*// Block 1 ---------------------------------------------------------------------------------------------------------------------------------------*

*//*

*//        Find the smallest missing positive element in a sorted array of positive integers. O(log(n))*

*//        Example:   [1, 2, 6, 31]*

*//        Result: 3*

*//*

*//        Example:  [2, 3, 4, 6, 9, 11, 15]*

*//        Result: 1*

public static int smallestMissing(int[] arr) {

       throw new UnsupportedOperationException();

   }

*//*

*//        Find the 'floor' and 'ceiling' of a given number in the sorted array.*

*//        Example:   arr = [0, 1, 2, 6, 31], n = 5*

*//        Result: floor = 2, ceiling = 6*

*//*

*//        Example:   arr = [7, 10, 15, 21, 29], n = 31*

*//        Result: floor = 29, ceiling = -1*

*//        Example:   arr = [7, 10, 15, 21, 29], n = 5*

*//        Result: floor = -1, ceiling = 7*

public static int findFloor(int[] arr) {

       throw new UnsupportedOperationException();

   }

   public static int findCeiling(int[] arr) {

       throw new UnsupportedOperationException();

   }

*//        \**

*//        Merge sort algorithm for a singly linked list*

class Node {

       private int data;

       private Node next;

       Node(int data, Node next) {

           this.data = data;

           this.next = next;

       }

       public int getData() {

           return data;

       }

       public void setData(int data) {

           this.data = data;

       }

       public Node getNext() {

           return next;

       }

       public void setNext(Node next) {

           this.next = next;

       }

   }

   public static Node mergeSort(Node head) {

       throw new UnsupportedOperationException();

   }

*// 5 -> 3 -> 6 -> 2*

*// 2 -> 3 -> 5 -> 6*

*//        Find `k` closest elements to a given value in an array*

*//        Example: arr =  [0, 5, 8, 10, 12, 16, 17, 22],  k = 3, n = 11*

*//        Result: 8, 10, 12*

*//*

*//        Example: arr =  [8, 9, 11, 15, 19,22, 25, 26, 27],  k = 4, n = 22*

*//        Result: 19, 22, 25, 26*

public static int[] findCeiling(int[] arr, int k, int n) {

       throw new UnsupportedOperationException();

   }

*//        \*\**

*//        Sort a doubly-linked list using merge sort*

class Node2 {

       int data;

       Node next;

       Node prev;

       public Node2(int data, Node next, Node prev) {

           this.data = data;

           this.next = next;

           this.prev = prev;

       }

       public int getData() {

           return data;

       }

       public void setData(int data) {

           this.data = data;

       }

       public Node getNext() {

           return next;

       }

       public void setNext(Node next) {

           this.next = next;

       }

       public Node getPrev() {

           return prev;

       }

       public void setPrev(Node prev) {

           this.prev = prev;

       }

   }

   public static Node2 mergeSort(Node2 head) {

       throw new UnsupportedOperationException();

   }

*//        You are given an array that consists of positive and negative integers. Find the maximum subarray*

*//        Example: arr =  [0, -5, -3, 10, 9, -11, 17, -22]*

*//        Result: 25 = (10 + 9 - 11 + 17)*

*//*

*//        Example: arr =  [8, -9, 11, -15, 9, -5, 6, -1, 3, 4]*

*//        Result: 16 = 9 - 5 + 6 - 1 + 3 + 4*

public static int findTheMax(int[] arr) {

       throw new UnsupportedOperationException();

   }

*//*

*//*

*//*

*//2 ------------------------------------------------------------------------------------------------------------------------------------------------------*

*//*

*//       Given a number dist count total number of ways to cover the distance with 1, 2 and 3 steps.*

*//    Example: n = 3*

*//    Result: 4*

*//    Notes:*

*//      1  + 1  + 1*

*//      1  + 2*

*//      2  + 1*

*//      3*

*//*

*//    Example: n = 4*

*//    Result: 7*

*//    Notes:*

*//      1  + 1  + 1  + 1*

*//      1  + 2  + 1*

*//      2  + 1  + 1*

*//      1  + 1  + 2*

*//      2  + 2*

*//      3  + 1*

*//      1  + 3*

*//        \**

*//        Catalan numbers are defined as a mathematical sequence that consists of positive integers, which can be used to find the number of possibilities of various combinations.*

*//*

*//        The nth term in the sequence denoted Cn, is found in the following formula:*

*//        Coin Change Problem*

*//        - any solution*

*//        - optimal solution \**

*//        - all solutions \*\**

*//        - coins are not indefinite \*\**

*//    Given an integer array representing coins*

*//    You can consider each coin can be obtained infinite number of times*

*//*

*//    You have to find the number of ways to make sum by using different combinations from array.*

*//    Example: sum = 4, coins[] = {1,2,3},*

*//    Result: {1, 1, 1, 1} or {1, 1, 2} or {2, 2} or {1, 3}.*

*//    Optimal {2, 2} or {1, 3}*

public static int[] findCoins(int[] arr, int sum) {

       throw new UnsupportedOperationException();

   }

*// ----------------------------------------------------------------------------------------------------------------------------------------------------------------*

*//    Validate brackets sequence*

*//    Example (())*

*//    Result: true*

*//    Example (()()*

*//    Result: false*

*//    Example )()(*

*//    Result: false*

*// \**

*// Validate arithmetic expression with numbers and + - \* /*

*//    Example 4+5-6\*6*

*//    Result: true*

*//    Example 4+-5-6\*6*

*//    Result: false*

*//    Example -4/6//6+1-2*

*//    Result: false*

*// \**

*// Validate arithmetic expression with numbers and + - \* / and brackets*

*//    Example (4+5-6)\*6*

*//    Result: true*

*//    Example 4+5-(6\*)6*

*//    Result: false*

*//    Example (-)4/6/6+1-2*

*//    Result: false*