**1.** 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.

ANSWER:

def sum(a,b):

d={}

for i,num in enumerate(a):

x=b-num

if x in d:

return [d[x],i]

d[num]=i

n=int(input("Enter a range: "))

a=[]

for j in range(n):

c=int(input("Enter the value: "))

a.append(c)

print(sum(a,int(input("Enter the target: "))))

Input: n=3

a=[2,4,6]

b=10

Output: [1,2]

Time Complexity: O(n)

Space Complexity: O(n)

**2.**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.

ANSWER:

class x:

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

self.val=val

self.next=next

def create(l):

head=x(0)

temp=head

for i in l:

temp.next=x(i)

temp=temp.next

return head.next

def add (l1,l2):

head=x(0)

temp=head

carry=0

while l1 or l2:

v1=l1.val if l1 else 0

v2=l2.val if l2 else 0

total=carry+v1+v2

carry=total//10

temp.next=x(total%10)

temp=temp.next

if l1: l1=l1.next

if l2: l2=l2.next

if carry:

temp.next=x(carry)

return head.next

def priint(l):

temp=l

while temp:

print(temp.val,end="->" if temp.next else "\n")

temp=temp.next

l1=create([2,4,3])

l2=create([5,6,4])

res=add(l1,l2)

priint(res)

Output: 7->0->8

Time Complexity: O(n)

Space Complexity: O(n)

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

ANSWER:

def length(s):

c = {}

left = 0

max\_len = 0

for right, char in enumerate(s):

if char in c and c[char] >= left:

left = c[char] + 1

c[char] = right

max\_len = max(max\_len, right - left + 1)

return max\_len

print(length("abcabcbb"))

Output: 3

Time Complexity: O(n)

Space Complexity: O(n)

**4.**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)).

ANSWER:

def Median(n1, n2):

if len(n1) > len(n2):

n1, n2 = n2, n1

m, n = len(n1), len(n2)

imin, imax, half = 0, m, (m + n + 1) // 2

while imin <= imax:

i = (imin + imax) // 2

j = half - i

if i < m and n1[i] < n2[j - 1]:

imin = i + 1

elif i > 0 and n1[i - 1] > n2[j]:

imax = i - 1

else:

if i == 0:

ml = n2[j - 1]

elif j == 0:

ml = n1[i - 1]

else:

ml = max(n1[i - 1], n2[j - 1])

if (m + n) % 2 == 1:

return ml

if i == m:

mr = n2[j]

elif j == n:

mr = n1[i]

else:

mr = min(n1[i], n2[j])

return (ml + mr) / 2.0

print(Median([1, 3,5], [2]))

OUTPUT: 2.5

Time Complexity: O(log n)

Space Complexity: O(1)

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

ANSWER:

def longestPalindrome(s: str) -> str:

if len(s) < 2:

return s

def expand(left, right):

while left >= 0 and right < len(s) and s[left] == s[right]:

left -= 1

right += 1

return s[left + 1:right]

longest = ""

for i in range(len(s)):

palindrome\_odd = expand(i, i)

palindrome\_even = expand(i, i + 1)

longest = max(longest, palindrome\_odd, palindrome\_even, key=len)

return longest

print(longestPalindrome("babad"))

Output: bab

Time Complexity: O(n)

Space Complexity: O(1)

**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);

ANSWER:

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

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

return s

rows = [''] \* numRows

current\_row = 0

direction = 1

for char in s:

rows[current\_row] += char

current\_row += direction

if current\_row == 0 or current\_row == numRows - 1:

direction \*= -1

zigzag = ''.join(rows)

return zigzag

# Test cases

print(convert("PAYPALISHIRING", 3))

Output: PAHNAPLSIIGYIR

Time Complexity: O(n)

Space Complexity: O(n)

**7.**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).

ANSWER:

x=int(input("Enter a number: "))

z=abs(x)

rev=int(str(z)[::-1])

if x<0:

print(rev\*-1)

else:

print(rev)

Input: -12

Output: -21

Time Complexity: O(log z)

Space Complexity: O(log z)

**8.** 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.

ANSWER:

def myAtoi(s: str) -> int:

result = 0

sign = 1

i = 0

n = len(s)

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

i += 1

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

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

i += 1

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

digit = int(s[i])

if result > (2\*\*31 - digit) // 10:

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

result = result \* 10 + digit

i += 1

return sign \* result

print(myAtoi("42 without alphabets"))

Output: 42

Time Complexity: O(n)

Space Complexity: O(n)

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

ANSWER:

x=int(input("Enter a number: "))

rev=0

y=x

while y>0:

b=y%10

rev=rev\*10+b

y=y//10

print("Palindrome"if x==rev else "Not palindrome")

Input: 12121

Output: Palindrome

Time Complexity: O(log x)

Space Complexity: O(1)

**10.**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).

ANSWER:

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

dp = [[False] \* (len(p) + 1) for \_ in range(len(s) + 1)]

dp[0][0] = True

for j in range(2, 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] or (dp[i - 1][j] and (s[i - 1] == p[j - 2] or p[j - 2] == '.'))

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

print(Match("aa", "a"))

print(Match("aa", "a\*"))

print(Match("ab", ".\*"))

Output: False

True

True

Time Complexity: O(m\*n)

Space Complexity: O(m\*n)