1. Two Sum

Given an array of integers nums and an integer target, return *indices of the two numbers suchthat 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 = 9Output: [0,1]
Explanation: Because nums[0] + nums[1] == 9, we return [0, 1].

Example 2:

Input: nums = [3,2,4],
target = 6Output: [1,2]

Example 3:

Input: nums = [3,3],
target = 6Output: [0,1]

Constraints:

- 2 <= nums.length <= 104
- -109 <= nums[i] <= 109
- -109 <= target <= 109
- Only one valid answer exists.

```
num=[2,7,11,15]
target = 9
n = len(num)
for i in range(n-1):
   for j in range(i+1 , n):
      if num[i]+num[j]==target:
         print(f 'target value found in [{i},{j}]')
```

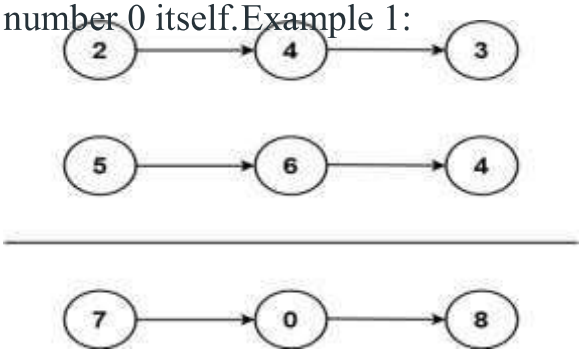
output

target value found in [0,1]

time complexity
$O(n^2)$

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 andreturn the sum as a linked list.
You may assume the two numbers do not contain any leading zero, except the

number 0 itself.Example 1:



Input: l1 = [2,4,3], l2 = [5,6,4]
Output: [7,0,8]
Explanation: 342+465 = 807

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

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

    while l1 or l2:
        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

        if carry > 0:
            current.next = ListNode(carry)

        return dummy_head.next

def create(lst):
    dummy_head = ListNode(0)
    current = dummy_head
    for number in lst:
        current.next = ListNode(number)
        current = current.next
    return dummy_head.next

def linked(node):
    lst = []
    while node:
        lst.append(node.val)
        node = node.next
    return lst

l1 = create([2, 4, 3])
l2 = create([5, 6, 4])
result = addTwoNumbers(l1, l2)
print(linked(result))
```

output
[7, 0, 8]

Time complexity
O(max(m, n))

3. 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.

Example 2:
Input: s =
"bbbbb"Output:
1
Explanation: The answer is "b", with the length of 1.

Example 3:
Input: s =
"pwwkew"
Output: 3
Explanation: The answer is "wke", with the length of 3.
Notice that the answer must be a substring, "pwke" is a subsequence and not a substring.

Constraints:

- 0 <= s.length <= 5 * 104
- s consists of English letters, digits, symbols and spaces.

```
def lengthOfLongestSubstring(s: str)->int:
    n = len(s)
    if (n <= 1):
        return n

    maxlength = 0
    i, j = 0, 0
    charFoundIndex = dict()
    while j < n:
        if s[j] in charFoundIndex:
```

```
            i = max(i, charFoundIndex[s[j]]+1)
        charFoundIndex[s[j]] = j
        maxlength= max(maxlength, j-i+1)
        j += 1

    return maxlength

print(lengthOfLongestSubstring("abcabcbb"))
```

output
3

Time complexity
O(n)

4. Median of Two Sorted Arrays

Given two sorted arrays nums1 and nums2 of size m and n respectively, return the median of thetwo sorted arrays.
The overall run time complexity should be O(log (m+n)).

Example 1:
Input: nums1 = [1,3], nums2 = [2]
Output: 2.00000
Explanation: merged array = [1,2,3] and median is 2

Input: nums1 = [1,2], nums2 = [3,4]
Output: 2.50000
Explanation: merged array = [1,2,3,4] and median is (2 + 3) / 2 = 2.5.

Constraints:

- nums1.length == m
- nums2.length == n
- 0 <= m <= 1000
- 0 <= n <= 1000
- 1 <= m + n <= 2000
- -106 <= nums1[i], nums2[i] <= 106

```
list1 = [1,3]
list2 = [2,4]
nums = list1 + list2
nums.sort()
print(nums)
n = len(nums)
if n % 2 == 1:
   print(nums[n // 2])
else:
   print((nums[n // 2 - 1] + nums[n // 2]) / 2)
```

output

```
[1,2,3,4]
2.5
```

Time complexity

O(n log n)

5. 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.

Example 2:
Input: s =
"cbbd"Output:
"bb"

Constraints:

- 1 <= s.length <= 1000
- s consist of only digits and English letters.

```
def longestPalindrome(s: str) -> str:
    if len(s) <= 1:
        return s

    start = 0
    max_length = 1

    for i in range(len(s)):
        left, right = i, i
        while left >= 0 and right < len(s) and s[left] == s[right]:
            if right - left + 1 > max_length:
                start = left
                max_length = right - left + 1
            left -= 1
            right += 1

        left, right = i, i + 1
        while left >= 0 and right < len(s) and s[left] == s[right]:
            if right - left + 1 > max_length:
                start = left
                max_length = right - left + 1
            left -= 1
            right += 1
```

```
    return s[start:start+max_length]


print(longestPalindrome("babad"))
print(longestPalindrome("cbbd"))
```

output
bab
bb

Time complexity
$O(n^2)$

6. Zigzag Conversion

The string "PAYPALISHIRING" is written in a zigzag pattern on a given number of rowslike 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 GY 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);

Example 1:
Input: s = "PAYPALISHIRING", numRows
= 3Output: "PAHNAPLSIIGYIR"

Example 2:
Input: s = "PAYPALISHIRING", numRows
= 4Output: "PINALSIGYAHRPI"

Explanation:
P I N
A L S I G
Y A H R
P I

Example 3:
Input: s = "A", numRows
= 1Output: "A"

Constraints:

- 1 <= s.length <= 1000
- s consists of English letters (lower-case and upper-case), ',' and '.'.
- 1 <= numRows <= 1000

```
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
print(convert(s, numRows))
```

OUTPUT
PAHNAPLSIIGYIR

Time complexity
O(n)

7. Reverse Integer

Given a signed 32-bit integer x, return x *with its digits reversed*. If reversing x causes the valueto 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).

Example 1:
Input: x = 123Output: 321

Example 2:
Input: x = -123Output: -321

Example 3:
Input: x = 120Output: 21

Constraints:

- -231 <= x <= 231 – 1

```
def reverse_number(num: int) -> int:

    reversed_str = str(num)[::-1]

    if num < 0:
        reversed_str = reversed_str[:-1]
        reversed_num = -int(reversed_str)
    else:

        reversed_num = int(reversed_str)
    return reversed_num


num = 123
```

print(reverse_number(num))

num = -456
print(reverse_number(num))

output
321
-654

Time complexity
O(n)

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:

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 positiverespectively. 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 isreached. 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 stringafter the digits.

Example 1:
Input: s = "42"Output: 42

Explanation: The underlined characters are what is read in, the caret is the current readerposition.
Step 1: "42" (no characters read because there is no leading whitespace)
^
Step 2: "42" (no characters read because there is neither a '-' nor '+')
^
Step 3: "42" ("42" is read in)
^
The parsed integer is 42.
Since 42 is in the range [-231, 231 - 1], the final result is 42.

Example 2:
Input: s = " -42"
Output: -42
Explanation:
Step 1: "_-42" (leading whitespace is read and ignored)
^
Step 2: " _-42" ('-' is read, so the result should be negative)
^
Step 3: " -42" ("42" is read in)
^
The parsed integer is -42.
Since -42 is in the range [-231, 231 - 1], the final result is -42.

Example 3:
Input: s = "4193 with
words"Output: 4193
Explanation:
Step 1: "4193 with words" (no characters read because there is no leading whitespace)
^

Step 2: "4193 with words" (no characters read because there is neither a '-' nor '+')
^
Step 3: "4193 with words" ("4193" is read in; reading stops because the next character is a non-digit)
^
The parsed integer is 4193.
Since 4193 is in the range [-231, 231 - 1], the final result is 4193.

Constraints:

- 0 <= s.length <= 200

s consists of English letters (lower-case and upper-case), digits (0-9), ' ', '+', '-', and '

```
def myAtoi(s: str) -> int:
    s = s.strip()
    if not s:
        return 0
    if s[0] in ['+', '-']:
        sign = -1 if s[0] == '-' else 1
        s = s[1:]
    else:
        sign = 1
    result = 0
    for char in s:
        if char.isdigit():
            result = result * 10 + int(char)
        else:
            break

    result *= sign
    INT_MAX = 2**31 - 1
    INT_MIN = -2**31

 result = max(INT_MIN, min(INT_MAX, result))

 return result


s = "   -42"
print(myAtoi(s))
```

output
-42

Time complexity
O(n)

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

Example 1:
Input: x =
121Output:
true
Explanation: 121 reads as 121 from left to right and from right to left.

Example 2:
Input: x = -
121Output:
false
Explanation: From left to right, it reads -121. From right to left, it becomes 121-.
Therefore it isnot a palindrome.

Example 3:
Input: x =
10Output:
false
Explanation: Reads 01 from right to left. Therefore it is not a palindrome.

Constraints:

- -231 <= x <= 231 – 1

```
def pali(num):
    num = str(num)
    p = num[::-1]
    n = len(num)

if num = = p:
        return True
    else:
        if p[n-1]== "-":
            return False
        else:
            return False
 num = 121
print( pali(num))

num = -121
print(pali(num))
```

output
True
False

Time complexity
O(n)

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.

The matching should cover the entire input string (not partial).

Example 1:
Input: s = "aa", p = "a"

Output: false
Explanation: "a" does not match the entire string "aa".

Example 2:
Input: s = "aa", p = "a*"Output: true
Explanation: '*' means zero or more of the preceding element, 'a'. Therefore, by repeating 'a'once, it becomes "aa".

Example 3:
Input: s = "ab", p = ".*"Output: true
Explanation: ".*" means "zero or more (*) of any character (.)".

Constraints:

- 1 <= s.length <= 20
- 1 <= p.length <= 30
- s contains only lowercase English letters.
- p contains only lowercase English letters, '.', and '*'.
- It is guaranteed for each appearance of the character '*', there will be a previous validcharacter to match.

```
def string(str1, str2):
    if len(str1)==len(str2):
        return True
    else:
        return False

str1 = "qwer"
str2 = ",532"
print(string(str1,str2))
str3 = "345,"
str4 = "5"
print(string(str3,str4))
```

output
True
False

Time complexity
O(n)