https://betterprogramming.pub/coding-interview-question-convert-roman-numerals-to-decimals-868b2694f497

Coding Interview Question: Convert Roman Numerals to Decimals

Master this simple yet intimidating algorithm

[](https://blakesanie.medium.com/?source=post_page-----868b2694f497--------------------------------)

[Blake Sanie](https://blakesanie.medium.com/?source=post_page-----868b2694f497--------------------------------)

[Follow](https://medium.com/m/signin?actionUrl=https%3A%2F%2Fmedium.com%2F_%2Fsubscribe%2Fuser%2F8d1a76d48b32&operation=register&redirect=https%3A%2F%2Fbetterprogramming.pub%2Fcoding-interview-question-convert-roman-numerals-to-decimals-868b2694f497&user=Blake%20Sanie&userId=8d1a76d48b32&source=post_page-8d1a76d48b32----868b2694f497---------------------follow_byline-----------)

[Jan 4](https://betterprogramming.pub/coding-interview-question-convert-roman-numerals-to-decimals-868b2694f497?source=post_page-----868b2694f497--------------------------------) · 4 min read



Photo by [Thomas Bormans](https://unsplash.com/@thomasbormans?utm_source=medium&utm_medium=referral) on [Unsplash](https://unsplash.com/?utm_source=medium&utm_medium=referral).

I see this come up time and time again. Because the Roman numeral number system contains a tally-like, sometimes out-of-order layout, countless software engineers are intimidated when asked to programmatically convert from Roman numeral to decimal (base 10).

Anecdotally speaking, I’ve been given this task during coding interviews countless times, and it wasn’t until the third or fourth encounter that I confidently delivered an ace response.

Even though the question isn’t super complex, the stakes are high. This is typically meant to be a warm-up, with harder questions to follow. Therefore, it’s important that you do not immediately trip up and instead wow your interviewer with your analytical and technical skills.

If you’re in the market for a software engineering job, you’ll see this at some point — guaranteed. But don’t worry, I’ll walk you through a series of steps you can take to knock this question out of the park.

Foundation and Approach

Take some time to show your interviewer that you truly understand the nature of the problem. It is imperative that you explain how Roman numerals operate — especially how addition and subtraction can coexist within each numeral.

Remember that the basic numerals — not digits — once used by the Romans have tally values: I: 1, V: 5, X: 10, L: 50, C: 100, D: 500, M: 1000, and so on.

Then, explain in basic terms the process behind converting a number from Roman numeral to decimal. In case you’re unaware, this process goes as follows:

1. Traverse the Roman numeral string from left to right, investigating two adjacent characters at a time. If you really want, specify that the direction of iteration does not matter as long as the comparisons (next step) are implemented accordingly.
2. If the character on the left is of higher ranking (tally value) than the character on the right, subtract the tally at that position from the final sum. Otherwise, add it.
3. After the string traversal is complete, the final sum represents the numeral’s modern counterpart.

I’d even suggest white-boarding an example conversion. For instance, demonstrate how CMXCVIII yields the expression -100 + 1000 — 10 + 100 + 5 + 1 + 1 + 1 whose value is 998.

Building a foundation of understanding is crucial to your success. Oftentimes, your interviewer will even correct you if you’re straying toward the wrong path. There’s nothing your interviewer will hate more than you diving head-first into the coding without first investigating the problem at hand.

The Code

Remember that explanation (in words) of the algorithm? Time to translate it for the computer to understand.

First, define the tallies associated with each numeral for each numeral with a map using the given values:

You’ll also need to keep track of the numerals’ ongoing sum, starting at zero:

sum = 0

Now, iterate across the numeral string and extract the current pair of numerals. Be sure to iterate until the second-to-last numeral to prevent an out-of-bounds exception:

for i in range(len(romanNumeral) - 1):  
 left = romanNumeral[i]  
 right = romanNumeral[i + 1]

Make the pivotal comparison and add to/subtract from sum accordingly:

if tallies[left] < tallies[right]:  
 sum -= tallies[left]  
 else:  
 sum += tallies[left]

Note that once you exit the for loop, the right-most numeral is not counted. We will account for this by updating the sum immediately after the for loop. Since there is no numeral to its right, addition is the only possibility in this case:

sum += tallies[romanNumeral[-1]]

That’s pretty much it! Just wrap the code in a callable function, return sum, and the code is ready to be run against test cases. With our implementation, the edge case of “” (empty string) as an input still provides the appropriate output of 0:

Theory and Analysis

You can probably guess what comes next: Your interviewer asks, “What’s the Big-O?”

Since each character in the romanNumeral string must be traversed twice, the algorithm exhibits a best-case, average-case, and worst-case time complexity of O(n) or *linear time*. This is supported by the algorithm’s single for loop.

Your interviewer might also ask about *space complexity* or how memory usage grows with input size. Our function only uses memory to instantiate and update the variables sum, i, left, and right. All of these variables have one fixed-size instance regardless of the length of romanNumeral. Therefore, the algorithm’s space complexity is *constant* or O(1).

Since this solution is optimal, you can show off your skills even further by explaining why it’s optimal and what makes other methods inferior. To start, this solution processes each character once as a left numeral and once as a right numeral. Achieving a more efficient working solution is not possible. Also, we managed to build the function with a single loop, whereas solutions with multiple nested loops are held back by a time complexity above *linear time*. Stylistically, the code is readable, organized, and easy to follow. Pointing this out doesn’t hurt since the other side wants to make sure you understand real-world best practices.

Our Python implementation in its as-is form is faster than 92.08% of submissions to [LeetCode’s assessment](https://leetcode.com/problems/roman-to-integer/), with a memory usage below 72.29% of all submissions. I encourage you to make more improvements and have a stab at rising higher through these percentiles on your own.

tallies = {

'I': 1,

'V': 5,

'X': 10,

'L': 50,

'C': 100,

'D': 500,

'M': 1000,

# specify more numerals if you wish

}

def RomanNumeralToDecimal(romanNumeral):

sum = 0

for i in range(len(romanNumeral) - 1):

left = romanNumeral[i]

right = romanNumeral[i + 1]

if tallies[left] < tallies[right]:

sum -= tallies[left]

else:

sum += tallies[left]

sum += tallies[romanNumeral[-1]]

return sum