

N.SAI SANKAR

192311187

## ASSIGNMENT – 2

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. Example 1: Input: height = [1,8,6,2,5,4,8,3,7] Output: 49 Explanation: The above vertical lines are represented by array [1,8,6,2,5,4,8,3,7]. In this case, the max area of water (blue section) the container can contain is 49. Example 2: Input: height = [1,1] Output: 1 Constraints: •  $n == \text{height.length}$  •  $2 \leq n \leq 105$  •  $0 \leq \text{height}[i] \leq 104$

```
def maxArea(height):
```

```
    left, right = 0, len(height) - 1
```

```
    max_area = 0
```

```
    while left < right:
```

```
        width = right - left    min_height =
```

```
        min(height[left], height[right])    current_area =
```

```
        width * min_height    max_area =
```

```
        max(max_area, current_area)
```

```
        if height[left] < height[right]:
```

```
            left += 1
```

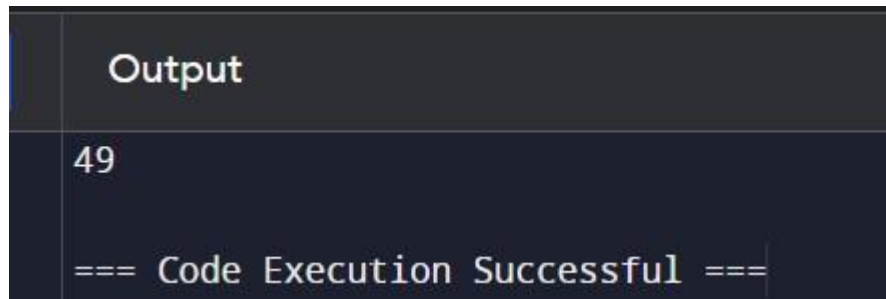
```
        else:
```

```
            right -= 1
```

```
    return max_area
```

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

```
print(maxArea(height))
```



```
Output
49
=== Code Execution Successful ===
```

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. Example 1: Input: num = 3 Output: "III" Explanation: 3 is represented as 3 ones. Example 2: Input: num = 58 Output: "LVIII" Explanation: L = 50, V = 5, III = 3. Example 3: Input: num = 1994 Output: "MCMXCIV" Explanation: M = 1000, CM = 900, XC = 90 and IV = 4. Constraints: • 1 <= num <= 3999

```
def intToRoman(num):    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_numeral = ""
```

```
    for value, roman in value_to_roman:
```

```
        while num >= value:
```

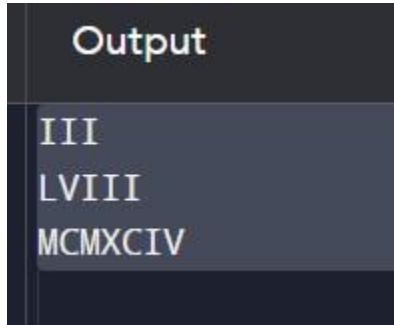
```
            roman_numeral += roman
```

```
            num -= value
```

```
    return roman_numeral
```

```
print(intToRoman(3)) print(intToRoman(58))
```

```
print(intToRoman(1994))
```



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. Example 1: Input: s = "III" Output: 3 Explanation: III = 3. Example 2: Input: s = "LVIII" Output: 58 Explanation: L = 50, V= 5, III = 3. Example 3: Input: s = "MCMXCIV" Output: 1994 Explanation: M = 1000, CM = 900, XC = 90 and IV = 4. Constraints: • 1 <= s.length <= 15 • s contains only the characters ('I', 'V', 'X', 'L', 'C', 'D', 'M'). • It is guaranteed that s is a valid roman numeral in the range [1, 3999].

```
def romanToInt(s: str) -> int:
```

```
    roman_to_int = {
```

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

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

```
    }
```

```
    total = 0
```

```
    i = 0
```

```
    n = len(s)
```

```
    while i < n:        if i + 1 < n and roman_to_int[s[i]] <
```

```
        roman_to_int[s[i + 1]]:
```

```

        total -= roman_to_int[s[i]]
    else:
        total += roman_to_int[s[i]]
    i += 1

```

```

return total

```

```

print(romanToInt("III"))    print(romanToInt("LVIII"))
print(romanToInt("MCMXCIV"))

```

Output
3
58
1994

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 "". Example 1: Input: strs = ["flower", "flow", "flight"] Output: "fl" Example 2: Input: strs = ["dog", "racecar", "car"] Output: "" Explanation: There is no common prefix among the input strings. Constraints: •  $1 \leq \text{strs.length} \leq 200$  •  $0 \leq \text{strs}[i].\text{length} \leq 200$  •  $\text{strs}[i]$  consists of only lowercase English letters.

```

def longestCommonPrefix(strs):

```

```

    if not strs:

```

```

    return ""

```

```

    prefix = strs[0]

```

```

    for string in strs[1:]:        while

```

```

    string[:len(prefix)] != prefix:

```

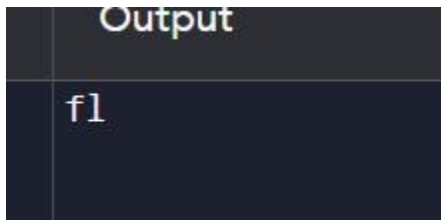
```

        prefix = prefix[:-1]
    if not prefix:
        return ""

    return prefix

print(longestCommonPrefix(["flower", "flow", "flight"]))
print(longestCommonPrefix(["dog", "racecar", "car"]))

```



15. 3Sum Given an integer array `nums`, return all the triplets `[nums[i], nums[j], nums[k]]` such that  $i \neq j$ ,  $i \neq k$ , and  $j \neq k$ , and  $nums[i] + nums[j] + nums[k] == 0$ . Notice that the solution set must not contain duplicate triplets. Example 1: Input: `nums = [-1,0,1,2,-1,-4]` Output: `[[-1,-1,2],[-1,0,1]]` Explanation:  $nums[0] + nums[1] + nums[2] = (-1) + 0 + 1 = 0$ .  $nums[1] + nums[2] + nums[4] = 0 + 1 + (-1) = 0$ .  $nums[0] + nums[3] + nums[4] = (-1) + 2 + (-1) = 0$ . The distinct triplets are `[-1,0,1]` and `[-1,-1,2]`. Notice that the order of the output and the order of the triplets does not matter. Example 2: Input: `nums = [0,1,1]` Output: `[]` Explanation: The only possible triplet does not sum up to 0. Example 3: Input: `nums = [0,0,0]` Output: `[[0,0,0]]` Explanation: The only possible triplet sums up to 0. Constraints: 

- $3 \leq \text{nums.length} \leq 3000$
- $-105 \leq \text{nums}[i] \leq 105$

 def

```
threeSum(nums):
```

```

    nums.sort()

    result = []

    n = len(nums)

    for i in range(n - 2):
        if i > 0 and
        nums[i] == nums[i - 1]:
            continue

        left, right = i + 1, n - 1

        while left < right:

```

```

s = nums[i] + nums[left] + nums[right]
if s == 0:
    result.append([nums[i], 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
elif s < 0:
    left += 1
else:
    right -= 1

return result

```

```

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

```

Output
<pre> [[[-1, -1, 2], [-1, 0, 1]] [] [[0, 0, 0]] </pre>

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. Example 1: Input: `nums = [-1,2,1,-4]`, `target = 1` Output: 2 Explanation: The sum that is closest to the target is 2. (-1 + 2 + 1 = 2). Example 2: Input: `nums = [0,0,0]`, `target = 1` Output: 0 Explanation: The sum that is closest to the target is 0. (0 + 0 + 0 = 0). Constraints: • 3 ≤ `nums.length` ≤ 500 • -1000 ≤ `nums[i]` ≤ 1000 • -104 ≤ `target` ≤ 104

```
def threeSumClosest(nums, target):
```

```

    nums.sort()
closest_sum = float('inf')
n = len(nums)

for i in range(n - 2):
    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)) print(threeSumClosest([0,
0, 0], 1))

```

Output
<pre> [[-1, -1, 2], [-1, 0, 1]] [] [[0, 0, 0]] </pre>

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. Example 1: Input: digits = "23" Output: ["ad","ae","af","bd","be","bf","cd","ce","cf"] Example 2: Input: digits = "" Output: [] Example 3: Input: digits = "2" Output: ["a","b","c"] Constraints: •  $0 \leq \text{digits.length} \leq 4$  •  $\text{digits}[i]$  is a digit in the range ['2', '9']. def letterCombinations(digits):

if not digits:

return []

digit\_to\_char = {

'2': 'abc', '3': 'def', '4': 'ghi', '5': 'jkl', '6': 'mno',

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

}

def backtrack(index, path):

if index == len(digits):

result.append("".join(path))

return

current\_digit = digits[index] for

char in digit\_to\_char[current\_digit]:

path.append(char)

backtrack(index + 1, path)

path.pop()



```

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

print(letterCombinations("23"))
print(letterCombinations(""))
print(letterCombinations("2"))

```

Output
['ad', 'ae', 'af', 'bd', 'be', 'bf', 'cd', 'ce', 'cf']
[]
['a', 'b', 'c']

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: •  $0 \leq a, b, c, d < n$  • a, b, c, and d are distinct. •  $nums[a] + nums[b] + nums[c] + nums[d] == target$  You may return the answer in any order. Example 1: Input: nums = [1,0,-1,0,-2,2], target = 0 Output: [[-2,-1,1,2],[-2,0,0,2],[-1,0,0,1]] Example 2: Input: nums = [2,2,2,2,2], target = 8 Output: [[2,2,2,2]] Constraints: •  $1 \leq nums.length \leq 200$  •  $-109 \leq nums[i] \leq 109$  •  $-109 \leq target \leq 109$

```

def fourSum(nums, target):
    nums.sort()
    result = []
    n = len(nums)

    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:
                sum_ = nums[i] + nums[j] + nums[left] + nums[right]
                if sum_ == target:

```

```

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

    elif sum_ < target:

        left += 1

else:

        right -= 1

return result

```

```

print(fourSum([1, 0, -1, 0, -2, 2], 0)) print(fourSum([2,
2, 2, 2, 2], 8))

```

Output
<pre> [[-2, -1, 1, 2], [-2, 0, 0, 2], [-1, 0, 0, 1]] [[2, 2, 2, 2]] </pre>

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. Example 1: Input: head = [1,2,3,4,5], n = 2 Output: [1,2,3,5] Example 2:

Input: head = [1], n = 1 Output: [] Example 3: Input: head = [1,2], n = 1 Output: [1] Constraints: • The

number of nodes in the list is sz. •  $1 \leq sz \leq 30$  •  $0 \leq \text{Node.val} \leq 100$  •  $1 \leq n \leq sz$  class

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

self.val = val

self.next = next

```
def removeNthFromEnd(head, n):  
    dummy = ListNode(0, head)  
    first = dummy    second = dummy
```

```
    for _ in range(n + 1):  
        first = first.next
```

```
    while first:  
        first = first.next  
    second = second.next
```

```
    second.next = second.next.next
```

```
    return dummy.next
```

```
def create_linked_list(lst):  
    dummy = ListNode(0)    current  
    = dummy  
    for val in lst:  
        current.next = ListNode(val)  
    current = current.next    return  
    dummy.next
```

```
def print_linked_list(head):  
    current = head  
    result = []  
    while current:
```

```

        result.append(current.val)
    current = current.next    return
    result

```

```

head = create_linked_list([1, 2, 3, 4, 5]) n
= 2
new_head = removeNthFromEnd(head, n)
print(print_linked_list(new_head))

```

```

head = create_linked_list([1]) n
= 1
new_head = removeNthFromEnd(head, n) print(print_linked_list(new_head))

```

```

head = create_linked_list([1, 2]) n
= 1
new_head = removeNthFromEnd(head, n) print(print_linked_list(new_head))

```

Output
[1, 2, 3, 5]
[]
[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. Example 1: Input: *s* = "()" Output: true Example 2: Input: *s* = "()[]{}" Output: true Example 3: Input: *s* = "(]" Output: false Constraints: • 1 ≤ *s*.length ≤ 104 • *s* consists of parentheses only '()[]{}' def isValid(*s*): stack = [] mapping = {'(': ')', '[': ']', '{': '}'

```

    for char in s:
        if char in
mapping.values():
            stack.append(char)
        elif char in
mapping.keys():
            if not stack or stack.pop()
!= mapping[char]:
                return False
    else:
        return False

    return not stack

```

```

print(isValid("()"))
print(isValid("{}[]{}"))
print(isValid("{}"))

```

```

Output
True
True
False

```