ASSSIGNMENT 1

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]

Code:-

class Solution:

def twoSum(self, nums, target):

num\_map = {}

for i, num in enumerate(nums):

complement = target - num

if complement in num\_map:

return [num\_map[complement], i]

num\_map[num] = i

# Test the function

solution = Solution()

nums = [2, 7, 11, 15]

target = 9

print(solution.twoSum(nums, target))

1. 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 Solution:

def addTwoNumbers(self, l1: Optional[ListNode], l2: Optional[ListNode]) -> Optional[ListNode]:

if not l1:

return l2

if not l2:

return l1

head = l1

prev = l1

carry = 0

while l1 or l2:

sum\_ = (l1.val if l1 else 0) + (l2.val if l2 else 0) + carry

carry = sum\_//10

sum\_ = sum\_%10

if l1:

l1.val = sum\_

prev = l1

l1 = l1.next

else:

newNode = ListNode(sum\_, None)

prev.next = newNode

prev = prev.next

if l2:

l2 = l2.next

if carry == 1:

newNode = ListNode(carry, None)

prev.next = newNode

prev = prev.next

return head

1. 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.

Code:-

def areDistinct(str, i, j):

visited = [0] \* (256)

for k in range(i, j + 1):

if (visited[ord(str[k])] == True):

return False

visited[ord(str[k])] = True

return True

def longestUniqueSubsttr(str):

n = len(str)

res = 0

for i in range(n):

for j in range(i, n):

if (areDistinct(str, i, j)):

res = max(res, j - i + 1)

return res

if \_\_name\_\_ == '\_\_main\_\_':

str = "greenforest"

print("The input is ", str)

len = longestUniqueSubsttr(str)

print("The length of the longest "

"non-repeating character substring is ", len)

1. 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.

Code:-

import sys

from typing import List

def findMedianSortedArrays(nums1: List[int], nums2: List[int]) -> float:

if len(nums1) > len(nums2):

nums1, nums2 = nums2, nums1

m = len(nums1)

n = len(nums2)

start = 0

end = m

while start <= end:

partition\_nums1 = (start + end) // 2

partition\_nums2 = (m + n + 1) // 2 - partition\_nums1

side after partition

maxLeftNums1 = -sys.maxsize if partition\_nums1 == 0 else nums1[partition\_nums1 - 1]

side after partition

minRightNums1 = sys.maxsize if partition\_nums1 == m else nums1[partition\_nums1]

maxLeftNums2 = -sys.maxsize if partition\_nums2 == 0 else nums2[partition\_nums2 - 1]

minRightNums2 = sys.maxsize if partition\_nums2 == n else nums2[partition\_nums2]

if maxLeftNums1 <= minRightNums2 and maxLeftNums2 <= minRightNums1:

if (m + n) % 2 == 0:

return (max(maxLeftNums1, maxLeftNums2) + min(minRightNums1, minRightNums2)) / 2

else:

return max(maxLeftNums1, maxLeftNums2)

elif maxLeftNums1 > minRightNums2:

end = partition\_nums1 - 1

else:

start = partition\_nums1 + 1

raise Exception("IllegalArgumentException")

1. 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.

Code:-

def printSubStr(str, low, high):

for i in range(low, high + 1):

print(str[i], end="")

def longestPalSubstr(str):

n = len(str)

maxLength = 1

start = 0

for i in range(n):

for j in range(i, n):

flag = 1

for k in range(0, ((j - i) // 2) + 1):

if (str[i + k] != str[j - k]):

flag = 0

if (flag != 0 and (j - i + 1) > maxLength):

start = i

maxLength = j - i + 1

print("Longest palindrome substring is: ", end="")

printSubStr(str, start, start + maxLength - 1)

# Return length of LPS

return maxLength

if \_\_name\_\_ == '\_\_main\_\_':

str = "forgeeksskeegfor"

print("\nLength is:", longestPalSubstr(str))

1. 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)

Code:-

class Solution(object):

def convert(self, s, numRows):

"""

:type s: str

:type numRows: int

:rtype: str

"""

if numRows == 1 or numRows >= len(s):

return s

L = [''] \* numRows

index, step = 0, 1

for x in s:

L[index] += x

if index == 0:

step = 1

elif index == numRows -1:

step = -1

index += step

return ''.join(L)

1. 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:-

def reverse\_integer(x):

sign = -1 if x < 0 else 1

x \*= sign

reversed\_x = int(str(x)[::-1])

if reversed\_x > 2\*\*31 - 1:

return 0

else:

return reversed\_x \* sign

1. 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: 1. Read in and ignore any leading whitespace. 2. 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. 3. 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. 4. 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). 5. 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. 6. 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.

Code:-

class Solution {

public int myAtoi(String str) {

long out = 0;

int i = 0;

int len = str.length();

char firstNotSpace = ' ';

boolean positive = true;

int digit = 0;

while ( i < len )

{

if (str.charAt(i) != ' ' && firstNotSpace == ' ')

{

if (str.charAt(i) == '+')

{

firstNotSpace = '+';

positive = true;

i++;

}

else if (str.charAt(i) == '-')

{

firstNotSpace = '-';

positive = false;

i++;

}

else if (str.charAt(i) >= '0' && str.charAt(i) <= '9')

{

digit = Character.getNumericValue(str.charAt(i));

out = out \* 10 + digit;

firstNotSpace = str.charAt(i);

i++;

if ((out > Integer.MAX\_VALUE) && positive == true)

return Integer.MAX\_VALUE;

if ((-out < Integer.MIN\_VALUE) && positive == false)

return Integer.MIN\_VALUE;

}

else

{

return 0;

}

}

else if (str.charAt(i) == ' ' && firstNotSpace == ' ')

{

i++;

continue;

}

else if (str.charAt(i) >= '0' && str.charAt(i) <= '9')

{

digit = Character.getNumericValue(str.charAt(i));

out = out \* 10 + digit;

i++;

if ((out > Integer.MAX\_VALUE) && positive == true)

return Integer.MAX\_VALUE;

if ((-out < Integer.MIN\_VALUE) && positive == false)

return Integer.MIN\_VALUE;

}

else if (firstNotSpace != ' ')

{

break;

}

}

if (positive == true)

{

return (int)out;

}

else

{

return (int)-out;

}

}

}

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

Code:-

class Solution:

def isPalindrome(self, x: int) -> bool:

x = str(x)

if x == x[::-1]:

return True

else:

return False

1. 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).

Code:-

class Solution:

def isMatch(self, s: str, p: str) -> bool:

s, p = ' '+ s, ' '+ p

lenS, lenP = len(s), len(p)

dp = [[0]\*(lenP) for i in range(lenS)]

dp[0][0] = 1

for j in range(1, lenP):

if p[j] == '\*':

dp[0][j] = dp[0][j-2]

for i in range(1, lenS):

for j in range(1, lenP):

if p[j] in {s[i], '.'}:

dp[i][j] = dp[i-1][j-1]

elif p[j] == "\*":

dp[i][j] = dp[i][j-2] or int(dp[i-1][j] and p[j-1] in {s[i], '.'})

return bool(dp[-1][-1])