11. Container With Most Water You are given an integer array height of length n. There are n vertical lines drawn such that the two endpoints of the ith line are (i, 0) and (i, height[i]). Find two lines that together with the x-axis form a container, such that the container contains the most water. Return the maximum amount of water a container can store. Notice that you may not slant the container.

**Code with output :**

def maxArea(height):

left = 0

right = len(height) - 1

max\_area = 0

while left < right:

# Calculate the area with the current left and right pointers

width = right - left

current\_height = min(height[left], height[right])

current\_area = width \* current\_height

max\_area = max(max\_area, current\_area)

# Move the pointer pointing to the shorter line inward

if height[left] < height[right]:

left += 1

else:

right -= 1

return max\_area

# Example usage

height1 = [1, 8, 6, 2, 5, 4, 8, 3, 7]

height2 = [1, 1]

# Output for the provided examples

print(maxArea(height1))

print(maxArea(height2))

Output: 49

Output: 1

Time complexity:O(n)

12. Integer to Roman

Roman numerals are represented by seven different symbols: I, V, X, L, C, D and M. Symbol Value I 1 V 5 X 10 L 50 C 100 D 500 M 1000 For example, 2 is written as II in Roman numeral, just two one's added together. 12 is written as XII, which is simply X + II. The number 27 is written as XXVII, which is XX + V + II. Roman numerals are usually written largest to smallest from left to right. However, the numeral for four is not IIII. Instead, the number four is written as IV. Because the one is before the five we subtract it making four. The same principle applies to the number nine, which is written as IX. There are six instances where subtraction is used: ● I can be placed before V (5) and X (10) to make 4 and 9. ● X can be placed before L (50) and C (100) to make 40 and 90. ● C can be placed before D (500) and M (1000) to make 400 and 900. Given an integer, convert it to a roman numeral.

**Code with output:**

def intToRoman(num):

# List of tuples representing the Roman numeral symbols and their corresponding values

value\_to\_roman = [

(1000, 'M'), (900, 'CM'), (500, 'D'), (400, 'CD'),

(100, 'C'), (90, 'XC'), (50, 'L'), (40, 'XL'),

(10, 'X'), (9, 'IX'), (5, 'V'), (4, 'IV'), (1, 'I')

]

roman = []

for value, symbol in value\_to\_roman:

if num == 0:

break

count, num = divmod(num, value)

roman.append(symbol \* count)

return ''.join(roman)

# Example usage

print(intToRoman(3)) Output: "III"

print(intToRoman(58)) Output: "LVIII"

print(intToRoman(1994)) Output: "MCMXCIV"

Time complexity:O(1)

13. Roman to Integer

Roman numerals are represented by seven different symbols: I, V, X, L, C, D and M. Symbol Value I 1 V 5 X 10 L 50 C 100 D 500 M 1000 For example, 2 is written as II in Roman numeral, just two ones added together. 12 is written as XII, which is simply X + II. The number 27 is written as XXVII, which is XX + V + II. Roman numerals are usually written largest to smallest from left to right. However, the numeral for four is not IIII. Instead, the number four is written as IV. Because the one is before the five we subtract it making four. The same principle applies to the number nine, which is written as IX. There are six instances where subtraction is used: ● I can be placed before V (5) and X (10) to make 4 and 9. ● X can be placed before L (50) and C (100) to make 40 and 90. ● C can be placed before D (500) and M (1000) to make 400 and 900. Given a roman numeral, convert it to an integer.

**Code with output:**

def romanToInt(s):

roman\_to\_value = {

'I': 1, 'V': 5, 'X': 10, 'L': 50,

'C': 100, 'D': 500, 'M': 1000

}

total = 0

prev\_value = 0

for char in reversed(s):

value = roman\_to\_value[char]

if value < prev\_value:

total -= value

else:

total += value

prev\_value = value

return total

print(romanToInt("III")) Output: 3

print(romanToInt("LVIII")) Output: 58

print(romanToInt("MCMXCIV")) Output: 1994

Time complexity:O(n)

Space complexity:O(1)

14. Longest Common Prefix

Write a function to find the longest common prefix string amongst an array of strings. If there is no common prefix, return an empty string "".

**Code with output:**

def longestCommonPrefix(strs):

if not strs:

return ""

prefix = strs[0]

for s in strs[1:]:

while s[:len(prefix)] != prefix:

# Reduce the prefix by one character at a time

prefix = prefix[:-1]

# If there's no common prefix, return an empty string

if not prefix:

return ""

return prefix

print(longestCommonPrefix(["flower","flow","flight"])) Output: "fl"

print(longestCommonPrefix(["dog","racecar","car"])) Output: ""

**Time Complexity**: O(n⋅m)

**Space Complexity**: O(1)

15. 3Sum

Given an integer array nums, return all the triplets [nums[i], nums[j], nums[k]] such that i != j, i != k, and j != k, and nums[i] + nums[j] + nums[k] == 0. Notice that the solution set must not contain duplicate triplets.

**Code with output:**

def threeSum(nums):

nums.sort()

result = []

n = len(nums)

for i in range(n):

if i > 0 and nums[i] == nums[i-1]:

continue

left, right = i + 1, n - 1

while left < right:

total = nums[i] + nums[left] + nums[right]

if total == 0:

result.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

elif total < 0:

left += 1

else:

right -= 1

return result

print(threeSum([-1, 0, 1, 2, -1, -4])) Output: [[-1, -1, 2], [-1, 0, 1]]

print(threeSum([0, 1, 1])) Output: []

print(threeSum([0, 0, 0])) Output: [[0, 0, 0]]

**Time Complexity**: O(n\*n)

**Space Complexity**: O(1)

16. 3Sum Closest

Given an integer array nums of length n and an integer target, find three integers in nums such that the sum is closest to target. Return the sum of the three integers. You may assume that each input would have exactly one solution.

**Code with output:**

def threeSumClosest(nums, target):

nums.sort()

closest\_sum = float('inf')

n = len(nums)

for i in range(n):

left, right = i + 1, n - 1

while left < right:

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

if current\_sum == target:

return current\_sum

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

closest\_sum = current\_sum

if current\_sum < target:

left += 1

else:

right -= 1

return closest\_sum

print(threeSumClosest([-1, 2, 1, -4], 1)) Output: 2

print(threeSumClosest([0, 0, 0], 1)) Output: 0

**Time Complexity**: O(n^2)

**Space Complexity**: O(1)

17. Letter Combinations of a Phone Number

Given a string containing digits from 2-9 inclusive, return all possible letter combinations that the number could represent. Return the answer in any order. A mapping of digits to letters (just like on the telephone buttons) is given below. Note that 1 does not map to any letters.

**Code with output:**

def letterCombinations(digits):

if not digits:

return []

digit\_to\_letters = {

'2': 'abc', '3': 'def', '4': 'ghi', '5': 'jkl',

'6': 'mno', '7': 'pqrs', '8': 'tuv', '9': 'wxyz'

}

results = []

def backtrack(index, path):

if index == len(digits):

results.append("".join(path))

return

possible\_letters = digit\_to\_letters[digits[index]]

for letter in possible\_letters:

path.append(letter)

backtrack(index + 1, path)

path.pop()

backtrack(0, [])

return results

print(letterCombinations("23")) # Expected Output: ["ad","ae","af","bd","be","bf","cd","ce","cf"]

print(letterCombinations("")) Output: []

print(letterCombinations("2")) Output: ["a","b","c"]

**Time Complexity**: O(4^n)

**Space Complexity**: O(n)

18. 4Sum

Given an array nums of n integers, return an array of all the unique quadruplets [nums[a], nums[b], nums[c], nums[d]] such that: ● 0 <= a, b, c, d < n ● a, b, c, and d are distinct. ● nums[a] + nums[b] + nums[c] + nums[d] == target You may return the answer in any order.

**Code with output:**

def fourSum(nums, target):

nums.sort()

n = len(nums)

results = []

for i in range(n - 3):

if i > 0 and nums[i] == nums[i - 1]:

continue

for j in range(i + 1, n - 2):

if j > i + 1 and nums[j] == nums[j - 1]:

continue

left, right = j + 1, n - 1

while left < right:

total = nums[i] + nums[j] + nums[left] + nums[right]

if total == target:

results.append([nums[i], nums[j], 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

elif total < target:

left += 1

else:

right -= 1

return results

print(fourSum([1, 0, -1, 0, -2, 2], 0)) Output: [[-2, -1, 1, 2], [-2, 0, 0, 2], [-1, 0, 0, 1]]

print(fourSum([2, 2, 2, 2, 2], 8)) Output: [[2, 2, 2, 2]]

**Time Complexity**: O(n^3)

**Space Complexity**: O(1)

19. Remove Nth Node From End of List

Given the head of a linked list, remove the nth node from the end of the list and return its head.

**Code with output:**

class ListNode:

def \_\_init\_\_(self, val=0, next=None):

self.val = val

self.next = next

def removeNthFromEnd(head, n):

dummy = ListNode(0)

dummy.next = head

fast = slow = dummy

for \_ in range(n + 1):

fast = fast.next

while fast:

fast = fast.next

slow = slow.next

slow.next = slow.next.next

return dummy.next

def printLinkedList(head):

result = []

while head:

result.append(head.val)

head = head.next

return result

head1 = ListNode(1)

head1.next = ListNode(2)

head1.next.next = ListNode(3)

head1.next.next.next = ListNode(4)

head1.next.next.next.next = ListNode(5)

print(printLinkedList(removeNthFromEnd(head1, 2))) Output: [1, 2, 3, 5]

head2 = ListNode(1)

print(printLinkedList(removeNthFromEnd(head2, 1))) Output: []

head3 = ListNode(1)

head3.next = ListNode(2)

print(printLinkedList(removeNthFromEnd(head3, 1))) Output: [1]

**Time Complexity**: O(n)

**Space Complexity**: O(1)

20. Valid Parentheses

Given a string s containing just the characters '(', ')', '{', '}', '[' and ']', determine if the input string is valid. An input string is valid if:

1. Open brackets must be closed by the same type of brackets.

2. Open brackets must be closed in the correct order.

3. Every close bracket has a corresponding open bracket of the same type.

**Code with output:**

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

print(isValid("()")) Output: True

print(isValid("()[]{}")) Output: True

print(isValid("(]")) Output: False

**Time Complexity**: O(n)

**Space Complexity**: O(n)