315. Count of Smaller Numbers After Self
We will use binary search and iterate from the last to the head of the list nums. TC is O(n**n)

```
from bisect import *
class Solution:
    def countSmaller(self, nums: List[int]) -> List[int]:
        cur = []
        res = []
        for i in reversed(nums):
            idx = bisect_left(cur, i)
            res.append(idx)
            cur.insert(idx, i)
        return reversed(res)
```

17. Letter Combinations of a Phone Number
We will use dfs to combine all possible combination and once its length is equal to our digits length we will append it to our res. TC is O(4**n)

```
class Solution:
    def letterCombinations(self, digits: str) -> List[str]:
        res = []
        mapping = {'2': 'abc', '3': 'def', '4': 'ghi', '5': 'jkl',
          '6': 'mno', '7': 'pqrs', '8': 'tuv', '9': 'wxyz'}
        if not digits:
            return []
        def dfs(cur_str, cur_idx, rest_len):
            if rest_len == 0:
                res.append(cur_str)
            elif rest_len > 0:
                for i in mapping[digits[cur_idx]]:
                    dfs(cur_str + i, cur_idx + 1, rest_len - 1)
        dfs('', 0, len(digits))
        return res
```

17. Letter Combinations of a Phone Number
We could use bfs to append current digits’ letters to the previous substrings. TC is O(4 ** n)

```
class Solution:
    def letterCombinations(self, digits: str) -> List[str]:
        res = ['']
        mapping = {'2': 'abc', '3': 'def', '4': 'ghi', '5': 'jkl',
          '6': 'mno', '7': 'pqrs', '8': 'tuv', '9': 'wxyz'}
        if not digits:
            return []
        for i in digits:
            next_ite = []
            for e in mapping[i]:
                for cur in res:
                    next_ite.append(cur + e)
            res = next_ite
        return res
```

39. Combination Sum

We will use dfs to add all combination, to prevent duplication, we will sort our candidates and in every iteration, once current number is equal to the previous one, we will skip it. Once current sum is larger than our target, we will break directly. TC is O(target * target)

```
class Solution:
    def combinationSum(self, candidates: List[int], target: int) -> List[List[int]]:
        res = []
        def dfs(cur, cur_idx, cur_sum, length):
            if cur_sum == target:
                res.append(cur)
            if cur_sum < target:
                for i in range(cur_idx, length):
                    if i > cur_idx and candidates[i] == candidates[i - 1]:
                        continue
                    if cur_sum + candidates[i] > target:
                        break
                    else:
                        dfs(cur + [candidates[i]], i, cur_sum + candidates[i], length)
        candidates.sort()
        dfs([], 0, 0, len(candidates))
        return res
```

40. Combination Sum II

This question is similar to the previous one, the only difference is that we will start from i + 1 rather than i every time when we dfs.

```
class Solution:
    def combinationSum2(self, candidates: List[int], target: int) -> List[List[int]]:
        res = []
        def dfs(cur, cur_idx, cur_sum, length):
            if cur_sum == target:
                res.append(cur)
            if cur_sum < target:
                for i in range(cur_idx, length):
                    if i > cur_idx and candidates[i] == candidates[i - 1]:
                        continue
                    if cur_sum + candidates[i] > target:
                        break
                    else:
                        dfs(cur + [candidates[i]], i + 1, cur_sum + candidates[i], length)
        candidates.sort()
        dfs([], 0, 0, len(candidates))
        return res
```