Assignment 2 Solution - Arrays | DSA

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

Explanation: All possible pairings (ignoring the ordering of elements) are:

1. (1, 4), (2, 3) -> min(1, 4) + min(2, 3) = 1 + 2 = 3
2. (1, 3), (2, 4) -> min(1, 3) + min(2, 4) = 1 + 2 = 3
3. (1, 2), (3, 4) -> min(1, 2) + min(3, 4) = 1 + 3 = 4

So the maximum possible sum is 4

Solution:

```
package in.ineuron.pptAssignment02;
import java.util.Arrays;
public class ArrayPiarSum {

    public static void main(String[] args) {
        int[] nums = { 1, 4, 3, 2 };
        System.out.println("The maximum possible sum is " + arrayPairSum(nums));

    }

    public static int arrayPairSum(int[] nums) {
        // sort the array
        Arrays.sort(nums);
        int sum = 0;
        for (int i = 1; i < nums.length; i += 2) {
            sum += Math.min(nums[i], nums[i - 1]);
        }
        return sum;
    }
}
```

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
Explanation: Alice can only eat 6 / 2 = 3 candies. Since there are only 3 types, she can eat one of each type.

Solution:

```
package in.ineuron.pptAssignment02;
import java.util.Arrays;
public class DistibutedCandies {

    public static void main(String[] args) {
        int[] candTypes = { 1, 1, 2, 2, 3, 3 };
        System.out.println(distributeCandies(candTypes));

    }

    public static int distributeCandies(int[] candyType) {
        int n = candyType.length;
        if (n == 0) {
            return 0;
        }

        int type = 1;

        Arrays.sort(candyType);
        for (int i = 1; i < n; i++) {
            if (candyType[i] != candyType[i - 1]) {
                type++;
            }
        }

        return Math.min(type, n / 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].

Solution:

```
package in.ineuron.pptAssignment02;
import java.util.HashMap;
import java.util.Map;

public class FindLHS {

    public static void main(String[] args) {
        int[] nums = { 1, 3, 2, 2, 5, 2, 3, 7 };
        System.out.println("Longest Harmonious Subsequence :: " + findLHS(nums));

    }

    private static int findLHS(int[] nums) {
        int result = 0;
        Map<Integer, Integer> count = new HashMap<>();
        for (int i : nums) {
            count.put(i, count.getOrDefault(i, 0) + 1);
        }
        for (int i : count.keySet()) {
            if (count.containsKey(i + 1)) {
                result = Math.max(result, count.get(i) + count.get(i + 1));
            }
        }
        return result;
    }

}
//TC:O(n)
//SC:O(n)
```

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**

Solution:

```
package in.ineuron.pptAssignment02;
public class CanPlaceFlowers {
	public static void main(String[] args) {

		int[] flowerbed = { 1, 0, 0, 0, 1 };
		int n = 1;
		System.out.println("Can Place Flowers :: " + canPlaceFlowers(flowerbed, n));

	}
	private static boolean canPlaceFlowers(int[] flowerbed, int n) {
		int count = 0;

		for (int i = 0; i < flowerbed.length; i++) {

			if (flowerbed[i] == 0) {
				int prev = (i == 0 || flowerbed[i - 1] == 0) ? 0 : 1;
				int next = (i == flowerbed.length - 1 || flowerbed[i + 1] == 0) ? 0 : 1;

				if (prev == 0 && next == 0) {
					flowerbed[i] = 1;
					count++;
				}
			}
			if (count >= n) {
				return true;
			}
		}
		return false;
	}
}
```

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

Solution:

```
package in.ineuron.pptAssignment02;

public class MaximumProductofThreeNumbers {

    public static void main(String[] args) {

        int[] nums = { 1, 2, 3 };
        System.out.println("Maximum Product of Three Numbers::" +
        maximumProduct(nums));

    }

    public static int maximumProduct(int[] nums) {
        // Find the three largest numbers in the array.
        int max1 = Integer.MIN_VALUE;
        int max2 = Integer.MIN_VALUE;
        int max3 = Integer.MIN_VALUE;

        for (int num : nums) {
            if (num > max1) {
                max3 = max2;
                max2 = max1;
                max1 = num;
            } else if (num > max2) {
                max3 = max2;
                max2 = num;
            } else if (num > max3) {
                max3 = num;
            }
        }

        // Return the product of the three largest numbers.
        return max1 * max2 * max3;
    }

}
```

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

Solution:

```
package in.ineuron.pptAssignment02;

public class BinarySearch {

    public static void main(String[] args) {
        int nums[] = { -1, 0, 3, 5, 9, 12 };
        int target = 9;
        System.out.println(" 9 exists in nums and its index is :: " + search(nums, target));

    }

    public static int search(int[] nums, int target) {
        int low = 0;
        int high = nums.length - 1;

        while (low <= high) {
            int mid = (low + high) / 2;

            if (nums[mid] == target) {
                return mid;
            } else if (nums[mid] < target) {
                low = mid + 1;
            } else {
                high = mid - 1;
            }
        }

        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

Solution:

```
package in.ineuron.pptAssignment02;

public class MonotonicArray {

    public static void main(String[] args) {
        int[] nums = { 1, 2, 2, 3 };
        System.out.println("Monotonic Array :: " + isMonotonic(nums));
    }

    public static boolean isMonotonic(int[] nums) {

        boolean increasing = true;
        boolean decreasing = true;

        for (int i = 0; i < nums.length - 1; i++) {
            if (nums[i + 1] > nums[i]) {
                decreasing = false;
            }

            if (nums[i + 1] < nums[i]) {
                increasing = false;
            }

            if (increasing == false && decreasing == false) {
                return false;
            }
        }
        return true;
    }

}
```

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.

Solution:

```
package in.ineuron.pptAssignment02;
public class MinScore {
	public static void main(String[] args) {
		int[] nums = { 1};
		int k = 0;
		System.out.println("Min Score :: " + minScore(nums, k));

	}

	public static int minScore(int[] nums, int k) {
		// Find the minimum and maximum elements in the array.
		int min = Integer.MAX_VALUE;
		int max = Integer.MIN_VALUE;

		for (int num : nums) {
			min = Math.min(min, num);
			max = Math.max(max, num);
		}

		int minScore = max - min;
		for (int i = 0; i < nums.length; i++) {
			for (int x = -k; x <= k; x++) {
				int newNum = nums[i] + x;
				int newScore = max - newNum;

				if (newScore < minScore) {
					minScore = newScore;
				}
			}
		}
		return minScore;
	}
}
```