8. String to Integer (atoi) **Leetcode Link** Medium String

Problem Description

this integer.

The myAtoi function is designed to mimic the behavior of the atoi function in C/C++, converting a string to a 32-bit signed integer following a set of specific rules. The steps of the algorithm are:

- 1. Start by skipping any leading whitespace in the input string, as these are irrelevant for number conversion. 2. Identify if the first non-whitespace character is a sign indicator ('-' for negative, '+' for positive). If found, note the sign for the
- final number. If no sign is specified, the number is considered positive by default. 3. Read the subsequent characters until a non-digit is encountered or the end of the string is reached. These characters represent
- the numeric part of the string. 4. Convert the sequence of digits into an integer. If no digits are found, the result should be 0. Apply the previously noted sign to
- 5. Make sure the resulting integer fits within the range of a 32-bit signed integer. If it does not, "clamp" the value to the nearest boundary of the range, which is either -2**31 or 2**31 - 1. 6. Return the processed integer as the final result.

All characters after the numeric part of the string should not be ignored, as they can influence the final output if they are non-

- A couple of important notes to consider: The only whitespace character to be considered is the space character ' '.
- To convert the string to an integer while following the rules, we need to go step by step: 1. Skip Whitespaces: Check each character until we encounter a non-whitespace character or the end of the string. This is

string.

Intuition

digits.

2. Check for a Sign: Once we hit a non-whitespace character, we decide on the sign of our result. If it's a '-' we set the sign to -1,

achieved by using a while loop that increments an index whenever a space is found.

- otherwise, we assume it's positive (1). 3. Convert String to Integer: We parse the string character by character as long as they are numeric (digits). We need to be
- value when multiplied by 10 or when adding the next digit.

mindful of possible overflow past the 32-bit integer range, which happens when our number exceeds the maximum allowed

4. Handle Overflows: Since we're working within the bounds of a signed 32-bit integer, we ensure that any number that goes beyond the limits is clamped to the closest limit, either the lower bound -2**31 or the upper bound 2**31 - 1.

By carefully following each of these steps, we simulate the atoi function's behavior, producing a 32-bit signed integer from an input

Solution Approach The solution implements the conversion process through a series of steps that correspond to the rules provided in the problem

1. Initialization: The function first checks if the string s is not empty. It then initializes important variables such as n for the length of the string, i for the current index, and res for the resulting integer value.

2. Whitespace Skipping: Using a while loop, the implementation skips over the leading whitespace characters ' ' until a non-

whitespace character is encountered by incrementing the i index. If i becomes equal to n (the length of the string), that means the string contained only whitespace, and we return 0.

description. Here's a walk-through of the key components:

The code demonstrates good practices such as:

3. Sign Detection: The next character is checked to determine the sign of the resultant integer. If it's a '-', the variable sign is set to -1, otherwise 1. If a sign character is found, i is incremented to move past it.

4. Integer Conversion: A while loop is used to iterate over the remaining characters of the string, as long as they are digits. Each digit is converted to an integer using int(s[i]) and added to the result res after multiplying the current res by 10 (shifting the number one decimal place to the left). The loop stops if a non-digit character appears.

5. Overflow Handling: Before adding the new digit to res, we check for potential overflow. For positive numbers, it checks if res is

greater than (2**31 - 1) // 10, or if res is equal to (2**31 - 1) // 10 and the next digit (c) is greater than 7 (since 2**31 - 1)

- ends in 7). For negative numbers, the same logic is applied but with the limits of -2**31. If an overflow is detected, the maximum or minimum 32-bit signed integer value is returned based on the sign.
- Efficient string traversal by using index arithmetic rather than creating new strings or arrays. The algorithm does not use any complex data structures, as it only requires integer variables to keep track of the result, the index, and the sign. It is a straightforward and efficient implementation governed by control statements to ensure that the resulting integer adheres to the constraints defined in the problem.

• The string s is not empty, so we initialize n=5 (n being the string's length), i=0 for the current index, and res=0 for the

∘ The next character is '-', which indicates a negative number. We set sign=-1 and increment i to move past the sign, and

• We reach the digits now. With a while loop, we start adding the digits to res. The first character '4' is converted to an

□ During integer conversion, we did not encounter any potential overflow cases, as the number -42 is within the range of a 32-

Let's illustrate the solution approach with a small example. Consider the input string s = " -42" which we want to convert to a 32-bit

6. Result Return: Once the loop finishes, the integer is adjusted with the sign and returned as the result.

• Early exits to avoid unnecessary processing (empty string or string with only spaces).

Overflow protection by checking the conditions before the actual overflow could occur.

 We use a while loop to skip the leading whitespaces. After skipping, i=3 as the first three characters are spaces. 3. Sign Detection:

now i=4.

4. Integer Conversion:

bit signed integer.

6. Result Return:

class Solution:

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

30

31

32

33

34

35

36

37

38

39

35

36

37

38

39

40

41

42

43

44

45

46

47

48

49

50

51

52

53

54

55

56

57

58

59

60

61

63

10

13

14

15

16

17

18

19

20

21

22

23

24

62 }

C++ Solution

public:

#include <climits>

#include <string>

class Solution {

int result = 0;

// Convert the number

break;

index++;

return sign * result;

int myAtoi(std::string s) {

int i = 0;

int answer = 0;

// Trim leading whitespace

while (index < length) {</pre>

// Pre-calculate the threshold to check for overflow

// Break if the current character is not a digit

if (currentChar < '0' || currentChar > '9') {

// Check for overflow when adding a new digit

result = result * 10 + (currentChar - '0');

// Apply the determined sign to the result and return

* Converts the string argument to an integer (32-bit signed integer).

bool isPositive = true; // Flag indicating if the number is positive

// Index for iteration

* or the number goes out of the 32-bit signed integer range.

* @return The parsed integer, or 0 if no conversion is possible.

* @param s - the string to convert to an integer.

size_t start = s.find_first_not_of(" \t\n\r");

s = s.substr(start); // Trim leading whitespace

* Trims whitespace, manages sign, and processes characters until a non-digit is found,

if (start == std::string::npos) return 0; // Return 0 if the string is just whitespace

// Variable to accumulate the parsed number

if (result > threshold || (result == threshold && currentChar > '7')) {

return sign == 1 ? Integer.MAX_VALUE : Integer.MIN_VALUE;

int threshold = Integer.MAX_VALUE / 10;

char currentChar = str.charAt(index);

// Update result with the new digit

if not str:

index = 0

return 0

index += 1

return 0

index += 1

index += 1

result = 0

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

length_of_string = len(str)

Skip leading whitespaces

if index == length_of_string:

digit = int(str[index])

Check for overflow cases

Append current digit to result

result = result * 10 + digit

sign = -1 if str[index] == '-' else 1

while index < length_of_string and str[index] == ' ':</pre>

Check if we have a sign, and set the sign accordingly

if str[index] in ['-', '+']: # skip the sign for next calculations

max_safe_int = (2 ** 31 - 1) // 10 # Precomputed to avoid overflow

if result > max_safe_int or (result == max_safe_int and digit > 7):

return 2 ** 31 - 1 if sign == 1 else -2 ** 31 # Clamp to INT_MIN or INT_MAX

Return 0 if string is empty

2. Whitespace Skipping:

Example Walkthrough

1. Initialization:

signed integer following the defined rules.

resulting integer value.

integer 4, and res becomes 4. Then i is incremented.

- Next, i=5 and the character is '2'. We multiply res by 10 (shifting to the left) and add 2, making res=42. \circ i is incremented again, and with i=n, we have reached the end of the string, and the loop ends. 5. Overflow Handling:
- This example adhered to the steps of the algorithm, resulting in the correct 32-bit signed integer conversion of the input string s.

After skipping whitespaces, if we're at the end it means it's an empty string

○ The final step is to apply the sign to res. With sign=-1, the result becomes -42, which is then returned.

appropriate operations to avoid overflows. **Python Solution**

The key to solving this problem was carefully handling each character by categorizing it as whitespace, sign, or digit, and applying

25 26 while index < length_of_string:</pre> 27 # If the current character is not a digit, break from loop 28 if not str[index].isdigit(): 29 break

```
40
             # Apply sign and return final result
 41
 42
             return sign * result
 43
Java Solution
  1 class Solution {
         public int myAtoi(String str) {
             // Ensure the input string is not null
             if (str == null) {
                 return 0;
  6
  8
             int length = str.length();
  9
             // If the string is empty, return 0
 10
 11
             if (length == 0) {
 12
                 return 0;
 13
 14
 15
             int index = 0;
 16
 17
             // Skip whitespace characters
 18
             while (index < length && str.charAt(index) == ' ') {</pre>
 19
                 index++;
 20
 21
 22
             // If we reached the end of string after skipping spaces, return 0
 23
             if (index == length) {
 24
                 return 0;
 25
 26
 27
             // Determine the sign based on the current character
 28
             int sign = 1;
 29
             if (str.charAt(index) == '-') {
 30
                 sign = -1;
 31
                 index++;
 32
             } else if (str.charAt(index) == '+') {
 33
                 index++;
 34
```

25 // Check if there is a sign in the beginning 26 if (s[i] == '+') { 27 isPositive = true; // Positive number 28 ++i; 29

```
} else if (s[i] == '-') {
                 isPositive = false; // Negative number
 30
 31
                 ++i;
 32
 33
 34
             // Process each character in the string
 35
             for (; i < s.length(); ++i) {</pre>
 36
                 // Calculate the numeric value of the current character
 37
                 int currentValue = s[i] - '0';
 38
 39
                 // Break the loop if the character is not a digit
 40
                 if (currentValue < 0 || currentValue > 9) break;
 41
 42
                 // Check for overflow: if the current answer is already at the risk of overflow
 43
                 // with an additional digit, return the respective limit
 44
                 if (answer > INT_MAX / 10 ||
 45
                     (answer == INT_MAX / 10 && currentValue > INT_MAX % 10)) {
 46
                     return isPositive ? INT_MAX : INT_MIN;
 47
 48
 49
                 // Update the answer by adding the current digit
 50
                 answer = answer * 10 + currentValue;
 51
 52
 53
             // Return the final answer, considering the sign
             return isPositive ? answer : -answer;
 54
 55
 56 };
 57
Typescript Solution
     * Converts the string argument to an integer (32-bit signed integer).
     * Trims whitespace, manages sign, and processes characters until a non-digit is found,
     * or the number goes out of the 32-bit signed integer range.
     * @param str - the string to convert to an integer.
     * @returns The parsed integer, or 0 if no conversion is possible.
     */
    function myAtoi(str: string): number {
        // Trim leading or trailing whitespace
 10
         str = str.trim();
 11
 12
         // Return 0 if the string is empty after trimming
         if (!str) return 0;
 13
 14
 15
         let isPositive: boolean = true; // Flag indicating if the number is positive
 16
         let i: number = 0;
                                        // Index for iteration
 17
         let answer: number = 0;
                                        // Variable to accumulate the parsed number
 18
 19
         // Check if there is a sign in the beginning
 20
         if (str[i] === '+') {
             isPositive = true; // Positive number
 21
 22
             i++;
         } else if (str[i] === '-') {
 23
 24
             isPositive = false; // Negative number
```

48 return isPositive ? answer : -answer; 49 50

Time and Space Complexity

i++;

// Process each character in the string

// Calculate the numeric value of the current character

// with an additional digit, return the respective limit

return isPositive ? 2147483647 : -2147483648;

// Update the answer by adding the current digit

// Break the loop if the character is not a digit

if (currentValue > 9 || currentValue < 0) break;</pre>

if (answer > Math.floor(2147483647 / 10) ||

answer = answer * 10 + currentValue;

// Return the final answer, considering the sign

let currentValue: number = str.charCodeAt(i) - '0'.charCodeAt(0);

answer > Math.floor((2147483647 - currentValue) / 10)) {

// Check for overflow: if the current answer is already at the risk of overflow

for (; i < str.length; i++) {</pre>

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

40

41

42

43

44

45

46

47

The time complexity of the given function is O(n), where n is the length of the string. Here's the breakdown: • We iterate once over the string to trim whitespaces, which takes O(n) in the worst case (if all characters are spaces).

Time Complexity

 Checking if the character is a sign ('+' or '-') is 0(1). The following while loop iterates over the rest of the string, but it runs at most n times, which is also 0(n) in the worst case.

- Each operation inside the while loop, including checking if a character is a digit, converting it to an integer, checking for overflow, and updating the result, is done in constant time 0(1).
- Therefore, the dominating factor in the time complexity is the length of the string n, resulting in O(n) overall. **Space Complexity**

The space complexity of the function is 0(1) because we use a fixed number of integer variables (i, sign, res, flag, and n) and they do not depend on the size of the input string. No additional structures are allocated that would grow with the input size. The space used by s is not counted since it is an input to the function and not additional space allocated by the function itself.