2. Add Two Numbers
We will add one digit by one digit. If there is a carry, we will also add it up. TC is O(max(m, n))

```
class Solution:
    def addTwoNumbers(self, l1: ListNode, l2: ListNode) -> ListNode:
      dummy = ListNode(0)
      dummy_mem = dummy
      carry = 0
      while l1 and l2:
        carry, value = divmod(l1.val + l2.val + carry, 10)
        dummy.next = ListNode(value)
        dummy = dummy.next
        l1 = l1.next
        l2 = l2.next

      while l1:
        carry, value = divmod(l1.val + carry, 10)
        dummy.next = ListNode(value)
        dummy = dummy.next
        l1 = l1.next
      while l2:
        carry, value = divmod(l2.val + carry, 10)
        dummy.next = ListNode(value)
        dummy = dummy.next
        l2 = l2.next
      if carry:
        dummy.next = ListNode(carry)
        dummy = dummy.next
      return dummy_mem.next
```

5. Longest Palindromic Substring
We will iterate through all elements in the string and see whether there is a longer palindromic substring starting from this letter to both ends. TC is O(n*n)

```
class Solution:
    def longestPalindrome(self, s: str) -> str:
      max_length = 0
      max_substring = ''
      for i, _ in enumerate(s):
        sub = self.helper(i, i + 1, s)
        if len(sub) > max_length:
          max_length = len(sub)
          max_substring = sub
        sub = self.helper(i - 1, i + 1, s)
        if len(sub) > max_length:
```

```
            max_length = len(sub)
            max_substring = sub
        return max_substring

    def helper(self, l, r, s):
        while l >= 0 and r < len(s):
            if s[l] == s[r]:
                l -= 1
                r += 1
            else:
                break

        return s[l + 1: r]
```

200. Number of Islands
We will use dfs to go through all nodes and calculate all adjacent islands as 1. TC is O(n)

```
class Solution:
    def numIslands(self, grid: List[List[str]]) -> int:
        count = 0
        if not grid or not grid[0]:
            return 0
        rows, cols = len(grid), len(grid[0])

        def visit(i, j):
            grid[i][j] = '0'
            for d_i, d_j in [[1, 0], [-1, 0], [0, -1], [0, 1]]:
                new_i, new_j = i + d_i, j + d_j
                if 0 <= new_i < rows and 0 <= new_j < cols and grid[new_i][new_j] == '1':
                    visit(new_i, new_j)


        for i in range(rows):
            for j in range(cols):
                if grid[i][j] == '1':
                    count += 1
                    visit(i, j)

        return count
```

15. 3Sum
We will sort our numbers and iterate all numbers and then after selecting one number we will select from right side, it will downgrade to 2-sum question. TC is O(n*n)

```
class Solution:
```

```
    def threeSum(self, nums: List[int]) -> List[List[int]]:
        result = []
        if len(nums) < 3:
          return []
        nums.sort()

        for i in range(len(nums) - 2):
          l, r = i + 1, len(nums) - 1
          if i > 0 and nums[i] == nums[i - 1]:
            continue
          while l < r:
            if nums[l] + nums[r] == -nums[i]:
              result.append([nums[i], nums[l], nums[r]])
              l += 1
              while nums[l] == nums[l - 1] and l < r:
                l += 1
              r -= 1
              while nums[r] == nums[r + 1] and l < r:
                r -= 1
            elif nums[l] + nums[r] > -nums[i]:
              r -= 1
              while nums[r] == nums[r + 1] and l < r:
                r -= 1
            else:
              l += 1
              while nums[l] == nums[l - 1] and l < r:
                l += 1
        return result
```

973. K Closest Points to Origin

We will use a minheap to maintain the K closest points. TC is O(n)

```
from heapq import *
class Solution:
    def kClosest(self, points: List[List[int]], K: int) -> List[List[int]]:
      result = []
      for x, y in points:
        heappush(result, (-x**2 - y**2, x, y))
        if len(result) > K:
          heappop(result)
      return list(map(lambda a: [a[1], a[2]], result))
```