# 1\_Two Sum

Given an array of integers, return **indices** of the two numbers such that they add up to a specific target.

You may assume that each input would have ***exactly*** one solution, and you may not use the *same* element twice.

**Example:**

Given nums = [2, 7, 11, 15], target = 9,

Because nums[0] + nums[1] = 2 + 7 = 9,

return [0, 1].

class Solution:

def twoSum(self, nums: List[int], target: int) -> List[int]:

nu = 0

for i in nums:

try:

num = nums.index(target - i,nu + 1)

return [nu, num]

except:

nu = nu + 1

continue

Runtime: 620 ms, faster than 31.78% of Python3 online submissions for Two Sum.

Memory Usage: 13.8 MB, less than 75.81% of Python3 online submissions for Two Sum.

**bobobo~**

class Solution:

def twoSum(self, nums, target):

"""

:type nums: List[int]

:type target: int

:rtype: List[int]

"""

h = {}

for i, num in enumerate(nums):

n = target - num

if n not in h:

h[num] = i

else:

return [h[n], i]

Runtime: 48 ms, faster than 93.52% of Python3 online submissions for Two Sum.

Memory Usage: 14.1 MB, less than 60.93% of Python3 online submissions for Two Sum.

# 3\_Longest Substring Without Repeating Characters

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

**Example 1:**

Input: "abcabcbb"

Output: 3

Explanation: The answer is "abc", with the length of 3.

**Example 2:**

Input: "bbbbb"

Output: 1

Explanation: The answer is "b", with the length of 1.

**Example 3:**

Input: "pwwkew"

Output: 3

Explanation: The answer is "wke", with the length of 3.

Note that the answer must be a substring, "pwke" is a subsequence and not a substring.

class Solution:

def lengthOfLongestSubstring(self, s: str) -> int:

leng = len(s)

if leng <= 1:

return leng

res = 1

for i in range(leng):

h = {}

mid\_res = 0

if leng - i < res:

break

for j,n in enumerate(s[i::]):

if n not in h:

h[n] = j

mid\_res += 1

else:

break

res = max(res, mid\_res)

return res

Runtime: 488 ms, faster than 15.28% of Python3 online submissions for Longest Substring Without Repeating Characters.

Memory Usage: 12.9 MB, less than 100.00% of Python3 online submissions for Longest Substring Without Repeating Characters.

# 4\_Median of Two Sorted Arrays

There are two sorted arrays **nums1** and **nums2** of size m and n respectively.

Find the median of the two sorted arrays. The overall run time complexity should be O(log (m+n)).

You may assume **nums1** and **nums2** cannot be both empty.

**Example 1:**

nums1 = [1, 3]

nums2 = [2]

The median is 2.0

**Example 2:**

nums1 = [1, 2]

nums2 = [3, 4]

The median is (2 + 3)/2 = 2.5

class Solution:

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

num = nums1 + nums2

num.sort()

m = len(num)

if m%2:

return num[m//2]

else:

return (num[m//2] + num[m//2 - 1])/2

Runtime: 92 ms, faster than 93.94% of Python3 online submissions for Median of Two Sorted Arrays.

Memory Usage: 12.8 MB, less than 100.00% of Python3 online submissions for Median of Two Sorted Arrays.

# 5\_Longest Palindromic Substring

Given a string **s**, find the longest palindromic substring in **s**. You may assume that the maximum length of **s** is 1000.

**Example 1:**

**Input:** "babad"

**Output:** "bab"

**Note:** "aba" is also a valid answer.

**Example 2:**

**Input:** "cbbd"

**Output:** "bb"

class Solution:

def longestPalindrome(self, s: str) -> str:

if len(s) < 2:

return s

ans = 1

t = s[0]

for i in range(1, len(s)):

j = min(i, len(s) - i - 1)

if 2 \* j + 1 <= ans:

pass

else:

#print(i, j)

flag = 1

for m in range(j):

if s[i - m - 1] != s[i + m + 1]:

flag = 0

break

if ans < 2 \* (m + flag) + 1:

ans = 2 \* (m + flag) + 1

t = s[(i - m - flag): (i + m + flag + 1)]

#print("single: t = {0}, i = {1}, j = {2}".format(t,i,m + flag))

for i in range(len(s)):

if i + 2 > len(s):

break

elif s[i] != s[i + 1]:

pass

else:

j = min(i, len(s) - i - 2)

#print("judge i = {0}, j = {1}, ans = {2}".format(i,j,ans))

if 2 \* j + 2 <= ans:

pass

else:

flag = 1

if j == 0:

ans = 2

t = s[i: i + 2]

else:

for m in range(j):

if s[i - m - 1] != s[i + m + 2]:

#print("flag is changing to 0")

#print(" m =", m,"ans =",ans)

flag = 0

break

if ans < 2 \* (m + flag) + 2:

ans = 2 \* (m + flag) + 2

t = s[i - m - flag: i + m + flag + 2]

#print("dual: t = {0}, i = {1}, j = {2}".format(t,i,m + flag))

return t

Runtime: 356 ms, faster than 91.51% of Python3 online submissions for Longest Palindromic Substring.

Memory Usage: 12.9 MB, less than 100.00% of Python3 online submissions for Longest Palindromic Substring.

# 7\_Reverse Integer

Given a 32-bit signed integer, reverse digits of an integer.

**Example 1:**

Input: 123

Output: 321

**Example 2:**

Input: -123

Output: -321

**Example 3:**

Input: 120

Output: 21

**Note:**  
Assume we are dealing with an environment which could only store integers within the 32-bit signed integer range: [−231,  231− 1]. For the purpose of this problem, assume that your function returns 0 when the reversed integer overflows.

class Solution:

def reverse(self, x: int) -> int:

t = 0

flag = 0

if x < 0:

flag = 1

x = abs(x)

while x > 0:

re = x % 10

t = t \* 10 + re

x = x // 10

if flag == 1:

t = -t

if (t >= 2\*\*31) or (t < -2\*\*31):

return 0

return t

Runtime: 28 ms, faster than 93.06% of Python3 online submissions for Reverse Integer.

Memory Usage: 12.7 MB, less than 100.00% of Python3 online submissions for Reverse Integer.

class Solution:

def reverse(self, x: int) -> int:

if x < 0:

flag = 1

else:

flag = 0

x = str(abs(x))

a = int(x[::-1])

if flag == 1:

a = -a

if a >= 2\*\*31 or a < -2\*\*31:

return 0

return a

Runtime: 28 ms, faster than 93.06% of Python3 online submissions for Reverse Integer.

Memory Usage: 12.8 MB, less than 100.00% of Python3 online submissions for Reverse Integer.

# 8\_String to Integer (atoi)

Implement atoi which converts a string to an integer.

The function first discards as many whitespace characters as necessary until the first non-whitespace character is found. Then, starting from this character, takes an optional initial plus or minus sign followed by as many numerical digits as possible, and interprets them as a numerical value.

The string can contain additional characters after those that form the integral number, which are ignored and have no effect on the behavior of this function.

If the first sequence of non-whitespace characters in str is not a valid integral number, or if no such sequence exists because either str is empty or it contains only whitespace characters, no conversion is performed.

If no valid conversion could be performed, a zero value is returned.

**Note:**

* Only the space character ' ' is considered as whitespace character.
* Assume we are dealing with an environment which could only store integers within the 32-bit signed integer range: [−231,  231− 1]. If the numerical value is out of the range of representable values, INT\_MAX (231− 1) or INT\_MIN (−231) is returned.

**Example 1:**

Input: "42"

Output: 42

**Example 2:**

Input: " -42"

Output: -42

Explanation: The first non-whitespace character is '-', which is the minus sign.

  Then take as many numerical digits as possible, which gets 42.

**Example 3:**

Input: "4193 with words"

Output: 4193

Explanation: Conversion stops at digit '3' as the next character is not a numerical digit.

**Example 4:**

Input: "words and 987"

Output: 0

Explanation: The first non-whitespace character is 'w', which is not a numerical

  digit or a +/- sign. Therefore no valid conversion could be performed.

**Example 5:**

Input: "-91283472332"

Output: -2147483648

Explanation: The number "-91283472332" is out of the range of a 32-bit signed integer.

  Thefore INT\_MIN (−231) is returned.

class Solution:

def myAtoi(self, str: str) -> int:

flag = 0

num = 0

num\_flag = 0

INT\_MAX = 2\*\*31 - 1

INT\_MIN = -2\*\*31

for i in str:

if (num\_flag + flag) and not('0' <= i <= '9'):

break

if i != ' ':

if i == '-':

flag = 1

elif i =='+':

flag = 2

elif '0' <= i <= '9':

num = num \* 10 + eval(i)

num\_flag = 1

else:

break

if flag == 1:

num = -num

if num > INT\_MAX:

return INT\_MAX

elif num < INT\_MIN:

return INT\_MIN

return num

class Solution:

def myAtoi(self, str: str) -> int:

s = str.lstrip()

if not s:

return 0

if s[0].isnumeric() or s[0] in ["+", "-"]:

j = 1

while j < len(s):

if s[j].isdigit():

j += 1

else:

break

res = s[0:j]

if res == "+" or res == "-":

return 0

return max(-2\*\*31, min(2\*\*31 - 1, int(res)))

return 0

# 9\_Palindrome Number

Determine whether an integer is a palindrome. An integer is a palindrome when it reads the same backward as forward.

**Example 1:**

**Input:** 121

**Output:** true

**Example 2:**

**Input:** -121

**Output:** false

**Explanation:** From left to right, it reads -121. From right to left, it becomes 121-. Therefore it is not a palindrome.

**Example 3:**

**Input:** 10

**Output:** false

**Explanation:** Reads 01 from right to left. Therefore it is not a palindrome.

class Solution:

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

if x < 0:

return False

num = 0

xx = x

while x > 0:

num = num \* 10 + x % 10

x = x // 10

if num == xx:

return True

else:

return False

Runtime: 64 ms, faster than 75.72% of Python3 online submissions for Palindrome Number.

Memory Usage: 12.7 MB, less than 100.00% of Python3 online submissions for Palindrome Number.

class Solution:

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

if x < 0:

return False

s = str(x)

for i in range(len(s)//2):

if s[i] != s[-i - 1]:

return False

return True

Runtime: 52 ms, faster than 95.34% of Python3 online submissions for Palindrome Number.

Memory Usage: 12.6 MB, less than 100.00% of Python3 online submissions for Palindrome Number.

# 214. Shortest Palindrome

# 65\_Valid Number

Validate if a given string can be interpreted as a decimal number.

Some examples:  
"0" => true  
" 0.1 " => true  
"abc" => false  
"1 a" => false  
"2e10" => true  
" -90e3   " => true  
" 1e" => false  
"e3" => false  
" 6e-1" => true  
" 99e2.5 " => false  
"53.5e93" => true  
" --6 " => false  
"-+3" => false  
"95a54e53" => false

**Note:** It is intended for the problem statement to be ambiguous. You should gather all requirements up front before implementing one. However, here is a list of characters that can be in a valid decimal number:

* Numbers 0-9
* Exponent - "e"
* Positive/negative sign - "+"/"-"
* Decimal point - "."

Of course, the context of these characters also matters in the input.

class Solution:

def isNumber(self, s: str) -> bool:

s = s.lstrip()

eflag = 0

dot\_flag = 0

nflag = 0

#print(s)

if not s:

return False

if s[0].isnumeric() or s[0] in ["+", "-", "."]:

j = 1

if s[0] in ["."]:

dot\_flag = 1

while j < len(s):

if s[j].isdigit():

nflag = 0

elif nflag == 1:

return False

elif eflag == 0 and s[j] == "e":

if s[j-1].isdigit():

eflag = 1

elif s[j-1] == '.' and j>1 and s[j-2].isdigit():

eflag = 1

else:

return False

elif dot\_flag == 0 and ( 1 - eflag) and s[j] == '.':

if not s[j-1].isdigit():

nflag = 1

dot\_flag = 1

elif s[j-1] == 'e' and s[j] in ["+","-"]:

nflag = 1

else:

break

j += 1

ans = s[0:j]

res = s[j:]

#print(res)

res = res.lstrip()

#print("ans = {0}, res = {1}".format(ans,res))

if res:

#print("Brand res")

return False

if s[j-1] == 'e':

#print("Brand s[j-1]")

return False

if ans == ".":

return False

if nflag == 1:

return False

return True

else:

return False

Runtime: 28 ms, faster than 94.81% of Python3 online submissions for Valid Number.

Memory Usage: 12.5 MB, less than 100.00% of Python3 online submissions for Valid Number.