

# CSA0670-Design and Analysis of Algorithms for Tractability Problems.

## Assignment

### 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:

- $2 \leq \text{nums.length} \leq 10^4$
- $-10^9 \leq \text{nums}[i] \leq 10^9$
- $-10^9 \leq \text{target} \leq 10^9$
- Only one valid answer exists.

## Program:

```
def two_sum(nums, target):
    num_to_index = {}

    for index, num in enumerate(nums):
        complement = target - num

        if complement in num_to_index:
            return [num_to_index[complement], index]

        num_to_index[num] = index

nums = [2, 7, 11, 15]
target = 9
result = two_sum(nums, target)
print(result) # Output: [0, 1]
```

## Output:

```
"C:\Program Files\Python311\python.exe" C:\Users\shoha\PycharmProjects\pythonProject2\tut1457.py
[0, 1]

Process finished with exit code 0
```

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.

### Program:

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

def addTwoNumbers(l1, l2):
    dummy_head = ListNode(0)
    current = dummy_head
    carry = 0

    while l1 or l2 or carry:
        val1 = l1.val if l1 else 0
        val2 = l2.val if l2 else 0
        total = val1 + val2 + carry
        carry = total // 10
        current.next = ListNode(total % 10)
        current = current.next

        if l1:
            l1 = l1.next
        if l2:
            l2 = l2.next

    return dummy_head.next

def create_linked_list(nums):
    dummy_head = ListNode(0)
    current = dummy_head
    for num in nums:
        current.next = ListNode(num)
        current = current.next
    return dummy_head.next

def print_linked_list(l):
    while l:
        print(l.val, end=" -> " if l.next else "\n")
        l = l.next

l1 = create_linked_list([2, 4, 3])
l2 = create_linked_list([5, 6, 4])

print("Input:")
print("List 1:")
print_linked_list(l1)
print("List 2:")
print_linked_list(l2)
```

```
result = addTwoNumbers(11, 12)

print("Output:")
print_linked_list(result)
```

## Output:

```
"C:\Program Files\Python311\python.exe" C:\Users\shoba\PycharmProjects\pythonProject2\tut1457.py
Input:
List 1:
2 -> 4 -> 3
List 2:
5 -> 6 -> 4
Output:
7 -> 0 -> 8

Process finished with exit code 0
```

## 3. Longest Substring without Repeating Characters

### Program:

```
def length_of_longest_substring(s):
    char_index = {}
    start = 0
    max_length = 0

    for i, char in enumerate(s):
        if char in char_index and char_index[char] >= start:
            start = char_index[char] + 1
        char_index[char] = i
        max_length = max(max_length, i - start + 1)

    return max_length
```

```

s = "abcabcbb"
print(f"The length of the longest substring without repeating characters in '{s}' is {length_of_longest_substring(s)}")

s = "bbbbbb"
print(f"The length of the longest substring without repeating characters in '{s}' is {length_of_longest_substring(s)}")

s = "pwwkew"
print(f"The length of the longest substring without repeating characters in '{s}' is {length_of_longest_substring(s)}")

s = ""
print(f"The length of the longest substring without repeating characters in '{s}' is {length_of_longest_substring(s)}")

```

### Output:

```

"C:\Program Files\Python311\python.exe" C:\Users\shoba\PycharmProjects\pythonProject2\tut1457.py
The length of the longest substring without repeating characters in 'abcabcbb' is 3
The length of the longest substring without repeating characters in 'bbbbbb' is 1
The length of the longest substring without repeating characters in 'pwwkew' is 3
The length of the longest substring without repeating characters in '' is 0

Process finished with exit code 0

```

## 4. Median of Two Sorted Arrays.

Program:

```

def findMedianSortedArrays(nums1, nums2):
    if len(nums1) > len(nums2):
        nums1, nums2 = nums2, nums1

    x, y = len(nums1), len(nums2)
    low, high = 0, x

    while low <= high:
        partitionX = (low + high) // 2
        partitionY = (x + y + 1) // 2 - partitionX

        maxX = float('-inf') if partitionX == 0 else nums1[partitionX - 1]
        minX = float('inf') if partitionX == x else nums1[partitionX]

        maxY = float('-inf') if partitionY == 0 else nums2[partitionY - 1]
        minY = float('inf') if partitionY == y else nums2[partitionY]

        if maxX <= minY and maxY <= minX:
            if (x + y) % 2 == 0:
                return (max(maxX, maxY) + min(minX, minY)) / 2
            else:
                return max(maxX, maxY)
        elif maxX > minY:
            high = partitionX - 1
        else:
            low = partitionX + 1

    raise ValueError("Input arrays are not sorted")

nums1 = [1, 3]
nums2 = [2]

print("Median of arrays {} and {} is: {}".format(nums1, nums2,
findMedianSortedArrays(nums1, nums2)))

nums1 = [1, 2]
nums2 = [3, 4]

print("Median of arrays {} and {} is: {}".format(nums1, nums2,
findMedianSortedArrays(nums1, nums2)))

```

## Output

```
"C:\Program Files\Python311\python.exe" C:\Users\shoba\PycharmProjects\pythonProject2\tut1457.py
Median of arrays [1, 3] and [2] is: 2
Median of arrays [1, 2] and [3, 4] is: 2.5

Process finished with exit code 0
```

## 5. Longest Palindromic Substring.

### Program:

```
def longest_palindromic_substring(s):
    if not s:
        return ""

    start, end = 0, 0

    for i in range(len(s)):
        len1 = expand_around_center(s, i, i)
        len2 = expand_around_center(s, i, i + 1)
        max_len = max(len1, len2)

        if max_len > (end - start):
            start = i - (max_len - 1) // 2
            end = i + max_len // 2

    return s[start:end + 1]

def expand_around_center(s, left, right):
    while left >= 0 and right < len(s) and s[left] == s[right]:
        left -= 1
        right += 1
    return right - left - 1

s = "babad"
print("Longest palindromic substring of '{}' is: {}".format(s,
longest_palindromic_substring(s)))
```

```

s = "cbbd"
print("Longest palindromic substring of '{}' is: '{}'".format(s,
longest_palindromic_substring(s)))

s = "a"
print("Longest palindromic substring of '{}' is: '{}'".format(s,
longest_palindromic_substring(s)))

s = "ac"
print("Longest palindromic substring of '{}' is: '{}'".format(s,
longest_palindromic_substring(s)))

```

## Output:

```

"C:\Program Files\Python311\python.exe" C:\Users\shoba\PycharmProjects\pythonProject2\tut1457.py
Longest palindromic substring of 'babad' is: 'aba'
Longest palindromic substring of 'cbbd' is: 'bb'
Longest palindromic substring of 'a' is: 'a'
Longest palindromic substring of 'ac' is: 'c'

Process finished with exit code 0

```

## 6. ZigZag Conversion.

### Program:

```

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

```



```

        return s

    rows = [''] * numRows
    current_row = 0
    going_down = False

    for char in s:
        rows[current_row] += char
        if current_row == 0 or current_row == numRows - 1:
            going_down = not going_down
        current_row += 1 if going_down else -1

    return ''.join(rows)

s = "PAYPALISHIRING"
numRows = 3
print("Zigzag conversion of '{}' with {} rows is: '{}'".format(s, numRows,
convert(s, numRows)))

s = "PAYPALISHIRING"
numRows = 4
print("Zigzag conversion of '{}' with {} rows is: '{}'".format(s, numRows,
convert(s, numRows)))

s = "A"
numRows = 1
print("Zigzag conversion of '{}' with {} rows is: '{}'".format(s, numRows,
convert(s, numRows)))

```

## Output:

```

"C:\Program Files\Python311\python.exe" C:\Users\shoba\PycharmProjects\pythonProject2\tut1457.py
Zigzag conversion of 'PAYPALISHIRING' with 3 rows is: 'PAHNAPLSIIGYIR'
Zigzag conversion of 'PAYPALISHIRING' with 4 rows is: 'PINALSIGYAHRPI'
Zigzag conversion of 'A' with 1 rows is: 'A'

Process finished with exit code 0

```

## 7. Reverse Integer.

### Program:

```
def reverse(x):
    INT_MAX = 2 ** 31 - 1
    INT_MIN = -2 ** 31

    result = 0
    negative = x < 0
    x = abs(x)

    while x != 0:
        pop = x % 10
        x //= 10

        if result > (INT_MAX - pop) // 10:
            return 0

        result = result * 10 + pop

    if negative:
        result = -result

    return result if INT_MIN <= result <= INT_MAX else 0

print("Reversed integer of {} is: {}".format(123, reverse(123)))
print("Reversed integer of {} is: {}".format(-123, reverse(-123)))
print("Reversed integer of {} is: {}".format(120, reverse(120)))
print("Reversed integer of {} is: {}".format(0, reverse(0)))
print("Reversed integer of {} is: {}".format(1534236469,
reverse(1534236469)))
```

### Output:

```
"C:\Program Files\Python311\python.exe" C:\Users\shoba\PycharmProjects\pythonProject2\tut1457.py
Reversed integer of 123 is: 321
Reversed integer of -123 is: -321
Reversed integer of 120 is: 21
Reversed integer of 0 is: 0
Reversed integer of 1534236469 is: 0

Process finished with exit code 0
```

## 8. String to Integer.

Program:

```
def myAtoi(s):
    INT_MAX = 2**31 - 1
    INT_MIN = -2**31

    i = 0
    n = len(s)
    while i < n and s[i].isspace():
        i += 1

    sign = 1
    if i < n and s[i] == '-':
        sign = -1
        i += 1
    elif i < n and s[i] == '+':
        i += 1

    result = 0
    while i < n and s[i].isdigit():
        digit = int(s[i])
        if result > (INT_MAX - digit) // 10:
            return INT_MAX if sign == 1 else INT_MIN
        result = result * 10 + digit
        i += 1
```

```

        return sign * result

print("Converted integer from '{}' is: {}".format("42", myAtoi("42")))
print("Converted integer from '{}' is: {}".format("-42", myAtoi("-42")))
print("Converted integer from '{}' is: {}".format("4193 with words", myAtoi("4193 with words")))
print("Converted integer from '{}' is: {}".format("words and 987", myAtoi("words and 987")))
print("Converted integer from '{}' is: {}".format("-91283472332", myAtoi("-91283472332")))

```

## Output:

```

"C:\Program Files\Python311\python.exe" C:\Users\shoba\PycharmProjects\pythonProject2\tut1457.py
Converted integer from '42' is: 42
Converted integer from '-42' is: -42
Converted integer from '4193 with words' is: 4193
Converted integer from 'words and 987' is: 0
Converted integer from '-91283472332' is: -2147483648

Process finished with exit code 0

```

## 9. Palindrome Number.

Program:

```

def isPalindrome(x):
    if x < 0:
        return False

```

```

original = x
reversed_num = 0

while x != 0:
    pop = x % 10
    x //= 10
    reversed_num = reversed_num * 10 + pop

return original == reversed_num

print("Is {} a palindrome? {}".format(121, isPalindrome(121)))
print("Is {} a palindrome? {}".format(-121, isPalindrome(-121)))
print("Is {} a palindrome? {}".format(10, isPalindrome(10)))
print("Is {} a palindrome? {}".format(12321, isPalindrome(12321)))

```

Output:

```

"C:\Program Files\Python311\python.exe" C:\Users\shoba\PycharmProjects\pythonProject2\tut1457.py
Is 121 a palindrome? True
Is -121 a palindrome? False
Is 10 a palindrome? False
Is 12321 a palindrome? True

Process finished with exit code 0

```

## 10. Regular Expression Matching.

Program:

```

def isMatch(s, p):
    dp = [[False] * (len(p) + 1) for _ in range(len(s) + 1)]

    dp[0][0] = True

```

```

    for j in range(1, len(p) + 1):
        if p[j - 1] == '*':
            dp[0][j] = dp[0][j - 2]

    for i in range(1, len(s) + 1):
        for j in range(1, len(p) + 1):
            if p[j - 1] == '.' or p[j - 1] == s[i - 1]:
                dp[i][j] = dp[i - 1][j - 1]
            elif p[j - 1] == '*':
                dp[i][j] = dp[i][j - 2]
                if p[j - 2] == '.' or p[j - 2] == s[i - 1]:
                    dp[i][j] = dp[i][j] or dp[i - 1][j]

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

print("Does '{}' match pattern '{}'? {}".format("aa", "a", isMatch("aa",
"a")))
print("Does '{}' match pattern '{}'? {}".format("aa", "a*", isMatch("aa",
"a*")))
print("Does '{}' match pattern '{}'? {}".format("ab", ".*", isMatch("ab",
".*")))
print("Does '{}' match pattern '{}'? {}".format("aab", "c*a*b",
isMatch("aab", "c*a*b")))
print("Does '{}' match pattern '{}'? {}".format("mississippi",
"mis*is*p*.",
isMatch("mississippi",
"mis*is*p*.")))

```

Output:

```
"C:\Program Files\Python311\python.exe" C:\Users\shoba\PycharmProjects\pythonProject2\tut1457.py
Does 'aa' match pattern 'a'? False
Does 'aa' match pattern 'a*'? True
Does 'ab' match pattern '.*'? True
Does 'aab' match pattern 'c*a*b'? True
Does 'mississippi' match pattern 'mis*is*p*.'? False

Process finished with exit code 0
```