

PRACTICE ASSIGNMENT 4

1. Kth Smallest

Given an array `arr[]` and an integer `k` where `k` is smaller than the size of the array, the task is to find the `k`th smallest element in the given array. Follow up: Don't solve it using the inbuilt sort function.

Input: `arr[] = [7, 10, 4, 3, 20, 15]`, `k = 3` Output: 7

Explanation: 3rd smallest element in the given array is 7.

CODE:

```
import java.util.PriorityQueue;
class Solution {
    public static int findKthSmallest(int[] nums, int start, int end, int position) {
        PriorityQueue<Integer> heap = new PriorityQueue<>((x, y) -> y - x);
        for (int i = 0; i < nums.length; i++) {
            heap.add(nums[i]);
            if (heap.size() > position) {
                heap.poll();
            }
        }
        return heap.peek();
    }
    public static void main(String[] args) {
        int[] arr = {7, 10, 4, 3, 20, 15};
        int k = 3;
        int result = findKthSmallest(arr, 0, arr.length - 1, k);
        System.out.println(result);
    }
}
```

OUTPUT:

```
X:\Desktop\java\assignment4>javac Solution.java
X:\Desktop\java\assignment4>java Solution
7
```

TIME COMPLEXITY: $O(\log k)$

2. Minimize the Heights II

Given an array `arr[]` denoting heights of `N` towers and a positive integer `K`.

For each tower, you must perform exactly one of the following operations exactly once.

Increase the height of the tower by `K`
Decrease the height of the tower by `K`
Find out the minimum possible difference between the height of the shortest and tallest towers after you have modified each tower. You can find a slight modification of the problem [here](#).

Note: It is compulsory to increase or decrease the height by `K` for each tower. After the operation, the resultant array should not contain any negative integers.

Input: `k = 2, arr[] = {1, 5, 8, 10}`

Output: 5

Explanation: The array can be modified as $\{1+k, 5-k, 8-k, 10-k\} = \{3, 3, 6, 8\}$. The difference between the largest and the smallest is $8-3 = 5$.

CODE:

```
import java.util.Arrays;
```

```
class Solution {
    public static int getMinDifference(int[] arr, int n, int k) {
        Arrays.sort(arr);
        int result = arr[n - 1] - arr[0];
        int small = arr[0] + k, large = arr[n - 1] - k;
        for (int i = 1; i < n - 1; i++) {
            int add = arr[i] + k;
            int sub = arr[i] - k;
            if (sub >= small || add <= large)
                continue;
            if (large - add < sub - small) {
                small = sub;
            } else {
                large = add;
            }
        }
        return Math.min(result, large - small);
    }

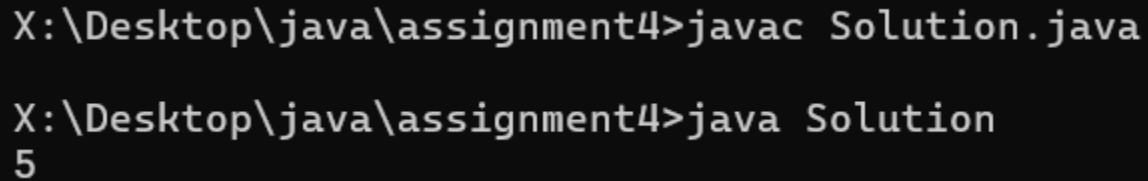
    public static void main(String[] args) {
        int[] arr1 = {1, 5, 8, 10};
    }
}
```

```

        int k1 = 2;
        System.out.println(getMinDifference(arr1, arr1.length, k1));
    }
}

```

OUTPUT:



```

X:\Desktop\java\assignment4>javac Solution.java

X:\Desktop\java\assignment4>java Solution
5

```

TIME COMPLEXITY: $O(n \log n)$.

3 . Parenthesis Checker

You are given a string *s* representing an expression containing various types of brackets: {}, (), and []. Your task is to determine whether the brackets in the expression are balanced. A balanced expression is one where every opening bracket has a corresponding closing bracket in the correct order.

Input: *s* = "{([])}"

Output: true

Input: *s* = "()"

Output: true

CODE:

```

import java.util.Stack;
class Solution {
    public static boolean isValid(String s) {
        Stack<Character> stack = new Stack<>();
        for (char c : s.toCharArray()) {
            if (c == '(' || c == '{' || c == '[') {
                stack.push(c);
            } else {
                if (stack.isEmpty()) return false;
                char top = stack.pop();
                if (c == ')' && top != '(' || c == '}' && top != '{' || c == ']' && top != '[') {

```

```

        return false;
    }
}
return stack.isEmpty();
}

public static void main(String[] args) {
    System.out.println(isValid("{([])}"));
    System.out.println(isValid("("));
}
}

```

OUTPUT:

```

X:\Desktop\java\assignment4>javac Solution.java

X:\Desktop\java\assignment4>java Solution
true
true

```

TIME COMPLEXITY: $O(n)$

4. Equilibrium Point

Given an array `arr` of non-negative numbers. The task is to find the first equilibrium point in an array. The equilibrium point in an array is an index (or position) such that the sum of all elements before that index is the same as the sum of elements after it.

Note: Return equilibrium point in 1-based indexing. Return -1 if no such point exists.

Input: `arr[] = [1, 3, 5, 2, 2]`

Output: 3

Explanation: The equilibrium point is at position 3 as the sum of elements before it (1+3) = sum of elements after it (2+2).

CODE:

```

class Solution {
    public static int equilibriumPoint(int arr[]) {
        int n = arr.length;
    }
}

```

```

        if (n == 1) return 1;
        int totalSum = 0;
        for (int num : arr) totalSum += num;
        int leftSum = 0;
        for (int i = 0; i < n; i++) {
            totalSum -= arr[i];
            if (leftSum == totalSum) return i + 1;
            leftSum += arr[i];
        }
        return -1;
    }
}
public static void main(String[] args) {
    int arr1[] = {1, 3, 5, 2, 2};
    System.out.println(equilibriumPoint(arr1));
}
}

```

OUTPUT:

```

X:\Desktop\java\assignment4>javac Solution.java
X:\Desktop\java\assignment4>java Solution
3

```

TIME COMPLEXITY: $O(N)$

5. Binary Search

Given a sorted array arr and an integer k, find the position(0-based indexing) at which k is present in the array using binary search.

Note: If multiple occurrences are there, please return the smallest index.

Input: arr[] = [1, 2, 3, 4, 5], k = 4 Output: 3

Explanation: 4 appears at index 3.

Input: arr[] = [11, 22, 33, 44, 55], k = 445 Output: -1

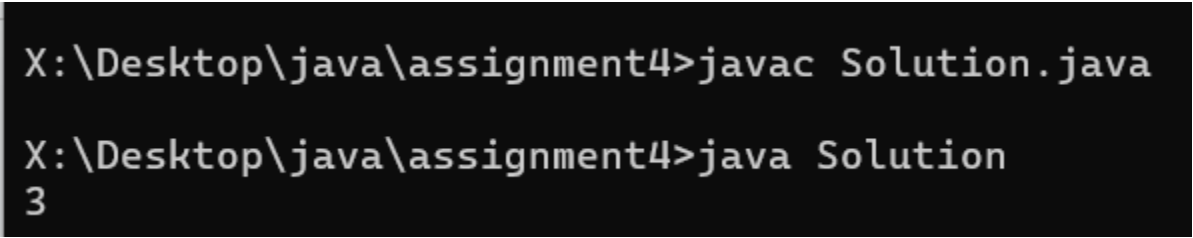
Explanation: 445 is not present.

CODE:

```
class Solution {
    public static int binarySearch(int[] arr, int k) {
        int left = 0, right = arr.length - 1;
        int result = -1;
        while (left <= right) {
            int mid = left + (right - left) / 2;
            if (arr[mid] == k) {
                result = mid;
                right = mid - 1;
            } else if (arr[mid] < k) {
                left = mid + 1;
            } else {
                right = mid - 1;
            }
        }

        return result;
    }

    public static void main(String[] args) {
        int[] arr = {1, 2, 3, 4, 5};
        int k = 4;
        System.out.println(binarySearch(arr, k));
    }
}
```

OUTPUT:

```
X:\Desktop\java\assignment4>javac Solution.java
X:\Desktop\java\assignment4>java Solution
3
```

TIME COMPLEXITY: $O(\log n)$

6. Next Greater Element

Given an array `arr[]` of integers, the task is to find the next greater element for each element of the array in order of their appearance in the array. Next greater element of an element in the array is the nearest element on the right which is greater than the current element.

If there does not exist next greater of current element, then next greater element for current element is -1. For example, next greater of the last element is always -1.

Input: `arr[] = [1, 3, 2, 4]` Output: `[3, 4, 4, -1]`.

Input: `arr[] = [6, 8, 0, 1, 3]` Output: `[8, -1, 1, 3, -1]`

CODE:

```
import java.util.Stack;

class Solution {
    public static int[] nextGreaterElement(int[] arr) {
        int n = arr.length;
        int[] result = new int[n];
        Stack<Integer> stack = new Stack<>();

        for (int i = n - 1; i >= 0; i--) {
            while (!stack.isEmpty() && stack.peek() <= arr[i]) {
                stack.pop();
            }
            result[i] = stack.isEmpty() ? -1 : stack.peek();
            stack.push(arr[i]);
        }

        return result;
    }

    public static void main(String[] args) {
        int[] arr = {1, 3, 2, 4};
        int[] res = nextGreaterElement(arr);
        for (int i : res) {
            System.out.print(i + " ");
        }
    }
}
```

OUTPUT:

```
X:\Desktop\java\assignment4>javac Solution.java
X:\Desktop\java\assignment4>java Solution
3 4 4 -1
```

TIME COMPLEXITY: $O(n)$

7. Union of Two Sorted Arrays with Duplicate Elements

Given two sorted arrays $a[]$ and $b[]$, where each array may contain duplicate elements , the task is to return the elements in the union of the two arrays in sorted order.

Union of two arrays can be defined as the set containing distinct common elements that are present in either of the array.

Input: $a[] = [1, 2, 3, 4, 5]$, $b[] = [1, 2, 3, 6, 7]$ Output: 1 2 3 4 5 6 7

Input: $a[] = [2, 2, 3, 4, 5]$, $b[] = [1, 1, 2, 3, 4]$ Output: 1 2 3 4 5

CODE:

```
import java.util.*;
public class UnionOfArrays {
    public static void findUnion(int[] a, int[] b) {
        Set<Integer> resultSet = new HashSet<>();
        for (int num : a) {
            resultSet.add(num);
        }

        for (int num : b) {
            resultSet.add(num);
        }
        List<Integer> resultList = new ArrayList<>(resultSet);
        Collections.sort(resultList);
        for (int num : resultList) {
            System.out.print(num + " ");
        }
    }
}
```



```
}

public static void main(String[] args) {
    int[] a = {1, 2, 3, 4, 5};
    int[] b = {1, 2, 3, 6, 7};

    findUnion(a, b);
}
}
```

OUTPUT:

```
X:\Desktop\java\assignment4>javac UnionOfArrays.java

X:\Desktop\java\assignment4>java UnionOfArrays
1 2 3 4 5 6 7
```

TIME COMPLEXITY: $O((n + m) \log(n + m))$
