Assignment solution

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

```
import java.util.Arrays;


public class ArrayPairSum {

    public static int arrayPairSum(int[] nums) {

        Arrays.sort(nums);

        int maxSum = 0;

        for (int i = 0; i < nums.length; i += 2) {

            maxSum += nums[i];

        }

        return maxSum;

    }


    public static void main(String[] args) {

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

        int maxSum = arrayPairSum(nums);

        System.out.println("Maximum possible sum: " + 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

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

```
import java.util.HashSet;


public class MaxCandies {
    public static int maxCandies(int[] candyType) {
        int maxTypes = candyType.length / 2;
        HashSet<Integer> uniqueCandies = new HashSet<>();

        for (int candy : candyType) {
            uniqueCandies.add(candy);
            if (uniqueCandies.size() == maxTypes) {
                break;
            }
        }

        return uniqueCandies.size();
    }
    public static void main(String[] args) {

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

        int maxTypes = maxCandies(candyType);

        System.out.println("Maximum number of different types of candies Alice can eat: " +
maxTypes);

    }

}
```

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].

```
import java.util.HashMap;

public class LongestHarmoniousSubsequence {
    public static int findLHS(int[] nums) {
        HashMap<Integer, Integer> countMap = new HashMap<>();
        int longestSubsequence = 0;

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

        // Check each number and its adjacent numbers to find the longest harmonious
subsequence
        for (int num : nums) {
            if (countMap.containsKey(num + 1)) {
                int subsequenceLength = countMap.get(num) + countMap.get(num + 1);
                longestSubsequence = Math.max(longestSubsequence, subsequenceLength);
            }
        }

        return longestSubsequence;
    }

    public static void main(String[] args) {
        int[] nums = {1, 3, 2, 2, 5, 2, 3, 7};
        int longestSubsequence = findLHS(nums);
        System.out.println("Length of the longest harmonious subsequence: " +
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

```
public class CanPlaceFlowers {
    public static boolean canPlaceFlowers(int[] flowerbed, int n) {
        int count = 0;
        int length = flowerbed.length;
        int i = 0;

        while (i < length) {
            if (flowerbed[i] == 0 && (i == 0 || flowerbed[i - 1] == 0) && (i == length - 1 ||
flowerbed[i + 1] == 0)) {
                flowerbed[i] = 1;
                count++;

                if (count >= n) {
                    return true;
                }
            }
            i++;
        }

        return false;
    }

    public static void main(String[] args) {
        int[] flowerbed = {1, 0, 0, 0, 1};
        int n = 1;
        boolean canPlace = canPlaceFlowers(flowerbed, n);
        System.out.println("Can place " + n + " new flower(s) in the flowerbed? " + canPlace);
    }
}
```

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

```
import java.util.Arrays;

public class MaximumProduct {
    public static int maximumProduct(int[] nums) {
        Arrays.sort(nums);
        int n = nums.length;

        // The maximum product can be either the product of the three largest numbers
        // or the product of the two smallest numbers and the largest number
        int option1 = nums[n - 1] * nums[n - 2] * nums[n - 3];
        int option2 = nums[0] * nums[1] * nums[n - 1];

        return Math.max(option1, option2);
    }

    public static void main(String[] args) {
        int[] nums = {1, 2, 3};
        int maxProduct = maximumProduct(nums);
        System.out.println("Maximum product: " + 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

```
public class BinarySearch {
    public static int search(int[] nums, int target) {
        int left = 0;
        int right = nums.length - 1;

        while (left <= right) {
            int mid = left + (right - left) / 2;
            if (nums[mid] == target) {
                return mid;
            } else if (nums[mid] < target) {
                left = mid + 1;
            } else {
                right = mid - 1;
            }
        }

        return -1;
    }

    public static void main(String[] args) {
        int[] nums = {-1, 0, 3, 5, 9, 12};
        int target = 9;

        int index = search(nums, target);
        System.out.println("Index of target " + target + ": " + index);
    }
}
```

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

```
public class MonotonicArray {
    public static boolean isMonotonic(int[] nums) {
        boolean increasing = true;
        boolean decreasing = true;

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

        return increasing || decreasing;
    }

    public static void main(String[] args) {
        int[] nums = {1, 2, 2, 3};
        boolean isMonotonic = isMonotonic(nums);
        System.out.println("Is the array monotonic? " + isMonotonic);
    }
}
```

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.

```
import java.util.Arrays;

public class MinimumScore {
    public static int minimumScore(int[] nums, int k) {
        int n = nums.length;
        Arrays.sort(nums);

        int minScore = nums[n - 1] - nums[0];

        for (int i = 1; i <= k; i++) {
            int low = nums[i];
            int high = nums[n - k + i - 1];
            minScore = Math.min(minScore, high - low);
        }

        return minScore;
    }

    public static void main(String[] args) {
        int[] nums = {1};
        int k = 0;

        int minScore = minimumScore(nums, k);
        System.out.println("Minimum score: " + minScore);
    }
}
```