*Assignment Questions 2*

**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: import java.util.Arrays;

import java.util.Scanner;

public class MaximizePairsSum {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter the number of integers (2n): ");

int n = scanner.nextInt();

int[] nums = new int[2 \* n];

System.out.println("Enter the integers:");

for (int i = 0; i < 2 \* n; i++) {

nums[i] = scanner.nextInt();

}

int maxSum = arrayPairSum(nums);

System.out.println("Maximized sum: " + maxSum);

}

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;

}

}

**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: import java.util.HashSet;

import java.util.Set;

public class MaxCandies {

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: " + maxTypes);

}

public static int maxCandies(int[] candyType) {

int maxTypes = candyType.length / 2; // Maximum number of candies Alice can eat

Set<Integer> uniqueTypes = new HashSet<>();

for (int type : candyType) {

uniqueTypes.add(type); // Add each candy type to the set

}

// If the number of unique candy types is less than maxTypes,

// return the number of unique types. Otherwise, return maxTypes.

return Math.min(uniqueTypes.size(), 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].

Solution: import java.util.HashMap;

import java.util.Map;

import java.util.Scanner;

public class LongestHarmoniousSubsequence {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter the number of integers: ");

int n = scanner.nextInt();

int[] nums = new int[n];

System.out.println("Enter the integers:");

for (int i = 0; i < n; i++) {

nums[i] = scanner.nextInt();

}

int longestSubsequenceLength = findLHS(nums);

System.out.println("Length of the longest harmonious subsequence: " + longestSubsequenceLength);

}

public static int findLHS(int[] nums) {

Map<Integer, Integer> countMap = new HashMap<>();

int longestSubsequenceLength = 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 in the array

for (int num : nums) {

// Check if there exists a number with a difference of 1

if (countMap.containsKey(num + 1)) {

int subsequenceLength = countMap.get(num) + countMap.get(num + 1);

longestSubsequenceLength = Math.max(longestSubsequenceLength, subsequenceLength);

}

}

return longestSubsequenceLength;

}

}

**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: public class CanPlaceFlowers {

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 flowers: " + canPlace);

}

public static boolean canPlaceFlowers(int[] flowerbed, int n) {

int count = 0;

int i = 0;

while (i < flowerbed.length) {

// Check if the current plot is empty (0)

if (flowerbed[i] == 0) {

// Check if the previous and next plots are also empty

boolean prevEmpty = (i == 0 || flowerbed[i - 1] == 0);

boolean nextEmpty = (i == flowerbed.length - 1 || flowerbed[i + 1] == 0);

// If both previous and next plots are empty, plant a flower

if (prevEmpty && nextEmpty) {

flowerbed[i] = 1;

count++;

}

}

// If we have planted enough flowers, return true

if (count >= n) {

return true;

}

i++;

}

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: import java.util.Arrays;

import java.util.Scanner;

public class MaximumProduct {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter the number of integers: ");

int n = scanner.nextInt();

int[] nums = new int[n];

System.out.println("Enter the integers:");

for (int i = 0; i < n; i++) {

nums[i] = scanner.nextInt();

}

int maxProduct = maximumProduct(nums);

System.out.println("Maximum product: " + maxProduct);

}

public static int maximumProduct(int[] nums) {

Arrays.sort(nums);

int n = nums.length;

int product1 = nums[n - 1] \* nums[n - 2] \* nums[n - 3]; // Product of the three largest numbers

int product2 = nums[0] \* nums[1] \* nums[n - 1]; // Product of the two smallest numbers and the largest number

return Math.max(product1, product2);

}

}

**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: public class BinarySearch {

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: " + index);

}

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;

}

}

**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: import java.util.Scanner;

public class MonotonicArray {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter the number of integers: ");

int n = scanner.nextInt();

int[] nums = new int[n];

System.out.println("Enter the integers:");

for (int i = 0; i < n; i++) {

nums[i] = scanner.nextInt();

}

boolean isMonotonic = isMonotonic(nums);

System.out.println("Is the array monotonic? " + isMonotonic);

}

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;

}

}

**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: import java.util.Arrays;

import java.util.Scanner;

public class MinimumScore {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter the number of integers: ");

int n = scanner.nextInt();

int[] nums = new int[n];

System.out.println("Enter the integers:");

for (int i = 0; i < n; i++) {

nums[i] = scanner.nextInt();

}

System.out.print("Enter the value of k: ");

int k = scanner.nextInt();

int minScore = minimumScore(nums, k);

System.out.println("Minimum score: " + minScore);

}

public static int minimumScore(int[] nums, int k) {

Arrays.sort(nums);

int n = nums.length;

int min = nums[0] + k;

int max = nums[n - 1] - k;

return Math.max(0, max - min);

}

}