

```
            printSolution();
            return true;
        }

        boolean result = false;
        for (int i = 0; i < N; i++) {
            // Check if it's safe to place a queen at board[i][col]
            if (isSafe(i, col)) {
                board[i][col] = 1; // Place the queen

                // Recursively place the queen in the next column
                result = solveNQueens(col + 1) || result;

                // Backtrack and remove the queen from board[i][col]
                board[i][col] = 0;
            }
        }
        return result;
    }

    public void solve() {
        if (!solveNQueens(0)) {
            System.out.println("No solution exists.");
        }
    }

    public static void main(String[] args) {
        int N = 6;
        NQueens nQueens = new NQueens(N);
        nQueens.solve();
    }
}
```

**OUTPUT:-**



## PRACTICAL - 11

**Write a program to solve Sum of subsets problem for a given set of distinct numbers.**

```
import java.util.ArrayList;
import java.util.List;

public class SumofSubsets {

    public static void findSubsets(int[] set, int targetSum) {
        List<Integer> subset = new ArrayList<>();
        findSubsetsHelper(set, subset, targetSum, 0);
    }

    // Helper function to generate subsets using backtracking
    private static void findSubsetsHelper(int[] set, List<Integer> subset, int targetSum, int
index) {
        if (targetSum == 0) {
            System.out.println(subset);
            return;
        }

        // If targetSum becomes negative or we reach the end, stop the recursion
        if (targetSum < 0 || index == set.length) {
            return;
        }

        // Include the current element and recurse
        subset.add(set[index]);
        findSubsetsHelper(set, subset, targetSum - set[index], index + 1);
```

```java
            // Exclude the current element and recurse
            subset.remove(subset.size() - 1);
            findSubsetsHelper(set, subset, targetSum, index + 1);
        }

        public static void main(String[] args) {

            int[] set = {10, 7, 5, 18, 12, 20, 15, 28};
            int targetSum = 35;

            System.out.println("Subsets that sum to " + targetSum + ":");
            findSubsets(set, targetSum);
        }
    }
```

**OUTPUT: -**

```
PS C:\Users\91708\Desktop\java folder>
Subsets that sum to 35:
[10, 7, 18]
[10, 5, 20]
[7, 28]
[5, 18, 12]
[20, 15]
```