Enlai Yii
2064210

Programming Assignment 3

1. Algorithm Explanation

An array of integers with duplicates, called nums, will be taken as a parameter. nums will be sorted at the beginning to have an efficient way to get all of the frequencies based on each element in nums. Once the dictionary is created called count, which will save all of the frequencies, all of the duplicates (if any) will be removed by creating a list based on each key in count. Which will be saved in nums. A cache will be created for memorization for the count dictionary and nums array. And it'll be an efficient, fast way to access count and nums with cache and keep track of what's inside count and nums.

The MAX-POINTS function will be the score system for the problem. Where we will decide whether the difference between 2 elements is 1 or not 1. If the difference is 1, then we will find the maximum value to include the chosen element and skip the next element, or exclude the chosen element and choose the next element. If the difference is not 1, then we include the chosen element and choose the next element.

(a) Let MAX-POINTS(i) denote the maximum points earned when the ai is chosen, then ai, ai - 1, and ai + 1 is removed from nums. Then ai points will be added into the score.

(b) The recursive formulation for the algorithm will be:

- Max-Points(i) = max(ai * count[ai] + Max-Point(i - 2), Max-Point(n-1))
  When the difference of ai and a(i-1) is 1.
- Max-Points(i) = ai * count[ai] + Max-Point(i - 1)
  When the difference of ai and a(i-1) is not 1.

The base cases will be:

- When n = 0, we return the only element in the array and the count associated to the element
- When n = 1, we find the difference between the current element and the next element. When the difference is 1, then we return the maximum value: a0*count[a0] or a1*count[a1]. When the difference is not 1, then we return the both elements combined.

2. Pseudocode

2a) Pseudocode for DELETE-AND-EARN

```
function DELETE-AND-EARN(nums):
    nums.sort()

    count = {}
    for num in nums:
       if num not in count:
            count[num] = 1
       else:
            count[num] += 1

    nums = list(count.key())
    cache = {}
    return MAX-POINTS(i)
```

2b) Pseudocode for MAX-POINTS

```
function Max-Points(i):
    if i = 0:
       return a0 * count[a0]
    if i = 1:
       if a1 - a0 ≠ 1:
            return a0 * count[a0] + a1 * count[a1]
       else:
            return max(a0 * count[a0], a1 * count[a1])

    if ai - a(i-1) ≠ 1:
       return ai * count[ai] + Max-Point(i - 1)
    else:
       include = ai * count[ai] + Max-Point(i - 2)
```

```
1           exclude = Max-Point(i - 1)
2           return max(include, exclude)
1
3
1
4
1
5
```

3. Runtime

The runtime of the solution will be O(nlogn) when the array is sorted. Then O(n) when counting the frequency of each element in the array and creating an array based on the dictionary keys will be less than or equal to O(n). For the MAX-POINTS function, accessing each element from the array and the dictionary, adding, multiplying, and getting the maximum element will be O(1). For each time we call Max-Points(), the time complexity will be O(n). The overall runtime and worst case will be O(nlogn).

4. Correctness

The algorithm will always give the maximum points based on the requirements. When the array is sorted at the beginning, the dictionary will be created efficiently as each element found more than once will be incremented immediately and a new key is added when the element changes. A new num will be based on the keys and a cache will be established to access every element efficiently.

Since we want to remove ai + 1 and ai - 1, then return ai points into a sum, we should find the difference between the current element and the previous element. If the difference is 1, then we will find the maximum value between including the current element with its frequency and skipping the next element or excluding the current element and choosing the next element. This is because we should add the current element and skip the next position since that should be removed. The other option will not add the current element and move to the next position because the current element is

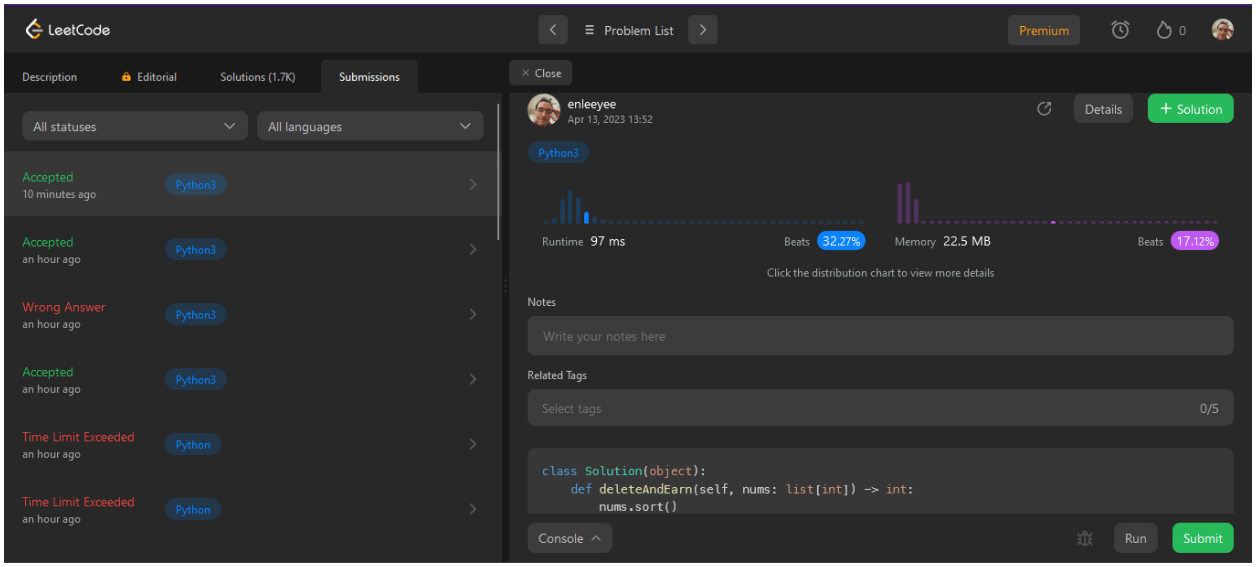
removed and move on. If the difference is not 1, then we include the current element with its frequency and choose the next element because only the current element is added, nothing is removed except the current element, and we move to the next element.

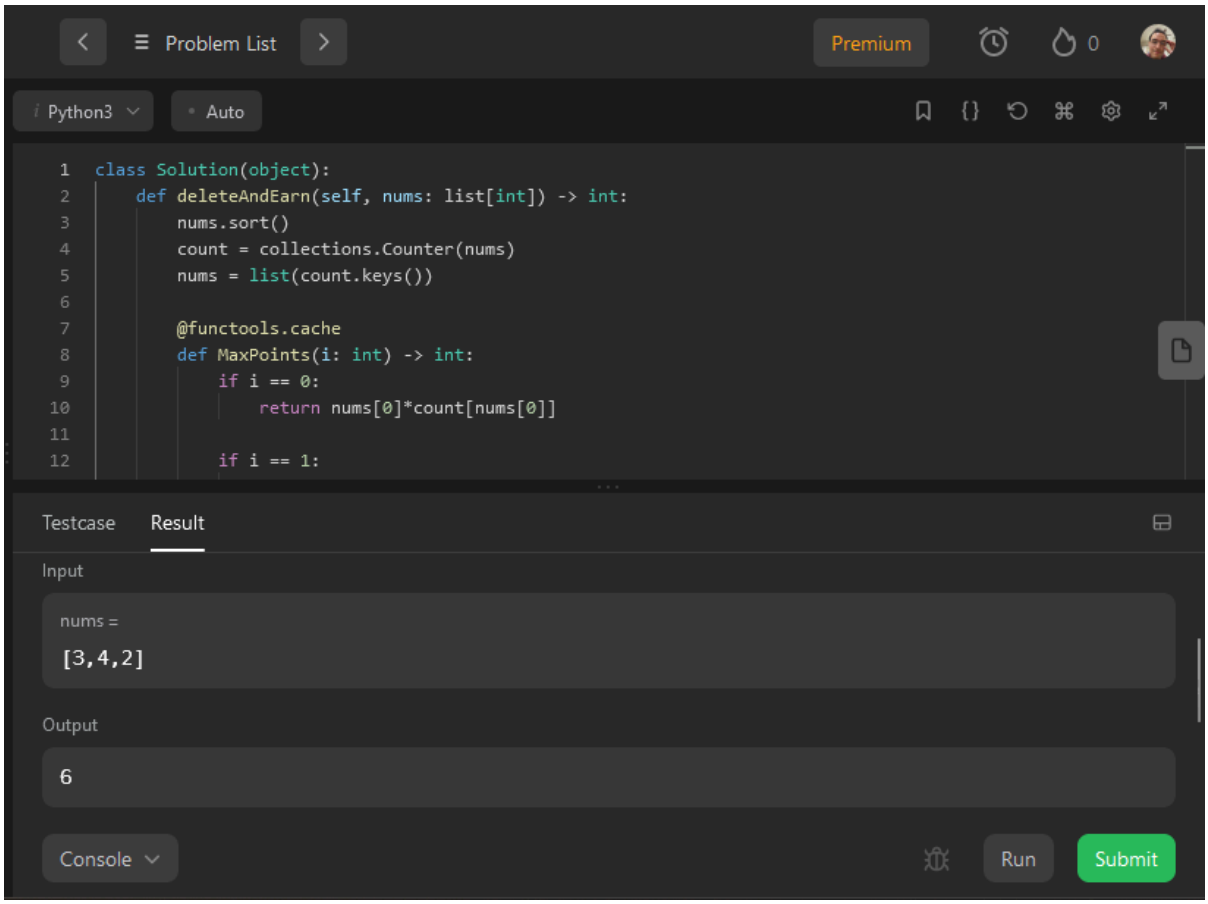
The base cases will prevent the function from going out of bounds as our cases are 0 and 1. If the base case is 0, then the current element and its frequency will be returned. If the base case is 1, then the difference between the first and final element will be calculated. We will include both elements with their frequency when the difference is not 1. Else, we will take the maximum value because the biggest value should be taken as points as biggest value - 1 will be the other element in the array.

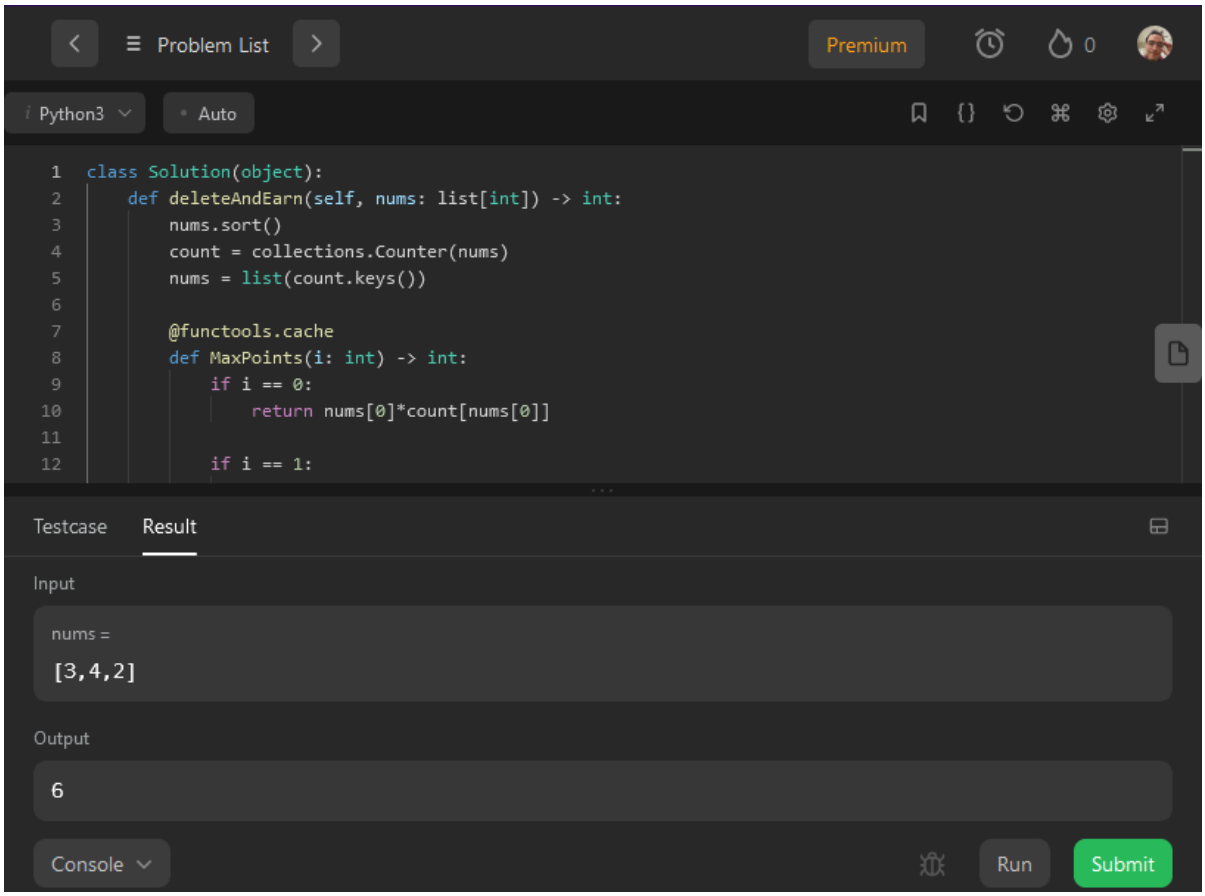
The runtime, in the worst case, will be O(nlogn). So the algorithm is efficient and will always return the most amount of points from nums.

5. LeetCode Submission

https://leetcode.com/problems/delete-and-earn/submissions/933244103/




Problem List
Premium
0
Python3
Auto
class Solution(object):
def deleteAndEarn(self, nums: list[int]) -> int:
nums.sort()
count = collections.Counter(nums)
nums = list(count.keys())
@functools.cache
def MaxPoints(i: int) -> int:
if i == 0:
return nums[0]*count[nums[0]]
if i == 1:
Testcase
Result
Input
nums =
[3,4,2]
Output
6
Console
Run
Submit


Problem List
Premium
0
Python3
Auto
class Solution(object):
def deleteAndEarn(self, nums: list[int]) -> int:
nums.sort()
count = collections.Counter(nums)
nums = list(count.keys())
@functools.cache
def MaxPoints(i: int) -> int:
if i == 0:
return nums[0]*count[nums[0]]
if i == 1:
Testcase
Result
Input
nums =
[3,4,2]
Output
6
Console
Run
Submit

6. Code

```
class Solution(object):
    def deleteAndEarn(self, nums: list[int]) -> int:
        nums.sort()
        count = collections.Counter(nums)
        nums = list(count.keys())

        @functools.cache
        def MaxPoints(i: int) -> int:
            if i == 0:
                return nums[0]*count[nums[0]]

            if i == 1:
                if nums[1] - nums[0] != 1:
                    return nums[0]*count[nums[0]] + nums[1]*count[nums[1]]

                else:
                    return max(nums[0]*count[nums[0]], nums[1]*count[nums[1]])


            if nums[i] - nums[i - 1] != 1:
                return nums[i]*count[nums[i]] + MaxPoints(i - 1)

            else:
                include = nums[i]*count[nums[i]] + MaxPoints(i - 2)
                exclude = MaxPoints(i - 1)
                return max(include, exclude)


        return MaxPoints(len(nums) - 1)
```