Assignment 2 (Arrays)

Question 1 Given an integer array nums of 2n integers, group these integers into n pairs (a1, b1), (a2, b2),..., (an, bn) such that the sum of min(ai, bi) for all i is maximized. Return the maximized sum.

Example 1:

Input: nums = [1,4,3,2]

Output: 4

Sol:-

```java
// Time Complexity: O(nlogn)
// Space Complexity: O(n)
class Solution {
    public int arrayPairSum(int[] nums) {
        //Sort the array in ascending order
        Arrays.sort(nums);
        // initialize sum to zero
        int maxSum = 0;
        for(int i=0;i<nums.length; i +=2){
            // Add everey element at even positions(0-indexed)
            maxSum += nums[i];
        }
        return maxSum;
    }
}
```

Question 2 Alice has n candies, where the ith candy is of type candyType[i]. Alice noticed that she started to gain weight, so she visited a doctor. The doctor advised Alice to only eat n / 2 of the candies she has (n is always even). Alice likes her candies very much, and she wants to eat the maximum number of different types of candies while still following the doctor's advice. Given the integer array candyType of length n, return the maximum number of different types of candies she can eat if she only eats n / 2 of them. **Example 1:** Input: candyType = [1,1,2,2,3,3] Output: 3

Sol:-

```java
// TIme Complexity: O(n)
// Space Complexity: O(n)
class Solution {
    public int distributeCandies(int[] candyType) {
        // Create an empty Hash Set, and add each candy into it.
        Set<Integer> uniqueCandiesSet = new HashSet<>();
        for (int candy: candyType) {
            uniqueCandiesSet.add(candy);
        }
        // Then, find the answer in the same way as before.
        return Math.min(uniqueCandiesSet.size(), candyType.length / 2);
    }
}
```

Question 3 We define a harmonious array as an array where the difference between its maximum value and its minimum value is exactly 1. Given an integer array nums, return the length of its longest harmonious subsequence among all its possible subsequences. A subsequence of an array is a sequence that can be derived from the array by deleting some or no elements without changing the order of the remaining elements.

Example 1: Input: nums = [1,3,2,2,5,2,3,7]

Output: 5

Explanation: The longest harmonious subsequence is [3,2,2,2,3].

Sol:-

```
// Time Complexity: O(n)
// Space Complexity: O(n)
class Solution {
    public int findLHS(int[] nums) {
        Map<Integer, Integer> frequencyMap = new HashMap<>();

        // Count the frequency of each number in nums
        for (int num : nums) {
            frequencyMap.put(num, frequencyMap.getOrDefault(num, 0) + 1);
        }

        int longestSubsequence = 0;

        // Check each number in nums
        for (int num : nums) {
            // Check if there is a number with a difference of 1
            if (frequencyMap.containsKey(num + 1)) {
                int currentSubsequence = frequencyMap.get(num) + frequencyMap.get(num + 1);
                longestSubsequence = Math.max(longestSubsequence, currentSubsequence);
            }
        }

        return longestSubsequence;
    }
}
```

Question 4 You have a long flowerbed in which some of the plots are planted, and some are not. However, flowers cannot be planted in adjacent plots. Given an integer array flowerbed containing 0's and 1's, where 0 means empty and 1 means not empty, and an integer n, return true if n new flowers can be planted in the flowerbed without violating the no-adjacent-flowers rule and false otherwise.

Example 1: Input: flowerbed = [1,0,0,0,1], n = 1

Output: true

Sol:-

```
// Time Complexity: O(n)
// Space Complexity: O(1)
class Solution {
    public boolean canPlaceFlowers(int[] flowerbed, int n) {
        int count = 0;
        int i = 0;
        while (i < flowerbed.length) {
            // Check if the current plot is empty and its adjacent plots are also empty
           if (flowerbed[i] == 0 && (i == 0 || flowerbed[i - 1] == 0) && (i == flowerbed.length
- 1 || flowerbed[i + 1] == 0)) {
                flowerbed[i] = 1;  // Plant a flower
                count++;  // Increment the count of planted flowers
            }
            i++;  // Move to the next plot
        }
         return count >= n;  // Check if the number of planted flowers is greater than or
equal to n
    }
```

```
}
```

Question 5 Given an integer array nums, find three numbers whose product is maximum and return the maximum product.

Example 1: Input: nums = [1,2,3]

Output: 6

Sol:-

```
// Time Complexity: O(nlogn)
// Space Complexity: O(logn)
class Solution {
    public int maximumProduct(int[] nums) {
        Arrays.sort(nums);
        int case1 = nums[0]*nums[1]*nums[nums.length-1];
        int case2 = nums[nums.length-1]* nums[nums.length-2]*nums[nums.length-3];

        int maxProduct = Math.max(case1, case2);
        return maxProduct;
    }
}
```

Question 6 Given an array of integers nums which is sorted in ascending order, and an integer target, write a function to search target in nums. If target exists, then return its index. Otherwise, return -1. You must write an algorithm with O(log n) runtime complexity.

Input: nums = [-1,0,3,5,9,12], target = 9

Output: 4

Explanation: 9 exists in nums and its index is 4

Sol:-

```
// Time COmplexity: O(logn)
// Space Complexity: O(1)
class Solution {
    public int search(int[] nums, int target) {
        // set the left and right boundaries
        int low = 0;
        int high = nums.length - 1;
        // chcek the condition
        while(low <= high){
            // get the middle index and the middle value
            int mid = low + (high - low)/2;
            // case1: if target is equal to mid so, return mid
            if(nums[mid] == target){
                return mid;
            }
            // case2: if target is smaller than mid so, find the value left side of the mid
            else if(target < nums[mid]){
                high = mid - 1;
            }
            // case3: if target is greater than mid so, find the value right side of the mid
            else{
                low = mid + 1;
            }
        }
        // element not found in the list
```

```
            return -1;
        }
    }
```

Question 7 An array is monotonic if it is either monotone increasing or monotone decreasing. An array nums is monotone increasing if for all i <= j, nums[i] <= nums[j]. An array nums is monotone decreasing if for all i <= j, nums[i] >= nums[j]. Given an integer array nums, return true if the given array is monotonic, or false otherwise.

Example 1:

Input: nums = [1,2,2,3]

Output: true

Sol:-

```
// Time Complexity: O(n)
// Space Complexity: O(1)
class Solution {
    public boolean isMonotonic(int[] nums) {
        boolean increasing = true;
        boolean decreasing = true;
        // traverse the whole array
        for (int i = 0; i < nums.length - 1; i++) { // O(n)
            //case1: check the condtion if(i>i+1) so, array is not in ascending order
            if (nums[i] > nums[i+1])
                increasing = false;
                //case2: check the condtion if(i<i+1) so, array is not in decending order
            if (nums[i] < nums[i+1])
                decreasing = false;
        }

        return increasing || decreasing;
    }
}
```

Question 8 You are given an integer array nums and an integer k. In one operation, you can choose any index i where 0 <= i < nums.length and change nums[i] to nums[i] + x where x is an integer from the range [-k, k]. You can apply this operation at most once for each index i. The score of nums is the difference between the maximum and minimum elements in nums. Return the minimum score of nums after applying the mentioned operation at most once for each index in it.

Example 1:

Input: nums = [1], k = 0

Output: 0

Explanation: The score is max(nums) - min(nums) = 1 - 1 = 0.

Sol:-

```
// TIme Complexity: O(n)
// Space Complexity: O(1)
class Solution {
    public int smallestRangeI(int[] nums, int k) {
        int min = 10001, max = -1;
        // max - k would give us the minimum max
        // min + k would give us the maximum min
        // Their difference would give us the minimum score (difference)
        for(int i = 0; i< nums.length; i++){
            min = Math.min(min, nums[i]);
```

```
            max = Math.max(max, nums[i]);
        }
        int diff = (max - k) - (min + k);
        return Math.max(0, diff);
    }
}
```