PRACTICAL NO: 1

```
public class IterativeRecursiveComparison {
  public static int iterativeSum(int n) {
      int sum = 0;
      for (int i = 1; i <= n; i++) {
         sum += i;
      }
      return sum;
  }
  public static int recursiveSum(int n) {
      if (n == 1) {
         return 1;
      }
      return n + recursiveSum(n - 1);
  }
  public static void main(String[] args) {
      int n = 10; // Sum of first 10 numbers = 55

      int iterativeResult = iterativeSum(n);
      int recursiveResult = recursiveSum(n);

      System.out.println("Iterative approach result: " + iterativeResult);
      System.out.println("Recursive approach result: " + recursiveResult);
  }
}
```

```
PS C:\Users\HP\.vscode>  & 'C:\Program Files\Java\jdk-24\bin\java.exe' '--enable-preview' '-X
ge\1d54edd17598dbeb22dfa40695392ed8\redhat.java\jdt_ws\.vscode_fa16ff10\bin' 'SumNumbers'
Iterative approach result: 55
Recursive approach result: 55
PS C:\Users\HP\.vscode>
```

PRACTICAL NO:2

```
import java.util.PriorityQueue;
import java.util.Scanner;
class HuffmanNode {
   int freq;
   char ch;
   HuffmanNode left, right;
}
class MyComparator implements java.util.Comparator<HuffmanNode> {
   public int compare(HuffmanNode x, HuffmanNode y) {
      return x.freq - y.freq;
   }
}
public class HuffmanCoding {
      public static void printCode(HuffmanNode root, String s) {
      if (root.left == null && root.right == null && Character.isLetter(root.ch)) {
         System.out.println(root.ch + " -> " + s);
         return;
      }
      printCode(root.left, s + "0");
      printCode(root.right, s + "1");
   }
public static void main(String[] args) {
      Scanner sc = new Scanner(System.in);
      System.out.print("Enter characters (without spaces): ");
      String chars = sc.next();
      System.out.print("Enter corresponding frequencies (separated by spaces): ");
      int[] freq = new int[chars.length()];
      for (int i = 0; i < chars.length(); i++) {
         freq[i] = sc.nextInt();
      }
```

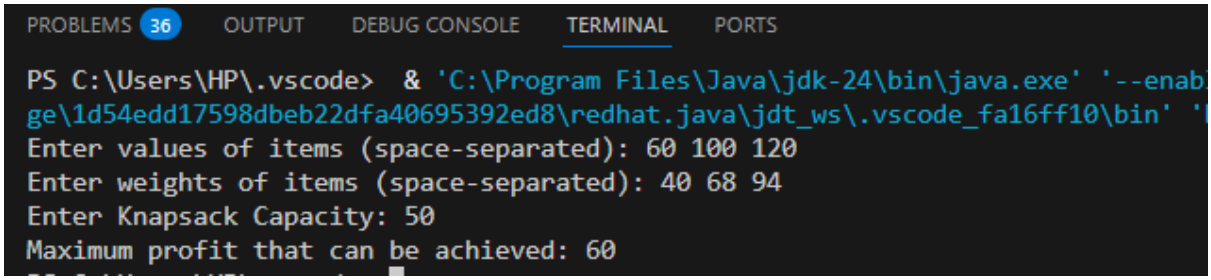
```
        PriorityQueue<HuffmanNode> q = new PriorityQueue<>(chars.length(), new MyComparator());
        for (int i = 0; i < chars.length(); i++) {
            HuffmanNode node = new HuffmanNode();
            node.ch = chars.charAt(i);
            node.freq = freq[i];
            node.left = null;
            node.right = null;
            q.add(node);
        }       while (q.size() > 1) {
            HuffmanNode x = q.poll();
            HuffmanNode y = q.poll();
            HuffmanNode f = new HuffmanNode();
            f.freq = x.freq + y.freq;
            f.ch = '-';
            f.left = x;
            f.right = y;
            q.add(f);
        }
        HuffmanNode root = q.peek();
        System.out.println("Huffman Codes:");
        printCode(root, "");
    }
}
```

```
PS C:\Users\HP\.vscode>  & 'C:\Program Files\Java\jdk-24\bin\java.exe' '--enable-preview' '-XX:+
ge\1d54edd17598dbeb22dfa40695392ed8\redhat.java\jdt_ws\.vscode_fa16ff10\bin' 'HuffmanCoding'
Enter characters (without spaces): abcde
Enter corresponding frequencies (separated by spaces): 5 9 12 13 16
Huffman Codes:
c -> 00
d -> 01
a -> 100
b -> 101
e -> 11
```

PRACTICAL NO:3

```
import java.util.Scanner;
public class Knapsack01 {
   public static int knapSack(int W, int wt[], int val[], int n) {
      int i, w;
      int K[][] = new int[n + 1][W + 1];
      for (i = 0; i <= n; i++) {
         for (w = 0; w <= W; w++) {
            if (i == 0 || w == 0)
               K[i][w] = 0;
            else if (wt[i - 1] <= w)
               K[i][w] = Math.max(val[i - 1] + K[i - 1][w - wt[i - 1]], K[i - 1][w]);
            else
               K[i][w] = K[i - 1][w];
         }
      }
      return K[n][W]; // Maximum profit
   }
   public static void main(String[] args) {
      Scanner sc = new Scanner(System.in);
      System.out.print("Enter values of items (space-separated): ");
      String[] valStr = sc.nextLine().split(" ");
      int n = valStr.length;
      int[] val = new int[n];
      for (int i = 0; i < n; i++) val[i] = Integer.parseInt(valStr[i]);
      System.out.print("Enter weights of items (space-separated): ");
      String[] wtStr = sc.nextLine().split(" ");
      int[] wt = new int[n];
      for (int i = 0; i < n; i++) wt[i] = Integer.parseInt(wtStr[i]);
      System.out.print("Enter Knapsack Capacity: ");
      int W = sc.nextInt();
```

```
        int maxProfit = knapSack(W, wt, val, n);
        System.out.println("Maximum profit that can be achieved: " + maxProfit);
    }
}
```



PRACTICAL NO:4

```
import java.util.Scanner;
public class NQueens {
   static int N;
   static void printSolution(int board[][]) {
      for (int i = 0; i < N; i++) {
         System.out.print("[");
         for (int j = 0; j < N; j++) {
            if (board[i][j] == 1)
               System.out.print(" Q ");
            else
               System.out.print(" - ");
         }
         System.out.println("]");
      }
      System.out.println();
   }
   static boolean isSafe(int board[][], int row, int col) {
      int i, j;
      for (i = 0; i < col; i++)
         if (board[row][i] == 1)
            return false;
      for (i = row, j = col; i >= 0 && j >= 0; i--, j--)
         if (board[i][j] == 1)
            return false;
      for (i = row, j = col; j >= 0 && i < N; i++, j--)
         if (board[i][j] == 1)
            return false;
      return true;
   }
   static boolean solveNQUtil(int board[][], int col) {
```

```
    if (col >= N) {
        printSolution(board);
        return true;
    }
    boolean res = false;
    for (int i = 0; i < N; i++) {
        if (isSafe(board, i, col)) {
            board[i][col] = 1;
            res = solveNQUtil(board, col + 1) || res;
            board[i][col] = 0;
        }
    }
    return res;
}
static void solveNQ() {
    int board[][] = new int[N][N];
    if (!solveNQUtil(board, 0)) {
        System.out.println("Solution does not exist");
    }
}
public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    System.out.print("Enter value of N (number of queens): ");
    N = sc.nextInt();
    solveNQ();
```

```
Enter value of N (number of queens): 4
[ -  -  Q  - ]
[ Q  -  -  - ]
[ -  -  -  Q ]
[ -  Q  -  - ]

[ -  Q  -  - ]
[ -  -  -  Q ]
[ Q  -  -  - ]
[ -  -  Q  - ]
}
```

PRACTICAL NO:5

```
import java.util.Random;
public class QuickSortComparison {
    public static void deterministicQuickSort(int[] arr, int low, int high) {
        if (low < high) {
            int pi = deterministicPartition(arr, low, high);
            deterministicQuickSort(arr, low, pi - 1);
            deterministicQuickSort(arr, pi + 1, high);
        }
    }
    private static int deterministicPartition(int[] arr, int low, int high) {
        int pivot = arr[high];
        int i = low - 1;
        for (int j = low; j < high; j++) {
            if (arr[j] < pivot) {
                i++;
                swap(arr, i, j);
            }
        }
        swap(arr, i + 1, high);
        return i + 1;
    }
    public static void randomizedQuickSort(int[] arr, int low, int high) {
        if (low < high) {
            int pi = randomizedPartition(arr, low, high);
            randomizedQuickSort(arr, low, pi - 1);
            randomizedQuickSort(arr, pi + 1, high);
        }
    }
    private static int randomizedPartition(int[] arr, int low, int high) {
        Random rand = new Random();
```

```
        int randomIndex = rand.nextInt(high - low + 1) + low;
        swap(arr, randomIndex, high);
        return deterministicPartition(arr, low, high);
    }
    private static void swap(int[] arr, int i, int j) {
        int temp = arr[i];
        arr[i] = arr[j];
        arr[j] = temp;
    }
    public static void main(String[] args) {
        int[] sizes = {100, 1000, 10000, 100000};
        for (int size : sizes) {
            int[] arr = generateRandomArray(size);
            int[] arrCopy = arr.clone();
            long startTime = System.nanoTime();
            deterministicQuickSort(arr, 0, arr.length - 1);
            long endTime = System.nanoTime();
            double deterministicTime = (endTime - startTime) / 1e9;
            startTime = System.nanoTime();
            randomizedQuickSort(arrCopy, 0, arrCopy.length - 1);
            endTime = System.nanoTime();
            double randomizedTime = (endTime - startTime) / 1e9;
            System.out.println("Array size: " + size);
            System.out.println("Deterministic Quick Sort time: " + String.format("%.6f",
deterministicTime) + " seconds");
            System.out.println("Randomized Quick Sort time: " + String.format("%.6f", randomizedTime)
+ " seconds");
            System.out.println();
        }
    }
    private static int[] generateRandomArray(int size) {
        Random rand = new Random();
```

```
        int[] arr = new int[size];
        for (int i = 0; i < size; i++) {
            arr[i] = rand.nextInt();
        }
        return arr;
    }
}
```