1. Two Sum
   Given an array of integers nums and an integer target, return *indices of the two numbers such that they add up to target*.
   You may assume that each input would have *exactly* one solution, and you may not use the *same* element twice.
   You can return the answer in any order.

   Example 1:
   Input: nums = [2,7,11,15], target = 9
   Output: [0,1]
   Explanation: Because nums[0] + nums[1] == 9, we return [0, 1].

   Example 2:
   Input: nums = [3,2,4], target = 6
   Output: [1,2]

   Example 3:
   Input: nums = [3,3], target = 6
   Output: [0,1]

   Constraints:
   - 2 <= nums.length <= 104
   - -109 <= nums[i] <= 109
   - -109 <= target <= 109
   - Only one valid answer exists.

PROGRAM:

```
def two_sum(nums, target):
    seen = {}
    for i, num in enumerate(nums):
        if target - num in seen:
            return [seen[target - num], i]
        seen[num] = i
```

OUTPUT:

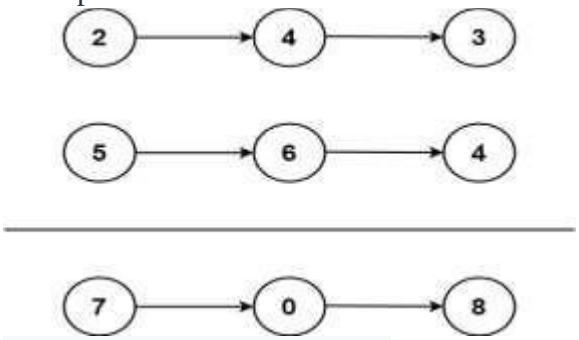
```
[3, 4]
```

2. Add Two Numbers

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

You may assume the two numbers do not contain any leading zero, except the number 0 itself.

Example 1:



Input: l1 = [2,4,3], l2 = [5,6,4]
Output: [7,0,8]
Explanation: 342 + 465 = 807.

Example 2:
Input: l1 = [0], l2 = [0]
Output: [0]

Example 3:
Input: l1 = [9,9,9,9,9,9,9], l2 = [9,9,9,9]
Output: [8,9,9,9,0,0,0,1]

Constraints:

- The number of nodes in each linked list is in the range [1, 100].
- 0 <= Node.val <= 9
- It is guaranteed that the list represents a number that does not have leading zeros.

PROGRAM:

```
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:
        sum_val = (l1.val if l1 else 0) + (l2.val if l2 else 0) + carry
        carry, val = divmod(sum_val, 10)

        current.next = ListNode(val)
        current = current.next

        l1 = l1.next if l1 else None
        l2 = l2.next if l2 else None

    return dummy.next
```

OUTPUT:

3. Longest Substring without Repeating Characters
Given a string s, find the length of the longest substring without repeating characters.

Example 1:
Input: s = "abcabcbb"
Output: 3
Explanation: The answer is "abc", with the length of 3.

Example 2:
Input: s = "bbbbb"
Output: 1
Explanation: The answer is "b", with the length of 1.

Example 3:
Input: s = "pwwkew"
Output: 3
Explanation: The answer is "wke", with the length of 3.
Notice that the answer must be a substring, "pwke" is a subsequence and not a substring.

Constraints:

- 0 <= s.length <= 5 * 104
- s consists of English letters, digits, symbols and spaces.

PROGRAM:

```
class Solution:
    def lengthOfLongestSubstring(self, s: str) -> int:
        start = maxLength = 0
        usedChars = {}

        for i, char in enumerate(s):
            if char in usedChars and start <= usedChars[char]:
                start = usedChars[char] + 1
            else:
                maxLength = max(maxLength, i - start + 1)

            usedChars[char] = i

        return maxLength
```

OUTPUT:

4. Median of Two Sorted Arrays
Given two sorted arrays nums1 and nums2 of size m and n respectively, return the median of the two sorted arrays.
The overall run time complexity should be O(log (m+n)).

Example 1:

Input: nums1 = [1,3], nums2 = [2]
Output: 2.00000
Explanation: merged array = [1,2,3] and median is 2.

Example 2:
Input: nums1 = [1,2], nums2 = [3,4]
Output: 2.50000
Explanation: merged array = [1,2,3,4] and median is (2 + 3) / 2 = 2.5.

Constraints:

- nums1.length == m
- nums2.length == n
- 0 <= m <= 1000
- 0 <= n <= 1000
- 1 <= m + n <= 2000
- -106 <= nums1[i], nums2[i] <= 106

PROGRAM:

```
def findMedianSortedArrays(nums1, nums2):
    nums = sorted(nums1 + nums2)
    n = len(nums)
    if n % 2 == 0:
        return (nums[n // 2 - 1] + nums[n // 2]) / 2
    else:
        return nums[n // 2]

# Example 1
nums1 = [1, 3]
nums2 = [2]
print(findMedianSortedArrays(nums1, nums2))  # Output: 2.00000

# Example 2
nums1 = [1, 2]
nums2 = [3, 4]
print(findMedianSortedArrays(nums1, nums2))  # Output: 2.50000
```

5. Longest Palindromic Substring

Given a string s, return *the longest palindromic substring* in s.

Example 1:
Input: s = "babad"
Output: "bab"
Explanation: "aba" is also a valid answer.

Example 2:
Input: s = "cbbd"
Output: "bb"

Constraints:

- 1 <= s.length <= 1000
- s consist of only digits and English letters.

PROGRAM:

```
class Solution:
    def longestPalindrome(self, s: str) -> str:
        def expandAroundCenter(left, right):
            while left >= 0 and right < len(s) and s[left] == s[right]:
                left -= 1
                right += 1
            return s[left + 1:right]

        if len(s) == 0:
            return ""

        longest = ""
        for i in range(len(s)):
            palindrome1 = expandAroundCenter(i, i)
            palindrome2 = expandAroundCenter(i, i + 1)
            longest = max(longest, palindrome1, palindrome2, key=len)

        return longest

# Example 1
s1 = "babad"
print(Solution().longestPalindrome(s1))  # Output: "bab"

# Example 2
s2 = "cbbd"
print(Solution().longestPalindrome(s2))  # Output: "bb"
```

OUTPUT:

6. Zigzag Conversion
   The string "PAYPALISHIRING" is written in a zigzag pattern on a given number of rows like this: (you may want to display this pattern in a fixed font for better legibility)
   P A H N
   A P L S I I G
   Y I R

   And then read line by line: "PAHNAPLSIIGYIR"
   Write the code that will take a string and make this conversion given a number of rows:
   string convert(string s, int numRows);

   Example 1:
   Input: s = "PAYPALISHIRING", numRows = 3
   Output: "PAHNAPLSIIGYIR"

Example 2:
Input: s = "PAYPALISHIRING", numRows = 4
Output: "PINALSIGYAHRPI"

Explanation:
P I N
A L S I G
Y A H R
P I

Example 3:
Input: s = "A", numRows = 1
Output: "A"

Constraints:

- 1 <= s.length <= 1000
- s consists of English letters (lower-case and upper-case), ',' and '.'.
- 1 <= numRows <= 1000

PROGRAM:

```
def convert(s, numRows):
    if numRows == 1 or numRows >= len(s):
        return s

    rows = [''] * numRows
    index, step = 0, 1

    for char in s:
        rows[index] += char
        if index == 0:
            step = 1
        elif index == numRows - 1:
            step = -1
        index += step

    return ''.join(rows)

# Example 1
s = "PAYPALISHIRING"
numRows = 3
output1 = convert(s, numRows)
print(output1)

# Example 2
numRows = 4
output2 = convert(s, numRows)
print(output2)
```

7. Reverse Integer
   Given a signed 32-bit integer x, return x *with its digits reversed*. If reversing x causes the value to go outside the signed 32-bit integer range [-231, 231 - 1], then return 0.
   Assume the environment does not allow you to store 64-bit integers (signed or unsigned).

   Example 1:
   Input: x = 123
   Output: 321

   Example 2:
   Input: x = -123
   Output: -321

   Example 3:
   Input: x = 120
   Output: 21

   Constraints:
   - -231 <= x <= 231 - 1

PROGRAM:

```
def reverse(x):
    sign = [1, -1][x < 0]
    rev = sign * int(str(abs(x))[::-1])
    return rev if -(2**31) <= rev <= 2**31 - 1 else 0


# Test Cases
print(reverse(123))  # Output: 321
print(reverse(-123))  # Output: -321
print(reverse(120))  # Output: 21
```

OUTPUT:

8. String to Integer (atoi)
   Implement the myAtoi(string s) function, which converts a string to a 32-bit signed integer (similar to C/C++'s atoi function).
   The algorithm for myAtoi(string s) is as follows:
   - Read in and ignore any leading whitespace.
   - Check if the next character (if not already at the end of the string) is '-' or '+'. Read this character in if it is either. This determines if the final result is negative or positive respectively. Assume the result is positive if neither is present.
   - Read in next the characters until the next non-digit character or the end of the input is reached. The rest of the string is ignored.

- Convert these digits into an integer (i.e. "123" -> 123, "0032" -> 32). If no digits were read, then the integer is 0. Change the sign as necessary (from step 2).
- If the integer is out of the 32-bit signed integer range [-231, 231 - 1], then clamp the integer so that it remains in the range. Specifically, integers less than -231 should be clamped to -231, and integers greater than 231 - 1 should be clamped to 231 - 1.
- Return the integer as the final result.

Note:

- Only the space character ' ' is considered a whitespace character.
- Do not ignore any characters other than the leading whitespace or the rest of the string after the digits.

Example 1:
Input: s = "42"
Output: 42
Explanation: The underlined characters are what is read in, the caret is the current reader position.
Step 1: "42" (no characters read because there is no leading whitespace)
^
Step 2: "42" (no characters read because there is neither a '-' nor '+')
^
Step 3: "42" ("42" is read in)
^
The parsed integer is 42.
Since 42 is in the range [-231, 231 - 1], the final result is 42.

Example 2:
Input: s = " -42"
Output: -42
Explanation:
Step 1: "__-42" (leading whitespace is read and ignored)
^
Step 2: " -42" ('-' is read, so the result should be negative)
^
Step 3: " -42" ("42" is read in)
^
The parsed integer is -42.
Since -42 is in the range [-231, 231 - 1], the final result is -42.

Example 3:
Input: s = "4193 with words"
Output: 4193
Explanation:
Step 1: "4193 with words" (no characters read because there is no leading whitespace)
^
Step 2: "4193 with words" (no characters read because there is neither a '-' nor '+')
^

Step 3: "4193 with words" ("4193" is read in; reading stops because the next character is a non-digit)
^
The parsed integer is 4193.
Since 4193 is in the range [-231, 231 - 1], the final result is 4193.
Constraints:

- 0 <= s.length <= 200
- s consists of English letters (lower-case and upper-case), digits (0-9), ' ', '+', '-', and '.'.

PROGRAM:

```
def myAtoi(s):
    s = s.strip()
    if not s:
        return 0

    sign = 1
    if s[0] == '-':
        sign = -1
        s = s[1:]
    elif s[0] == '+':
        s = s[1:]

    num = 0
    for char in s:
        if not char.isdigit():
            break
        num = num * 10 + int(char)

    num = max(-2**31, min(sign * num, 2**31 - 1))

    return num
```

OUTPUT:

9. Palindrome Number

Given an integer x, return true *if* x *is a palindrome, and* false *otherwise*.

Example 1:
Input: x = 121
Output: true
Explanation: 121 reads as 121 from left to right and from right to left.

Example 2:
Input: x = -121
Output: false
Explanation: From left to right, it reads -121. From right to left, it becomes 121-. Therefore it is not a palindrome.

Example 3:
Input: x = 10
Output: false
Explanation: Reads 01 from right to left. Therefore it is not a palindrome.

Constraints:

- -231 <= x <= 231 - 1

PROGRAM:

```
class Solution:
    def isPalindrome(self, x: int) -> bool:
        if x < 0:
            return False
        return str(x) == str(x)[::-1]
```

OUTPUT:

10. Regular Expression Matching

Given an input string `s` and a pattern `p`, implement regular expression matching with support for `'.'` and `'*'` where:

- `'.'` Matches any single character.
- `'*'` Matches zero or more of the preceding element.

The matching should cover the entire input string (not partial).

Example 1:
Input: s = "aa", p = "a"
Output: false
Explanation: "a" does not match the entire string "aa".

Example 2:
Input: s = "aa", p = "a*"
Output: true

Explanation: '*' means zero or more of the preceding element, 'a'. Therefore, by repeating 'a' once, it becomes "aa".

Example 3:
Input: s = "ab", p = ".*"
Output: true
Explanation: ".*" means "zero or more (*) of any character (.)".

Constraints:

- 1 <= s.length <= 20
- 1 <= p.length <= 30
- s contains only lowercase English letters.
- p contains only lowercase English letters, '.', and '*'.
- It is guaranteed for each appearance of the character '*', there will be a previous valid character to match.

PROGRAM:

```
class Solution:
    def isMatch(self, s: str, p: str) -> bool:
        if not p:
            return not s

        first_match = bool(s) and p[0] in {s[0], '.'}

        if len(p) >= 2 and p[1] == '*':
            return (self.isMatch(s, p[2:]) or
                    first_match and self.isMatch(s[1:], p))
        else:
            return first_match and self.isMatch(s[1:], p[1:])
```

OUTPUT:

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

PROGRAM:

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

    while left < right:
        width = right - left
        h = min(height[left], height[right])
        max_area = max(max_area, width * h)

        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(max_area(height1))  # Output: 49

# Example 2
height2 = [1, 1]
print(max_area(height2))  # Output: 1
```

OUTPUT:

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

PROGRAM:

```
class Solution:
    def intToRoman(self, num: int) -> str:
        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_num = ''
        i = 0
```

```
        while num > 0:
            for _ in range(num // val[i]):
                roman_num += syms[i]
                num -= val[i]
            i += 1
        return roman_num

# Example 1
num = 3
sol = Solution()
print(sol.intToRoman(num))  # Output: "III"

# Example 2
num = 58
sol = Solution()
print(sol.intToRoman(num))  # Output: "LVIII"
```

OUTPUT:

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].

PROGRAM:

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

        for char in s:
            curr_value = roman_dict[char]
            total += curr_value

            if curr_value > prev_value:
                total -= 2 * prev_value
```

```
            prev_value = curr_value

        return total

# Test cases
solution = Solution()
print(solution.romanToInt("III"))  # Output: 3
print(solution.romanToInt("LVIII"))  # Output: 58
print(solution.romanToInt("MCMXCIV"))  # Output: 1994
```
OUTPUT:

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 <= strs.length <= 200
- 0 <= strs[i].length <= 200
- strs[i] consists of only lowercase English letters.

PROGRAM:

```
def longestCommonPrefix(strs):
    if not strs:
        return ""
    shortest = min(strs, key=len)
    for i, char in enumerate(shortest):
        for other in strs:
            if other[i] != char:
                return shortest[:i]
    return shortest
```

OUTPUT:

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.

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 <= nums.length <= 3000
- -105 <= nums[i] <= 105

PROGRAM:

```
def threeSum(nums):
    nums.sort()
    res = []
    for i in range(len(nums)-2):
        if i > 0 and nums[i] == nums[i-1]:
            continue
        l, r = i+1, len(nums)-1
        while l < r:
            total = nums[i] + nums[l] + nums[r]
            if total < 0:
                l += 1
            elif total > 0:
                r -= 1
            else:
                res.append([nums[i], nums[l], nums[r]])
                while l < r and nums[l] == nums[l+1]:
                    l += 1
                while l < r and nums[r] == nums[r-1]:
                    r -= 1
                l += 1
                r -= 1
    return res

# Example 1
nums1 = [-1, 0, 1, 2, -1, -4]
print(threeSum(nums1))

# Example 2
nums2 = [0, 1, 1]
print(threeSum(nums2))

# Example 3
nums3 = [0, 0, 0]
print(threeSum(nums3))
```

OUTPUT:

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).

PROGRAM:

```
class Solution:
    def threeSumClosest(self, nums, target):
        nums.sort()
        closest_sum = float('inf')

        for i in range(len(nums) - 2):
            left, right = i + 1, len(nums) - 1

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

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

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

        return closest_sum

# Example
nums = [-1, 2, 1, -4]
target = 1
solution = Solution()
output = solution.threeSumClosest(nums, target)
print(output)
```

OUTPUT:

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 <= digits.length <= 4
- digits[i] is a digit in the range ['2', '9'].

PROGRAM:

```
from typing import List

class Solution:
    def letterCombinations(self, digits: str) -> List[str]:
        if not digits:
            return []

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

        def backtrack(index, path):
            if index == len(digits):
                combinations.append(''.join(path))
                return

            for letter in phone[digits[index]]:
                path.append(letter)
                backtrack(index + 1, path)
                path.pop()

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

return combinations

OUTPUT:

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.

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 <= nums.length <= 200
- -109 <= nums[i] <= 109
- -109 <= target <= 109

PROGRAM:

```
def fourSum(nums, target):
    nums.sort()
    res = []
    n = len(nums)
    for i in range(n):
        for j in range(i + 1, n):
            left = j + 1
            right = n - 1
            while left < right:
                total = nums[i] + nums[j] + nums[left] + nums[right]
                if total == target:
                    res.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
                while j + 1 < n and nums[j] == nums[j + 1]:
                    j += 1
            while i + 1 < n and nums[i] == nums[i + 1]:
                i += 1
    return res
```

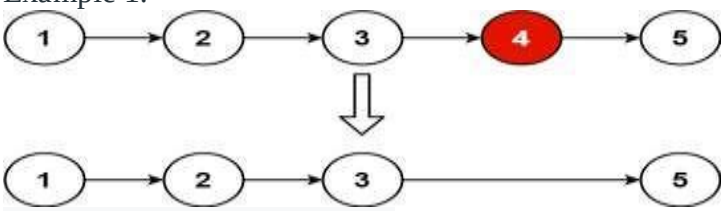
```
# Example 1
nums1 = [1, 0, -1, 0, -2, 2]
target1 = 0
output1 = fourSum(nums1, target1)
print(output1)

# Example 2
nums2 = [2, 2, 2, 2, 2]
target2 = 8
output2 = fourSum(nums2, target2)
print(output2)
```
OUTPUT:

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]

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 i in range(1, n+2):
        first = first.next

    while first is not None:
        first = first.next
        second = second.next

    second.next = second.next.next

    return dummy.next
```

OUTPUT:

Constraints:

- The number of nodes in the list is sz.
- 1 <= sz <= 30
- 0 <= Node.val <= 100
- 1 <= n <= sz

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:

- Open brackets must be closed by the same type of brackets.
- Open brackets must be closed in the correct order.
- 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 '()[]{}'.

PROGRAM:

```
def is_valid(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

# Test Cases
print(is_valid("()"))      # Output: True
print(is_valid("()[]{}"))  # Output: True
print(is_valid("(]"))      # Output: False
```

OUTPUT: