NAME: BHARATH V
DEPT: CSBS

1.Kth Smallest Element

```
import java.util.*;
public class KthSmallestElement {
   public static int kthSmallest(int[] arr, int k) {
      PriorityQueue<Integer> maxHeap = new PriorityQueue<>(Collections.reverseOrder());
      for (int i = 0; i < k; i++) {
         maxHeap.add(arr[i]);
      }
      for (int i = k; i < arr.length; i++) {
         if (arr[i] < maxHeap.peek()) {
            maxHeap.poll();
            maxHeap.add(arr[i]);
         }
      }
      return maxHeap.peek();
   }

   public static void main(String[] args) {
      int[] arr = {12, 3, 5, 7, 19, 2};
      int k = 4;
      int result = kthSmallest(arr, k);
      System.out.println("The " + k + "-th smallest element is: " + result);
   }
}
```

OUTPUT:

The 4-th smallest element is 7

Complexity:

Time: O(nlogn)

Space: O(n)

2.Minimize the heights

```
import java.util.Arrays;
public class MinimizeHeightsII {
   public static int minimizeHeightDifference(int[] arr, int k) {
      int n = arr.length;
      if (n == 1) return 0;
      Arrays.sort(arr);
      int result = arr[n-1] - arr[0];
      for (int i = 1; i < n; i++) {
         int minHeight = Math.min(arr[0] + k, arr[i] - k);
         int maxHeight = Math.max(arr[n-1] - k, arr[i-1] + k);
         result = Math.min(result, maxHeight - minHeight);
      }

      return result;
   }

   public static void main(String[] args) {
      int[] arr = {1, 5, 8, 10};
      int k = 2;

      int result = minimizeHeightDifference(arr, k);
      System.out.println("The minimum possible difference is: " + result);
   }
}
```

OUTPUT:

The minimum possible difference is: 5

Complexity:

Time: O(nlogn)

Space: O(n)

3.Paranthesis Chechup

```
import java.util.Stack;
public class ParenthesisCheck {
    public static boolean isBalanced(String expression) {
        Stack<Character> stack = new Stack<>();
        for (char ch : expression.toCharArray()) {
            if (ch == '(' || ch == '{' || ch == '[') {
                stack.push(ch);
            }
            else if (ch == ')' || ch == '}' || ch == ']') {
                if (stack.isEmpty() || !isMatchingPair(stack.pop(), ch)) {
                    return false;
                }
            }
        }
        return stack.isEmpty();
    }
    private static boolean isMatchingPair(char opening, char closing) {
        return (opening == '(' && closing == ')') ||
               (opening == '{' && closing == '}') ||
               (opening == '[' && closing == ']');
    }

    public static void main(String[] args) {
        String expression = "{[()]}";
```

```
        if (isBalanced(expression)) {
            System.out.println("The parentheses are balanced.");
        } else {
            System.out.println("The parentheses are not balanced.");
        }
    }
}
```

OUTPUT:

The parentheses are balanced.

Complexity:

Time: O(n)

Space: O(n)

4.Equilibrium Points

```
public class EquilibriumPoint {
    public static int findEquilibriumPoint(int[] arr) {
        int n = arr.length;
        if (n == 0) return -1;

        int totalSum = 0;
        int leftSum = 0;
        for (int i = 0; i < n; i++) {
            totalSum += arr[i];
        }
        for (int i = 0; i < n; i++) {
            totalSum -= arr[i];
            if (leftSum == totalSum) {
                return I;
            }
            leftSum += arr[i];
```

```
        }
        return -1;
    }

    public static void main(String[] args) {
        int[] arr = {1, 3, 5, 2, 2};
        int result = findEquilibriumPoint(arr);
        if (result == -1) {
            System.out.println("No equilibrium point found.");
        } else {
            System.out.println("Equilibrium point found at index: " + result);
        }
    }
}
```

OUTPUT:

Equilibrium point found at index: 2

Complexity:

Time: O(n)

Space: O(1)

5.Binary Search

```
public class BinarySearch {
    public static int binarySearch(int[] arr, int target) {
        int low = 0;
        int high = arr.length - 1;

        while (low <= high) {
            int mid = low + (high - low) / 2;
            if (arr[mid] == target) {
                return mid;
            }
```

```
            if (arr[mid] < target) {
                low = mid + 1;
            }

            else {
                high = mid - 1;
            }
}
        return -1;
    }

    public static void main(String[] args) {
        int[] arr = {1, 3, 5, 7, 9, 11, 13};
        int target = 7;
        int result = binarySearch(arr, target);
        if (result == -1) {
            System.out.println("Element not found.");
        } else {
            System.out.println("Element found at index: " + result);
        }
    }
}
```

OUTPUT:

Element found at index: 3

Complexity:

Time:O(logn)

Space:O(1)

6.Next Greater Element

```
import java.util.Stack;
public class NextGreaterElement {
```

```
    public static void findNextGreaterElement(int[] arr) {
        int n = arr.length;
        Stack<Integer> stack = new Stack<>();
        for (int i = n - 1; i >= 0; i--) {
            while (!stack.isEmpty() && stack.peek() <= arr[i]) {
                stack.pop();
            }
            if (stack.isEmpty()) {
                System.out.println(arr[i] + " --> -1");
            } else {

                System.out.println(arr[i] + " --> " + stack.peek());
            }
            stack.push(arr[i]);
        }
    }

    public static void main(String[] args) {
        int[] arr = {4, 5, 2, 10, 8};
        findNextGreaterElement(arr);
    }
}
```

OUTPUT:

4 --> 5

5 --> 10

2 --> 10

10 --> -1

8 --> -1

Complexity:

Time: O(n)

Space:O(n)

7. union of two arrays with duplicate elements

```
import java.util.*;
public class UnionOfArrays {
   public static int[] findUnion(int[] arr1, int[] arr2) {
      Set<Integer> unionSet = new HashSet<>();
      for (int num : arr1) {
         unionSet.add(num);
      }
      for (int num : arr2) {
         unionSet.add(num);
      }
      int[] result = new int[unionSet.size()];
      int i = 0;
      for (int num : unionSet) {
         result[i++] = num;
      }

      return result;
   }

   public static void main(String[] args) {
      int[] arr1 = {1, 2, 2, 4, 5};
      int[] arr2 = {2, 3, 4, 6};
      int[] union = findUnion(arr1, arr2);
      System.out.println("Union of two arrays:");
      System.out.println(Arrays.toString(union));
   }
}
```

OUTPUT:

Union of two arrays:

[1, 2, 3, 4, 5, 6]

Complexity:

Time: O(n+m)

Space:O(n+m)