ASSIGNMENT – 1

QUESTION 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.

CODE :

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

QUESTION 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.

CODE :

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


def addTwoNumbers(l1, l2):
    dummy = ListNode(0)
    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 = sum_val // 10
        current.next = ListNode(sum_val % 10)
        current = current.next
        l1 = l1.next if l1 else None
        l2 = l2.next if l2 else None
        return dummy.next
```

QUESTION 3:

Longest Substring without Repeating Characters

Given a string s, find the length of the longest substring without repeating characters.

CODE:

```
def length_of_longest_substring(s):
    start = maxLength = 0
    usedChars = {}

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

        usedChars[s[i]] = i

    return maxLength

# Test the function with examples
print(length_of_longest_substring("abcabcbb"))
print(length_of_longest_substring("bbbbb"))
print(length_of_longest_substring("pwwkew"))
```

QUESTION 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)).

CODE :

```
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: 2.0


# Example 2
nums1 = [1, 2]
nums2 = [3, 4]
print(findMedianSortedArrays(nums1, nums2))  # Output: 2.5
```

QUESTION 5:

Longest Palindromic Substring

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

CODE ;

```
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 Usage
solution = Solution()
print(solution.longestPalindrome("babad"))  # Output: "bab"
print(solution.longestPalindrome("cbbd"))   # Output: "bb"
```

QUESTION 6:

Zigzag Conversion

The string "PAYPALISHIRING" is written in a zigzag pattern on a given number of rows

CODE:

```
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)
```

QUESTION 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).

CODE :

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

QUESTION 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).

CODE :

```
def myAtoi(s: str) -> int:
    # Remove leading whitespaces
    s = s.lstrip()
    # Check if the string is empty
    if not s:
        return 0
    # Initialize variables
    sign = 1
    result = 0
    i = 0
    # Handle sign
    if s[i] in ('+', '-'):
        sign = -1 if s[i] == '-' else 1
        i += 1
    # Convert characters to integer
    while i < len(s) and s[i].isdigit():
        result = result * 10 + int(s[i])
        i += 1
    # Apply sign and check for overflow
```

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

    return result

# Example usage
input_str = "   -42"
print(myAtoi(input_str))
```

QUESTION 9:

Palindrome Number

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

CODE :

```
def is_palindrome(x: int) -> bool:
    # Convert negative numbers to positive for comparison
    if x < 0:
        return False
    # Convert the integer to a string
    num_str = str(x)
    # Check if the string is equal to its reverse
    return num_str == num_str[::-1]
# Example usage
print(is_palindrome(121))  # Output: True
print(is_palindrome(-121))  # Output: False
print(is_palindrome(10))  # Output: False
```

QUESTION 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.

CODE :

```
def is_match(s: str, p: str) -> bool:
    # Initialize a 2D DP table
    dp = [[False] * (len(p) + 1) for _ in range(len(s) + 1)]
    dp[0][0] = True

    # Fill in the first row (empty string s)
    for j in range(1, len(p) + 1):
        if p[j - 1] == '*':
            dp[0][j] = dp[0][j - 2]

    # Fill in the rest of the DP table
    for i in range(1, len(s) + 1):
        for j in range(1, len(p) + 1):
            if p[j - 1] == s[i - 1] or p[j - 1] == '.':
                dp[i][j] = dp[i - 1][j - 1]
            elif p[j - 1] == '*':
```

```
                dp[i][j] = dp[i][j - 2] or (dp[i - 1][j] and (s[i - 1] == p[j - 2] or p[j - 2]
== '.'))

    return dp[len(s)][len(p)]

# Example usage
print(is_match("aa", "a"))  # Output: False
print(is_match("aa", "a*"))  # Output: True
print(is_match("ab", ".*"))  # Output: True
```