1. We will first traverse from left to right and multiply all previous numbers and append to result, then we will multiply from right to left. TC is O(n)

```
class Solution:
    def productExceptSelf(self, nums: List[int]) -> List[int]:
        result = [1]
        product = 1
        for num in nums[:-1]:
          result.append(num * result[-1])

        for i in range(len(nums) - 2, -1, -1):
          product *= nums[i + 1]
          result[i] = result[i] * product
        return result
```

2. Isomorphic Strings

We will iterate all letters in word, using a memo to record existed letter. We will use two arrays to record each letter's index. If we encounter the same letter, we would set that same index, or we will set that one current index. The same as t.

```
class Solution:
    def isIsomorphic(self, s: str, t: str) -> bool:
        s1 = []
        s2 = []
        memo = {}
        for idx, i in enumerate(s):
          if i in memo:
            s1.append(memo[i])
          else:
            s1.append(idx)
            memo[i] = idx
        memo.clear()
        for idx, i in enumerate(t):
          if i in memo:
            s2.append(memo[i])
          else:
            s2.append(idx)
            memo[i] = idx
        return s1 == s2
```

3. Minimum Index Sum of Two Lists

We will iterate the first list and use memo to record its index. Then we will iterate through the second one and if there is one existing in memo, we will compare index sum with min length, if it's smaller, we will empty our result and append current string to result, if they are equal, we will append string. TC is O(n)

```
from collections import defaultdict
class Solution:
    def findRestaurant(self, list1: List[str], list2: List[str]) -> List[str]:
        memo = defaultdict(int)
        result, min_length = [], 2001
        for idx, n in enumerate(list1):
          memo[n] = idx
        for idx, n in enumerate(list2):
          if n in memo:
            if min_length > idx + memo[n]:
              result = []
              min_length = idx + memo[n]
              result.append(n)
            elif min_length == idx + memo[n]:
              result.append(n)
        return result
```

4. Merge Two Sorted Lists

We will merge two lists by comparing node.val. In the end, we will connect the rest if one of them could connect. TC is O(n)

```
class Solution:
    def mergeTwoLists(self, l1: ListNode, l2: ListNode) -> ListNode:
        dummy = ListNode(0)
        dummy_mem = dummy
        while l1 and l2:
          if l1.val > l2.val:
            dummy.next = l2
            l2 = l2.next
          elif l1.val <= l2.val:
            dummy.next = l1
            l1 = l1.next
          dummy = dummy.next

        if l1:
          dummy.next = l1

        if l2:
          dummy.next = l2
        return dummy_mem.next
```

5. Number of Islands

We will use bfs to traverse all connected islands. We will use a dictionary to record visited coordinates to prevent duplication. TC is O(n)

```
class Solution:
    def numIslands(self, grid: List[List[str]]) -> int:
```

```
        count = 0
        visited = set()
        cur = set()
        if not grid or not grid[0]:
          return 0
        rows, cols = len(grid), len(grid[0])
        for i in range(rows):
          for j in range(cols):
            if grid[i][j] == '1':
              if (i, j) not in visited:
                count += 1
                visited.add((i, j))
                cur = set([(i, j)])
                while len(cur) > 0:
                  next_cur = set()
                  for x, y in cur:
                    for d_x, d_y in [[1, 0], [-1, 0], [0, 1], [0, -1]]:
                      new_x, new_y = x + d_x, y + d_y
                      if 0 <= new_x < len(grid) and 0 <= new_y < len(grid[0]) and (new_x,
new_y) not in visited and grid[new_x][new_y] == '1':
                        visited.add((new_x, new_y))
                        next_cur.add((new_x, new_y))
                  cur = next_cur
        return count
```