Practical No.1

```
import java.util.Scanner;

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

        System.out.print("Enter the number of terms: ");
        int n = sc.nextInt();

        int first = 0, second = 1;

        System.out.println("Fibonacci Series up to " + n + " terms:");

        for (int i = 1; i <= n; i++) {
            System.out.print(first + " ");
            int next = first + second;
            first = second;
            second = next;
        }

        sc.close();
    }
}
```

Practical No.2

```
import java.util.Scanner;

class FibonacciRecursive {
    static int fib(int n) {
        if (n <= 1)
            return n;
        return fib(n - 1) + fib(n - 2);
    }

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

        System.out.println("Fibonacci Series (Recursive):");
        for (int i = 0; i < n; i++) {
            System.out.print(fib(i) + " ");
        }
        sc.close();
    }
}
```

Practical No.3

```
import java.util.PriorityQueue;
import java.util.Scanner;

class HuffmanNode {
   int data;
   char c;
   HuffmanNode left, right;
}

class ImplementComparator implements java.util.Comparator<HuffmanNode>
{
   public int compare(HuffmanNode x, HuffmanNode y) {
      return x.data - y.data;
   }
}

 class HuffmanEncoding {
   static void printCode(HuffmanNode root, String s) {
      if (root.left == null && root.right == null && Character.isLetter(root.c)) {
         System.out.println(root.c + ":" + 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 number of characters: ");
      int n = sc.nextInt();
      char[] charArray = new char[n];
      int[] charfreq = new int[n];

      System.out.println("Enter characters and their frequencies:");
      for (int i = 0; i < n; i++) {
         charArray[i] = sc.next().charAt(0);
         charfreq[i] = sc.nextInt();
      }
```

```
        PriorityQueue<HuffmanNode> q = new PriorityQueue<>(n, new
ImplementComparator());

        for (int i = 0; i < n; i++) {
            HuffmanNode hn = new HuffmanNode();
            hn.c = charArray[i];
            hn.data = charfreq[i];
            hn.left = null;
            hn.right = null;
            q.add(hn);
        }

        HuffmanNode root = null;

        while (q.size() > 1) {
            HuffmanNode x = q.poll();
            HuffmanNode y = q.poll();

            HuffmanNode f = new HuffmanNode();
            f.data = x.data + y.data;
            f.c = '-';
            f.left = x;
            f.right = y;

            root = f;
            q.add(f);
        }

        System.out.println("Huffman Codes are:");
        printCode(root, "");
        sc.close();
    }
}
```

Practical No.4

```
import java.util.Arrays;
import java.util.Comparator;
import java.util.Scanner;

class Item {
    int weight, value;

    Item(int weight, int value) {
        this.weight = weight;
        this.value = value;
    }
}

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

        Item[] items = new Item[n];
        for (int i = 0; i < n; i++) {
            System.out.print("Enter weight and value of item " + (i + 1) + ": ");
            int w = sc.nextInt();
            int v = sc.nextInt();
            items[i] = new Item(w, v);
        }

        System.out.print("Enter capacity of knapsack: ");
        int capacity = sc.nextInt();

        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); // Sort in descending order
            }
        });

        double totalValue = 0.0;

        for (Item item : items) {
```

```
            if (capacity >= item.weight) {
                capacity -= item.weight;
                totalValue += item.value;
            } else {
                totalValue += ((double) item.value / item.weight) * capacity;
                break;
            }
        }

        System.out.println("Maximum value in knapsack (Profit) = " + totalValue);
        sc.close();
    }
}
```

Practical No.5 (Using Dynammic Programming)

```
import java.util.Scanner;

class Knapsack01 {
   static int knapSack(int W, int wt[], int val[], int n) {
      int[][] dp = new int[n + 1][W + 1];

      for (int i = 0; i <= n; i++) {
         for (int w = 0; w <= W; w++) {
            if (i == 0 || w == 0)
               dp[i][w] = 0;
            else if (wt[i - 1] <= w)
               dp[i][w] = Math.max(val[i - 1] + dp[i - 1][w - wt[i - 1]], dp[i -
1][w]);
            else
               dp[i][w] = dp[i - 1][w];
         }
      }
      return dp[n][W];
   }

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

      System.out.print("Enter number of items: ");
      int n = sc.nextInt();

      int[] values = new int[n];
      int[] weights = new int[n];

      System.out.println("Enter weight and value for each item:");
      for (int i = 0; i < n; i++) {
         weights[i] = sc.nextInt();
         values[i] = sc.nextInt();
      }

      System.out.print("Enter knapsack capacity: ");
      int W = sc.nextInt();

      int maxValue = knapSack(W, weights, values, n);
      System.out.println("Maximum value in Knapsack (Profit) = " +
maxValue);
```

```
        sc.close();
    }
}
```