763. Partition Labels
We will iterate through all elements and indexes into list, we will use a variable to record all fatherest index we need to arrive. TC is O(n)

```
from collections import defaultdict
class Solution:
    def partitionLabels(self, S: str) -> List[int]:
        memo = defaultdict(list)
        result = []
        farthest = 0
        prev = -1
        for idx, c in enumerate(S):
            Memo[c] = idx

        for idx, c in enumerate(S):
            if idx > farthest:
                result.append(farthest - prev)
                prev = farthest
            farthest = max(farthest, memo[c])

        result.append(len(S) - 1 - prev)
        return result
```

442. Find All Duplicates in an Array
We will use mark to record its occuring time. TC is O(n), SC is O(1)

```
class Solution:
    def findDuplicates(self, nums: List[int]) -> List[int]:
        res = []
        for num in nums:
            if nums[abs(num) - 1] < 0:
                res.append(abs(num))
            else:
                nums[abs(num) - 1] *= -1
        return res
```

735. Asteroid Collision
We will use a stack to remember the number if it's a positive number, if it's a negative and the last element in result is positive, we will compare them and pop or break. TC is O(n)

```
class Solution:
    def asteroidCollision(self, asteroids: List[int]) -> List[int]:
        result = []
        for a in asteroids:
            if not result or a > 0:
                result.append(a)
```

```
            else:
                while result and result[-1] > 0:
                    if result[-1] == -a:
                        result.pop()
                        break
                    elif result[-1] > -a:
                        break
                    else:
                        result.pop()
                else:
                    result.append(a)

        return result
```

509. Fibonacci Number

We will iterate based on the previous number and get the number we want. TC is O(n)

```
class Solution:
    def fib(self, N: int) -> int:
        if N <= 1:
            return N
        i = 1
        n1, n2 = 0, 1
        while i < N:
            i += 1
            ret = n1 + n2
            n1, n2 = n2, ret
        return ret
```

141. Linked List Cycle

We will use one slow node and one fast node to iterate through all nodes until the slow one meets the fast one. TC is O(n)

```
class Solution:
    def hasCycle(self, head: ListNode) -> bool:
        if not head or not head.next:
            return False

        slow, fast = head, head.next
        while fast and fast.next:
            if slow == fast:
                return True
            slow = slow.next
            if not fast or not fast.next:
                return False
            fast = fast.next.next
```

```
return False
```