41. First Missing Positive
Technically, we will take advantage of nums' index and store each number in the index of num - 1. There is the key for this, we should keep swapping until current element is out of length of array or the same as the element we are gonna swap. In the second iteration, we will find the first element that's not equal to the element's index + 1. Or we will return len(nums) + 1. TC is O(n), SC is O(1).

```
class Solution:
    def firstMissingPositive(self, nums: List[int]) -> int:
        length = len(nums)

        for idx in range(length):
            while 0 < nums[idx] <= length and nums[idx] != nums[nums[idx] - 1]:
                dest_index = nums[idx] - 1
                nums[idx], nums[dest_index] = nums[dest_index], nums[idx]

        for idx, num in enumerate(nums):
            if idx + 1 != num:
                return idx + 1
        return len(nums) + 1
```

39. Combination Sum
Because each element is unique. We will iterate through the array. If current array's sum is equal to target, we will push it to our result array, if less than target, we will call our dfs function again, but will start from current index to prevent duplication. The TC is unexpected.

```
class Solution:
    def combinationSum(self, candidates: List[int], target: int) -> List[List[int]]:

        def dfs(result, cur, start_index):
            cur_sum = sum(cur)
            for i in range(start_index, len(candidates)):
                if candidates[i] + cur_sum == target:
                    result.append(cur + [candidates[i]])
                elif candidates[i] + cur_sum < target:
                    dfs(result, cur + [candidates[i]], i)

        result = []
        dfs(result, [], 0)

        return result
```

25. Reverse Nodes in k-Group

There are two ways. First, we will check after dummy node, whether there is k nodes. If there is, we will reverse k nodes after that. Then we will move dummy afterwords for k nodes and repeat the previous steps. TC is O(n)

```
class Solution:
    def reverseKGroup(self, head: ListNode, k: int) -> ListNode:
        dummy = ListNode(0)
        dummy_mem = dummy
        dummy.next = head

        def qualifiedForReverse(dummy, k):
            count = 0
            if not dummy:
                return False
            dummy = dummy.next
            while count < k and dummy:
                dummy = dummy.next
                count += 1
            return count == k

        while qualifiedForReverse(dummy, k):
            cur = dummy.next
            for _ in range(k - 1):
                original_last = cur.next.next
                cur.next.next = dummy.next
                dummy.next = cur.next
                cur.next = original_last
            for _ in range(k):
                dummy = dummy.next
        return dummy_mem.next
```

716. Max Stack
For popMax, we will reverse the memo and delete the max, and reverse it again, so that we could delete the last max number. TC O(n)

```
class MaxStack:

    def __init__(self):
        """
        initialize your data structure here.
        """
        self.memo = []
    def push(self, x: int) -> None:
        self.memo.append(x)

    def pop(self) -> int:
        return self.memo.pop()

    def top(self) -> int:
        return self.memo[-1]

    def peekMax(self) -> int:
        return max(self.memo)

    def popMax(self) -> int:
        temp = max(self.memo)
        self.memo = self.memo[::-1]
        self.memo.remove(temp)
        self.memo = self.memo[::-1]
        return temp
```

6. Zigzag Conversion
We will use string list to store strings on each line. We will accumulate index until index == numsRow and then deduct it to 0, repeat this process until the end of s. Then we will join the string list and return it. TC is O(n)

```
class Solution:
    def convert(self, s: str, numRows: int) -> str:
        memo = [''] * numRows
        ind, mark = 0, True
        if numRows == 1:
            return s

        for e in s:
            memo[ind] += e
            if ind == numRows - 1:
                mark = False
            elif ind == 0 and not mark:
                mark = True
            if mark:
                ind += 1
            else:
                ind -= 1
        return ''.join(memo)
```