Untitled4

June 26, 2023

[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
'''

# Program :-
def arrayPairSum(nums):
    nums.sort()
    sum = 0
    for i in range(0, len(nums), 2):
        sum += min(nums[i], nums[i+1])
    return sum

# Test the function
nums = [1, 4, 3, 2]
print(arrayPairSum(nums))
```

```
4
```

[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.
'''

# Program :-
def maxCandies(candyType):
    uniqueTypes = set()
    for candy in candyType:
        uniqueTypes.add(candy)
    return min(len(uniqueTypes), len(candyType) // 2)

# Test the function
candyType = [1, 1, 2, 2, 3, 3]
print(maxCandies(candyType))
```

```
3
```

[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].
'''

# Program :-
from collections import defaultdict

def findLHS(nums):
    counter = defaultdict(int)
    for num in nums:
        counter[num] += 1

    max_length = 0
    for num in counter:
        if num + 1 in counter:
            max_length = max(max_length, counter[num] + counter[num + 1])

    return max_length

# Test the function
nums = [1, 3, 2, 2, 5, 2, 3, 7]
print(findLHS(nums))
```

```
5
```

[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
'''

# Program :-
def canPlaceFlowers(flowerbed, n):
    count = 0
    length = len(flowerbed)
    for i in range(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
    return False

# Test the function
flowerbed = [1, 0, 0, 0, 1]
n = 1
print(canPlaceFlowers(flowerbed, n))
```

```
True
```

[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
'''

# Program :-
def maximumProduct(nums):
    nums.sort()
    n = len(nums)
    return max(nums[n-1] * nums[n-2] * nums[n-3], nums[0] * nums[1] * nums[n-1])

# Test the function
nums = [1, 2, 3]
print(maximumProduct(nums))
```

```
6
```

[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
'''

# Program :-
def search(nums, target):
    left = 0
    right = len(nums) - 1

    while left <= right:
        mid = left + (right - left) // 2

        if nums[mid] == target:
            return mid
        elif nums[mid] < target:
            left = mid + 1
        else:
            right = mid - 1

    return -1

# Test the function
nums = [-1, 0, 3, 5, 9, 12]
target = 9
print(search(nums, target))
```

```
4
```

[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
'''

# Program :-
def isMonotonic(nums):
    isIncreasing = True
    isDecreasing = True

    for i in range(1, len(nums)):
        if nums[i] > nums[i - 1]:
            isDecreasing = False
        if nums[i] < nums[i - 1]:
            isIncreasing = False
        if not isIncreasing and not isDecreasing:
            return False

    return True

# Test the function
nums = [1, 2, 2, 3]
print(isMonotonic(nums))
```

```
True
```

[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.
'''

# Program :-
def minimumScore(nums, k):
    minVal = float('inf')
    maxVal = float('-inf')

    for num in nums:
        minVal = min(minVal, num + k)
        maxVal = max(maxVal, num - k)
    if maxVal - minVal <= 2 * k:
        return 0

    midValue = (minVal + maxVal) // 2

    minVal = float('inf')
    maxVal = float('-inf')

    for num in nums:
        if num <= midValue:
            minVal = max(minVal, num + k)
        if num > midValue:
            maxVal = min(maxVal, num - k)

    return maxVal - minVal

# Test the function
nums = [1]
k = 0
print(minimumScore(nums, k))
```

```
0
```