NAME – AMEY HUSHAR PATKAR

ASSIGMENT NO 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

ANS –

To maximize the sum of the minimum values in pairs of an integer array nums, you can sort the array and pair adjacent elements.

Here's the step-by-step solution:

Sort the nums array in non-decreasing order.

Initialize a variable max\_sum to 0.

Iterate through the sorted array by taking two elements at a time:

Pair the current element nums[i] with the next element nums[i+1].

Add the minimum value between nums[i] and nums[i+1] to the max\_sum.

Return the max\_sum.

Here's the implementation in Python:

def arrayPairSum(nums):

nums.sort()

max\_sum = 0

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

max\_sum += nums[i]

return max\_sum

Let's test the function using the example you provided:

nums = [1, 4, 3, 2]

result = arrayPairSum(nums)

print("Output:", result)

The output will be:

Output: 4

It means that for the given array [1, 4, 3, 2], the maximum possible sum of the minimum values in pairs is 4. The function returns 4 as the output.

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.

ANS –

To find the maximum number of different types of candies Alice can eat while following the doctor's advice, you need to determine the number of unique candy types in the given array candyType and compare it with half the length of the array.

Here's the step-by-step solution:

Initialize a set called unique\_candy\_types to keep track of the unique candy types.

Iterate through each element candy in candyType:

Add candy to the unique\_candy\_types set.

Calculate the maximum number of different types of candies Alice can eat:

Take the minimum value between the length of the unique\_candy\_types set and half the length of the candyType array.

Return the maximum number of different types of candies Alice can eat.

Here's the implementation in Python:

def distributeCandies(candyType):

unique\_candy\_types = set()

for candy in candyType:

unique\_candy\_types.add(candy)

max\_candy\_types = min(len(unique\_candy\_types), len(candyType) // 2)

return max\_candy\_types

Let's test the function using the example you provided:

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

result = distributeCandies(candyType)

print("Output:", result)

The output will be:

Output: 3

It means that for the given array [1, 1, 2, 2, 3, 3], Alice can eat a maximum of 3 different types of candies. The function returns 3 as the output.

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].

ANS -

To find the length of the longest harmonious subsequence in an integer array nums, you can count the frequencies of each number in the array and check for pairs of numbers whose frequencies differ by 1.

Here's the step-by-step solution:

Initialize a dictionary called num\_freq to store the frequencies of numbers in nums.

Iterate through each element num in nums:

If num is not in num\_freq, add it as a key and set its frequency to 1.

If num is already in num\_freq, increment its frequency by 1.

Initialize a variable max\_length to 0.

Iterate through each key num in num\_freq:

If num + 1 is also a key in num\_freq, update max\_length to the maximum value between max\_length and the sum of the frequencies of num and num + 1.

Return max\_length.

Here's the implementation in Python:

def findLHS(nums):

num\_freq = {}

for num in nums:

if num not in num\_freq:

num\_freq[num] = 1

else:

num\_freq[num] += 1

max\_length = 0

for num in num\_freq:

if num + 1 in num\_freq:

max\_length = max(max\_length, num\_freq[num] + num\_freq[num + 1])

return max\_length

Let's test the function using the example you provided:

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

result = findLHS(nums)

print("Output:", result)

The output will be:

Output: 5

It means that for the given array [1, 3, 2, 2, 5, 2, 3, 7], the longest harmonious subsequence is [3, 2, 2, 2, 3] with a length of 5. The function returns 5 as the output.

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

ANS –

To determine whether n new flowers can be planted in a flowerbed, we can iterate through the flowerbed and check if each position is available for planting based on the adjacent positions.

Here's the step-by-step solution:

Initialize a variable count to 0 to keep track of the number of flowers that can be planted.

Iterate through the flowerbed from left to right:

Check if the current position i is empty (flowerbed[i] == 0) and its adjacent positions are also empty (flowerbed[i-1] != 1 and flowerbed[i+1] != 1 or handling boundary cases).

If the conditions are met, increment count by 1 and mark the current position as planted by setting flowerbed[i] to 1.

Stop iterating if count reaches n.

Return True if count is equal to n, indicating that n new flowers can be planted. Otherwise, return False.

Here's the implementation in Python:

def canPlaceFlowers(flowerbed, n):

count = 0

length = len(flowerbed)

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)

):

count += 1

flowerbed[i] = 1

if count == n:

return True

i += 1

return False

Let's test the function using the example you provided:

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

n = 1

result = canPlaceFlowers(flowerbed, n)

print("Output:", result)

The output will be:

vbnet

Output: True

It means that for the given flowerbed [1, 0, 0, 0, 1], it is possible to plant 1 new flower without violating the adjacent flowers rule. The function returns True as the output.

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

ANS –

To find the maximum product of three numbers in an integer array nums, we can sort the array in non-decreasing order and consider two cases:

The maximum product is the product of the three largest numbers in nums. We take the product of nums[-1], nums[-2], and nums[-3].

The maximum product is the product of the largest number and the two smallest numbers (if the array contains negative numbers). We take the product of nums[-1], nums[0], and nums[1].

We compare the results from these two cases and return the larger product.

Here's the step-by-step solution:

Sort the array nums in non-decreasing order.

Compute the maximum product using the first case: max\_product\_1 = nums[-1] \* nums[-2] \* nums[-3].

Compute the maximum product using the second case: max\_product\_2 = nums[-1] \* nums[0] \* nums[1].

Return the maximum of max\_product\_1 and max\_product\_2.

Here's the implementation in Python:

def maximumProduct(nums):

nums.sort()

max\_product\_1 = nums[-1] \* nums[-2] \* nums[-3]

max\_product\_2 = nums[-1] \* nums[0] \* nums[1]

return max(max\_product\_1, max\_product\_2)

Let's test the function using the example you provided:

nums = [1, 2, 3]

result = maximumProduct(nums)

print("Output:", result)

The output will be:

Output: 6

It means that for the given array [1, 2, 3], the maximum product of three numbers is 6. The function returns 6 as the output.

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

ANS –

To search for a target value in a sorted array nums with O(log n) runtime complexity, we can use the binary search algorithm.

Here's the step-by-step solution:

Initialize two pointers, left and right, pointing to the first and last elements of the array, respectively.

While left is less than or equal to right, do the following:

Calculate the middle index as mid = (left + right) // 2.

If the middle element nums[mid] is equal to the target, return mid.

If the middle element is greater than the target, update right = mid - 1 to search in the left half of the array.

If the middle element is less than the target, update left = mid + 1 to search in the right half of the array.

If the target is not found after the loop, return -1 to indicate that the target does not exist in the array.

Here's the implementation in Python:

def search(nums, target):

left = 0

right = len(nums) - 1

while left <= right:

mid = (left + right) // 2

if nums[mid] == target:

return mid

elif nums[mid] > target:

right = mid - 1

else:

left = mid + 1

return -1

Let's test the function using the example you provided:

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

target = 9

result = search(nums, target)

print("Output:", result)

The output will be:

Output: 4

It means that the target value 9 exists in the nums array, and its index is 4. The function returns 4 as the output.

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

ANS –

To determine whether an array nums is monotonic, we can iterate through the array and check if it is either monotone increasing or monotone decreasing. If we find any violation of the monotonic condition, we return False. Otherwise, we return True.

Here's the step-by-step solution:

Initialize two boolean variables, isIncreasing and isDecreasing, as True.

Iterate through the array nums starting from the second element:

If the current element nums[i] is less than the previous element nums[i-1], set isIncreasing to False.

If the current element nums[i] is greater than the previous element nums[i-1], set isDecreasing to False.

If both isIncreasing and isDecreasing are False, return False as the array is neither monotone increasing nor monotone decreasing.

If the loop completes without any violation, return True.

Here's the implementation in Python:

def isMonotonic(nums):

isIncreasing = True

isDecreasing = True

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

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

isIncreasing = False

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

isDecreasing = False

if not isIncreasing and not isDecreasing:

return False

return True

Let's test the function using the example you provided:

nums = [1, 2, 2, 3]

result = isMonotonic(nums)

print("Output:", result)

The output will be:

Output: True

It means that the given array [1, 2, 2, 3] is monotonic, and the function returns True as the output.

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.

ANS –

To find the minimum score of the given array nums after applying the operation at most once for each index, we can take the following steps:

Find the minimum and maximum elements in the array nums.

Calculate the initial score as the difference between the maximum and minimum elements: score = max(nums) - min(nums).

If the difference score is already 0, return 0 since no operation is needed.

For each index i in the array nums, calculate the potential minimum and maximum values after applying the operation with the range [-k, k]:

The potential minimum value is nums[i] - k.

The potential maximum value is nums[i] + k.

Update the minimum and maximum values accordingly.

Calculate the new score as the difference between the updated maximum and minimum values: new\_score = max\_val - min\_val.

Return the minimum of the initial score and the new score.

Here's the implementation in Python:

def minimumScore(nums, k):

min\_val = min(nums)

max\_val = max(nums)

score = max\_val - min\_val

if score == 0:

return 0

for num in nums:

potential\_min = num - k

potential\_max = num + k

min\_val = min(min\_val, potential\_min)

max\_val = max(max\_val, potential\_max)

new\_score = max\_val - min\_val

return min(score, new\_score)

Let's test the function using the example you provided:

nums = [1]

k = 0

result = minimumScore(nums, k)

print("Output:", result)

The output will be:

Output: 0

It means that the minimum score of the array [1] after applying the operation is 0, and the function returns 0 as the output.