Enlai Yii

2064210

Programming Assignment 1

1. Algorithm Explanation

An integer array, nums, will be taken as a parameter with a size of n. An empty array called result will be created to save any new permutations created from the function. Then, new\_element will get the final element from nums. The function will get every element except the last element when the function is recursively called. The next for loop will get every single position where the last element can be inserted into the current permutation. Once the permutation is created, then the current permutation will be added to the array result. Once all of the permutations (or n! permutations) are created, then the function will return the next function, even\_odd\_lists.

**Given an integer array, i.e. (3,4,6), with the size of n. We can store and remove the last element from the array. Now, from the current array, we can recursively create permutations and insert the stored element into the array from 0 … n - 1. Finally, output all of the permutations of n distinct elements.**

The function, even\_odd\_lists, will create two empty arrays, even and odd. The function will sort the array result into even and odd lists based on their last element. Once the array result has been iterated through, then both arrays will be merged as the arrays are returned.

Input = [1, 2, 3]

Permutation for [1,2] is recursively solved.

[1,2] [2,1]

You can add the last element (3) at the index 0, 1, 2 or 0 to n - 1

[3, 1, 2] [3, 2, 1]

[1, 3, 2] [2, 3, 1]

[1, 2, 3] [2, 1, 3]

n = 3, we get 3! = 6 permutations.

1. Pseudocode

2a) Pseudocode for permutation

1 function perms(nums):

2 if len(nums) = 0

3 return [[]]

4 new\_element = nums[-1]

5 result = []

**# recursive call to divide nums into permutations**

6 for perm in perms(nums[:-1])

**# for loop to get every single position to place the**

**new element in the 0th index to the (n-1)th index**

7 for i in 0 . . . n - 1

8 new\_perm = perm.insert(i, new\_element)

9 result.append(new\_perm)

10 return even\_odd\_lists(result)

2b) Pseudocode for Even/Odd function

1 function even\_odd\_lists(result):

2 even = []

3 odd = []

4 for element in result:

5 if element[-1] mod 2 = 0

6 even.append(element)

7 else

8 odd.append(element)

9 return even + odd

1. Runtime

For the permutation function, on line 6, the for loop is called n - 1 \* n - 2 \* … \* 1 times. Which is equivalent to n! times. Then, on line 7, the for loop will be called 1 + (n - 1) times = n times. Finally, the function will be called n(n!) times = O(n!).

For the even/odd function, on line 4, the for loop will be called 1 + (n - 1) times = n times, similar to the second for loop (previous function on line 5). From line 5 through 8, an if-statement will only compare itself, so n times. The total runtime for the function will be n + n = 2n times = O(n).

The total runtime for the whole entire algorithm will be O(n!).

1. **Proof by Induction**

**Base Case:**

**n = 0, there is only one permutation, the empty set.**

**Induction Hypothesis:**

**Assume the algorithm is correct for n elements in the array nums. We will get n! permutations for an array with size n. We want to prove that the algorithm is correct for**

**n + 1 elements with (n + 1)! permutations.**

**Inductive Step:**

**For an array with size n + 1, the new\_element will be the final element, or (n + 1) - 1 = nth element. Since an array with size n contains n! permutations, we can assume the recursive call will be called n \* (n - 1) \* (n - 2) \* … \* 1 = n\*n! times. Finally, there will be n + 1 positions where the final element can be placed. So, there will be (n + 1)(n)(n!) … (2)(1) permutations. Therefore, we will have (n + 1)n! = (n + 1)! permutations.**

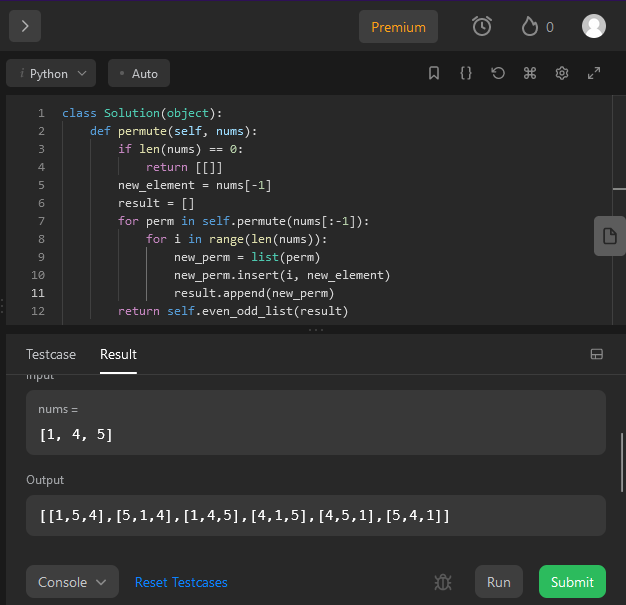
**Thus, the algorithm is correct for an array with n + 1 elements.**  ⏹

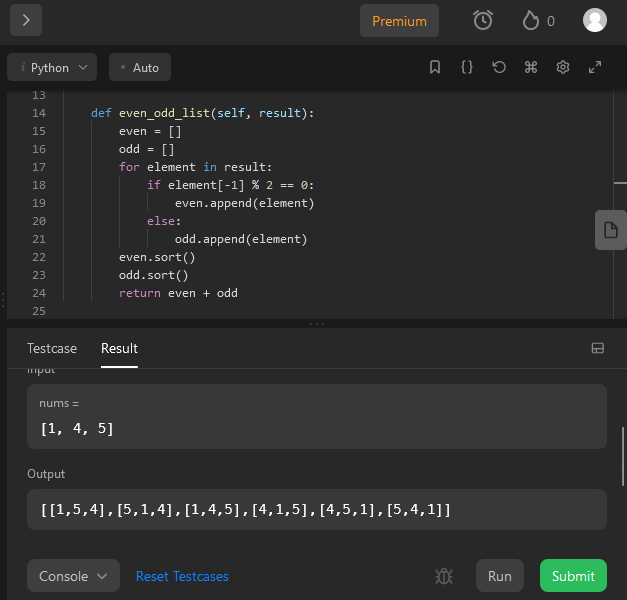
1. LeetCode link

<https://leetcode.com/problems/permutations/submissions/896911561/>

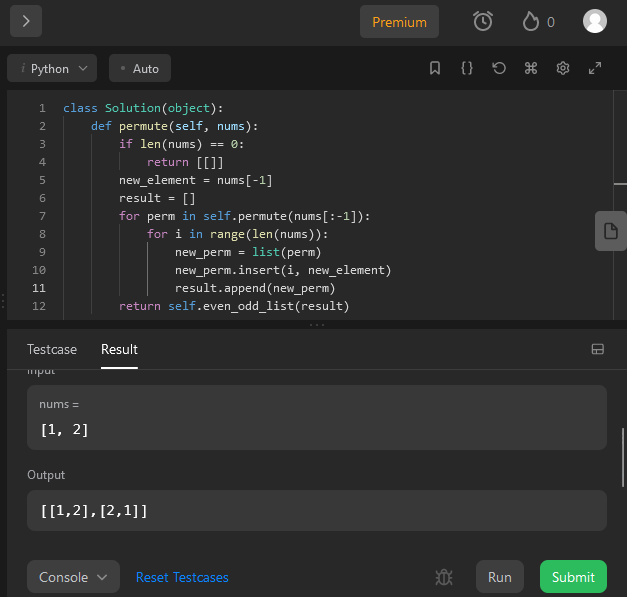
1. LeetCode Screenshot

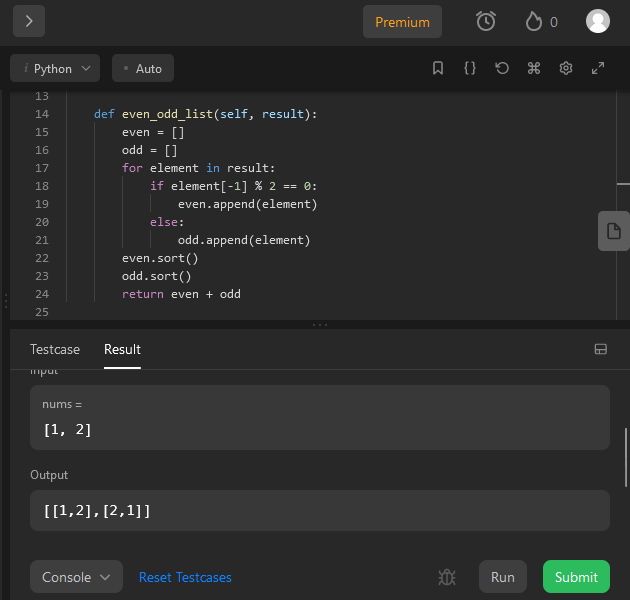
Example 1:



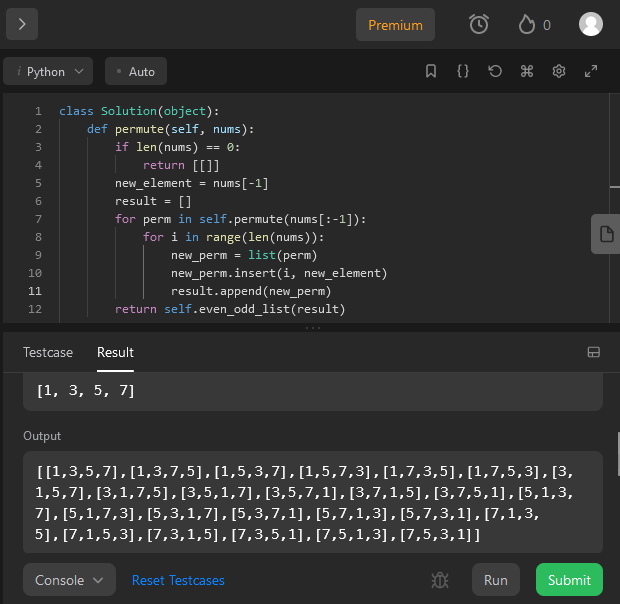


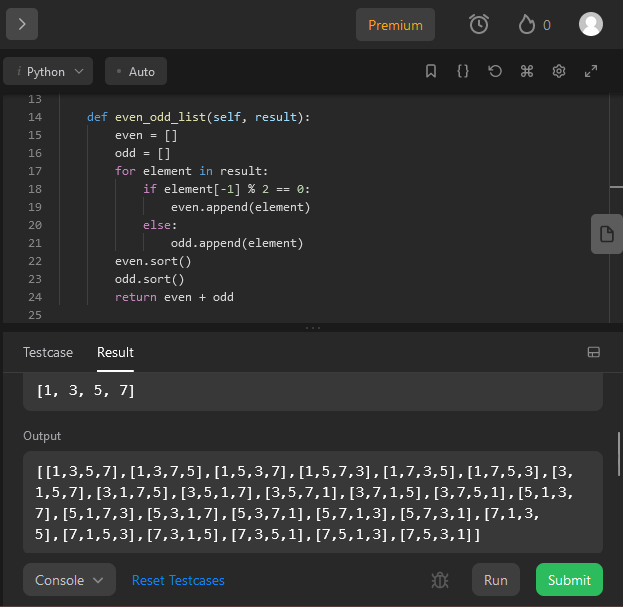
Example 2:





Example 3:





1. Code

1 class Solution(object):

2 def deleteAndEarn(self, nums: list[int]) -> int:

3 nums.sort()

4 count = collections.Counter(nums)

5 nums = list(count.keys())

6

7 @functools.cache

8 def MaxPoints(i: int) -> int:

9 if i == 0:

10 return nums[0]\*count[nums[0]]

11 if i == 1:

12 if nums[1] - nums[0] != 1:

13 return nums[0]\*count[nums[0]] + nums[1]\*count[nums[1]]

14 else:

15 return max(nums[0]\*count[nums[0]], nums[1]\*count[nums[1]])

16 if nums[i] - nums[i - 1] != 1:

17 return nums[i]\*count[nums[i]] + MaxPoints(i - 1)

18 else:

19 include = nums[i]\*count[nums[i]] + MaxPoints(i - 2)

20 exclude = MaxPoints(i - 1)

21 return max(include, exclude)

22 return MaxPoints(len(nums) - 1)