**DSA – ASSIGNMENT 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. :-**

* First, we sort the array nums in ascending order.
* We then iterate through the sorted array, incrementing i by 2 in each iteration to ensure we select every other element.
* In each iteration, we add the value of nums[i] (which represents the minimum element of the pair) to the max\_sum.
* Finally, we return max\_sum, which represents the maximized sum of the minimum elements in each pair.

**def arrayPairSum(nums):**

**nums.sort() # Sort the array in ascending order**

**max\_sum = 0**

**for i in range(0, len(nums), 2):**

**max\_sum += nums[i] # Add the minimum element of each pair to the sum**

**return max\_sum**

**# Test the function with the given example**

**nums = [1, 4, 3, 2]**

**maximized\_sum = arrayPairSum(nums)**

**print(maximized\_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. :-**

* First, we calculate the number of unique types of candies by converting candyType into a set (which automatically removes duplicates) and taking the length of the set.
* Next, we calculate the maximum number of candies Alice can eat by dividing the length of candyType by 2 (using integer division to ensure an integer result).
* Finally, we return the minimum value between the number of unique candies and the maximum number of candies Alice can eat. This ensures we don't exceed the limit imposed by the doctor.

**def maxCandies(candyType):**

**unique\_candies = len(set(candyType)) # Count the unique types of candies**

**max\_candies = len(candyType) // 2 # Calculate the limit**

**return min(unique\_candies, max\_candies) # Return the minimum of the two values**

**# Test the function with the given example**

**candyType = [1, 1, 2, 2, 3, 3]**

**max\_diff\_types = maxCandies(candyType)**

**print(max\_diff\_types)**

💡 **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. :-**

* First, we iterate through the array nums and count the frequencies of each number using a dictionary num\_counts.
* Then, for each number num, we check if there exists another number num + 1 in num\_counts. If so, we calculate the length of the harmonious subsequence formed by those two numbers, which is the sum of their frequencies.
* Finally, we update the max\_length variable if a longer harmonious subsequence is found.
* At the end, we return the max\_length, which represents the length of the longest harmonious subsequence.

**def findLHS(nums):**

**num\_counts = {} # Dictionary to store the frequencies of numbers**

**max\_length = 0**

**# Count the frequencies of numbers**

**for num in nums:**

**num\_counts[num] = num\_counts.get(num, 0) + 1**

**# Check for each number if there exists another number with difference 1**

**for num in num\_counts:**

**if num + 1 in num\_counts:**

**length = num\_counts[num] + num\_counts[num + 1]**

**max\_length = max(max\_length, length)**

**return max\_length**

**# Test the function with the given example**

**nums = [1, 3, 2, 2, 5, 2, 3, 7]**

**longest\_subsequence\_length = findLHS(nums)**

**print(longest\_subsequence\_length)**

💡 **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. :-**

* We initialize variables count and i to keep track of the number of flowers planted and the current index, respectively.
* We iterate through the flowerbed using a while loop until we reach the end or plant all the required flowers.
* In each iteration, we check if the current plot flowerbed[i] is empty (0) and its adjacent plots (if exist) are also empty. If so, we plant a flower by setting flowerbed[i] to 1 and increment the count variable.
* If count reaches n, we have successfully planted all the flowers, and we return True.
* If we reach the end of the iteration and count is still less than n, it means we were not able to plant all the flowers, and we return False.

**def canPlaceFlowers(flowerbed, n):**

**length = len(flowerbed)**

**count = 0**

**i = 0**

**while i < length:**

**if (**

**flowerbed[i] == 0**

**and (i == 0 or flowerbed[i - 1] == 0)**

**and (i == length - 1 or flowerbed[i + 1] == 0)**

**):**

**flowerbed[i] = 1 # Plant a flower**

**count += 1 # Increment the count**

**if count == n:**

**return True**

**i += 1**

**return False**

**# Test the function with the given example**

**flowerbed = [1, 0, 0, 0, 1]**

**n = 1**

**can\_plant = canPlaceFlowers(flowerbed, n)**

**print(can\_plant)**

💡 **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. :-**

* We start by sorting the array nums in ascending order using the sort() method.
* Then, we calculate two possible maximum products:
  + product1 is the product of the three largest numbers in the sorted array nums[-1], nums[-2], and nums[-3].
  + product2 is the product of the two smallest (negative) numbers nums[0] and nums[1], and the largest number nums[-1].
* Finally, we compare product1 and product2 using the max() function and return the maximum product.

**def maximumProduct(nums):**

**nums.sort() # Sort the array in ascending order**

**# Calculate the two possible maximum products**

**product1 = nums[-1] \* nums[-2] \* nums[-3] # Product of three largest numbers**

**product2 = nums[0] \* nums[1] \* nums[-1] # Product of two smallest and one largest number**

**return max(product1, product2) # Return the maximum product**

**# Test the function with the given example**

**nums = [1, 2, 3]**

**max\_product = maximumProduct(nums)**

**print(max\_product)**

💡 **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. :-**

* We initialize two pointers, left and right, representing the start and end indices of the search range, respectively. Initially, left is set to 0, and right is set to the last index of the array nums.
* Inside the while loop, we calculate the middle index mid using the formula left + (right - left) // 2.
* We compare the value at the middle index nums[mid] with the target. If they are equal, we have found the target and return the index mid.
* If nums[mid] is less than the target, it means the target is in the right half of the search range. We update left to mid + 1 to search in the right half.
* If nums[mid] is greater than the target, it means the target is in the left half of the search range. We update right to mid - 1 to search in the left half.
* The while loop continues until left becomes greater than right, indicating that the target is not found in the array. In this case, we return -1.

**def search(nums, target):**

**left = 0**

**right = len(nums) - 1**

**while left <= right:**

**mid = left + (right - left) // 2 # Calculate the middle index**

**if nums[mid] == target:**

**return mid # Found the target, return the index**

**elif nums[mid] < target:**

**left = mid + 1 # Target is in the right half**

**else:**

**right = mid - 1 # Target is in the left half**

**return -1 # Target not found**

**# Test the function with the given example**

**nums = [-1, 0, 3, 5, 9, 12]**

**target = 9**

**index = search(nums, target)**

**print(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

**Solution. :-**

* We initialize two boolean variables, is\_increasing and is\_decreasing, to True.
* We iterate through the array nums starting from index 1.
* In each iteration, we compare the current element nums[i] with the previous element nums[i - 1].
  + If nums[i] is less than nums[i - 1], it violates the monotone increasing condition, so we set is\_increasing to False.
  + If nums[i] is greater than nums[i - 1], it violates the monotone decreasing condition, so we set is\_decreasing to False.
* After the iteration, if either is\_increasing or is\_decreasing is True, it means the array is monotonic, and we return True. Otherwise, we return False.

**def isMonotonic(nums):**

**is\_increasing = True**

**is\_decreasing = True**

**for i in range(1, len(nums)):**

**if nums[i] < nums[i - 1]:**

**is\_increasing = False**

**if nums[i] > nums[i - 1]:**

**is\_decreasing = False**

**return is\_increasing or is\_decreasing**

**# Test the function with the given example**

**nums = [1, 2, 2, 3]**

**is\_monotonic = isMonotonic(nums)**

**print(is\_monotonic)**

💡 **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. :-**

* We start by sorting the array nums in ascending order using the sort() method.
* Next, we adjust the minimum element by adding k and the maximum element by subtracting k from them.
* After adjusting the elements, we find the new minimum and maximum elements using the min() and max() functions, respectively.
* Finally, we return the difference between the new maximum and minimum elements as the minimum score.

**def minimumScore(nums, k):**

**nums.sort() # Sort the array in ascending order**

**# Adjust the minimum and maximum elements**

**nums[0] += k**

**nums[-1] -= k**

**# Find the new minimum and maximum elements**

**new\_min = min(nums)**

**new\_max = max(nums)**

**return new\_max - new\_min**

**# Test the function with the given example**

**nums = [1]**

**k = 0**

**min\_score = minimumScore(nums, k)**

**print(min\_score)**