**Saveetha School of Engineering**

**Saveetha Institute of Medical And Technical Science**

**ASSIGNMENT - 01**

**PROGRAMMING LANGUAGE**

Python

**COURSE CODE / NAME**

CSA0666 - Design And Analysis Of Algorithm For Divide And Conquer Techniques

**SUBMITTED BY**

T Svetha – 192321155

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

def twoSum(nums, target):

seen = {}

for i, num in enumerate(nums):

complement = target - num

if complement in seen:

return [seen[complement], i]

seen[num] = i

return []

nums = [2, 7, 11, 15]

target = 9

result = twoSum(nums, target)

print(result)

Output: [0,1]

TIME COMPLEXITY: O(n)

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.

class ListNode:

def \_init\_(self, val=0, next=None):

self.val = val

self.next = next

class Solution:

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

dummy = head = ListNode(0)

carry = 0

while l1 or l2 or carry:

val1 = l1.val if l1 else 0

val2 = l2.val if l2 else 0

sum = val1 + val2 + carry

carry = sum // 10

head.next = ListNode(sum % 10)

head = head.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 = Solution().addTwoNumbers(l1, l2)

while result:

print(result.val, end=" -> ")

result = result.next

print("None")

Output: [7 -> 0 -> 8]

Time complexity: O(n)

1. **Longest Substring without Repeating Characters**

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

def longest\_substring\_brute\_force(s: str) -> int:

n = len(s)

max\_length = 0

for i in range(n):

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

substring = s[i:j]

if all(char not in substring[k:j] for k in range(i, j) if char != substring[k]):

max\_length = max(max\_length, len(substring))

return max\_length

s = "abcabcbb"

result = longest\_substring\_brute\_force(s)

print(result)

Output: 3

Time complexity: O(n^2)

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.

class Solution(object):

def findMedianSortedArrays(self, nums1, nums2):

num = sorted(nums1 + nums2)

n=len(num)

if n % 2 != 0:

mid = int(round(n/2))

return num[mid]

else:

mid= int(n/2)

return (num[mid-1] + num[mid])/2.0

nums1 = [1,3]

nums2 = [2]

print( Solution().findMedianSortedArrays(nums1,nums2)

Output: 2.00000

Time complexity: O(n)

1. **Longest Palindromic Substring**

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

def longestPalindrome(s: str) -> str:

n = len(s)

longest = ""

for i in range(n):

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

# Check if the substring is a palindrome

substring = s[i:j]

if substring == substring[::-1] and len(substring) > len(longest):

longest = substring

return longest

s = "babad"

result = longestPalindrome(s)

print(result)

Output: "bab"

Time complexity: O(n³)

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);

def convert(s: str, numRows: int) -> str:

if numRows == 1:

return s

n = len(s)

cycle = 2 \* numRows - 2

result = [""] \* numRows

for i in range(n):

row = i % cycle // 2

result[row] += s[i]

return "".join(result)

s = "PAYPALISHIRING"

numRows = 3

result = convert(s, numRows)

print(result)

Output: "PAHNAPLSIIGYIR"

Time complexity: O(n)

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).

class Solution(object):

def reverse(self, x):

y=0

if x< 0:

y= (-1 \* int(str(-1 \* x)[::-1]) )

elif x>0:

y= int(str(x)[::-1])

return y if -2\*31<=y<=(2\*31)+1 else 0

x=123

print(Solution().reverse(x)

Output: 321

Time complexity: O(n)

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).

def myAtoi(s: str) -> int:

n = len(s)

i = 0

# Skip leading whitespace

while i < n and s[i] == ' ':

i += 1

sign = 1

if i < n and (s[i] == '-' or s[i] == '+'):

sign = -1 if s[i] == '-' else 1

i += 1

result = 0

while i < n and s[i].isdigit():

digit = ord(s[i]) - ord('0')

if result > (2\*31 - 1) // 10 or (result == (2\*31 - 1) // 10 and digit > 7):

return 2\*31 - 1 if sign == 1 else -2\*31

result = result \* 10 + digit

i += 1

return max(-2\*31, min(result \* sign, 2\*31 - 1))

s = "42"

result = myAtoi(s)

print(result)

Output: 42

Time complexity: O(n)

1. **Palindrome Number**

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

class Solution(object):

def isPalindrome(self, x):

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

return True

else:

return False

x=121

print(Solution().isPalindrome(x)

Output: True

Time complexity: O(n)

1. **Regular Expression**

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

import re

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

pattern = re.compile(p)

return bool(pattern.fullmatch(s))

s = "aa"

p = "a"

result = isMatch(s, p)

print(result)

Output: false

Time complexity: SO(n)