1. Longest Substring Without Repeating Characters

**def lengthOfLongestSubstring(self, s: str) -> int:**

**char\_set = set()**

**left = 0**

**max\_length = 0**

**for right in range(len(s)):**

**while s[right] in char\_set:**

**char\_set.remove(s[left])**

**left += 1**

**char\_set.add(s[right])**

**max\_length = max(max\_length, right - left + 1)**

**return max\_length**

1. **Zig-Zag Pattern**

    def convert(self, s: str, numRows: int) -> str:

        if numRows == 1 or numRows >= len(s):

            return s

        rows = [''] \* numRows

        current\_row = 0

        going\_down = False

        for char in s:

            rows[current\_row] += char

            if current\_row == 0 or current\_row == numRows - 1:

                going\_down = not going\_down

            current\_row += 1 if going\_down else -1

        return ''.join(rows)

1. **3Sum Closest**

    def threeSumClosest(self, nums: List[int], target: int) -> int:

        nums.sort()

        n = len(nums)

        closest\_sum = nums[0] + nums[1] + nums[2]

        for i in range(n - 2):

            left, right = i + 1, n - 1

            while left < right:

                current\_sum = nums[i] + nums[left] + nums[right]

                if abs(current\_sum - target) < abs(closest\_sum - target):

                    closest\_sum = current\_sum

                if current\_sum < target:

                    left += 1

                elif current\_sum > target:

                    right -= 1

                else:

                    return current\_sum

        return closest\_sum

1. Generate-Paranthesis

        result= []

        def MakeParanthesis(str1, openCount, closeCount):

            if openCount == closeCount == n:

                result.append(str1)

                return

            if openCount < n :

                MakeParanthesis(str1 + '(', openCount+1, closeCount)

            if closeCount < openCount:

                MakeParanthesis(str1 + ')', openCount, closeCount+1)

        MakeParanthesis("", 0, 0)

        return result

1. **Multiply strings**

    def multiply(self, num1: str, num2: str) -> str:

        return str(int(num1)\*int(num2))

1. **Group Anagrams**

    def groupAnagrams(self, strs: List[str]) -> List[List[str]]:

        anagrams = defaultdict(list)

        for s in strs:

            sorted\_str = ''.join(sorted(s))

            anagrams[sorted\_str].append(s)

        return list(anagrams.values())

1. CanJump

        max\_reachable = 0

        for i, jump in enumerate(nums):

            if i > max\_reachable:

                return False

            max\_reachable = max(max\_reachable, i + jump)

        return max\_reachable >= len(nums) - 1

8.UniquePaths

def uniquePaths(self, m: int, n: int) -> int:

        dp = [[0] \* n for \_ in range(m)]

        for i in range(m):

            dp[i][0] = 1

        for j in range(n):

            dp[0][j] = 1

        for i in range(1, m):

            for j in range(1, n):

                dp[i][j] = dp[i-1][j] + dp[i][j-1]

        return dp[m-1][n-1]

9. Text Justification

 res, curr, num\_of\_letters = [], [], 0

        for word in words:

            if num\_of\_letters + len(word) + len(curr) > maxWidth:

                for i in range(maxWidth - num\_of\_letters):

                    curr[i % (len(curr) - 1 or 1)] += ' '

                res.append(''.join(curr))

                curr, num\_of\_letters = [], 0

            curr += [word]

            num\_of\_letters += len(word)

        res.append(' '.join(curr).ljust(maxWidth))

        return res