DSA Practice 3

1. Kth Smallest Element :

Solution :

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

    private static int quickSelect(int[] arr, int left, int right, int k) {
        if (left == right) {
            return arr[left];
        }

        // Choose a pivot and partition the array
        int pivotIndex = partition(arr, left, right);

        if (pivotIndex == k) {
            return arr[pivotIndex];
        } else if (pivotIndex > k) {
            return quickSelect(arr, left, pivotIndex - 1, k);
        } else {
            return quickSelect(arr, pivotIndex + 1, right, k);
    }
    }

    private static int partition(int[] arr, int left, int right) {
        int pivot = arr[right];
        int i = left;


    for (int j = left; j < right; j++) {
            if (arr[j] <= pivot) {
                swap(arr, i, j);
                i++;
            }
        }
```

```
        swap(arr, i, right);
        return i; // Return the pivot index
    }

    private static void swap(int[] arr, int i, int j) {
        int temp = arr[i];
        arr[i] = arr[j];
        arr[j] = temp;
    }
}
```

Time Complexity : Average Case - O(n)

Worst Case : O(n^2)

2. Parantheses Checker :

Solution :

```
class Solution {
    // Function to check if brackets are balanced or not.
    static boolean isParenthesisBalanced(String s) {
        Stack<Character> stack = new Stack<>();
        for(char ch : s.toCharArray()){
            if(ch == '{' || ch == '[' || ch == '('){
                stack.push(ch);
            }
            else if(ch == '}' || ch == ']' || ch == ')'){
                if(stack.isEmpty()) {
                    return false;
                }
                char top = stack.peek();

                if ((ch == '}' && top == '{') ||
                    (ch == ']' && top == '[') ||
                    (ch == ')' && top == '(')) {
                    stack.pop();
```

```
            } else {
                return false;
            }
        }
    }

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

Time Complexity : O(n)

Space Complexity : O(n)

3. Equilibrium Point :

Solution :

```
class Solution {
    // Function to find equilibrium point in the array.
    public static int equilibriumPoint(int arr[]) {
        // code here
        int n = arr.length;
        if(n == 1) {
            return n;
        }

        int totalSum = 0;

        for(int num : arr) {
            totalSum+=num;
        }

        int left = 0;
        for(int i=0;i<n;i++) {
            totalSum-=arr[i];

            if(totalSum == left){
                return i+1;
```

```
            }

            left+=arr[i];
        }

        return -1;

    }
}
```

Time Complexity : O(n)

Space Complexity : O(1)

4. Next Greater Element :

Solution :

```
import java.util.ArrayList;
import java.util.Stack;

class Solution {
    public ArrayList<Integer> nextLargerElement(int[] arr) {
        int n = arr.length;
        ArrayList<Integer> result = new ArrayList<>(n);
        for (int i = 0; i < n; i++) {
            result.add(-1);
        }

        Stack<Integer> stack = new Stack<>();
        for (int i = n - 1; i >= 0; i--) {
            while (!stack.isEmpty() && arr[stack.peek()] <= arr[i]) {
                stack.pop();
            }

            if (!stack.isEmpty()) {
                result.set(i, arr[stack.peek()]);
            }
```

```
            stack.push(i);
        }

        return result;
    }
}
```

Time Complexity : O(n)

Space Complexity : O(n)

5. Union of two Arrays :

Solution :

```
class Solution {
    public static int findUnion(int a[], int b[]) {
        // code here
        HashSet<Integer> set1 = new HashSet<>();

        for(int i=0;i<a.length;i++) {
            set1.add(a[i]);
        }

        for(int i = 0;i<b.length;i++) {
            set1.add(b[i]);
        }

        int result = set1.size();
        return result;
    }
}
```

Time Complexity : O(n+m)

Space Complexity : O(n+m)

6. Minimize the Heights II :

Solution :

```
class Solution {
    int getMinDiff(int[] arr, int k) {
        // code here
        int n = arr.length;
        Arrays.sort(arr);
        int ans = arr[n - 1] - arr[0];

        int tempmin, tempmax;
        tempmin = arr[0];
        tempmax = arr[n - 1];

        for (int i = 1; i < n; i++) {

            if (arr[i] - k < 0)
                continue;

            tempmin = Math.min(arr[0] + k, arr[i] - k);

            tempmax
                = Math.max(arr[i - 1] + k, arr[n - 1] - k);
            ans = Math.min(ans, tempmax - tempmin);
        }
        return ans;
    }
}
```

Time Complexity : O(n log n)

Space Complexity : O(1)