11. Container With Most Water

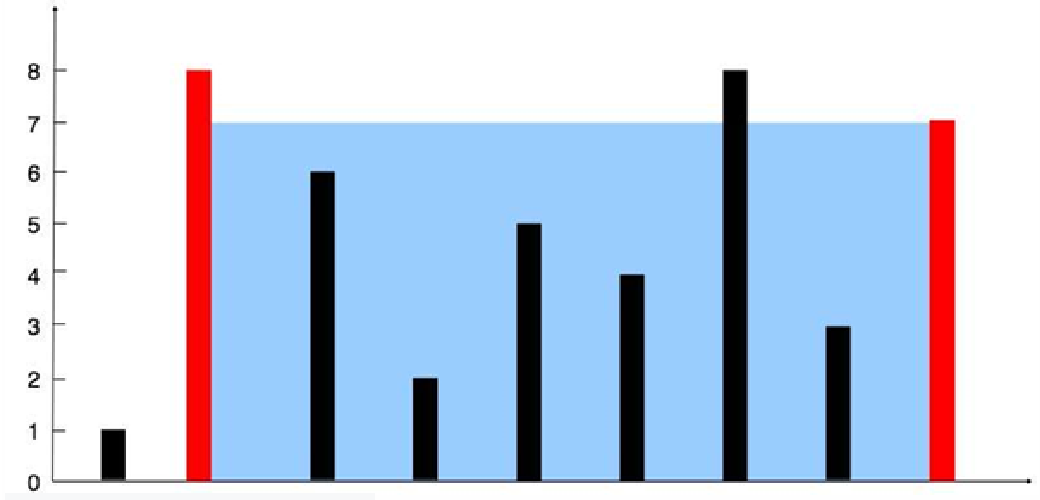
You are given an integer array height of length n. There are n vertical lines drawn such that the two endpoints of the ith line are (i, 0) and (i, height[i]).

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 == height.length
- 2 <= n <= 105
- 0 <= height[i] <= 104

SOLUTION

Initialize Two Pointers:

Set one pointer at the beginning (left = 0) and the other at the end (right = n - 1) of the array.

Calculate the Area:

For each pair of lines pointed to by the left and right pointers, calculate the area of the container they form. The area is given by the formula:

area=(right−left)×min(height[left],height[right])

area=(right−left)×min(height[left],height[right])

Keep track of the maximum area found during these calculations.

Move the Pointers:

To attempt to find a taller container, move the pointer pointing to the shorter line inward. This is because moving the shorter line might help find a taller line, which could potentially form a larger area with the other line.

If height[left] < height[right], move the left pointer to the right (left += 1).

Otherwise, move the right pointer to the left (right -= 1).

Continue Until the Pointers Meet:

Repeat the process until the left and right pointers meet.

IMPLEMENTATION

```
def maxArea(height):
    left, right = 0, len(height) - 1
    max_area = 0

    while left < right:
        width = right - left
        current_area = width * min(height[left], height[right])
        max_area = max(max_area, current_area)
        if height[left] < height[right]:
            left += 1
        else:
            right -= 1

    return max_area
# Example 1
height1 = [1, 8, 6, 2, 5, 4, 8, 3, 7]
print(maxArea(height1))
```

O/P:

```
49
```

12. integer to roman

```
 def intToRoman(num):
    val = [
        1000, 900, 500, 400,
        100, 90, 50, 40,
        10, 9, 5, 4,
        1
        ]
    syms = [
        "M", "CM", "D", "CD",
        "C", "XC", "L", "XL",
        "X", "IX", "V", "IV",
        "I"
        ]
    roman_numeral = ''
    for i in range(len(val)):
        while num >= val[i]:
            num -= val[i]
            roman_numeral += syms[i]
    return roman_numeral
print(intToRoman(58))
   Output: "LVIII"
```

13.roman to integer

```
def romanToInt(s):
    roman_to_int = {
        '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_int[char]
        if value >= prev_value:
            total += value
        else:
            total -= value
        prev_value = value
    return total
print(romanToInt("MCMXCIV"))
```

Output: 1994

14.longest common prefix

```
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
print(longestCommonPrefix(["flower","flow","flight"])) #
```

Output: "fl"

15.3 sum

```
def threeSum(nums):
    nums.sort()
    result = []
    n = len(nums)
    for i in range(n):
        # Skip the same element to avoid duplicates
        if i > 0 and nums[i] == nums[i - 1]:
            continue

        left, right = i + 1, n - 1

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

            if current_sum == 0:
                result.append([nums[i], nums[left], nums[right]])

                # Skip duplicates for the second number
                while left < right and nums[left] == nums[left + 1]:
                    left += 1
                # Skip duplicates for the third number
                while left < right and nums[right] == nums[right - 1]:
                    right -= 1

                left += 1
                right -= 1
            elif current_sum < 0:
                left += 1
            else:
                right -= 1

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

16.3 sum closest

```
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 abs(current_sum - target) < abs(closest_sum - target):
                closest_sum = current_sum

            if current_sum < target:
                left += 1
            elif current_sum > target:
                right -= 1
            else:
                return current_sum

    return closest_sum
print(threeSumClosest([0, 0, 0], 1))
```

```
    Output: 0
```

17.phone mapping

```
phone_map = {
    '2': 'abc', '3': 'def', '4': 'ghi', '5': 'jkl',
    '6': 'mno', '7': 'pqrs', '8': 'tuv', '9': 'wxyz'
}
def letterCombinations(digits):
    if not digits:
        return []

    phone_map = {
        '2': 'abc', '3': 'def', '4': 'ghi', '5': 'jkl',
        '6': 'mno', '7': 'pqrs', '8': 'tuv', '9': 'wxyz'
    }

    def backtrack(index, path):
        # If the path length is equal to digits length, we have a complete combination
        if index == len(digits):
            combinations.append(''.join(path))
            return

        # Get the letters that the current digit maps to, and iterate over them
        possible_letters = phone_map[digits[index]]
        for letter in possible_letters:
            path.append(letter)
            backtrack(index + 1, path)
            path.pop()  # Backtrack

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

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

18.4 sums

```
def fourSum(nums, target):
    nums.sort()
```

```
n = len(nums)
quadruplets = []

for i in range(n - 3):
    # Skip duplicates for the first number
    if i > 0 and nums[i] == nums[i - 1]:
        continue
    for j in range(i + 1, n - 2):
        # Skip duplicates for the second number
        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:
                quadruplets.append([nums[i], nums[j], nums[left], nums[right]])
                # Skip duplicates for the third number
                while left < right and nums[left] == nums[left + 1]:
                    left += 1
                # Skip duplicates for the fourth number
                while left < right and nums[right] == nums[right - 1]:
                    right -= 1
                left += 1
                right -= 1
            elif total < target:
                left += 1
            else:
                right -= 1

return quadruplets
```

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

o/p:

```
[[-2, -1, 1, 2], [-2, 0, 0, 2], [-1, 0, 0, 1]]
```

19. Remove Nth Node From End of List

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

def removeNthFromEnd(head: ListNode, n: int) -> ListNode:
    dummy = ListNode(0, head)  # Create a dummy node to handle edge cases smoothly
    first = dummy
    second = dummy

    # Move first n+1 steps ahead, so the gap between first and second is n nodes
    for _ in range(n + 1):
        first = first.next

    # Move both first and second until first reaches the end
    while first:
        first = first.next
        second = second.next

    # Now, second.next is the node to be removed
    second.next = second.next.next

    return dummy.next

# Helper function to create a linked list from a list and return the head
```

```
def create_linked_list(arr):
    if not arr:
        return None
    head = ListNode(arr[0])
    current = head
    for val in arr[1:]:
        current.next = ListNode(val)
        current = current.next
    return head

# Helper function to convert a linked list to a list
def linked_list_to_list(head):
    result = []
    while head:
        result.append(head.val)
        head = head.next
    return result

head = create_linked_list([1,2,3,4,5])
n = 2
new_head = removeNthFromEnd(head, n)
print(linked_list_to_list(new_head))  # Output: [1, 2, 3, 5]

head = create_linked_list([1])
n = 1
new_head = removeNthFromEnd(head, n)
print(linked_list_to_list(new_head))  # Output: []

head = create_linked_list([1,2])
n = 1
```

```
new_head = removeNthFromEnd(head, n)
print(linked_list_to_list(new_head))
```

```
o/p:
[1, 2, 3, 5]
[]
[1]
```

20.valid parenthesis

```
def isValid(s: str) -> bool:
    stack = []
    mapping = {')': '(', '}': '{', ']': '['}

    for char in s:
        if char in mapping:
            # If the character is a closing bracket
            # Pop the topmost element from the stack
            # If stack is empty, assign a dummy value '#'
            top_element = stack.pop() if stack else '#'

            # Check if the popped bracket corresponds to the current closing bracket
            if mapping[char] != top_element:
                return False
        else:
            # If the character is an opening bracket, push it onto the stack
            stack.append(char)

    # If stack is empty, all brackets were closed properly
    return not stack
print(isValid("()[]{}"))
```

```
 Output: True
```