1.Given an array of integers nums and an integer target, return indices of the two numbers such

that they add up to target.

Sol:-

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

nums1 = [2, 7, 11, 15]

target1 = 9

print(two\_sum(nums1, target1)) # Output: [0, 1]

nums2 = [3, 2, 4]

target2 = 6

print(two\_sum(nums2, target2)) # Output: [1, 2]

nums3 = [3, 3]

target3 = 6

print(two\_sum(nums3, target3)) # Output: [0, 1]

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.

Sol:-

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

l1 = ListNode(2, ListNode(4, ListNode(3)))

l2 = ListNode(5, ListNode(6, ListNode(4)))

result = addTwoNumbers(l1, l2)

while result:

print(result.val, end=" ")

result = result.next

print()

3. Longest Substring without Repeating Characters

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

Sol:- def length\_of\_longest\_substring(s):

start = 0

max\_length = 0

char\_index\_map = {}

for end in range(len(s)):

if s[end] in char\_index\_map:

start = max(start, char\_index\_map[s[end]] + 1)

char\_index\_map[s[end]] = end

max\_length = max(max\_length, end - start + 1)

return max\_length

s = "abcabcbb"

print(length\_of\_longest\_substring(s))

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)).

Sol:- 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]

nums1 = [1, 3]

nums2 = [2]

print(findMedianSortedArrays(nums1, nums2))

5. Longest Palindromic Substring

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

Sol:-

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) < 1:

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

s = "babad"

solution = Solution()

output = solution.longestPalindrome(s)

print(output)

6. 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);

sol:-

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)

s = "PAYPALISHIRING"

numRows = 3

output = convert(s, numRows)

print(output) # Output: "PAHNAPLSIIGYIR"

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).

Sol:-

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

8. 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:

Sol:-

class Solution:

def myAtoi(self, s: str) -> int:

s = s.strip()

if not s:

return 0

sign = 1

if s[0] in ['-', '+']:

if s[0] == '-':

sign = -1

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

9. Palindrome Number

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

Sol:- def is\_palindrome(x):

return str(x) == str(x)[::-1]

x = 121

print(is\_palindrome(x))

10. 10. Regular Expression Matching

Given an input string s and a pattern p, implement regular expression matching with support for

'.' and '\*' where:

Sol:-

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:])