ASSIGNMENT -2

ARRAYS

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:

Algorithm:

1) Sort the given array nums.
2) Use a loop to traverse through the array by a step count of 2, to fetch the sum of minimum numbers.
3) TC: O(nlogn) – as we sort the array
4) SC: O(1) – as no extra space was used

Code:

```python
class Solution:
    def arrayPairSum(self, nums: List[int]) -> int:
        nums.sort()
        res=0
        for i in range(0,len(nums),2):
            res = res+nums[i]
        return res
```

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:

Algorithm:

1) Divide the given length of array by 2 to get half the length of it and assign it to candy.
2) Get the length of candyType to know the unique types of candies using SET operation and assign it to types.
3) Finally, return the minimum among the two (i.e., candy and types)
4) TC: O(logn) – as we use set operation
5) SC: O(1) – as it uses a constant space.

Code:

```
class Solution:
    def distributeCandies(self, candyType: List[int]) -> int:
        candy = len(candyType)//2
        types= len(set(candyType))
        return min(candy,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:

Algorithm:

1) Initialize a variable called max_length to 0.
2) Iterate through each key in freq.
3) If key + 1 is also in freq, update max_length to be the maximum of its current value and the sum of the frequencies of key and key + 1 in freq.
4) Return max_length, which is the length of the longest harmonious subsequence in nums.
5) TC: O(n)
6) SC: O(n)

Code:

```
class Solution:
    def findLHS(self, nums: List[int]) -> int:
        occur = Counter(nums)
        max_len = 0

        for key in occur:
            if key + 1 in occur:
                max_len = max(max_len, occur[key]+ occur[key+1])
        return max_len
```

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:

TC: O(n)

SC: O(1)

Algorithm:

1) First initialize the given array adding two zeros in the front and back.
2) Use a loop to iterate over the length of the array starting from first index.
3) Check if the previous and the next indices are empty of the current index position.
4) If True, we can plant the flowerbed. Repeat this for the length of the given array and keep decrementing the value of n if the flowerbed can be planted.
5) Finally, check if the value of n is equal to zero. If yes, return True else return False.
6) TC: O(n) – as we traverse the length of the array elements.
7) SC: O(1) – done at constant space.

Code:

```
class Solution:
    def canPlaceFlowers(self, flowerbed: List[int], n: int) -> bool:
        s = [0] + flowerbed + [0] # to skip first & last
        for i in range(1,len(s)-1):
            if s[i-1] == 0 and s[i]== 0 and s[i+1]==0:
                s[i]=1
                n -= 1
        return n <= 0
```

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:

Algorithm:

Brute Force: This approach takes three loops – TC : O(n^3)

1) Firstly, sort the given array.
2) Then find the product of the last three sorted elements and the product of first two and last element.
3) Finally, find the max of the two products and return the result.
4) TC: O(nlogn) – as we sort the given array.
5) SC: O(1) – done at constant space.

Code:

```python
class Solution:
    def maximumProduct(self, nums: List[int]) -> int:
        nums = sorted(nums)
        prod1 = nums[-1]*nums[-2]*nums[-3]
        prod2 = nums[0]*nums[1]*nums[-1]
        return max(prod1,prod2)
```

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:

Algorithm: - Binary Search (two pointers)

1) Initialise two pointers left and right with 0 and length of array – 1.
2) Use a loop to traverse all the elements of the array. Find the mid value of the array and check if its equal to the target, if yes, return it.
3) If no, see if the mid value is greater or lesser than the target. When its greater decrement the right pointer and the vice versa.
4) If the target is not found return -1.
5) TC: O(logn) – as we use binary search.
6) SC: O(1) – its done at constant space.

Code:

```
class Solution:
    def search(self, nums: List[int], target: int) -> int:
        l,r = 0 , len(nums)-1
        while l <= r:
            mid = (l+r)//2
            if nums[mid] > target:
                r= mid -1
            elif nums[mid] < target:
                l = mid + 1
            else:
                return mid
        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:

Algorithm:

1) Initialize the two variables to True.
2) Use a loop to iterate over the elements, and check if the current value is greater than the previous values if yes, return True. If not return False.
3) TC: O(n) – as we iterate over the array once.
4) SC: O(1) – its performed at constant space.

Code:

```
class Solution:
    def isMonotonic(self, nums: List[int]) -> bool:
        incr = decr = True
        for i in range(1, len(nums)):
            if nums[i] > nums[i-1]:
                decr = False
            if nums[i] < nums[i-1]:
                incr = False
        return incr or decr
```

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:

Algorithm:

1) Initialize two variables mini and maxi values by finding the min value of the array and adding it to the k value. And find the max value of the array and subtract from the value k.
2) Check if the value of mini is greater than the maxi values, and return 0.
3) Else return the difference of the two variables.
4) TC: O(n) – as we use the min, max functions.
5) SC: O(1) – its done at a constant space.

Code:

```python
class Solution:
    def smallestRangeI(self, nums: List[int], k: int) -> int:
        mini = min(nums)+k
        maxi = max(nums)-k
# when min becomes greater than max return 0 as it gives -ve value
        if mini > maxi:
            return 0
        else:
            return maxi - mini
```