Assignment_02

July 19, 2024

1 Assignment 02

1.0.1 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

[ ]:
```
public static int maxPairSum(int[] nums) {
    // Sort the array nums in increasing order.
    Arrays.sort(nums);

    // Initialize the sum.
    int sum = 0;

    // Iterate through the array and add the minimum of each pair of adjacent␣
 ↪elements to the sum.
    for (int i = 0; i < nums.length / 2; i++) {
      sum += Math.min(nums[2 * i], nums[2 * i + 1]);
    }

    // Return the final sum.
    return sum;
  }
```

[ ]:
```
int[] nums = {1,4,3,2};
System.out.println(maxPairSum(nums));
```

4

1.0.2 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.

[ ]:
```
public static int maxDifferentCandies(int[] candyType) {
    // Check that the input is valid.
    if (candyType.length % 2 != 0) {
      throw new IllegalArgumentException("The input array must have an even␣
 ↪length.");
    }

    // Create a set to store the unique candy types.
    Set<Integer> candyTypes = new HashSet<>();

    // Iterate through the array and add each candy type to the set.
    for (int candy : candyType) {
      candyTypes.add(candy);
    }

    // Return the number of candy types in the set.
    return candyTypes.size();
  }
```

[ ]:
```
int[] candyType = {1, 1, 2, 2, 3, 3};
System.out.println(maxDifferentCandies(candyType));
```

```
3
```

1.0.3 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].

[ ]:
```
public static int longestHarmoniousSubsequence(int[] nums) {
    // Initialize the maximum length.
    int maxLen = 0;
```

```
    // Create a map to store the number of occurrences of each number.
    Map<Integer, Integer> countMap = new HashMap<>();

    // Iterate through the array and update the map.
    for (int num : nums) {
      countMap.put(num, countMap.getOrDefault(num, 0) + 1);
    }

    // Iterate through the map and find the longest harmonious subsequence.
    for (int num : countMap.keySet()) {
      int otherNum = num + 1;
      if (countMap.containsKey(otherNum)) {
        maxLen = Math.max(maxLen, countMap.get(num) + countMap.get(otherNum));
      }
    }

    // Return the maximum length.
    return maxLen;
  }
```

[ ]:
```
int[] nums = {1, 3, 2, 2, 5, 2, 3, 7};
System.out.println(longestHarmoniousSubsequence(nums));
```

```
5
```

1.0.4 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 static boolean canPlantFlowers(int[] flowerbed, int n) {
    // Initialize the count of empty plots.
    int count = 0;

    // Iterate through the array and count the number of empty plots.
    for (int i = 0; i < flowerbed.length; i++) {
      if (flowerbed[i] == 0) {
        if (i == 0 || flowerbed[i - 1] == 0) {
          count++;
        } else if (i == flowerbed.length - 1 || flowerbed[i + 1] == 0) {
          count++;
        }
      }
```

```
    }

    // Return true if the number of empty plots is at least n.
    return count >= n;
  }
```

[ ]:
```
int[] flowerbed = { 1, 0, 0, 0, 1};
int n = 1;
System.out.println(canPlantFlowers(flowerbed, n));
```

```
true
```

1.0.5 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

[ ]:
```
public static int maxProductOfThree(int[] nums) {
    // Initialize the maximum product.
    int maxProduct = Integer.MIN_VALUE;

    // Initialize the three largest numbers.
    int first = Integer.MIN_VALUE;
    int second = Integer.MIN_VALUE;
    int third = Integer.MIN_VALUE;

    // Iterate through the array and update the maximum product.
    for (int num : nums) {
      if (num > first) {
        third = second;
        second = first;
        first = num;
      } else if (num > second) {
        third = second;
        second = num;
      } else if (num > third) {
        third = num;
      }

      // Calculate the maximum product of three numbers.
      maxProduct = Math.max(maxProduct, first * second * third);
    }

    // Return the maximum product.
    return maxProduct;
  }
```

[ ]:
```
int[] nums = {1, 2, 3};
System.out.println(maxProductOfThree(nums));
```

6

1.0.6 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 static int search(int[] nums, int target) {
    // Initialize the left and right pointers.
    int left = 0;
    int right = nums.length - 1;

    // Iterate until the left pointer is greater than or equal to the right␣
 ↪pointer.
    while (left <= right) {
      // Calculate the middle index.
      int mid = (left + right) / 2;

      // Check if the middle element is equal to the target.
      if (nums[mid] == target) {
        return mid;
      } else if (nums[mid] < target) {
        // Update the left pointer.
        left = mid + 1;
      } else {
        // Update the right pointer.
        right = mid - 1;
      }
    }

    // Return -1 if the target is not found.
    return -1;
  }
```

[ ]:
```
int[] nums = {-1,0,3,5,9,12};
int target = 9;
System.out.println(search(nums, target));
```

4

1.0.7 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

[ ]:
```java
public static boolean isMonotonic(int[] nums) {
    // Initialize the increasing flag and decreasing flag.
    boolean increasing = true;
    boolean decreasing = true;

    // Iterate through the array.
    for (int i = 1; i < nums.length; i++) {
      // Check if the array is increasing.
      if (nums[i] < nums[i - 1]) {
        increasing = false;
      }

      // Check if the array is decreasing.
      if (nums[i] > nums[i - 1]) {
        decreasing = false;
      }

      // If the array is not increasing and not decreasing, return false.
      if (!increasing && !decreasing) {
        return false;
      }
    }

    // Return true if the array is monotonic.
    return true;
  }
```

[ ]:
```java
int[] nums = {1, 2,2,3};
System.out.println(isMonotonic(nums));
```

```
true
```

1.0.8 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.

[ ]:
```java
public static int minScore(int[] nums, int k) {
    // Initialize the minimum score.
    int minScore = Integer.MAX_VALUE;

    // Iterate through the array.
    for (int i = 0; i < nums.length; i++) {
      // Calculate the minimum score after applying the operation.
      int newScore = Math.min(nums[i] + k, nums[i] - k);

      // Update the minimum score.
      minScore = Math.min(minScore, newScore);
    }

    // Return the minimum score.
    return minScore;
  }
```

[ ]:
```java
int[] nums = {1};
int k = 0;
System.out.println(minScore(nums, k));
```

```
1
```

[ ]:
```java
int[] nums = {1, 3, 2, 4};
int k = 2;
System.out.println(minScore(nums, k));
```

```
-1
```

2 Thank You!