Topic Introduction

Welcome back, Interview Prepper! Today’s trio of problems all share a fascinating thread: **number system conversion**. Whether it’s ancient Romans or digital spreadsheets, humans have always found creative ways to represent numbers. Understanding how to convert between these systems is a classic interview topic—and a powerful mental tool.

Let’s first define the core concept: **number system conversion**. This is the process of changing a number from one form to another, like from decimal (our everyday base-10 system) to Roman numerals, or from spreadsheet columns (“AA”, “AB”, etc.) to numbers. The key is recognizing how each system encodes values, then applying the rules in reverse to convert back.

Why is this important?

- It checks your attention to detail.
- It tests your ability to generalize patterns and reverse them.
- It’s a great way to see how you handle edge cases.

Let’s warm up with a simple example: converting the decimal number 27 to base-2 (binary).
How does it work?

- Divide 27 by 2 repeatedly, keeping track of remainders:

27 / 2 = 13 remainder 1

13 / 2 = 6 remainder 1

6 / 2 = 3 remainder 0

3 / 2 = 1 remainder 1

1 / 2 = 0 remainder 1

- Write the remainders in reverse: 11011 (that’s 27 in binary).

When is this useful in interviews?

- When converting between numeral systems (Roman, binary, hexadecimal, etc.).
- When encoding and decoding (e.g., Excel columns, URL shorteners).
- When manipulating strings that represent numbers.

Today’s problems:

- Roman to Integer
- Integer to Roman
- Excel Sheet Column Number

Why are these grouped together?
They all require you to convert between different number representations. The Roman problems are about mapping between a “symbolic” numeral system and base-10, while the Excel column is a base-26 alphabetic system. Solving them will sharpen your ability to spot number patterns and implement conversion logic—core interview skills!

Problem 1: Roman to Integer

Roman to Integer

Problem Statement (rephrased):

Given a string representing a Roman numeral, convert it into its integer (decimal) value.

Example:

Input: `"MCMXCIV"`

Output: `1994`

How?

- M = 1000
- CM = 900 (C before M subtracts 100)
- XC = 90 (X before C subtracts 10)
- IV = 4 (I before V subtracts 1)

Total: 1000 + 900 + 90 + 4 = 1994

Thought Process:

Roman numerals have two main rules:

- If a symbol is followed by one of equal or lesser value, add its value.
- If a symbol is followed by a greater value, subtract its value.

Work through a few examples by hand to see the pattern.

Try yourself:

What does `"LVIII"` convert to?

Brute-force approach:

Scan each symbol, checking for subtraction cases (like IV, IX, etc.) using string matching. This gets the job done but is not elegant.

Optimal approach:

- Use a dictionary to map symbols to values.
- Iterate through the string:
    - If the current symbol is less than the next, subtract it.
    - Otherwise, add it.

Let's code it up:

```
def romanToInt(s):
    # Map Roman numerals to their integer values
    roman_map = {
        'I': 1,    'V': 5,    'X': 10,
        'L': 50,   'C': 100,  'D': 500,
        'M': 1000
    }
    total = 0
    prev_value = 0  # To store the value of the previous symbol

    # Iterate from right to left
    for char in reversed(s):
        value = roman_map[char]
        if value < prev_value:
            # Subtract if a smaller value comes before a larger one
            total -= value
        else:
            # Otherwise, add it
            total += value
        prev_value = value  # Update for next iteration

    return total
```

Time Complexity: O(n), where n is the length of the string

Space Complexity: O(1), as the mapping is fixed size

Code Explanation:

- We walk through the string from right to left.
- If the current symbol is less than the one to its right, we subtract; otherwise, we add.
- This neatly handles all subtractive cases (like IV, IX, etc.) without needing complex string matching.

Trace Example:

Input: `"MCMXCIV"`

- Start from 'V': 5 (added), prev=5
- Next 'I': 1 (1 < 5, so subtract) -> total = 5 - 1 = 4
- Next 'C': 100 (100 > 1, add) -> total = 4 + 100 = 104
- Next 'X': 10 (10 < 100, subtract) -> total = 104 - 10 = 94
- Next 'M': 1000 (1000 > 10, add) -> total = 94 + 1000 = 1094
- Next 'C': 100 (100 < 1000, subtract) -> total = 1094 - 100 = 994
- Next 'M': 1000 (add) -> total = 994 + 1000 = 1994

Test case for you:

Input: `"XLII"`

What’s the output? Try it on paper!

Take a moment to solve this on your own before jumping into the solution.

Problem 2: Integer to Roman

Integer to Roman

Problem Statement (rephrased):

Given an integer between 1 and 3999, convert it to its Roman numeral representation.

Similarities and Differences:

This is the reverse of the previous problem. Instead of decoding, you’re encoding numbers using Roman numeral rules.

Brute-force approach:

Try to build the numeral by repeatedly subtracting the largest Roman value that fits. But if you don’t handle the “subtractive” symbols (like IV, IX), your output will be incorrect.

Optimal approach:

- Use two arrays/lists: one for Roman symbols, one for their values.
- Start from the largest value and keep subtracting while adding the corresponding symbol to the result.
- Handle subtractive cases (like 900, 400, etc.) by including them in your value-symbol lists.

Step-by-step logic:

- Initialize a list of values and their corresponding Roman numerals, *including* subtractive forms.
- For each value (from largest to smallest):
    - While the input number is at least as big as the value:
        - Subtract the value from the number.
        - Append the symbol to the result string.

Example:

Input: `1994`

- 1000 (M): subtract once -> M, remaining: 994
- 900 (CM): subtract once -> CM, remaining: 94
- 90 (XC): subtract once -> XC, remaining: 4
- 4 (IV): subtract once -> IV, remaining: 0

Result: `"MCMXCIV"`

Another test case to dry-run:

Input: `58`

What’s the output?

Pseudocode:

```
Initialize values = [1000, 900, 500, 400, 100, 90, 50, 40, 10, 9, 5, 4, 1]
Initialize symbols = ["M", "CM", "D", "CD", "C", "XC", "L", "XL", "X", "IX", "V", "IV",
"I"]
Initialize result as empty string

For each value, symbol in values, symbols:
    While num >= value:
        Append symbol to result
        Subtract value from num

Return result
```

Step-by-step trace:

Input: `58`

- 50 (L): append 'L', num = 8
- 5 (V): append 'V', num = 3
- 1 (I): append 'I' three times, num = 0

Result: `"LVIII"`

Time Complexity: O(1) (since input is capped at 3999)

Space Complexity: O(1)

Test case for you:

Input: `44`

What’s the Roman numeral?

Problem 3: Excel Sheet Column Number

Excel Sheet Column Number

Problem Statement (rephrased):

Given a string representing an Excel sheet column title (like "AB"), return its corresponding column number.

What’s different here?

You’re converting from a *base-26* system (A=1, B=2, ..., Z=26, then AA=27, etc.) to decimal. Unlike Roman numerals, each letter is just a digit in this base-26 number system.

Brute-force approach:

You could try to build a mapping for every two-letter/three-letter combination, but this is inefficient and doesn’t scale.

Optimal approach:

- Think of the column title as a base-26 number.
- For each character, multiply the current result by 26, then add the letter's value (A=1, B=2, ..., Z=26).

Example:

Input: `"AB"`

- 'A' = 1
- 'B' = 2

So: (1 * 26) + 2 = 28

Another test case to try:

Input: `"ZY"`

What’s the output?

Pseudocode:

```
Initialize result = 0
For each character in the string:
    result = result * 26 + (char's value, where A=1, ..., Z=26)
Return result
```

Step-by-step trace:

Input: `"ZY"`

- 'Z' = 26 -> result = 0 * 26 + 26 = 26
- 'Y' = 25 -> result = 26 * 26 + 25 = 676 + 25 = 701

Time Complexity: O(n), n = length of the string

Space Complexity: O(1)

Test case for you:

Input: `"FX"`

What's the column number?

Hint: Look for the pattern. How would you implement this conversion in your favorite language?

Summary and Next Steps

Today, you tackled three classic **number system conversion** problems:

- Roman numerals to integers and back
- Excel column titles to numbers

Key patterns to remember:

- Map each symbol to its value.
- Scan and decide to add/multiply/subtract based on the system's rules.
- For base-N conversions, process each symbol as a digit in that base.

Common mistakes/traps:

- Forgetting to handle subtractive notation in Roman numerals (like IV, IX).
- Off-by-one errors in base-26 conversions (remember: A=1, not 0!).
- Not processing from the correct direction (sometimes right-to-left matters).

Action List:

- Solve all three problems on your own, even the one with code provided.
- Try re-solving Problem 2 and 3 using recursion or a different loop structure.
- Research other number system conversions (e.g., base-2, base-16).
- Compare your solution with others—especially how they handle edge cases.
- If you get stuck, break the problem into steps and try to solve a smaller case.

Keep practicing—mastering these patterns will make you a conversion pro in interviews and beyond!