

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 $0 \dots n - 1$. 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.

2. Pseudocode

2a) Pseudocode for permutation

```
1  function perms(nums):
2      if len(nums) == 0
3          return [[]]
4      new_element = nums[-1]
5      result = []
6
7      # recursive call to divide nums into permutations
8
9      for perm in perms(nums[:-1])
10         # for loop to get every single position to place the
11         new element in the 0th index to the (n-1)th index
12
13         for i in 0 . . . n - 1
14             new_perm = perm.insert(i, new_element)
15             result.append(new_perm)
16
17     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
10     return even + odd
```

3. Runtime

For the permutation function, on line 6, the for loop is called $n - 1 * n - 2 * \dots * 1$ 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 $O(n!)$.

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

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

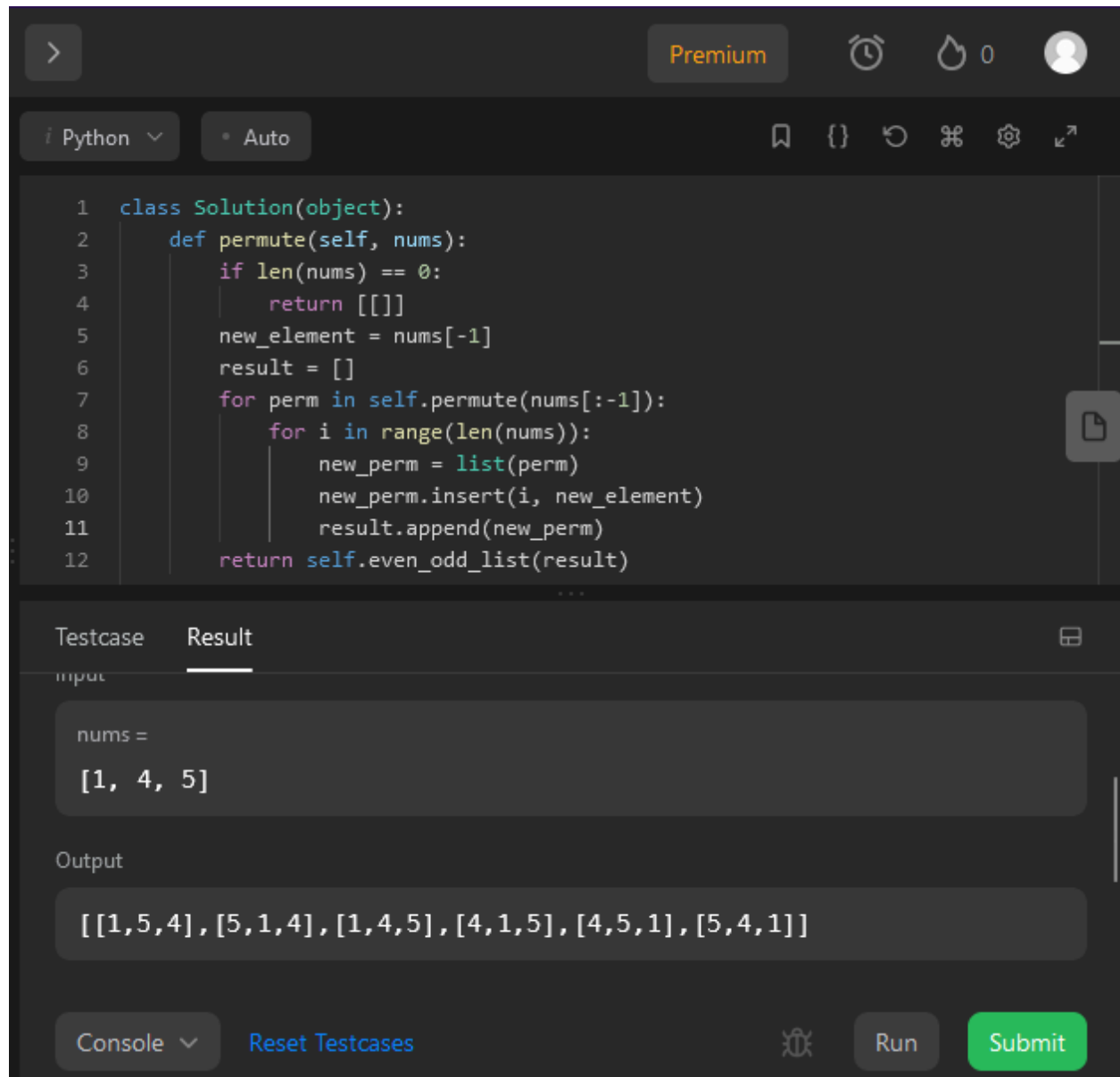


5. LeetCode link

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

6. LeetCode Screenshot

Example 1:



The screenshot shows a LeetCode submission interface. At the top, there's a navigation bar with a back arrow, a 'Premium' badge, and icons for a clock, a flame with '0', and a user profile. Below this is a toolbar with 'Python' and 'Auto' tabs, and icons for bookmarks, code folding, undo, redo, search, settings, and a share icon. The main area displays a Python solution for the 'Permutations' problem. The code defines a 'Solution' class with a 'permute' method that uses recursion to generate all permutations of the input list 'nums'. The method returns a list of lists. Below the code editor, there's a 'Testcase' tab and a 'Result' tab. The 'Result' tab is active, showing the input 'nums = [1, 4, 5]' and the output '[[1, 5, 4], [5, 1, 4], [1, 4, 5], [4, 1, 5], [4, 5, 1], [5, 4, 1]]'. At the bottom, there's a 'Console' dropdown, a 'Reset Testcases' button, and 'Run' and 'Submit' buttons.

```
1 class Solution(object):
2     def permute(self, nums):
3         if len(nums) == 0:
4             return [[]]
5         new_element = nums[-1]
6         result = []
7         for perm in self.permute(nums[:-1]):
8             for i in range(len(nums)):
9                 new_perm = list(perm)
10                new_perm.insert(i, new_element)
11                result.append(new_perm)
12        return self.even_odd_list(result)
```

Testcase Result

input

nums =
[1, 4, 5]




Output

[[1, 5, 4], [5, 1, 4], [1, 4, 5], [4, 1, 5], [4, 5, 1], [5, 4, 1]]

Console Reset Testcases Run Submit







>

Premium

 0

Python


Auto



```
13
14     def even_odd_list(self, result):
15         even = []
16         odd = []
17         for element in result:
18             if element[-1] % 2 == 0:
19                 even.append(element)
20             else:
21                 odd.append(element)
22         even.sort()
23         odd.sort()
24         return even + odd
25
```

Testcase

Result



input


nums =
[1, 4, 5]

Output

[[1,5,4], [5,1,4], [1,4,5], [4,1,5], [4,5,1], [5,4,1]]

Console

[Reset Testcases](#)



Run

Submit

Example 2:

The screenshot shows a coding interface with a dark theme. At the top, there's a navigation bar with a back arrow, a 'Premium' badge, and icons for a clock, a flame with '0', and a user profile. Below this is a toolbar with 'Python' and 'Auto' buttons, and icons for bookmarks, code folding, undo, redo, search, settings, and a share icon. The main editor displays a Python class `Solution` with a `permute` method. The code is as follows:

```
1 class Solution(object):
2     def permute(self, nums):
3         if len(nums) == 0:
4             return [[]]
5         new_element = nums[-1]
6         result = []
7         for perm in self.permute(nums[:-1]):
8             for i in range(len(nums)):
9                 new_perm = list(perm)
10                new_perm.insert(i, new_element)
11                result.append(new_perm)
12        return self.even_odd_list(result)
```

Below the editor is a 'Testcase' tab with a 'Result' sub-tab. The 'input' section shows `nums = [1, 2]`. The 'Output' section shows `[[1,2], [2,1]]`. At the bottom, there's a 'Console' dropdown, a 'Reset Testcases' link, a 'Run' button, and a green 'Submit' button.



Premium



Python

Auto



```
13
14     def even_odd_list(self, result):
15         even = []
16         odd = []
17         for element in result:
18             if element[-1] % 2 == 0:
19                 even.append(element)
20             else:
21                 odd.append(element)
22         even.sort()
23         odd.sort()
24         return even + odd
25
```

Testcase

Result



input

```
nums =
[1, 2]
```

Output

```
[[1,2], [2,1]]
```

Console

[Reset Testcases](#)



Run

Submit

Example 3:

The screenshot shows a coding environment with a dark theme. At the top, there's a navigation bar with a back arrow, a 'Premium' badge, and icons for a clock, a flame with '0', and a user profile. Below this is a toolbar with 'Python' and 'Auto' tabs, and icons for bookmarks, code folding, undo, redo, search, settings, and a share icon. The main area contains a Python code snippet for a class 'Solution' with a 'permute' method. The method uses recursion to generate all permutations of the input list 'nums'. Below the code editor, there's a 'Testcase' tab showing '[1, 3, 5, 7]' and a 'Result' tab showing the output: a list of 24 permutations. At the bottom, there's a 'Console' tab, a 'Reset Testcases' button, a 'Run' button, and a green 'Submit' button.

```
1 class Solution(object):
2     def permute(self, nums):
3         if len(nums) == 0:
4             return [[]]
5         new_element = nums[-1]
6         result = []
7         for perm in self.permute(nums[:-1]):
8             for i in range(len(nums)):
9                 new_perm = list(perm)
10                new_perm.insert(i, new_element)
11                result.append(new_perm)
12        return self.even_odd_list(result)
```

Testcase Result

[1, 3, 5, 7]




Output

[[1,3,5,7], [1,3,7,5], [1,5,3,7], [1,5,7,3], [1,7,3,5], [1,7,5,3], [3,1,5,7], [3,1,7,5], [3,5,1,7], [3,5,7,1], [3,7,1,5], [3,7,5,1], [5,1,3,7], [5,1,7,3], [5,3,1,7], [5,3,7,1], [5,7,1,3], [5,7,3,1], [7,1,3,5], [7,1,5,3], [7,3,1,5], [7,3,5,1], [7,5,1,3], [7,5,3,1]]

Console Reset Testcases Run Submit







>

Premium

  0 

Python


Auto

```
13
14     def even_odd_list(self, result):
15         even = []
16         odd = []
17         for element in result:
18             if element[-1] % 2 == 0:
19                 even.append(element)
20             else:
21                 odd.append(element)
22         even.sort()
23         odd.sort()
24         return even + odd
25
```

Testcase

Result




[1, 3, 5, 7]

Output

```
[[1,3,5,7], [1,3,7,5], [1,5,3,7], [1,5,7,3], [1,7,3,5], [1,7,5,3], [3,
1,5,7], [3,1,7,5], [3,5,1,7], [3,5,7,1], [3,7,1,5], [3,7,5,1], [5,1,3,
7], [5,1,7,3], [5,3,1,7], [5,3,7,1], [5,7,1,3], [5,7,3,1], [7,1,3,
5], [7,1,5,3], [7,3,1,5], [7,3,5,1], [7,5,1,3], [7,5,3,1]]
```

Console

[Reset Testcases](#)



Run

Submit

7. 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)
```