242. Valid Anagram
We will memorize s's every character's number. Then we will iterate through t and once one character's number is less than 0. We will return 0 directly. TC is O(m + n)\

```
from collections import defaultdict
class Solution:
    def isAnagram(self, s: str, t: str) -> bool:
        memo = defaultdict(int)
        if len(s) != len(t):
            return False
        for i in s:
            memo[i] += 1

        for i in t:
            if memo[i] == 0:
                return False
            memo[i] -= 1
        return True
```

243. Number of island
We will use bfs to traverse all islands. Once we encountered '1'. We will visit all its adjacent '1's, and put these coordinates into visit set. Then in the next time when we visit a '1' not in visit set. We will repeat the above process. TC is O(mn).

```
class Solution:
    def numIslands(self, grid: List[List[str]]) -> int:
        if not grid or not grid[0]:
            return 0
        visited = {}
        directions = [[0, 1], [0, -1], [1, 0], [-1, 0]]
        q = []
        count = 0
        rows, cols = len(grid), len(grid[0])
        for i in range(rows):
            for j in range(cols):
                if grid[i][j] == '1' and (i, j) not in visited:
                    count += 1
                    q.clear()
                    q.append((i, j))
                    visited[(i, j)] = True
                    while len(q) > 0:
                        x, y = q.pop()
                        for d_x, d_y in directions:
                            if 0 <= x + d_x < rows and 0 <= y + d_y < cols and grid[x + d_x][y + d_y] == '1'
and (x + d_x, y + d_y) not in visited:
```

```
                    q.append((x + d_x, y + d_y))
                    visited[(x + d_x, y + d_y)] = True
        return count
```

127. Word Ladder
We will use bfs to traverse all possible next ladder that in wordList and push them for next iteration. We will start from both ends, and always traverse the short end. Once we get our next ladder is in other end set, we will return our result. We will add count by 1 in each layer of ladder traverse.

```
class Solution:
    def ladderLength(self, beginWord: str, endWord: str, wordList: List[str]) -> int:
        wordSet = set(wordList)
        count = 1
        ite_words = 'abcdefghijklmnopqrstuvwxyz'

        if endWord not in wordList:
            return 0

        cur_ite = set([beginWord])
        if beginWord in wordSet:
            wordSet.remove(beginWord)
        wordSet.remove(endWord)
        other_ite = set([endWord])
        next_ite = set()
        visited = set()
        while cur_ite:
            for c in cur_ite:
                for i in range(len(c)):
                    for l in ite_words:
                        new_word = c[:i] + l + c[i + 1:]
                        if new_word in other_ite:
                            return count + 1
                        if new_word in wordSet:
                            next_ite.add(new_word)
                            wordSet.remove(new_word)
            count += 1
            if len(cur_ite) > len(other_ite):
                cur_ite = other_ite
                other_ite = next_ite
            else:
                cur_ite = next_ite
            next_ite = set()
        return 0
```

557. Reverse a words in a string
We will split the original string by blank and reverse each string and join them using blank. TC is O(n)

```
class Solution:
    def reverseWords(self, s: str) -> str:
        return ' '.join(list(map(lambda a: a[::-1], s.split(' '))))
```

136. Single number
We will use hashmap to memorize each character’s present number. Then traverse the hashmap and return the key whose value is 1. TC is O(n)

```
from collections import defaultdict
class Solution:
    def singleNumber(self, nums: List[int]) -> int:
        memo = defaultdict(int)
        for i in nums:
          memo[i] += 1
        for k, v in memo.items():
          if v == 1:
            return k
```