a) Implement Merge sort algorithm and plot its time complexity with reference to the size of the input.

```
import numpy as np

import matplotlib.pyplot as plt

import time


class MS:

    def sort(self, arr, low, high):

        if low < high:

            mid = (low + high) // 2

            self.sort(arr, low, mid)

            self.sort(arr, mid + 1, high)

            self.merge(arr, low, mid, high)


    def merge(self, arr, low, mid, high):

        left = arr[low:mid + 1]

        right = arr[mid + 1:high + 1]

        i = j = 0

        k = low

        while i < len(left) and j < len(right):

            if left[i] < right[j]:

                arr[k] = left[i]

                i += 1

            else:

                arr[k] = right[j]

                j += 1

            k += 1

        while i < len(left):

            arr[k] = left[i]

            i += 1
```

```
            k += 1
        while j < len(right):
            arr[k] = right[j]
            j += 1
            k += 1

def time_measures(sizes, trials):
    times = []
    for size in sizes:
        total_time = 0
        for _ in range(trials):
            arr = np.random.randint(1, 100, size=size)
            start = time.time()
            MS().sort(arr, 0, len(arr) - 1)
            total_time += time.time() - start
        times.append(total_time / trials)
    return times

if __name__ == "__main__":
    # Input array from the user
    user_arr = list(map(int, input("Enter numbers to sort (comma-separated): ").split(',')))
    ms = MS()
    ms.sort(user_arr, 0, len(user_arr) - 1)
    print("Sorted Array:", user_arr)

    # Time complexity for different sizes
    sizes = [10, 100, 500, 1000]
    trials = int(input("Enter the number of trials for each size: "))
    times = time_measures(sizes, trials)

    # Plotting the time complexity graph
```

```python
plt.plot(sizes, times, marker='o', label="Merge Sort")
plt.xlabel('Input Size')
plt.ylabel('Time (seconds)')
plt.title('Merge Sort Time Complexity')
plt.legend()
plt.grid(True)
plt.show()
```

b) Implement Quick sort algorithm and plot its time complexity regarding asymptotic notations (Best, average, and worst).

```
import numpy as np

import matplotlib.pyplot as plt

import time

class QS:

    def sort(self, a, low, high):

        if low < high:

            p = self.part(a, low, high)

            self.sort(a, low, p - 1)

            self.sort(a, p + 1, high)


    def part(self, a, low, high):

        mid = (low + high) // 2

        a[mid], a[high] = a[high], a[mid]

        p = a[high]

        i = low - 1

        for j in range(low, high):

            if a[j] < p:

                i += 1

                a[i], a[j] = a[j], a[i]

        a[i + 1], a[high] = a[high], a[i + 1]

        return i + 1


def time_measures(sizes, case='avg'):

    times = []

    for size in sizes:

        if case == 'best':

            a = np.arange(size)

        elif case == 'worst':

            a = np.arange(size, 0, -1)
```

```
        else:
            a = np.random.randint(1, 100, size=size)

        start = time.time()
        QS().sort(a, 0, len(a) - 1)
        times.append(time.time() - start)
    return times

if __name__ == "__main__":
    # Get input aay from the user and display sorted output
    user_arr = list(map(int, input("Enter numbers to sort (comma-separated): ").split(',')))
    qs = QS()
    qs.sort(user_arr, 0, len(user_arr) - 1)
    print("Sorted array:", user_arr)

    # Define sizes for time complexity measurement and plot the graph
    sizes = [10, 100, 500, 1000]
    best = time_measures(sizes, 'best')
    avg = time_measures(sizes, 'avg')
    worst = time_measures(sizes, 'worst')

    # Plotting time complexity graph
    plt.plot(sizes, best, marker='o', label="Best Case")
    plt.plot(sizes, avg, marker='o', label="Average Case")
    plt.plot(sizes, worst, marker='o', label="Worst Case")
    plt.xlabel('Input Size')
    plt.ylabel('Time (seconds)')
    plt.title('Quick Sort Time Complexity')
    plt.legend()
    plt.grid(True)
    plt.show()
```

Write a program to identify the articulation points present in a graph.

```
import java.util.*;

public class Articulation {
   private int n;
   private int[][] arr;
   private int[] dfn;
   private int[] low;
   private int num;
   private Set<Integer> s;

   public Articulation(int n) {
      this.n = n;
      arr = new int[n][n];
      dfn = new int[n];
      low = new int[n];
      num = 1;
      s = new HashSet<>();
   }

   public void read(Scanner scan) {
      for (int i = 0; i < n; i++) {
         for (int j = 0; j < n; j++)
            arr[i][j] = scan.nextInt();
         dfn[i] = 0;
      }
   }

   public void art(int u, int v) {
      dfn[u] = num;
```

```
    low[u] = num;
    num++;
    int child = 0;
    for (int j = 0; j < n; j++) {
        if (arr[u][j] == 1 && dfn[j] == 0) {
            if (v == -1)
                child++;
            art(j, u);
            if (v != -1 && low[j] >= dfn[u])
                s.add(u);
            low[u] = Math.min(low[u], low[j]);
        } else if (arr[u][j] == 1 && j != v)
            low[u] = Math.min(low[u], dfn[j]);
    }
    if (v == -1 && child > 1)
        s.add(u);
}

public void printVisited() {
    System.out.println("articulation points" + s);
    for (int i = 0; i < n; i++)
        System.out.print(dfn[i] - 1 + " ");
    System.out.println();
}

public static void main(String[] args) {
    Scanner scan = new Scanner(System.in);
    System.out.println("enter no of vertices");
    int n = scan.nextInt();
    Articulation art = new Articulation(n);
    System.out.println("adjacent matrix");
```

```java
        art.read(scan);
        art.art(0, -1);
        System.out.println("vertices");
        art.printVisited();
    }
}
```

```
enter no of vertices
4
adjacent matrix
0 1 1 0
1 0 1 1
1 1 0 0
0 1 0 0
vertices
articulation points[1]
0 1 2 3
```

Implement Job Sequencing with deadlines algorithm.

```
import java.util.*;

class Job {
    private int id;
    private int deadline;
    private int profit;

    Job(int id, int deadline, int profit) {
        this.id = id;
        this.deadline = deadline;
        this.profit = profit;
    }

    public int getId() {
        return id;
    }

    public int getDeadline() {
        return deadline;
    }

    public int getProfit() {
        return profit;
    }
}

public class JobSeq {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
```

```java
        System.out.println("enter no of jobs");
        int n = sc.nextInt();
        Job[] jobs = new Job[n];
        System.out.println("emter job id,deadline,profit");
        for (int i = 0; i < n; i++) {
            jobs[i] = new Job(sc.nextInt(), sc.nextInt(), sc.nextInt());
        }
        js(jobs);
    }

    public static void js(Job[] jobs) {
        Arrays.sort(jobs, new Comparator<Job>() {
            public int compare(Job j1, Job j2) {
                return j2.getProfit() - j1.getProfit();
            }
        });

        int maxDeadline = 0;
        for (Job job : jobs) {
            if (job.getDeadline() > maxDeadline) {
                maxDeadline = job.getDeadline();
            }
        }

        Job[] result = new Job[maxDeadline];
        boolean[] slot = new boolean[maxDeadline];

        for (Job job : jobs) {
            for (int j = Math.min(maxDeadline, job.getDeadline()) - 1; j >= 0; j--) {
                if (!slot[j]) { // If the slot is free
                    result[j] = job;
```

```
                    slot[j] = true;
                    break;
                }
            }
        }

        int totalProfit = 0;
        System.out.println("Scheduled jobs:");
        for (Job job : result) {
            if (job != null) {
                System.out.println("Job ID: " + job.getId() + ", Profit: " + job.getProfit());
                totalProfit += job.getProfit();
            }
        }
        System.out.println("Total profit: " + totalProfit);
    }

}
```

```
enter no of jobs
3
emter job id,deadline,profit
1 2 100
2 1 19
3 2 27
Scheduled jobs:
Job ID: 3, Profit: 27
Job ID: 1, Profit: 100
Total profit: 127
```

Implement Fractional Knapsack Algorithm.

```
import java.util.*;

public class Knapsack {
    public static double greedyKnapSack(Item[] arr, int capacity) {
        Arrays.sort(arr, new Comparator<Item>() {
            public int compare(Item i1, Item i2) {
                double cpr1 = (double) i1.p / i1.w;
                double cpr2 = (double) i2.p / i2.w;

                if (cpr1 < cpr2)
                    return 1;
                else
                    return -1;

            }
        });

        double total = 0;

        for (Item i : arr) {
            if ((capacity - i.w) >= 0) {
                capacity -= i.w;
                total += i.p;
            } else {
                double fract = (double) capacity / (double) i.w;
                total += fract * (i.p);
                capacity = 0;
                break;
            }
```

```java
        }
        return total;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter the number of items: ");
        int n = sc.nextInt();
        Item[] arr = new Item[n];
        System.out.println("Enter w, p of each item: ");
        for (int i = 0; i < n; i++) {
            arr[i] = new Item(sc.nextInt(), sc.nextInt());
        }
        System.out.println("Enter capacity :");
        int m = sc.nextInt();

        double pro = greedyKnapSack(arr, m);
        System.out.println(pro);
    }

}

class Item {
    public int w;
    public int p;

    public Item(int w, int p) {
        this.w = w;
        this.p = p;
    }
}
```

Enter the number of items:

3

Enter w, p of each item:

10 20

5 2

50 30

Enter capacity :

50

44.0

Implement OBST using dynamic programming.

```
import java.util.*;

public class OBST {
    public static void bst(double[] p, double[] q, int n) {
        double[][] w = new double[n + 1][n + 1];
        double[][] c = new double[n + 1][n + 1];
        int[][] r = new int[n + 1][n + 1];

        // Initialize base cases: w[i][i] = q[i-1], c[i][i] = 0
        for (int i = 0; i <= n; i++) {
            w[i][i] = q[i]; // for the first and last dummy keys
            c[i][i] = 0;
            r[i][i] = 0;
        }

        // Compute w, c, and r values for each pair of keys
        for (int m = 1; m <= n; m++) {
            for (int i = 0; i <= n - m; i++) {
                int j = i + m;
                w[i][j] = w[i][j - 1] + p[j] + q[j]; // cumulative weight for keys and dummy keys

                double minCost = Double.MAX_VALUE;
                int root = -1;

                // Calculate the minimum cost and the corresponding root
                for (int k = r[i][j - 1]; k <= r[i + 1][j]; k++) {
                    double cost = ((i <= k - 1) ? c[i][k - 1] : 0) + ((k <= j) ? c[k][j] : 0) + w[i][j];
                    if (cost < minCost) {
                        minCost = cost;
```

```java
                    root = k;
                }
            }

            c[i][j] = minCost;
            r[i][j] = root;
        }
    }

    // Print the results
    System.out.println("Minimum cost: " + c[0][n]);
    System.out.println("Weight: " + w[0][n]);
}

public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    System.out.print("Enter the number of keys: ");
    int n = sc.nextInt();

    double[] p = new double[n + 1];
    double[] q = new double[n + 1];

    System.out.println("Enter probabilities for the keys:");
    for (int i = 1; i <= n; i++) {
        System.out.print("p[" + i + "]: ");
        p[i] = sc.nextDouble();
    }

    System.out.println("Enter probabilities for the dummy keys:");
    for (int i = 0; i <= n; i++) {
        System.out.print("q[" + i + "]: ");
```

```
            q[i] = sc.nextDouble();
        }

        bst(p, q, n);
    }
}
```

Enter the number of keys: 4

Enter probabilities for the keys:

p[1]: 3

p[2]: 3

p[3]: 1

p[4]: 1

Enter probabilities for the dummy keys:

q[0]: 2

q[1]: 3

q[2]: 1

q[3]: 1

q[4]: 1

Minimum cost: 16.0

Weight: 16.0

Implement N-queen algorithm.

```
import java.util.*;

public class Solution {

    public List<List<String>> solveNQueens(int n) {
        HashSet<Integer> col = new HashSet<>();
        HashSet<Integer> posDiag = new HashSet<>();
        HashSet<Integer> negDiag = new HashSet<>();

        List<List<String>> res = new ArrayList<>();
        char[][] board = new char[n][n];
        for (int i = 0; i < n; i++) {
            Arrays.fill(board[i], '.');
        }
        backTrack(0, col, posDiag, negDiag, res, board, n);
        return res;
    }

    public void backTrack(int r, HashSet<Integer> col, HashSet<Integer> pd, HashSet<Integer> nd,
List<List<String>> res,
            char[][] board, int n) {
        if (r == n) {
            List<String> a = new ArrayList<>();
            for (char[] i : board) {
                a.add(new String(i));
            }
            res.add(a);
            return;
        }
```

```
    for (int c = 0; c < n; c++) {
        if (col.contains(c) || nd.contains(r - c) || pd.contains(r + c)) {
            continue;
        }
        col.add(c);
        nd.add(r - c);
        pd.add(r + c);
        board[r][c] = 'Q';

        backTrack(r + 1, col, pd, nd, res, board, n);

        col.remove(c);
        nd.remove(r - c);
        pd.remove(r + c);
        board[r][c] = '.';
    }
}

public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    Solution solution = new Solution();

    System.out.print("Enter the number of queens (n): ");
    int n = sc.nextInt(); // User input for the size of the board

    List<List<String>> result = solution.solveNQueens(n);

    if (result.isEmpty()) {
        System.out.println("No solution exists.");
    } else {
        for (List<String> solutionBoard : result) {
```

```
            for (String row : solutionBoard) {
                System.out.println(row);
            }
            System.out.println(); // Empty line between solutions
        }
    }
  }
}
```

```
Enter the number of queens (n): 4
.Q..
...Q
Q...
..Q.

..Q.
Q...
...Q
.Q..
```

Implement Prim’s algorithm.

```
import java.util.*;

public class PrimsAlgorithm {

    public static void primMST(int[][] graph) {
        int n = graph.length;
        boolean[] mstSet = new boolean[n];
        int[] parent = new int[n];
        int[] key = new int[n];

        Arrays.fill(key, Integer.MAX_VALUE);
        Arrays.fill(mstSet, false);

        key[0] = 0;
        parent[0] = -1;

        for (int count = 0; count < n - 1; count++) {
            int u = minKey(key, mstSet);

            mstSet[u] = true;

            for (int v = 0; v < n; v++) {
                if (graph[u][v] != 0 && !mstSet[v] && graph[u][v] < key[v]) {
                    key[v] = graph[u][v];
                    parent[v] = u;
                }
            }
        }
```

```java
    printMST(parent, graph);
}

public static int minKey(int[] key, boolean[] mstSet) {
    int min = Integer.MAX_VALUE;
    int minIndex = -1;

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

    return minIndex;
}

public static void printMST(int[] parent, int[][] graph) {
    System.out.println("Edge \tWeight");
    for (int i = 1; i < parent.length; i++) {
        System.out.println(parent[i] + " - " + i + "\t" + graph[i][parent[i]]);
    }
}

public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);

    System.out.print("Enter the number of vertices: ");
    int n = sc.nextInt();
    int[][] graph = new int[n][n];
```

```java
        System.out.println("Enter the adjacency matrix (weights of edges): ");
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                graph[i][j] = sc.nextInt();
            }
        }

        PrimsAlgorithm prim = new PrimsAlgorithm();
        prim.primMST(graph);
    }
}
```

```
Enter the number of vertices: 4
Enter the adjacency matrix (weights of edges):
0 2 0 6
2 0 3 8
0 3 0 0
6 8 0 0
Edge    Weight
0 - 1   2
1 - 2   3
0 - 3   6
```