41. First Missing Positive
We will put number in its index = number - 1. We will keep swapping until nums[idx] is out of (0, length] or nums[nums[idx] - 1] != nums[idx]. Then in the second iteration, we will return number that's not equal to idx + 1. Or we will return length + 1. TC is O(n)

```
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]:
                temp = nums[idx]
                nums[idx], nums[temp - 1] = nums[temp - 1], nums[idx]

        for i in range(length):
            if i + 1 != nums[i]:
                return i + 1
        return length + 1
```

78. Subsets
We will iterate through nums, and first we will memorize the original one as next_ite. Then we will add current number to each array of the result. Every time, we will add next_ite to our result and reset next_ite to []. We repeat this process until the end of the array.

```
class Solution:
    def subsets(self, nums: List[int]) -> List[List[int]]:
        nums_set = set(nums)
        result = [[]]

        for num in nums_set:
            result += [i + [num] for i in result]
        return result
```

39. Combination Sum
We will use dfs to find proper arrays. When current array's sum is equal to target, we will append this array to result. If current array's sum is less than our target, we will continue dfs from current index so that we could prevent duplication. TC is O(n*n).

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

      self.dfs([], candidates, 0, target, result)
      return result
```

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

32. Longest Valid Parentheses

We will iterate through s, once '(', we will push its index into our stack, when it's ')' and left is larger than 0, we will pop our stack and reduce left by 1. Or we just push its index to stack. In the end, we will go through our stack again, which contains string's breakpoint and get the maximum length. TC is O(n).

```
class Solution:
    def longestValidParentheses(self, s: str) -> int:
        stack = []
        left = 0

        for i, c in enumerate(s):
            if c == '(':
                stack.append(i)
                left += 1
            else:
                if left > 0:
                    stack.pop()
                    left -= 1
                else:
                    stack.append(i)
        if not stack:
            return len(s)
        prev = None
        cur = 0
        for i in stack:
            if prev == None:
                cur = i
            else:
                cur = max(cur, i - prev - 1)
            prev = i
        cur = max(cur,len(s) - prev - 1)
        return cur
```

202. Happy Number

We will slow fast way to check the cycle. Slow goes one step, fast proceeds by 2 steps. We will repeat this process until slow equals to fast. In the end, we only need to check whether slow == 1.

```
class Solution:
    def isHappy(self, n: int) -> bool:
      slow, fast = n, n

      while True:
        slow = self.getDigitsSquare(slow)
        fast = self.getDigitsSquare(fast)
        fast = self.getDigitsSquare(fast)
        if slow == fast:
          break
      return slow == 1

    def getDigitsSquare(self, n):
      total = 0
      while n > 0:
        total += (n % 10) ** 2
        n = n // 10
      return total
```