Programming Assignment 1

COSC 3320 Algorithms and Data Structures

Tristan Fry

1 Written Explanation

The algorithm is a divide and conquer / decrease and conquer algorithm, that uses backtracking to solve the problem. We first start by dividing the main array into smaller arrays, by n - 1. We do this by removing the first element in the array for which we store for later use. We continue this process until the size of the sub arrays is equal to 1. We then back track, and combine the sub arrays, then find all of the permutations of that sub array, and add the removed value to the end of the array. It is very important that the removed value does not get put back anywhere else except for at the back of the, in order to prevent duplicate permutations from happening. We then continue the process of back tracking all the way back until the problem has been fully solved. Then we finally check to see if the last element of a given set permutation is even, if it is we move it to the front of the array containing all sets of permutations as per the instruction requirement.

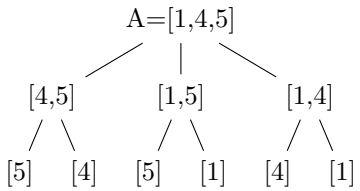
2 Pseudo Code

```
function PERMUTATIONS(A)
    R ← [∅]
    if array length equals 1 then
        return [nums]
    end if
    for i in array length do
        n ← A.pop(0)
        perms ← RECURSE(A)
        for each element in perms, y, do
            append the removed value n, to y
        end for
        add perms to result array
        append n to nums
    end for
    for x to length of R do
        if last index at given permutation x, is even then
            SWAP R[x] and R[0]
        end if
    end for
    return [R]
```

3 Proof Of Correctness

Base Case: This is the trivial case where S has a length of 1. The algorithm is correct, since the output will just be the single element in S which is all permutations of the given array S.
Induction Hypothesis: Assume that algorithm is correct for all sets of size less than n.
Induction Step: Assuming the induction hypothesis, we will show that the algorithm is correct for an input of length less than n. Suppose we call Permutation on an array of size n, it will recursively call Permutations and create sub arrays by $n-1$, until n has a length of one, thus we have already shown to be true. This concludes the proof.

4 Runtime



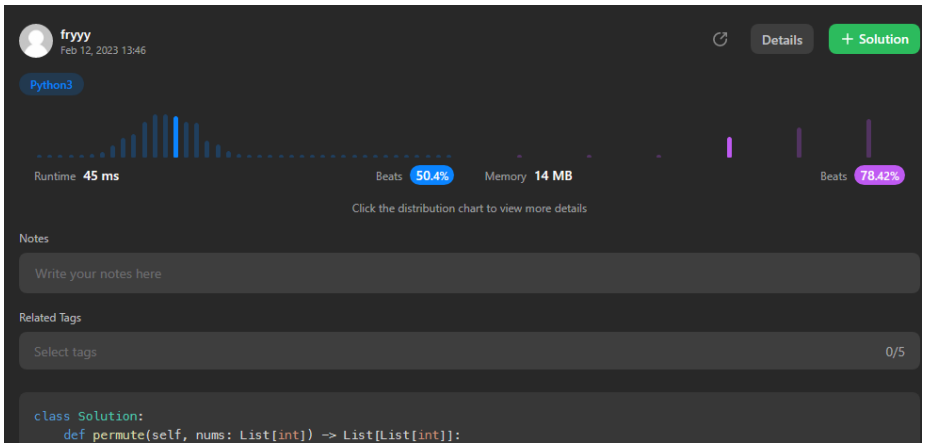
This gives us:
n
n*(n-1)
n*(n-1)*(n-2)
O(n!)

However, when looking at the code with the recursive call, we can see that it is in fact $O(n^2)$, thus giving us a final time complexity of $O(n^2(n!))$.

5 LeetCode Submission

See :

https://leetcode.com/problems/permutations/submissions/896211568/



6 Code

```python
class Solution:
    def permute(self, nums: List[int]) -> List[List[int]]:
        result = [ ]

        #base case
        if len(nums) == 1:
            return [nums.copy()]
        ##divide
        for i in range(len(nums)):
            removedVal = nums.pop(0)
            permutations = self.permute(nums)

            #conquer / combining elements
            for permutation in permutations:
                permutation.append(removedVal)
            result.extend(permutations)
            nums.append(removedVal)

        ## Reordering where permuations with an even number come before those with an odd number
        for x in range(len(result)):
            if result[x][-1] % 2 == 0:
                temp =  result.pop(x)
                result.insert(0, temp)

        return result
```