ASSIGNMENT-2

11.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
Program:

```
def maxArea(height):
    max_area = 0
    left = 0
    right = len(height) - 1
    while left < right:
        area = min(height[left], height[right]) * (right - left)
        max_area = max(max_area, area)
        if height[left] < height[right]:
            left += 1
        else:
            right -= 1
    return max_area
height1 = [1, 8, 6, 2, 5, 4, 8, 3, 7]
print(maxArea(height1))
height2 = [1, 1]
print(maxArea(height2))
```

Output:

```
the maximum areais : 49
the maximum area is : 1
```

12. 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.

Program:

```
def intToRoman(num):
    roman_numerals = {
        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'}
    result = ''
    for value, symbol in roman_numerals.items():
        while num >= value:
            result += symbol
            num -= value
    return result
num1 = 3
print(intToRoman(num1))
num2 = 58
print(intToRoman(num2))
num3 = 1994
print(intToRoman(num3))
```

Output:

```
interger to roman is : III
 integer to roman is :  LVIII
integer to roman  is :  MCMXCIV
```

13.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

Program:

```
def romanToInt(s):
    roman_numerals = {
        'I': 1, 'V': 5, 'X': 10, 'L': 50,
        'C': 100, 'D': 500, 'M': 1000  }
    total = 0
    for i in range(len(s)):
        # If the current numeral is smaller than the next one, subtract its value
        if i < len(s) - 1 and roman_numerals[s[i]] < roman_numerals[s[i + 1]]:
            total -= roman_numerals[s[i]]
        else:
            total += roman_numerals[s[i]]
    return total
s1 = "III"
print(romanToInt(s1))
s2 = "LVIII"
print(romanToInt(s2))
s3 = "MCMXCIV"
print(romanToInt(s3))
```

OUTPUT:

```
roman to integer is :  3
roman to integer is : 58
roman to integer is : 1994
```

14. 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.

Program:

```
def longestCommonPrefix(strs):
```

```
    if not strs:
        return ""
    for i in range(len(strs[0])):
        char = strs[0][i]
        for string in strs[1:]:
            # If the index is beyond the length of the string or the character is different
            if i >= len(string) or string[i] != char:
                return strs[0][:i]
    return strs[0]
strs1 = ["flower", "flow", "flight"]
print(longestCommonPrefix(strs1))
strs2 = ["dog", "racecar", "car"]
print(longestCommonPrefix(strs2))
```

Output:

```
longest common prefix is : fl
```

15. 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.

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.

Program:

```
def threeSum(nums):
    nums.sort()
    triplets = []
    for i in range(len(nums) - 2):
        if i > 0 and nums[i] == nums[i - 1]:
            continue
```

```
        left = i + 1
        right = len(nums) - 1
        while left < right:
            total = nums[i] + nums[left] + nums[right]
            if total == 0:
                triplets.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 triplets
nums1 = [-1, 0, 1, 2, -1, -4]
print(threeSum(nums1))
nums2 = [0, 1, 1]
print(threeSum(nums2))
nums3 = [0, 0, 0]
print(threeSum(nums3))
```

Output:

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

16. 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).

Program:

```
def threeSumClosest(nums, target):
    nums.sort()
    closest_sum = float('inf')
    for i in range(len(nums) - 2):
        left = i + 1
        right = len(nums) - 1
        while left < right:
            total = nums[i] + nums[left] + nums[right]
            if abs(total - target) < abs(closest_sum - target):
                closest_sum = total
            if total == target:
                return total
            elif total < target:
                left += 1
            else:
                right -= 1
    return closest_sum
nums1 = [-1, 2, 1, -4]
target1 = 1
print(threeSumClosest(nums1, target1))
nums2 = [0, 0, 0]
target2 = 1
print(threeSumClosest(nums2, target2))
```

Output:

```
three sumclosest is: 2
three sumclosest is : 0
```

17. 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"]

Program:

```
def letterCombinations(digits):
    if not digits:
        return []
    digit_map = {
        '2': 'abc',
        '3': 'def',
        '4': 'ghi',
        '5': 'jkl',
        '6': 'mno',
        '7': 'pqrs',
        '8': 'tuv',
        '9': 'wxyz'  }
    def backtrack(combination, next_digits):
        if len(next_digits) == 0:
            # Add the combination to the result
            combinations.append(combination)
        else:
            for letter in digit_map[next_digits[0]]:
                backtrack(combination + letter, next_digits[1:])
    combinations = []
    backtrack('', digits)
    return combinations
digits1 = "23"
print(letterCombinations(digits1))
digits2 = ""
print(letterCombinations(digits2))
digits3 = "2"
print(letterCombinations(digits3))
```

Output:

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

18. 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.

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]]

Program:

```
def fourSum(nums, target):
    nums.sort()
    quadruplets = []
    for i in range(len(nums) - 3):
        if i > 0 and nums[i] == nums[i - 1]:
            continue
        for j in range(i + 1, len(nums) - 2):
            if j > i + 1 and nums[j] == nums[j - 1]:
                continue
            left = j + 1
            right = len(nums) - 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]])
                    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 quadruplets
nums1 = [1, 0, -1, 0, -2, 2]
target1 = 0
print(fourSum(nums1, target1))
nums2 = [2, 2, 2, 2, 2]
target2 = 8
print(fourSum(nums2, target2))
```

Output:

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

19. 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]

Program:

```
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
    first = dummy
    second = dummy
    for _ in range(n + 1):
        first = first.next
    while first is not None:
        first = first.next
```

```
        second = second.next
    second.next = second.next.next
    return dummy.next
nodes = [ListNode(val) for val in [1, 2, 3, 4, 5]]
for i in range(len(nodes) - 1):
    nodes[i].next = nodes[i + 1]
head1 = nodes[0]
n1 = 2
new_head1 = removeNthFromEnd(head1, n1)
while new_head1 is not None:
    print(new_head1.val, end=" ")
    new_head1 = new_head1.next
print()
head2 = ListNode(1)
n2 = 1
new_head2 = removeNthFromEnd(head2, n2)
while new_head2 is not None:
    print(new_head2.val, end=" ")
    new_head2 = new_head2.next
print()
head3 = ListNode(1, ListNode(2))
n3 = 1
new_head3 = removeNthFromEnd(head3, n3)
while new_head3 is not None:
    print(new_head3.val, end=" ")
    new_head3 = new_head3.next
```

Output:

20. 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

Program:

```
def isValid(s):
    stack = []
    mapping = {')': '(', ']': '[', '}': '{'}
    for char in s:
        # If the character is an opening bracket, push it onto the stack
        if char in mapping.values():
            stack.append(char)
        else:
            if not stack or stack[-1] != mapping[char]:
                return False
            else:
                stack.pop()
    return not stack
s1 = "()"
print(isValid(s1))
s2 = "()[]{}"
print(isValid(s2))
s3 = "(]"
print(isValid(s3))
```

Output:

```
True
True
False
```