3. Longest Substring Without Repeating Characters

Problem:
Given a string, find the length of the longest substring without repeating characters.

Example:
Input: `"abcabcbb"`
Output: `3` (substring is `"abc"`)

In [1]:
```python
def lengthOfLongestSubstring(s: str) -> int:
    seen = set()
    left = 0
    max_len = 0

    for right in range(len(s)):
        # Shrink the window until s[right] is unique
        while s[right] in seen:
            seen.remove(s[left])
            left += 1

        seen.add(s[right])
        max_len = max(max_len, right - left + 1)

    return max_len

```

8. String to Integer (atoi)

Problem:
Convert a string to a 32-bit signed integer (like the C/C++ `atoi` function).
Discard leading whitespaces and handle optional `+` / `-` signs.

Example:
Input: `"   -42"`
Output: `-42`

In [2]:

```python
def myAtoi(s: str) -> int:
    i = 0
    n = len(s)
    result = 0
    sign = 1

    # Skip whitespaces
    while i < n and s[i] == ' ':
        i += 1

    # Check for sign
    if i < n and (s[i] == '-' or s[i] == '+'):
        sign = -1 if s[i] == '-' else 1
        i += 1

    # Convert characters to numbers
    while i < n and s[i].isdigit():
        result = result * 10 + int(s[i])
        i += 1

    result *= sign

    # Clamp to 32-bit range
    return max(min(result, 2**31 - 1), -2**31)

```

13. Roman to Integer

Problem:
Convert a Roman numeral to an integer.

Example:
Input: `"III"`
Output: `3`
Input: `"IV"`
Output: `4`

In [3]:

```python
def romanToInt(s: str) -> int:
    roman_map = {'I': 1, 'V': 5, 'X': 10, 'L': 50,
                 'C': 100, 'D': 500, 'M': 1000}
    total = 0

    for i in range(len(s)):
        # If current value is less than next, subtract it
        if i + 1 < len(s) and roman_map[s[i]] < roman_map[s[i+1
            total -= roman_map[s[i]]
        else:
            total += roman_map[s[i]]

    return total

```

15. 3Sum

Problem:
Given an array, return all unique triplets that sum to zero.

Example:
Input: `[-1, 0, 1, 2, -1, -4]`
Output: `[[-1, -1, 2], [-1, 0, 1]]`

In [4]:

```python
def threeSum(nums):
    nums.sort()
    res = []

    for i in range(len(nums)):
        if i > 0 and nums[i] == nums[i - 1]:
            continue  # skip duplicates

        left, right = i + 1, len(nums) - 1

        while left < right:
            total = nums[i] + nums[left] + nums[right]

            if total < 0:
                left += 1
            elif total > 0:
                right -= 1
            else:
                res.append([nums[i], nums[left], nums[right]])
                # skip duplicates
                while left < right and nums[left] == nums[left
                    left += 1
                while left < right and nums[right] == nums[righ
                    right -= 1
                left += 1
                right -= 1

    return res

```

26. Remove Duplicates from Sorted Array

Problem:
Given a sorted array, remove duplicates in-place and return the new length.

Example:
Input: `[1, 1, 2]`
Output: `2` , with array modified as `[1, 2, _]`

In [5]:
```python
def removeDuplicates(nums):
    if not nums:
        return 0

    index = 1  # place to insert the next unique number

    for i in range(1, len(nums)):
        if nums[i] != nums[i - 1]:
            nums[index] = nums[i]
            index += 1

    return index

```

31. Next Permutation

Problem:
Rearrange numbers to get the next lexicographically greater permutation. If no such permutation exists, rearrange to the lowest possible (sorted in ascending order).

Example:
Input: `[1,2,3]`
Output: `[1,3,2]`

In [6]:
```python
def nextPermutation(nums):
    # Step 1: Find the first decreasing element from the right
    i = len(nums) - 2
    while i >= 0 and nums[i] >= nums[i+1]:
        i -= 1

    if i >= 0:
        # Step 2: Find number just larger than nums[i]
        j = len(nums) - 1
        while nums[j] <= nums[i]:
            j -= 1
        # Step 3: Swap them
        nums[i], nums[j] = nums[j], nums[i]

    # Step 4: Reverse the remaining array
    nums[i+1:] = reversed(nums[i+1:])

```

43. Multiply Strings

Problem:
Multiply two large non-negative numbers represented as strings.

Example:
Input: `"123" * "456"`
Output: `"56088"`

In [7]:
```python

def multiply(num1: str, num2: str) -> str:
    if num1 == "0" or num2 == "0":
        return "0"

    res = [0] * (len(num1) + len(num2))

    # Reverse strings to make indexing easier
    num1, num2 = num1[::-1], num2[::-1]

    # Multiply each digit
    for i in range(len(num1)):
        for j in range(len(num2)):
            res[i + j] += int(num1[i]) * int(num2[j])
            # Handle carry
            res[i + j + 1] += res[i + j] // 10
            res[i + j] %= 10

    # Remove leading zeros
    while len(res) > 1 and res[-1] == 0:
        res.pop()

    return ''.join(map(str, res[::-1]))

```

49. Group Anagrams

Problem:
Group all anagrams together from a list of strings.

Example:
Input: `["eat","tea","tan","ate","nat","bat"]`
Output: `[["eat","tea","ate"],["tan","nat"],["bat"]]`

In [8]:
```python
from collections import defaultdict

def groupAnagrams(strs):
    anagram_map = defaultdict(list)

    for word in strs:
        # Sort characters in the word to use as a key
        key = ''.join(sorted(word))
        anagram_map[key].append(word)

    return list(anagram_map.values())
```

67. Add Binary

Problem:
Add two binary strings and return their sum as a binary string.

Example:
Input: `"11" + "1"`
Output: `"100"`

In [9]:
```python
def addBinary(a: str, b: str) -> str:
    result = []
    carry = 0

    i, j = len(a) - 1, len(b) - 1

    while i >= 0 or j >= 0 or carry:
        total = carry

        if i >= 0:
            total += int(a[i])
            i -= 1
        if j >= 0:
            total += int(b[j])
            j -= 1

        result.append(str(total % 2))
        carry = total // 2

    return ''.join(reversed(result))
```

88. Merge Sorted Array

Problem:
Merge two sorted arrays `nums1` and `nums2` into `nums1` in-place, assuming it has enough space at the end.

Example:

Input:

`nums1 = [1,2,3,0,0,0], m = 3`

`nums2 = [2,5,6], n = 3`

In [10]:

```python

def merge(nums1, m, nums2, n):
    # Start filling from the back
    i, j, k = m - 1, n - 1, m + n - 1

    while i >= 0 and j >= 0:
        if nums1[i] > nums2[j]:
            nums1[k] = nums1[i]
            i -= 1
        else:
            nums1[k] = nums2[j]
            j -= 1
        k -= 1

    # If nums2 still has elements
    while j >= 0:
        nums1[k] = nums2[j]
        j -= 1
        k -= 1


```

125. Valid Palindrome

Problem:

Check if a string is a palindrome, considering only alphanumeric characters and ignoring cases.

Example:

Input: `"A man, a plan, a canal: Panama"`

Output: `True`

In [11]:

```python

def isPalindrome(s: str) -> bool:
    filtered = [c.lower() for c in s if c.isalnum()]
    return filtered == filtered[::-1]
```

157. Read N Characters Given Read4

Problem:

You are given an API `read4(char *buf)` that reads 4 characters at a time. Implement `read(buf, n)` to read `n` characters.

Assume `read4` is already provided.

Python Code (with mock):

In [12]:
```python
def read4(buf4):
    # Mock implementation
    return 0

def read(buf, n):
    i = 0
    while i < n:
        buf4 = [""] * 4
        count = read4(buf4)
        if count == 0:
            break
        for j in range(min(count, n - i)):
            buf[i] = buf4[j]
            i += 1
    return i
```

161. One Edit Distance

Problem:
Given two strings, return `True` if they are one edit apart (insert, delete, or replace one character).

Example:
Input: `("ab", "acb")`
Output: `True`

In [13]:
```python

def isOneEditDistance(s: str, t: str) -> bool:
    if len(s) > len(t):
        return isOneEditDistance(t, s)

    if len(t) - len(s) > 1:
        return False

    for i in range(len(s)):
        if s[i] != t[i]:
            if len(s) == len(t):
                return s[i+1:] == t[i+1:]  # replace
            else:
                return s[i:] == t[i+1:]    # insert

    return len(s) + 1 == len(t)
```

238. Product of Array Except Self

Problem:
Return an array such that each element is the product of all other elements except itself.

Example:
Input: `[1,2,3,4]`
Output: `[24,12,8,6]`

In [14]:
```python

def productExceptSelf(nums):
    length = len(nums)
    output = [1] * length

    # Left pass
    left_product = 1
    for i in range(length):
        output[i] = left_product
        left_product *= nums[i]

    # Right pass
    right_product = 1
    for i in range(length - 1, -1, -1):
        output[i] *= right_product
        right_product *= nums[i]

    return output
```

283. Move Zeroes

Problem:
Move all `0` s to the end of the array while maintaining the order of non-zero elements. Modify the array in-place.

Example:
Input: `[0,1,0,3,12]`
Output: `[1,3,12,0,0]`

In [15]:

```python

def moveZeroes(nums):
    last_non_zero = 0  # Index to place the next non-zero

    # First pass: move non-zeroes forward
    for i in range(len(nums)):
        if nums[i] != 0:
            nums[last_non_zero] = nums[i]
            last_non_zero += 1

    # Second pass: fill the rest with zeroes
    for i in range(last_non_zero, len(nums)):
        nums[i] = 0
```

340. Longest Substring with At Most K Distinct Characters

Problem:
Given a string, return the length of the longest substring that contains at most **k** distinct characters.

Example:
Input: `"eceba"`, `k = 2`
Output: `3` (substring is `"ece"`)

In [16]:

```python

def lengthOfLongestSubstringKDistinct(s: str, k: int) -> int:
    if k == 0:
        return 0

    from collections import defaultdict

    left = 0
    max_len = 0
    char_count = defaultdict(int)

    for right in range(len(s)):
        char_count[s[right]] += 1

        while len(char_count) > k:
            char_count[s[left]] -= 1
            if char_count[s[left]] == 0:
                del char_count[s[left]]
            left += 1

        max_len = max(max_len, right - left + 1)

    return max_len
```

468. Validate IP Address

Problem:
Write a function to check whether an input string is a valid IPv4 or IPv6 address.

Example:
Input: `"172.16.254.1"` → Output: `"IPv4"`
Input: `"2001:0db8:85a3:0000:0000:8a2e:0370:7334"` → Output: `"IPv6"`
Input: `"256.256.256.256"` → Output: `"Neither"`

In [17]:

```python

def validIPAddress(IP: str) -> str:
    def isIPv4(s):
        parts = s.split(".")
        if len(parts) != 4:
            return False
        for part in parts:
            if not part.isdigit() or not 0 <= int(part) <= 255
                return False
        return True

    def isIPv6(s):
        parts = s.split(":")
        if len(parts) != 8:
            return False
        for part in parts:
            if len(part) == 0 or len(part) > 4:
                return False
            for char in part:
                if not (char.isdigit() or char.lower() in "abcd
                    return False
        return True

    if IP.count(".") == 3 and isIPv4(IP):
        return "IPv4"
    elif IP.count(":") == 7 and isIPv6(IP):
        return "IPv6"
    else:
        return "Neither"
```

560. Subarray Sum Equals K

Problem:

Find the number of continuous subarrays whose sum equals `k`.

Example:

Input: `nums = [1,1,1], k = 2`

Output: `2`

In [18]:

```python

def subarraySum(nums, k):
    from collections import defaultdict
    count = 0
    curr_sum = 0
    prefix_sums = defaultdict(int)
    prefix_sums[0] = 1

    for num in nums:
        curr_sum += num
        count += prefix_sums[curr_sum - k]
        prefix_sums[curr_sum] += 1

    return count
```

680. Valid Palindrome II

Problem:
Given a string, return `True` if it can be a palindrome by removing at most one character.

Example:
Input: `"abca"`
Output: `True` (remove `'b'` or `'c'`)

In [19]:

```python

def validPalindrome(s: str) -> bool:
    def is_palindrome(i, j):
        while i < j:
            if s[i] != s[j]:
                return False
            i += 1
            j -= 1
        return True

    left, right = 0, len(s) - 1
    while left < right:
        if s[left] != s[right]:
            return is_palindrome(left+1, right) or is_palindrom
        left += 1
        right -= 1

    return True
```

2. Add Two Numbers

Problem:
You are given two non-empty linked lists representing two non-negative integers. The digits are stored in reverse order, and each node contains a single digit. Add the two numbers and return the result as a linked list.

Example:
Input: (2 -> 4 -> 3) + (5 -> 6 -> 4)
Output: 7 -> 0 -> 8 (because 342 + 465 = 807)

In [20]:

```python

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

def addTwoNumbers(l1, l2):
    dummy = ListNode()
    current = dummy
    carry = 0

    while l1 or l2 or carry:
        val1 = l1.val if l1 else 0
        val2 = l2.val if l2 else 0
        total = val1 + val2 + carry

        carry = total // 10
        current.next = ListNode(total % 10)
        current = current.next

        if l1: l1 = l1.next
        if l2: l2 = l2.next

    return dummy.next
```

21. Merge Two Sorted Lists

Problem:
Merge two sorted linked lists and return it as a new sorted list.

Example:
Input: 1 -> 2 -> 4 and 1 -> 3 -> 4
Output: 1 -> 1 -> 2 -> 3 -> 4 -> 4

In [21]:

```python

def mergeTwoLists(l1, l2):
    dummy = ListNode()
    current = dummy

    while l1 and l2:
        if l1.val < l2.val:
            current.next = l1
            l1 = l1.next
        else:
            current.next = l2
            l2 = l2.next
        current = current.next

    current.next = l1 if l1 else l2
    return dummy.next
```

138. Copy List with Random Pointer

Problem:
A linked list has a `next` pointer and a `random` pointer. Copy the list with both pointers.

In [22]:

```python
class Node:
    def __init__(self, val, next=None, random=None):
        self.val = val
        self.next = next
        self.random = random

def copyRandomList(head):
    if not head:
        return None

    # Step 1: Clone nodes and insert right after original node
    curr = head
    while curr:
        new_node = Node(curr.val, curr.next)
        curr.next = new_node
        curr = new_node.next

    # Step 2: Set random pointers
    curr = head
    while curr:
        if curr.random:
            curr.next.random = curr.random.next
        curr = curr.next.next

    # Step 3: Separate the new list from original
    curr = head
    new_head = head.next
    while curr:
        copy = curr.next
        curr.next = copy.next
        if copy.next:
            copy.next = copy.next.next
        curr = curr.next

    return new_head
```

143. Reorder List

Problem:

Reorder a linked list like this:

`L0 → Ln → L1 → Ln-1 → L2 → Ln-2 → …`

Example:

Input: 1 → 2 → 3 → 4 → 5

Output: 1 → 5 → 2 → 4 → 3

In [23]:

```python

def reorderList(head):
    if not head:
        return

    # Step 1: Find the middle of the list
    slow, fast = head, head
    while fast and fast.next:
        slow = slow.next
        fast = fast.next.next

    # Step 2: Reverse the second half
    prev, curr = None, slow.next
    slow.next = None
    while curr:
        next_temp = curr.next
        curr.next = prev
        prev = curr
        curr = next_temp

    # Step 3: Merge two halves
    first, second = head, prev
    while second:
        tmp1, tmp2 = first.next, second.next
        first.next = second
        second.next = tmp1
        first, second = tmp1, tmp2
```

98. Validate Binary Search Tree

Problem:
Check if a binary tree is a valid binary search tree.

In [24]:

```python

def isValidBST(root):
    def helper(node, lower=float('-inf'), upper=float('inf')):
        if not node:
            return True
        if not (lower < node.val < upper):
            return False
        return (helper(node.left, lower, node.val) and
                helper(node.right, node.val, upper))

    return helper(root)
```

114. Flatten Binary Tree to Linked List

Problem:
Flatten a binary tree to a linked list in-place. The "linked list" should use the right pointers, and all left pointers should be null.

Example:
Input:

```
    1

/
2 5 / \
3 4 6
```

Output:
```
1 -> 2 -> 3 -> 4 -> 5 -> 6
```

In [25]:
```python

def flatten(root):
    if not root:
        return

    stack = [root]
    prev = None

    while stack:
        curr = stack.pop()
        if prev:
            prev.right = curr
            prev.left = None

        if curr.right:
            stack.append(curr.right)
        if curr.left:
            stack.append(curr.left)

        prev = curr
```

133. Clone Graph

Problem:
Given a reference of a node in a connected undirected graph, return a deep copy (clone) of the graph.

In [26]:

```python

class Node:
    def __init__(self, val = 0, neighbors = None):
        self.val = val
        self.neighbors = neighbors if neighbors is not None els

def cloneGraph(node):
    if not node:
        return None

    visited = {}

    def dfs(n):
        if n in visited:
            return visited[n]

        copy = Node(n.val)
        visited[n] = copy
        for neighbor in n.neighbors:
            copy.neighbors.append(dfs(neighbor))
        return copy

    return dfs(node)
```

199. Binary Tree Right Side View

Problem:
Given a binary tree, return the values of the nodes you can see from the right side, from top to bottom.

Example:
Input:
""" 1 /
2 3 \
5 4

Output: `[1, 3, 4]` """

In [27]:

```python

def rightSideView(root):
    if not root:
        return []

    from collections import deque
    queue = deque([root])
    result = []

    while queue:
        level_size = len(queue)
        for i in range(level_size):
            node = queue.popleft()
            if i == level_size - 1:
                result.append(node.val)
            if node.left:
                queue.append(node.left)
            if node.right:
                queue.append(node.right)

    return result
```

200. Number of Islands

Problem:
Given a 2D grid of `'1'` s (land) and `'0'` s (water), count the number of islands. An island is surrounded by water and is formed by connecting adjacent lands horizontally or vertically.

Example:
Input:

[["1","1","0","0","0"], ["1","1","0","0","0"], ["0","0","1","0","0"], ["0","0","0","1","1"]]

Output: `3`

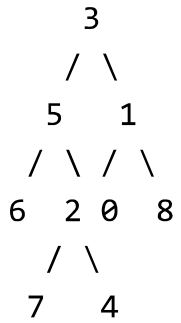
In [28]:

```python
def numIslands(grid):
    if not grid:
        return 0

    rows, cols = len(grid), len(grid[0])
    visited = set()
    islands = 0

    def dfs(r, c):
        if (r < 0 or r >= rows or
            c < 0 or c >= cols or
            grid[r][c] == "0" or
            (r, c) in visited):
            return

        visited.add((r, c))
        dfs(r+1, c)
        dfs(r-1, c)
        dfs(r, c+1)
        dfs(r, c-1)

    for r in range(rows):
        for c in range(cols):
            if grid[r][c] == "1" and (r, c) not in visited:
                dfs(r, c)
                islands += 1

    return islands
```

236. Lowest Common Ancestor of a Binary Tree

Problem:
Given a binary tree, find the lowest common ancestor (LCA) of two given nodes `p` and `q`.

Example:
Input:



`p = 5`, `q = 1` → Output: `3` (LCA)

In [29]:

```python

def lowestCommonAncestor(root, p, q):
    if not root or root == p or root == q:
        return root

    left = lowestCommonAncestor(root.left, p, q)
    right = lowestCommonAncestor(root.right, p, q)

    if left and right:
        return root
    return left if left else right
```

257. Binary Tree Paths

Problem:
Given a binary tree, return all root-to-leaf paths as strings.

Example:
Input:

```
   1

/
2 3
5
```

Output: `["1->2->5", "1->3"]`

In [30]:

```python

def binaryTreePaths(root):
    result = []

    def dfs(node, path):
        if not node:
            return
        if not node.left and not node.right:
            result.append(path + str(node.val))
            return
        dfs(node.left, path + str(node.val) + "->")
        dfs(node.right, path + str(node.val) + "->")

    dfs(root, "")
    return result
```

543. Diameter of Binary Tree

Problem:
Find the diameter (longest path between any two nodes) of a binary tree. The path may or may not pass through the root.

Example:
Input: `[1,2,3,4,5]`
Output: `3` (path: 4 → 2 → 1 → 3)

In [31]:
```python

def diameterOfBinaryTree(root):
    diameter = 0

    def dfs(node):
        nonlocal diameter
        if not node:
            return 0
        left = dfs(node.left)
        right = dfs(node.right)
        diameter = max(diameter, left + right)
        return 1 + max(left, right)

    dfs(root)
    return diameter
```

721. Accounts Merge

Problem:
Merge accounts that have common emails. Each account is a list: `[name, email1, email2, ...]`.

Example:
Input:

[["John", "johnsmith@mail.com (mailto:johnsmith@mail.com)", "john00@mail.com (mailto:john00@mail.com)"], ["John", "johnnybravo@mail.com (mailto:johnnybravo@mail.com)"], ["John", "johnsmith@mail.com (mailto:johnsmith@mail.com)", "john_newyork@mail.com (mailto:john_newyork@mail.com)"], ["Mary", "mary@mail.com (mailto:mary@mail.com)"]]

Output:

[["John", 'john00@mail.com (mailto:john00@mail.com)', 'john_newyork@mail.com (mailto:john_newyork@mail.com)', 'johnsmith@mail.com (mailto:johnsmith@mail.com)'], ["John",

"johnnybravo@mail.com (mailto:johnnybravo@mail.com)"], ["Mary",

In [32]:

```python
from collections import defaultdict

def accountsMerge(accounts):
    email_graph = defaultdict(list)  # Email graph (email -> [c
    email_to_name = {}               # Email -> Name mapping

    # Step 1: Build the graph
    for account in accounts:
        name = account[0]
        first_email = account[1]

        for email in account[1:]:
            email_graph[first_email].append(email)
            email_graph[email].append(first_email)
            email_to_name[email] = name

    # Step 2: DFS to find connected components (emails)
    visited = set()
    result = []

    def dfs(email, collected):
        visited.add(email)
        collected.append(email)
        for neighbor in email_graph[email]:
            if neighbor not in visited:
                dfs(neighbor, collected)

    for email in email_graph:
        if email not in visited:
            collected_emails = []
            dfs(email, collected_emails)
            result.append([email_to_name[email]] + sorted(colle

    return result

```

426. Convert Binary Search Tree to Sorted Doubly Linked List

Problem:

Convert a BST to a sorted circular doubly-linked list in-place.

In [33]:

```python

class Node:
    def __init__(self, val, left=None, right=None):
        self.val = val
        self.left = left
        self.right = right

def treeToDoublyList(root):
    if not root:
        return None

    first = last = None

    def helper(node):
        nonlocal first, last
        if not node:
            return
        helper(node.left)
        if last:
            last.right = node
            node.left = last
        else:
            first = node
        last = node
        helper(node.right)

    helper(root)
    first.left = last
    last.right = first
    return first
```

785. Is Graph Bipartite?

Problem:
Check if a graph is bipartite (can color the graph using 2 colors such that no two adjacent nodes have the same color).

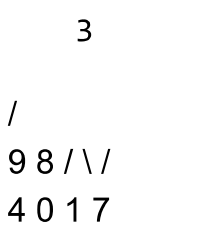
In [34]:

```python

def isBipartite(graph):
    color = {}

    def dfs(node, c):
        if node in color:
            return color[node] == c
        color[node] = c
        for neighbor in graph[node]:
            if not dfs(neighbor, 1 - c):
                return False
        return True

    for node in range(len(graph)):
        if node not in color:
            if not dfs(node, 0):
                return False
    return True
```

314. Binary Tree Vertical Order Traversal

Problem:
Given a binary tree, return its vertical order traversal (nodes grouped by their horizontal distance from the root).

Example:
Input:



Output: `[[4],[9],[3,0,1],[8],[7]]`

In [35]:

```python

from collections import defaultdict, deque

def verticalOrder(root):
    if not root:
        return []

    column_table = defaultdict(list)
    queue = deque([(root, 0)])

    while queue:
        node, col = queue.popleft()
        column_table[col].append(node.val)
        if node.left:
            queue.append((node.left, col - 1))
        if node.right:
            queue.append((node.right, col + 1))

    return [column_table[x] for x in sorted(column_table.keys()
```

17. Letter Combinations of a Phone Number

Problem:
Given a string of digits from 2-9, return all possible letter combinations that the number could represent using the phone keypad mapping.

Example:
Input: `"23"`
Output: `["ad","ae","af","bd","be","bf","cd","ce","cf"]`

In [36]:

```python

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 = []

    def backtrack(index, path):
        if index == len(digits):
            result.append("".join(path))
            return
        for char in phone[digits[index]]:
            backtrack(index + 1, path + [char])

    backtrack(0, [])
    return result
```

46. Permutations

Problem:
Given a list of distinct integers, return all possible permutations.

Example:
Input: `[1,2,3]`
Output:

[[1,2,3], [1,3,2], [2,1,3], [2,3,1], [3,1,2], [3,2,1]]

In [37]:

```python

def permute(nums):
    result = []

    def backtrack(path, used):
        if len(path) == len(nums):
            result.append(path[:])
            return
        for i in range(len(nums)):
            if used[i]:
                continue
            used[i] = True
            path.append(nums[i])
            backtrack(path, used)
            path.pop()
            used[i] = False

    backtrack([], [False] * len(nums))
    return result
```

47. Permutations II

Problem:
Given a list of numbers (may contain duplicates), return all unique permutations.

Example:
Input: `[1,1,2]`
Output:

[[1,1,2], [1,2,1], [2,1,1]]

In [38]:

```python

def permuteUnique(nums):
    nums.sort()
    result = []
    used = [False] * len(nums)

    def backtrack(path):
        if len(path) == len(nums):
            result.append(path[:])
            return
        for i in range(len(nums)):
            if used[i]:
                continue
            if i > 0 and nums[i] == nums[i - 1] and not used[i
                continue  # skip duplicates
            used[i] = True
            path.append(nums[i])
            backtrack(path)
            path.pop()
            used[i] = False

    backtrack([])
    return result
```

78. Subsets

Problem:
Return all possible subsets (the power set) of a given list of integers.

Example:
Input: `[1,2,3]`
Output: `[[],[1],[2],[3],[1,2],[1,3],[2,3],[1,2,3]]`

In [39]:

```python

def subsets(nums):
    result = []

    def backtrack(start, path):
        result.append(path[:])
        for i in range(start, len(nums)):
            path.append(nums[i])
            backtrack(i + 1, path)
            path.pop()

    backtrack(0, [])
    return result
```

33. Search in Rotated Sorted Array

Problem:
Given a rotated sorted array and a target, return the index of the target. If not found, return `-1`.

Example:
Input: `nums = [4,5,6,7,0,1,2], target = 0`
Output: `4`

In [40]:

```python

def search(nums, target):
    left, right = 0, len(nums) - 1

    while left <= right:
        mid = (left + right) // 2
        if nums[mid] == target:
            return mid

        # Left part is sorted
        if nums[left] <= nums[mid]:
            if nums[left] <= target < nums[mid]:
                right = mid - 1
            else:
                left = mid + 1
        # Right part is sorted
        else:
            if nums[mid] < target <= nums[right]:
                left = mid + 1
            else:
                right = mid - 1

    return -1
```

34. Find First and Last Position of Element in Sorted Array

Problem:
Given a sorted array and a target, return the starting and ending index of the target. If not found, return `[-1, -1]`.

Example:
Input: `nums = [5,7,7,8,8,10], target = 8`
Output: `[3, 4]`

In [41]:

```python

def searchRange(nums, target):
    def findBound(isFirst):
        left, right = 0, len(nums) - 1
        bound = -1
        while left <= right:
            mid = (left + right) // 2
            if nums[mid] == target:
                bound = mid
                if isFirst:
                    right = mid - 1
                else:
                    left = mid + 1
            elif nums[mid] < target:
                left = mid + 1
            else:
                right = mid - 1
        return bound

    return [findBound(True), findBound(False)]
```

50. Pow(x, n)

Problem:
Implement `pow(x, n)`, which calculates `x` raised to the power `n`.

Example:
Input: `x = 2.00000, n = 10`
Output: `1024.00000`

In [42]:

```python
def myPow(x: float, n: int) -> float:
    if n == 0:
        return 1

    # Handle negative exponent
    negative = n < 0
    n = abs(n)

    result = 1.0
    current_product = x

    while n > 0:
        if n % 2 == 1:
            result *= current_product  # Multiply when the curr
        current_product *= current_product  # Square the base
        n //= 2  # Shift right

    return 1 / result if negative else result

```

56. Merge Intervals

Problem:
Given a collection of intervals, merge all overlapping intervals.

Example:
Input: `[[1,3],[2,6],[8,10],[15,18]]`
Output: `[[1,6],[8,10],[15,18]]`

In [43]:

```python

def merge(intervals):
    if not intervals:
        return []

    intervals.sort(key=lambda x: x[0])
    merged = [intervals[0]]

    for current in intervals[1:]:
        prev = merged[-1]
        if current[0] <= prev[1]:  # overlapping
            prev[1] = max(prev[1], current[1])
        else:
            merged.append(current)

    return merged
```

162. Find Peak Element

Problem:
Find a peak element in the array. An element is a peak if it's greater than its neighbors. Return the index of any peak.

Example:
Input: `[1,2,3,1]`
Output: `2` (since 3 is a peak)

In [44]:
```python

def findPeakElement(nums):
    left, right = 0, len(nums) - 1

    while left < right:
        mid = (left + right) // 2
        if nums[mid] < nums[mid + 1]:
            left = mid + 1
        else:
            right = mid

    return left
```

278. First Bad Version

Problem:
You are given `n` versions and a function `isBadVersion(version)`. Find the first bad version.

Example:
Input: `n = 5, bad = 4`
Output: `4`

In [45]:

```python

def firstBadVersion(n):
    left, right = 1, n
    while left < right:
        mid = (left + right) // 2
        if isBadVersion(mid):
            right = mid
        else:
            left = mid + 1
    return left


```

349. Intersection of Two Arrays

Problem:
Given two arrays, return their intersection (unique elements).

Example:
Input: `nums1 = [1,2,2,1], nums2 = [2,2]`
Output: `[2]`

In [46]:

```python

def intersection(nums1, nums2):
    return list(set(nums1) & set(nums2))
```

350. Intersection of Two Arrays II

Problem:
Return the intersection of two arrays, including duplicate elements as many times as they appear in both.

Example:
Input: `nums1 = [1,2,2,1], nums2 = [2,2]`
Output: `[2,2]`

In [47]:

```python

from collections import Counter

def intersect(nums1, nums2):
    counts = Counter(nums1)
    result = []

    for num in nums2:
        if counts[num] > 0:
            result.append(num)
            counts[num] -= 1

    return result
```

5. Longest Palindromic Substring

Problem:
Find the longest palindromic substring in a given string.

Example:
Input: `"babad"`
Output: `"bab"` (or `"aba"`)

In [48]:

```python

def longestPalindrome(s):
    if not s:
        return ""

    start, end = 0, 0

    def expandAroundCenter(left, right):
        while left >= 0 and right < len(s) and s[left] == s[rig
            left -= 1
            right += 1
        return left + 1, right - 1

    for i in range(len(s)):
        l1, r1 = expandAroundCenter(i, i)
        l2, r2 = expandAroundCenter(i, i + 1)

        if r1 - l1 > end - start:
            start, end = l1, r1
        if r2 - l2 > end - start:
            start, end = l2, r2

    return s[start:end + 1]
```

91. Decode Ways

Problem:
Given a string containing digits, return the number of ways to decode it, where `'A'` = 1, `'B'` = 2, ..., `'Z'` = 26.

Example:
Input: `"226"`
Output: `3` (`"BZ"` , `"VF"` , `"BBF"`)

In [49]:
```python

def numDecodings(s):
    if not s or s[0] == '0':
        return 0

    dp = [0] * (len(s) + 1)
    dp[0] = dp[1] = 1

    for i in range(2, len(s) + 1):
        if s[i-1] != '0':
            dp[i] += dp[i-1]
        if 10 <= int(s[i-2:i]) <= 26:
            dp[i] += dp[i-2]

    return dp[len(s)]
```

121. Best Time to Buy and Sell Stock

Problem:
Find the maximum profit from one buy-sell transaction in a list of prices.

Example:
Input: `[7,1,5,3,6,4]`
Output: `5` (buy at 1, sell at 6)

In [50]:

```python

def maxProfit(prices):
    min_price = float('inf')
    max_profit = 0

    for price in prices:
        if price < min_price:
            min_price = price
        else:
            max_profit = max(max_profit, price - min_price)

    return max_profit


def maxProfit(prices):
    if not prices:
        return 0

    n = len(prices)
    dp = [[0] * 2 for _ in range(n)]

    # Day 0
    dp[0][0] = 0             # No stock, no profit
    dp[0][1] = -prices[0]    # Bought stock on day 0

    for i in range(1, n):
        # Either keep not holding or sell today
        dp[i][0] = max(dp[i-1][0], dp[i-1][1] + prices[i])
        # Either keep holding or buy today (only once allowed)
        dp[i][1] = max(dp[i-1][1], -prices[i])

    return dp[-1][0]

```

139. Word Break

Problem:
Given a string `s` and a dictionary of words, return `True` if `s` can be segmented into a space-separated sequence of dictionary words.

Example:
Input: `s = "leetcode", wordDict = ["leet","code"]`
Output: `True`

In [51]:

```python

def wordBreak(s, wordDict):
    word_set = set(wordDict)
    dp = [False] * (len(s)+1)
    dp[0] = True

    for i in range(1, len(s)+1):
        for j in range(i):
            if dp[j] and s[j:i] in word_set:
                dp[i] = True
                break

    return dp[len(s)]
```

146. LRU Cache

Problem:
Design a data structure that follows the Least Recently Used (LRU) cache eviction policy.

Operations:

- `get(key)` — Return value if key exists, else -1.
- `put(key, value)` — Insert/update key. If capacity is exceeded, evict the least recently used item.

Example:

cache = LRUCache(2) cache.put(1, 1) cache.put(2, 2) cache.get(1) # returns 1 cache.put(3, 3) # evicts key 2 cache.get(2) # returns -1

In [52]:

```python

from collections import OrderedDict

class LRUCache:

    def __init__(self, capacity: int):
        self.cache = OrderedDict()
        self.capacity = capacity

    def get(self, key: int) -> int:
        if key not in self.cache:
            return -1
        # Move to end to show it was recently used
        self.cache.move_to_end(key)
        return self.cache[key]

    def put(self, key: int, value: int) -> None:
        if key in self.cache:
            self.cache.move_to_end(key)
        self.cache[key] = value
        if len(self.cache) > self.capacity:
            self.cache.popitem(last=False)  # pop least recentl
```

567. Permutation in String

Problem:

Given two strings `s1` and `s2`, return `True` if `s2` contains a permutation of `s1`.

Example:

Input: `s1 = "ab"`, `s2 = "eidbaooo"`

Output: `True`

In [53]:

```python

from collections import Counter

def checkInclusion(s1, s2):
    s1_count = Counter(s1)
    window = Counter()

    for i in range(len(s2)):
        window[s2[i]] += 1
        if i >= len(s1):
            left_char = s2[i - len(s1)]
            window[left_char] -= 1
            if window[left_char] == 0:
                del window[left_char]
        if window == s1_count:
            return True

    return False
```

953. Verifying an Alien Dictionary

Problem:
Given a list of words and a custom alphabet order, return `True` if the words are sorted according to that order.

Example:
Input: `words = ["hello","leetcode"], order = "hlabcdefgijkmnopqrstuvwxyz"`
Output: `True`

In [54]:

```python

def isAlienSorted(words, order):
    order_map = {char: i for i, char in enumerate(order)}

    for i in range(len(words) - 1):
        w1, w2 = words[i], words[i + 1]
        for j in range(min(len(w1), len(w2))):
            if w1[j] != w2[j]:
                if order_map[w1[j]] > order_map[w2[j]]:
                    return False
                break
        else:
            if len(w1) > len(w2):
                return False

    return True
```

986. Interval List Intersections

Problem:
Given two lists of intervals, return their intersections.

Example:
Input:

A = [[0,2],[5,10],[13,23],[24,25]],
B = [[1,5],[8,12],[15,24],[25,26]]

Output: `[[1,2],[5,5],[8,10],[15,23],[24,24],[25,25]]`

In [55]:

```
def intervalIntersection(A, B):
    result = []
    i, j = 0, 0

    while i < len(A) and j < len(B):
        start = max(A[i][0], B[j][0])
        end = min(A[i][1], B[j][1])
        if start <= end:
            result.append([start, end])
        if A[i][1] < B[j][1]:
            i += 1
        else:
            j += 1

    return result
```

1. Add Strings

Problem:
Given two non-negative integers as strings, return their sum as a string. You should not convert the inputs to integers directly.

Example:
Input: `num1 = "11" , num2 = "123"`
Output: `"134"`

In [56]:

```python

def addStrings(num1: str, num2: str) -> str:
    i, j = len(num1) - 1, len(num2) - 1
    carry = 0
    result = []

    while i >= 0 or j >= 0 or carry:
        n1 = int(num1[i]) if i >= 0 else 0
        n2 = int(num2[j]) if j >= 0 else 0
        total = n1 + n2 + carry
        carry = total // 10
        result.append(str(total % 10))
        i -= 1
        j -= 1

    return ''.join(reversed(result))
```

2. Insert into a Sorted Circular Linked List

Problem:
Given a sorted circular linked list and a value to insert, insert the value into the list while maintaining its sorted order.

In [57]:

```python

class Node:
    def __init__(self, val, next=None):
        self.val = val
        self.next = next

def insert(head: 'Node', insertVal: int) -> 'Node':
    new_node = Node(insertVal)
    if not head:
        new_node.next = new_node
        return new_node

    prev, curr = head, head.next
    while True:
        if prev.val <= insertVal <= curr.val:
            break
        if prev.val > curr.val and (insertVal >= prev.val or in
            break
        prev, curr = curr, curr.next
        if prev == head:
            break

    prev.next = new_node
    new_node.next = curr
    return head
```

3. Kth Smallest Element in a Sorted Matrix

Problem:
Given an `n x n` matrix where each row and column is sorted in ascending order, return the `k` th smallest element.

Example:
Input:

matrix = [[1,5,9], [10,11,13], [12,13,15]] k = 8

Output: `13`

In [58]:

```python

import heapq

def kthSmallest(matrix, k):
    n = len(matrix)
    min_heap = []

    # Add the first element of each row to the heap
    for r in range(min(k, n)):
        heapq.heappush(min_heap, (matrix[r][0], r, 0))

    while k > 0:
        val, r, c = heapq.heappop(min_heap)
        if c + 1 < n:
            heapq.heappush(min_heap, (matrix[r][c + 1], r, c +
        k -= 1

    return val
```

5. Longest Substring with At Most Two Distinct Characters

Problem:
Return the length of the longest substring that contains at most 2 distinct characters.

Example:
Input: `"eceba"`
Output: `3` (substring is `"ece"`)

In [59]:

```python

def lengthOfLongestSubstringTwoDistinct(s: str) -> int:
    from collections import defaultdict
    left = 0
    max_len = 0
    char_count = defaultdict(int)

    for right in range(len(s)):
        char_count[s[right]] += 1
        while len(char_count) > 2:
            char_count[s[left]] -= 1
            if char_count[s[left]] == 0:
                del char_count[s[left]]
            left += 1
        max_len = max(max_len, right - left + 1)

    return max_len
```

6. Max Consecutive Ones III

Problem:
Given a binary array `nums` and an integer `k`, return the maximum number of consecutive `1`s in the array if you can flip at most `k` `0`s.

Example:
Input: `nums = [1,1,1,0,0,0,1,1,1,1,0], k = 2`
Output: `6`

In [60]:
```
1
2  def longestOnes(nums, k):
3      left = 0
4      max_len = 0
5
6      for right in range(len(nums)):
7          if nums[right] == 0:
8              k -= 1
9
10         while k < 0:
11             if nums[left] == 0:
12                 k += 1
13             left += 1
14
15         max_len = max(max_len, right - left + 1)
16
17     return max_len
```

7. Minimum Add to Make Parentheses Valid

Problem:
Given a string `s` of `'('` and `')'`, return the minimum number of parentheses needed to make the string valid.

Example:
Input: `"())"`
Output: `1`

In [61]:

```python

def minAddToMakeValid(s: str) -> int:
    balance = 0
    insertions = 0

    for char in s:
        if char == '(':
            balance += 1
        else:
            if balance == 0:
                insertions += 1
            else:
                balance -= 1

    return insertions + balance
```

8. Moving Average from Data Stream

Problem:
Design a class to calculate the moving average of all integers in the sliding window of size `size`.

Example:

m = MovingAverage(3) m.next(1) → 1.0 m.next(10) → (1+10)/2 = 5.5 m.next(3) → (1+10+3)/3 = 4.67 m.next(5) → (10+3+5)/3 = 6.0

In [62]:

```python

from collections import deque

class MovingAverage:

    def __init__(self, size: int):
        self.size = size
        self.window = deque()
        self.total = 0

    def next(self, val: int) -> float:
        self.window.append(val)
        self.total += val
        if len(self.window) > self.size:
            self.total -= self.window.popleft()
        return self.total / len(self.window)
```

9. Range Sum of BST

Problem:
Given a binary search tree and a range `[low, high]`, return the sum of all node values within the range.

Example:
Input:

```
    10

/
5 15 / \
3 7 18
```

Range = [7, 15] → Output: 32

In [63]:

```
def rangeSumBST(root, low, high):
    if not root:
        return 0

    if root.val < low:
        return rangeSumBST(root.right, low, high)
    elif root.val > high:
        return rangeSumBST(root.left, low, high)
    else:
        return (root.val +
                rangeSumBST(root.left, low, high) +
                rangeSumBST(root.right, low, high))
```

227. Basic Calculator II

Problem Explanation: You are given a string `s` representing an arithmetic expression that contains non-negative integers and the operators `+`, `-`, `*`, and `/`. Your task is to evaluate this expression and return the result as an integer. The division should truncate towards zero. The input is guaranteed to be valid.

Example:

Input: `"3+2*2"`
Output: `7`

In [64]:

```python
#### ✅ Python Code (with comments):

def calculate(s: str) -> int:
    stack = []       # Stack to keep numbers and intermediate r
    num = 0          # To form the current number
    sign = '+'       # Keeps track of last seen operator

    for i, ch in enumerate(s):
        if ch.isdigit():
            num = num * 10 + int(ch)  # Build the number digit

        if ch in "+-*/" or i == len(s) - 1:  # If we hit an ope
            if sign == '+':
                stack.append(num)     # Push the number as is
            elif sign == '-':
                stack.append(-num)    # Push the negative numbe
            elif sign == '*':
                stack[-1] *= num      # Multiply the top of the
            elif sign == '/':
                stack[-1] = int(stack[-1] / num)  # Integer div

            sign = ch                 # Update last seen operat
            num = 0                   # Reset number

    return sum(stack)                 # Sum all values in the s
```

9. Palindrome Number

Problem Explanation: Given an integer `x`, return `True` if `x` is a palindrome, and `False` otherwise.

A palindrome number reads the same backward as forward (e.g., 121, 1331).

Example:

Input: `121`
Output: `True`
Input: `-121`
Output: `False` (negative numbers aren't palindromes)

In [65]:
```python
#### ✅ Python Code:

def isPalindrome(x: int) -> bool:
    # Negative numbers cannot be palindromes
    if x < 0:
        return False

    # Convert number to string and compare with its reverse
    return str(x) == str(x)[::-1]
```

71. Simplify Path

Problem Explanation: Given a Unix-style absolute path, simplify it. The path may include `.` (current directory), `..` (parent directory), and multiple slashes.

Example:

Input: `"/a/./b/../../c/"`
Output: `"/c"`

In [66]:
```python
#### ✅ Python Code:

def simplifyPath(path: str) -> str:
    stack = []

    # Split by '/' and iterate over parts
    for part in path.split('/'):
        if part == '..':
            if stack:
                stack.pop()  # Go up one directory
        elif part and part != '.':
            stack.append(part)  # Go into the directory

    return '/' + '/'.join(stack)
```

1091. Shortest Path in Binary Matrix

Problem Explanation: You are given an `n x n` binary matrix where `0` represents open cells and `1` represents blocked ones. You need to find the shortest path from the top-left to the bottom-right corner, moving in 8 directions. Return the length of the shortest path, or `-1` if none exists.

✅ **Python Code:**

In [67]:

```python
from collections import deque

def shortestPathBinaryMatrix(grid):
    n = len(grid)

    # If start or end is blocked
    if grid[0][0] != 0 or grid[n-1][n-1] != 0:
        return -1

    # Directions for 8 possible moves
    directions = [(-1,-1), (-1,0), (-1,1), (0,-1), (0,1), (1,-1
    queue = deque([(0, 0, 1)])  # (row, col, path length)
    visited = set((0, 0))

    while queue:
        x, y, length = queue.popleft()
        if x == n-1 and y == n-1:
            return length

        for dx, dy in directions:
            nx, ny = x + dx, y + dy
            if 0 <= nx < n and 0 <= ny < n and grid[nx][ny] ==
                visited.add((nx, ny))
                queue.append((nx, ny, length + 1))

    return -1
```

339. Nested List Weight Sum

Problem Explanation: You are given a nested list of integers. Each element is either an integer or a list of integers. Return the sum of all integers in the list weighted by their depth (outermost list has depth 1).

Example:

Input: `[1,[4,[6]]]`
Output: `1*1 + 4*2 + 6*3 = 27`

In [68]:
```python
#### ✅ Python Code:

def depthSum(nestedList):
    def dfs(lst, depth):
        total = 0
        for el in lst:
            if isinstance(el, int):
                total += el * depth  # Multiply integer by its
            else:
                total += dfs(el, depth + 1)  # Recurse for inne
        return total

    return dfs(nestedList, 1)
```

23. Merge k Sorted Lists

Problem Explanation: You are given `k` sorted linked lists. Merge them all into one sorted linked list and return it.

✅ **Python Code:**

In [69]:
```python
from heapq import heappush, heappop
from typing import Optional

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

def mergeKLists(lists):
    heap = []
    for i, node in enumerate(lists):
        if node:
            heappush(heap, (node.val, i, node))  # Push (value,

    dummy = ListNode(0)
    curr = dummy

    while heap:
        val, i, node = heappop(heap)
        curr.next = node
        curr = curr.next
        if node.next:
            heappush(heap, (node.next.val, i, node.next))

    return dummy.next
```

921. Minimum Add to Make Parentheses Valid

Problem Explanation: Given a string of parentheses `'('` and `')'`, return the minimum number of parentheses you need to add to make it valid. A string is valid if every opening parenthesis has a matching closing parenthesis.

Example:

Input: `"())"`
Output: `1`

✅ **Python Code:**

In [70]:
```python
def minAddToMakeValid(s: str) -> int:
    open_needed = 0  # Count of '(' we need
    close_needed = 0 # Count of ')' we need to balance

    for ch in s:
        if ch == '(':
            open_needed += 1
        elif ch == ')':
            if open_needed > 0:
                open_needed -= 1  # Match one '(' with ')'
            else:
                close_needed += 1  # Need one '(' before this '

    return open_needed + close_needed
```

65. Valid Number

Problem Explanation: Check if a given string is a valid number. The number can be an integer, a float, or in scientific notation (like `2e10`). Leading/trailing spaces are allowed. Must handle edge cases like `1.`, `.1`, `+3`, `4e-2`.

✅ **Python Code:**

In [71]:
```python

def isNumber(s: str) -> bool:
    s = s.strip()  # Remove leading/trailing spaces
    try:
        float(s)    # Try to convert to float
        return True
    except:
        return False
```

129. Sum Root to Leaf Numbers

Problem Explanation: Given a binary tree where each node contains a digit `0–9`, each root-to-leaf path represents a number. Return the total sum of all those numbers.

Example:

Tree:

1 /
2 3

Paths: 12 and 13 → Sum = 25

✅ **Python Code:**

In [72]:
```python
def sumNumbers(root):
    def dfs(node, current_sum):
        if not node:
            return 0
        current_sum = current_sum * 10 + node.val
        if not node.left and not node.right:  # Leaf node
            return current_sum
        return dfs(node.left, current_sum) + dfs(node.right, cu

    return dfs(root, 0)
```

827. Making A Large Island

Problem Explanation: Given a binary grid (1 = land, 0 = water), you can change at most one 0 to 1. Return the size of the largest island possible.

✅ **Python Code:**

In [73]:

```python
def largestIsland(grid):
    n = len(grid)
    island_id = 2
    area = {}

    def dfs(x, y, id):
        if 0 <= x < n and 0 <= y < n and grid[x][y] == 1:
            grid[x][y] = id
            return 1 + dfs(x+1, y, id) + dfs(x-1, y, id) + dfs(
        return 0

    for i in range(n):
        for j in range(n):
            if grid[i][j] == 1:
                area[island_id] = dfs(i, j, island_id)
                island_id += 1

    max_area = max(area.values(), default=0)

    for i in range(n):
        for j in range(n):
            if grid[i][j] == 0:
                seen = set()
                for dx, dy in [(-1,0),(1,0),(0,-1),(0,1)]:
                    ni, nj = i+dx, j+dy
                    if 0 <= ni < n and 0 <= nj < n and grid[ni]
                        seen.add(grid[ni][nj])
                max_area = max(max_area, 1 + sum(area[k] for k

    return max_area
```

1004. Max Consecutive Ones III

Problem Explanation: You are given a binary array and can flip at most k zeros to ones. Return the length of the longest subarray with at most k zeros flipped.

In [74]:
```python
#### ✅ Python Code:

def longestOnes(nums, k):
    left = 0
    for right in range(len(nums)):
        if nums[right] == 0:
            k -= 1
        if k < 0:
            if nums[left] == 0:
                k += 1
            left += 1
    return right - left + 1
```

346. Moving Average from Data Stream

Problem Explanation: Design a class that calculates the moving average of the last `N` elements from a data stream.

✅ **Python Code:**

In [75]:
```python
from collections import deque

class MovingAverage:
    def __init__(self, size: int):
        self.queue = deque()
        self.size = size
        self.sum = 0

    def next(self, val: int) -> float:
        self.queue.append(val)
        self.sum += val
        if len(self.queue) > self.size:
            self.sum -= self.queue.popleft()
        return self.sum / len(self.queue)
```

415. Add Strings

Problem Explanation: Given two non-negative integers `num1` and `num2` represented as strings, return their sum as a string. You must do it without converting the inputs to integers directly or using big integer libraries.

✅ **Python Code:**

In [76]:

```python
def addStrings(num1: str, num2: str) -> str:
    i, j = len(num1) - 1, len(num2) - 1
    carry = 0
    result = []

    while i >= 0 or j >= 0 or carry:
        n1 = int(num1[i]) if i >= 0 else 0
        n2 = int(num2[j]) if j >= 0 else 0
        total = n1 + n2 + carry
        carry = total // 10
        result.append(str(total % 10))
        i -= 1
        j -= 1

    return ''.join(reversed(result))
```

378. Kth Smallest Element in a Sorted Matrix

Problem Explanation: You are given an `n x n` matrix where each row and column is sorted in ascending order. Find the `k` th smallest element in the matrix.

Example:

Matrix: [[1, 5, 9], [10, 11, 13], [12, 13, 15]] k = 8

Output: 13

In [77]:

```python
#### ✅ Python Code:

import heapq

def kthSmallest(matrix, k):
    n = len(matrix)
    min_heap = []

    # Insert the first element of each row into the heap
    for i in range(min(k, n)):
        heapq.heappush(min_heap, (matrix[i][0], i, 0))  # (valu

    count = 0
    while min_heap:
        val, r, c = heapq.heappop(min_heap)
        count += 1
        if count == k:
            return val
        if c + 1 < n:
            heapq.heappush(min_heap, (matrix[r][c+1], r, c+1))
```

938. Range Sum of BST

Problem Explanation: Given a binary search tree (BST) and two integers `low` and `high`, return the sum of all node values between `low` and `high` inclusive.

✅ **Python Code:**

In [78]:
```python

def rangeSumBST(root, low, high):
    if not root:
        return 0
    if root.val < low:
        return rangeSumBST(root.right, low, high)
    if root.val > high:
        return rangeSumBST(root.left, low, high)
    return root.val + rangeSumBST(root.left, low, high) + range
```

3527. Find the Most Common Response

Problem Explanation: You are given a list of responses (like survey answers). Return the most common one. If there are multiple with the same frequency, return the one that appears first.

✅ **Python Code:**

In [79]:
```python

from collections import defaultdict

def mostCommonResponse(responses):
    count = defaultdict(int)
    max_freq = 0
    result = None

    for response in responses:
        count[response] += 1
        if count[response] > max_freq:
            max_freq = count[response]
            result = response
    return result
```

159. Longest Substring with At Most Two Distinct Characters

Problem Explanation: Given a string, return the length of the longest substring that contains at most two distinct characters.

Example:

Input: `"eceba"` → Output: `3` (`"ece"`)

✅ **Python Code:**

In [80]:
```python
def lengthOfLongestSubstringTwoDistinct(s: str) -> int:
    from collections import defaultdict

    left = 0
    count = defaultdict(int)
    max_len = 0

    for right in range(len(s)):
        count[s[right]] += 1

        while len(count) > 2:
            count[s[left]] -= 1
            if count[s[left]] == 0:
                del count[s[left]]
            left += 1

        max_len = max(max_len, right - left + 1)

    return max_len
```

708. Insert into a Sorted Circular Linked List

Problem Explanation: Given a node from a **circular sorted** linked list, insert a new value into the list while maintaining sorted order.

✅ **Python Code:**

In [81]:

```python
class Node:
    def __init__(self, val, next=None):
        self.val = val
        self.next = next

def insert(head: 'Node', insertVal: int) -> 'Node':
    new_node = Node(insertVal)
    if not head:
        new_node.next = new_node
        return new_node

    prev, curr = head, head.next
    while True:
        # Case 1: Correct place to insert
        if prev.val <= insertVal <= curr.val:
            break
        # Case 2: At the turning point
        if prev.val > curr.val and (insertVal >= prev.val or in
            break
        prev, curr = curr, curr.next
        if prev == head:
            break

    prev.next = new_node
    new_node.next = curr
    return head
```

973. K Closest Points to Origin

Problem Explanation: You are given a list of points on a 2D plane and an integer `k`. Return the `k` points closest to the origin (0,0).

✅ **Python Code:**

In [82]:
```python
import heapq

def kClosest(points, k):
    max_heap = []  # Max-heap to store the closest k points (ne

    for (x, y) in points:
        dist = -(x ** 2 + y ** 2)  # Negate to simulate max-hea
        heapq.heappush(max_heap, (dist, x, y))
        if len(max_heap) > k:
            heapq.heappop(max_heap)  # Remove the farthest poin

    # Extract x, y from heap
    return [(x, y) for (_, x, y) in max_heap]

```

14. Longest Common Prefix

Problem Explanation: Given a list of strings, return the longest common prefix string shared by all the strings. If none exists, return an empty string.

In [83]:
```python
#### ✅ Python Code:

def longestCommonPrefix(strs):
    if not strs:
        return ""

    prefix = strs[0]
    for s in strs[1:]:
        while not s.startswith(prefix):
            prefix = prefix[:-1]
            if not prefix:
                return ""
    return prefix
```

977. Squares of a Sorted Array

Problem Explanation: Given a sorted array of integers (possibly negative), return a new array containing the squares of each number, also sorted in non-decreasing order.

Example:

Input: `[-4, -1, 0, 3, 10]` → Output: `[0, 1, 9, 16, 100]`

✅ **Python Code:**

In [84]:

```python
def sortedSquares(nums):
    n = len(nums)
    result = [0] * n
    left, right = 0, n - 1
    pos = n - 1

    while left <= right:
        if abs(nums[left]) > abs(nums[right]):
            result[pos] = nums[left] ** 2
            left += 1
        else:
            result[pos] = nums[right] ** 2
            right -= 1
        pos -= 1

    return result
```

498. Diagonal Traverse

Problem Explanation: Given a `m x n` matrix, return all elements of the matrix in a diagonal order (zigzag).

Example:

Input:

[[1, 2, 3], [4, 5, 6], [7, 8, 9]]

Output: `[1,2,4,7,5,3,6,8,9]`

✅ **Python Code:**

In [85]:

```python
def findDiagonalOrder(mat):
    if not mat or not mat[0]:
        return []

    m, n = len(mat), len(mat[0])
    result = []
    for d in range(m + n - 1):
        intermediate = []

        r = 0 if d < n else d - n + 1
        c = d if d < n else n - 1

        while r < m and c > -1:
            intermediate.append(mat[r][c])
            r += 1
            c -= 1

        if d % 2 == 0:
            result.extend(intermediate[::-1])
        else:
            result.extend(intermediate)

    return result
```

1868. Product of Two Run-Length Encoded Arrays

Problem Explanation: You're given two run-length encoded arrays (RLE). Compute the product of the two arrays and return the result also in RLE format.

✅ **Python Code:**

In [86]:

```python

def findRLEArray(encoded1, encoded2):
    res = []
    i = j = 0

    while i < len(encoded1) and j < len(encoded2):
        v1, c1 = encoded1[i]
        v2, c2 = encoded2[j]
        val = v1 * v2
        cnt = min(c1, c2)

        if res and res[-1][0] == val:
            res[-1][1] += cnt
        else:
            res.append([val, cnt])

        encoded1[i][1] -= cnt
        encoded2[j][1] -= cnt

        if encoded1[i][1] == 0:
            i += 1
        if encoded2[j][1] == 0:
            j += 1

    return res
```

489. Robot Room Cleaner

Problem Explanation: You're given a robot with API access to clean a room. Implement the `cleanRoom` function using DFS to ensure the entire room is cleaned.

- The robot can move forward, turn left, turn right, and clean the cell.
- The room is modeled as a grid of open/blocked cells.

In [87]:

```python
#### ✅ Python Code:

class Solution:
    def cleanRoom(self, robot):
        visited = set()
        directions = [(-1,0), (0,1), (1,0), (0,-1)]  # Up, Righ

        def go_back():
            robot.turnLeft()
            robot.turnLeft()
            robot.move()
            robot.turnLeft()
            robot.turnLeft()

        def dfs(x, y, d):
            visited.add((x, y))
            robot.clean()

            for i in range(4):
                new_d = (d + i) % 4
                dx, dy = directions[new_d]
                nx, ny = x + dx, y + dy

                if (nx, ny) not in visited and robot.move():
                    dfs(nx, ny, new_d)
                    go_back()

                robot.turnRight()

        dfs(0, 0, 0)
```

824. Goat Latin

Problem Explanation: Convert a sentence to "Goat Latin":

- If a word starts with a vowel, append "ma".
- If it starts with a consonant, move the first letter to the end, then add "ma".
- Add `'a'` repeated `(word index)` times to the end.

✅ **Python Code:**

In [88]:
```python
def toGoatLatin(sentence: str) -> str:
    vowels = set('aeiouAEIOU')
    words = sentence.split()
    result = []

    for i, word in enumerate(words):
        if word[0] in vowels:
            new_word = word + 'ma'
        else:
            new_word = word[1:] + word[0] + 'ma'
        new_word += 'a' * (i + 1)
        result.append(new_word)

    return ' '.join(result)
```

791. Custom Sort String

Problem Explanation: Given a custom order string `order` and a target string `s`, sort `s` such that characters appear in the same order as in `order`.

✅ **Python Code:**

In [89]:
```python
def customSortString(order: str, s: str) -> str:
    count = {ch: 0 for ch in s}
    for ch in s:
        count[ch] += 1

    result = []
    for ch in order:
        if ch in count:
            result.append(ch * count[ch])
            del count[ch]

    for ch in count:
        result.append(ch * count[ch])

    return ''.join(result)
```

1110. Delete Nodes And Return Forest

Problem Explanation: You're given a binary tree and a list of node values to delete. After deleting those nodes, return the remaining forest (a list of tree roots).

In [90]:

```python
#### ✅ Python Code:

def delNodes(root, to_delete):
    to_delete_set = set(to_delete)
    forest = []

    def dfs(node, is_root):
        if not node:
            return None

        root_deleted = node.val in to_delete_set
        if is_root and not root_deleted:
            forest.append(node)

        node.left = dfs(node.left, root_deleted)
        node.right = dfs(node.right, root_deleted)

        return None if root_deleted else node

    dfs(root, True)
    return forest
```

647. Palindromic Substrings

Problem Explanation: Given a string `s`, return the number of **palindromic substrings** in it. A substring is palindromic if it reads the same forwards and backwards.

Example:

Input: `"abc"` → Output: `3` (`"a"`, `"b"`, `"c"`)
Input: `"aaa"` → Output: `6` (`"a"`, `"a"`, `"a"`, `"aa"`, `"aa"`, `"aaa"`)

In [91]:

```python
#### ✅ Python Code:

def countSubstrings(s: str) -> int:
    def expand(l, r):
        count = 0
        while l >= 0 and r < len(s) and s[l] == s[r]:
            count += 1
            l -= 1
            r += 1
        return count

    result = 0
    for i in range(len(s)):
        result += expand(i, i)     # Odd length palindromes
        result += expand(i, i + 1) # Even length palindromes

    return result
```

76. Minimum Window Substring

Problem Explanation: Given two strings `s` and `t`, return the **smallest substring** of `s` that contains all the characters in `t`. If no such window exists, return an empty string.

Example:

Input: `s = "ADOBECODEBANC"`, `t = "ABC"` → Output: `"BANC"`

✅ **Python Code:**

In [92]:

```python

from collections import Counter

def minWindow(s: str, t: str) -> str:
    if not s or not t:
        return ""

    t_count = Counter(t)
    required = len(t_count)
    formed = 0
    l = 0
    window_counts = {}
    res = float('inf'), None, None

    for r, char in enumerate(s):
        window_counts[char] = window_counts.get(char, 0) + 1

        if char in t_count and window_counts[char] == t_count[c
            formed += 1

        while l <= r and formed == required:
            if r - l + 1 < res[0]:
                res = (r - l + 1, l, r)

            window_counts[s[l]] -= 1
            if s[l] in t_count and window_counts[s[l]] < t_coun
                formed -= 1
            l += 1

    return "" if res[0] == float('inf') else s[res[1]:res[2]+1]
```

4. Median of Two Sorted Arrays

Problem Explanation: You're given two sorted arrays `nums1` and `nums2`. Return the median of the merged sorted array in O(log(min(m, n))) time.

✅ **Python Code:**

In [93]:

```python

def findMedianSortedArrays(nums1, nums2):
    A, B = nums1, nums2
    total = len(A) + len(B)
    half = total // 2

    if len(B) < len(A):
        A, B = B, A

    l, r = 0, len(A) - 1
    while True:
        i = (l + r) // 2  # A
        j = half - i - 2  # B

        Aleft = A[i] if i >= 0 else float('-inf')
        Aright = A[i + 1] if (i + 1) < len(A) else float('inf')
        Bleft = B[j] if j >= 0 else float('-inf')
        Bright = B[j + 1] if (j + 1) < len(B) else float('inf')

        if Aleft <= Bright and Bleft <= Aright:
            if total % 2:
                return min(Aright, Bright)
            return (max(Aleft, Bleft) + min(Aright, Bright)) /
        elif Aleft > Bright:
            r = i - 1
        else:
            l = i + 1
```

480. Sliding Window Median

Problem Explanation: You're given a list of numbers and a window size `k`. Return a list of medians for each sliding window of size `k`.

In [94]:

```python
#### ✅ Python Code:

import heapq

def medianSlidingWindow(nums, k):
    import bisect

    window = sorted(nums[:k])
    res = []

    for i in range(k, len(nums)+1):
        mid = k // 2
        if k % 2:
            res.append(float(window[mid]))
        else:
            res.append((window[mid - 1] + window[mid]) / 2.0)

        if i == len(nums):
            break
        window.remove(nums[i - k])
        bisect.insort(window, nums[i])

    return res
```

863. All Nodes Distance K in Binary Tree

Problem Explanation: Given a binary tree, a target node, and an integer `k`, return all nodes that are `k` distance from the target node.

✅ **Python Code:**

In [95]:

```python
from collections import deque, defaultdict

def distanceK(root, target, k):
    graph = defaultdict(list)

    def build_graph(node, parent):
        if node:
            if parent:
                graph[node.val].append(parent.val)
                graph[parent.val].append(node.val)
            build_graph(node.left, node)
            build_graph(node.right, node)

    build_graph(root, None)

    visited = set()
    queue = deque([(target.val, 0)])
    result = []

    while queue:
        node, dist = queue.popleft()
        if dist == k:
            result.append(node)
        elif dist < k:
            for neighbor in graph[node]:
                if neighbor not in visited:
                    visited.add(neighbor)
                    queue.append((neighbor, dist + 1))

    return result
```

270. Closest Binary Search Tree Value

Problem Explanation: You're given a BST and a target value. Return the value in the BST that is closest to the target.

In [96]:

```python
#### ✅ Python Code:

def closestValue(root, target):
    closest = root.val

    while root:
        if abs(root.val - target) < abs(closest - target):
            closest = root.val
        root = root.left if target < root.val else root.right

    return closest
```

219. Contains Duplicate II

Problem Explanation: Given an array of integers and an integer `k`, return `True` if there are two distinct indices `i` and `j` such that `nums[i] == nums[j]` and `abs(i - j) <= k`.

In [97]:

```python
#### ✅ Python Code:

def containsNearbyDuplicate(nums, k):
    seen = {}
    for i, num in enumerate(nums):
        if num in seen and i - seen[num] <= k:
            return True
        seen[num] = i
    return False
```

Type *Markdown* and LaTeX: α^2