1. Container With Most Water:-

Code:-

```
def maxArea(height):
    left, right = 0, len(height) - 1
    max_water = 0
    while left < right:
        width = right - left
        max_water = max(max_water, min(height[left], height[right]) *
width)
        if height[left] < height[right]:
            left += 1
        else:
            right -= 1
    return max_water
```

Time complexity: O(n)

Space complexity: O(1)

2. Integer to Roman:-

Code:-

```
def intToRoman(num):
```

```
    val = [1000, 900, 500, 400, 100, 90, 50, 40, 10, 9, 5, 4, 1]
    syb = ["M", "CM", "D", "CD", "C", "XC", "L", "XL", "X", "IX",
"V", "IV", "I"]
    roman_num = ''
    for i in range(len(val)):
        while num >= val[i]:
            num -= val[i]
            roman_num += syb[i]
    return roman_num
```

Time Complexity: O(1)

Space Complexity: O(1)

3. Roman to Integer:-

Code:-

```
def romanToInt(s):
    roman = {'I': 1, 'V': 5, 'X': 10, 'L': 50, 'C': 100, 'D': 500, 'M': 1000}
    total = 0
    for i in range(len(s)):
        if i > 0 and roman[s[i]] > roman[s[i - 1]]:
            total += roman[s[i]] - 2 * roman[s[i - 1]]
        else:
```

```
            total += roman[s[i]]
    return total
```

Time complexity: O(n)

Space complexity: O(1)

4. Longest Common Prefix:-

Code:-

```
def longestCommonPrefix(strs):
    if not strs:
        return ""
    shortest = min(strs, key=len)
    for i, ch in enumerate(shortest):
        for other in strs:
            if other[i] != ch:
                return shortest[:i]
    return shortest
```

Time complexity: O(n*m)

Space complexity: O(1)

5. 3Sum:-

Code:-

```
def threeSum(nums):
    nums.sort()
    res = []
    for i in range(len(nums)):
        if i > 0 and nums[i] == nums[i - 1]:
            continue
        left, right = i + 1, len(nums) - 1
        while left < right:
            sum = nums[i] + nums[left] + nums[right]
            if sum < 0:
                left += 1
            elif sum > 0:
                right -= 1
            else:
                res.append([nums[i], nums[left], nums[right]])
                while left < right and nums[left] == nums[left + 1]:
                    left += 1
                while left < right and nums[right] == nums[right - 1]:
                    right -= 1
                left += 1
                right -= 1
    return res
```

Time complexity: O(n^2)

Space complexity: O(n)

6. 3Sum Closest:-

Code:-

```
def threeSumClosest(nums, target):
    nums.sort()
    closest_sum = float('inf')
    for i in range(len(nums) - 2):
        left, right = i + 1, len(nums) - 1
        while left < right:
            curr_sum = nums[i] + nums[left] + nums[right]
            if abs(curr_sum - target) < abs(closest_sum - target):
                closest_sum = curr_sum
            if curr_sum < target:
                left += 1
            elif curr_sum > target:
                right -= 1
            else:
                return curr_sum
    return closest_sum
```

Time complexity: O(n^2)

Space complexity:: O(1)

7. Letter Combinations of a Phone Number:-

Code:-

```
def letterCombinations(digits):
    if not digits:
        return []
    phone = {
        '2': 'abc', '3': 'def', '4': 'ghi', '5': 'jkl',
        '6': 'mno', '7': 'pqrs', '8': 'tuv', '9': 'wxyz'
    }
    result = ['']
    for digit in digits:
        result = [prefix + letter for prefix in result for letter in
phone[digit]]
    return result
```

Time Complexity: O(3^n * 4^m)
Space Complexity: O(3^n * 4^m)

8. 4Sum:-

Code:-

```
def fourSum(nums, target):
    nums.sort()
    res, quad = [], []
    def kSum(k, start, target):
```

```
    if k != 2:
        for i in range(start, len(nums) - k + 1):
            if i > start and nums[i] == nums[i - 1]:
                continue
            quad.append(nums[i])
            kSum(k - 1, i + 1, target - nums[i])
            quad.pop()
        return
    left, right = start, len(nums) - 1
    while left < right:
        curr_sum = nums[left] + nums[right]
        if curr_sum < target:
            left += 1
        elif curr_sum > target:
            right -= 1
        else:
            res.append(quad + [nums[left], nums[right]])
            left += 1
            right -= 1
            while left < right and nums[left] == nums[left - 1]:
                left += 1
            while left < right and nums[right] == nums[right + 1]:
                right -= 1
kSum(4, 0, target)
return res
```

Time Complexity: O(n^3)

Space Complexity: O(n)

9. Remove Nth Node From End of List:-

Code:-

```
class ListNode:
    def __init__(self, val=0, next=None):
        self.val = val
        self.next = next

def removeNthFromEnd(head, n):
    dummy = ListNode(0, head)
    first = second = dummy
    for _ in range(n + 1):
        first = first.next
    while first:
        first, second = first.next, second.next
    second.next = second.next.next
    return dummy.next
```

Time Complexity: O(n)

Space Complexity: O(1)

10. Valid Parentheses:-

Code:-

```
def isValid(s):
    stack = []
    mapping = {")": "(", "}": "{", "]": "["}
    for char in s:
        if char in mapping:
            top_element = stack.pop() if stack else '#'
            if mapping[char] != top_element:
                return False
        else:
            stack.append(char)
    return not stack
```

Time Complexity: O(n)

Space Complexity: O(n)