Date: 13/11/24

DSA Practice Problems

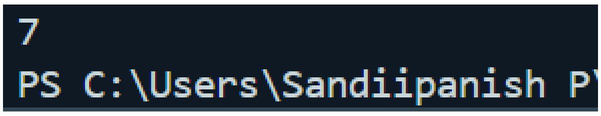
1. Kth Smallest

Code:

```
import java.util.Random;


public class KthSmallestElement {
   public static int kthSmallest(int[] arr, int k) {
      return quickSelect(arr, 0, arr.length - 1, k - 1);
   }


   private static int quickSelect(int[] arr, int low, int high, int k) {
      if (low == high) return arr[low];


      Random rand = new Random();
      int pivotIndex = low + rand.nextInt(high - low + 1);
      pivotIndex = partition(arr, low, high, pivotIndex);


      if (k == pivotIndex) return arr[k];
      else if (k < pivotIndex) return quickSelect(arr, low, pivotIndex - 1, k);
      else return quickSelect(arr, pivotIndex + 1, high, k);
   }


   private static int partition(int[] arr, int low, int high, int pivotIndex) {
      int pivotValue = arr[pivotIndex];
      swap(arr, pivotIndex, high);
      int storeIndex = low;
```

```
        for (int i = low; i < high; i++) {
            if (arr[i] < pivotValue) {
                swap(arr, storeIndex, i);
                storeIndex++;
            }
        }
        swap(arr, storeIndex, high);
        return storeIndex;
    }

    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[] arr1 = {7, 10, 4, 3, 20, 15};
        System.out.println(kthSmallest(arr1, 3));
    }
}
```

Output:



```
7
PS C:\Users\Sandiipanish P
```

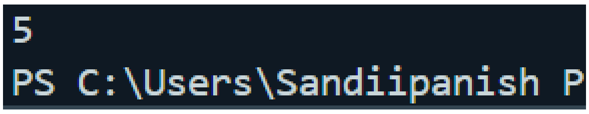
Time Complexity: O(n)

2. Minimize heights 2

Code:

```
import java.util.Arrays;


public class MinimizeHeightDifference {
    public static int getMinDiff(int[] arr, int n, int k) {
        Arrays.sort(arr);
        int minDiff = arr[n - 1] - arr[0];


        int smallest = arr[0] + k;
        int largest = arr[n - 1] - k;


        for (int i = 0; i < n - 1; i++) {
            int minHeight = Math.min(smallest, arr[i + 1] - k);
            int maxHeight = Math.max(largest, arr[i] + k);
            minDiff = Math.min(minDiff, maxHeight - minHeight);
        }


        return minDiff;
    }


    public static void main(String[] args) {
        int[] arr1 = {1, 5, 8, 10};
        int k1 = 2;
        System.out.println(getMinDiff(arr1, arr1.length, k1));
    }
}
```

Output:



```
5
PS C:\Users\Sandiipanish P
```

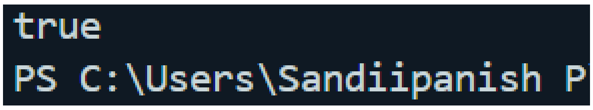
Time Complexity: O(n log n)

3. Parentheses Checker

Code:

```
import java.util.Stack;

public class BalancedBrackets {
    public static boolean isBalanced(String s) {
        Stack<Character> stack = new Stack<>();

        for (char c : s.toCharArray()) {
            if (c == '{' || c == '(' || c == '[') {
                stack.push(c);
            }
            else if (c == '}' || c == ')' || c == ']') {
                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) {

        String s1 = "{([])}";

        System.out.println(isBalanced(s1));

    }

}
```

Output:



```
true
PS C:\Users\Sandiipanish P
```

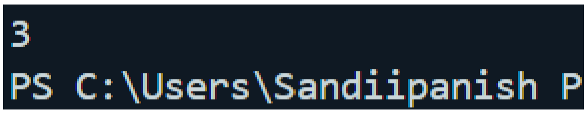
Time Complexity: O(n)

4.Equilibrium Point

Code:

```
public class EquilibriumPoint {

    public static int findEquilibriumPoint(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(findEquilibriumPoint(arr1));
    }
}
```

Output:



```
3
PS C:\Users\Sandiipanish P
```

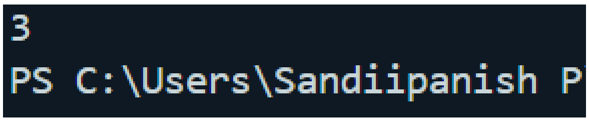
Time Complexity: O(n)

5. Binary Search

Code:

```
public class BinarySearch {
    public static int findPosition(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[] arr1 = {1, 2, 3, 4, 5};
        int k1 = 4;
        System.out.println(findPosition(arr1, k1));
    }
}
```

Output:



```
3
PS C:\Users\Sandiipanish P
```

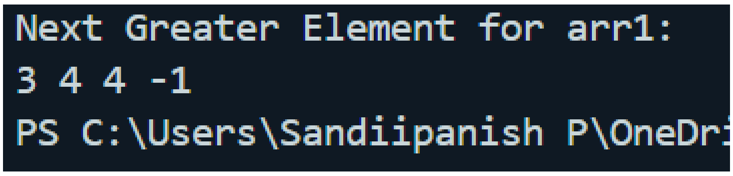
Time Complexity: O(log n)

6. Next Greater Element

Code:

```
import java.util.Stack;


public class NextGreaterElement {


    public static int[] findNextGreater(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();
            }


            if (!stack.isEmpty()) {
                result[i] = stack.peek();
            } else {
                result[i] = -1;
            }


            stack.push(arr[i]);
        }


        return result;
    }
    public static void main(String[] args) {
        int[] arr1 = {1, 3, 2, 4};

        System.out.println("Next Greater Element for arr1: ");
        printArray(findNextGreater(arr1));

    }

    public static void printArray(int[] arr) {
        for (int num : arr) {
            System.out.print(num + " ");
        }
        System.out.println();
    }
}
```

Output:



```
Next Greater Element for arr1:
3 4 4 -1
PS C:\Users\Sandiipanish P\OneDri
```

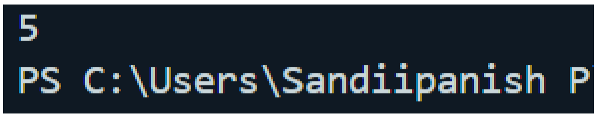
Time Complexity: O(n)

7. Union of two arrays with duplicate element

Code:

```
import java.util.HashSet;

public class UnionOfArrays {
    public static int findUnionCount(int[] a, int[] b) {
        HashSet<Integer> set = new HashSet<>();

        for (int num : a) set.add(num);

        for (int num : b) set.add(num);

        return set.size();

    }


    public static void main(String[] args) {

        int[] a1 = {1, 2, 3, 4, 5};

        int[] b1 = {1, 2, 3};

        System.out.println(findUnionCount(a1, b1));

    }

}
```

Output:



```
5
PS C:\Users\Sandiipanish P
```

Time Complexity: O(n+m)