Intuition

Define a hashmap of the mappings of the seven different romans symbols to their respective integers as well as the six extra edge cases. Decompose your given integer into parts that are equal to these 13 numbers that have their roman representation in the hashmap. Return the concatenation of the strings of the decomposed numbers.

Algorithm

```
roman_numerals = {1:"I",5:"V", 10:"X",50:"L",100:"C",
                  500:"D", 1000:"M",4:"IV",9:"IX",
                  40:"XL", 90:"XC", 400:"CD",900:"CM"}
```

We first define the aforementioned hashmap of key value pairs.

```
multiples = [1000,900,500,400,100,90,50,40,10,9,5,4,1]
roman = ""
while num > 0:
    for i in range(len(multiples)):
        if num >= multiples[i]:
            roman = roman + roman_numeral[multiples[i]]
            num -= multiples[i]
            break
return roman
```

The first two lines are defining data structures which we will be using in the algorithm.
The while loop is decomposing the number by iterating through the ordered list “multiples”. Each time we use one of the multiples as one of the decompositions, we find its corresponding roman representation and append that to the “roman” string we build up gradually. We then return the string as per the requirements.

Algorithm analysis

Optimising for time

The current run time is log(n) where n is the integer we are making roman. I have chosen to use a dictionary for its constant read time which is optimal in this case.

Optimising for space

The solution is also optimised for space as we are not creating unnecessary data structures. The only data structure which may be seen as redundant is the “multiples” list. We could have a single list of lists where the inner lists are of the following format [number,roman representation], however I think the code is more readable by using the method I have chosen.