1 – Explicacion:

Para explicar como funciona la conversion de un número entero a un número romano debemos tener en cuenta lo siguiente:

Ejemplo:
Tomemos el número 3893 y lo dividimos en múltiplos de 1000 hasta que el residuo sea menor que 1000.

En este caso, 3893 / 1000 = 3, y nos queda un residuo de 893.

Por lo tanto, escribimos "MMM" para representar el numero 3000 en números romanos.

Ahora tomamos el residuo que nos dio 893 y lo dividimos en múltiplos de 500 hasta que el residuo sea menor que 500. En este caso, no hay ningún múltiplo de 500 que divida a 893.

Continuamos entonces dividiendo el residuo (893) en múltiplos de 100 hasta que el residuo sea menor que 100.

Quedando asi entnces que 893 / 100 = 8 con un residuo de 93.

Por lo tanto, añadimos "CCC" para representar 300 en números romanos.

Nuevamente dividimos el residuoque nos dio 93 en múltiplos de 50 hasta que el residuo sea menor que 50.

Resulta entonces que 93 no es divisible por 50. Por lo que nos toca entonces volver a dividir el residuo 93 en múltiplos de 10 hasta que el residuo sea menor que 10.

93 / 10 = 9 con un residuo de 3.

Por lo tanto, añadimos "XC" para representar 90 en números romanos.

Ahora, dividimos el residuo 3 en múltiplos de 1.

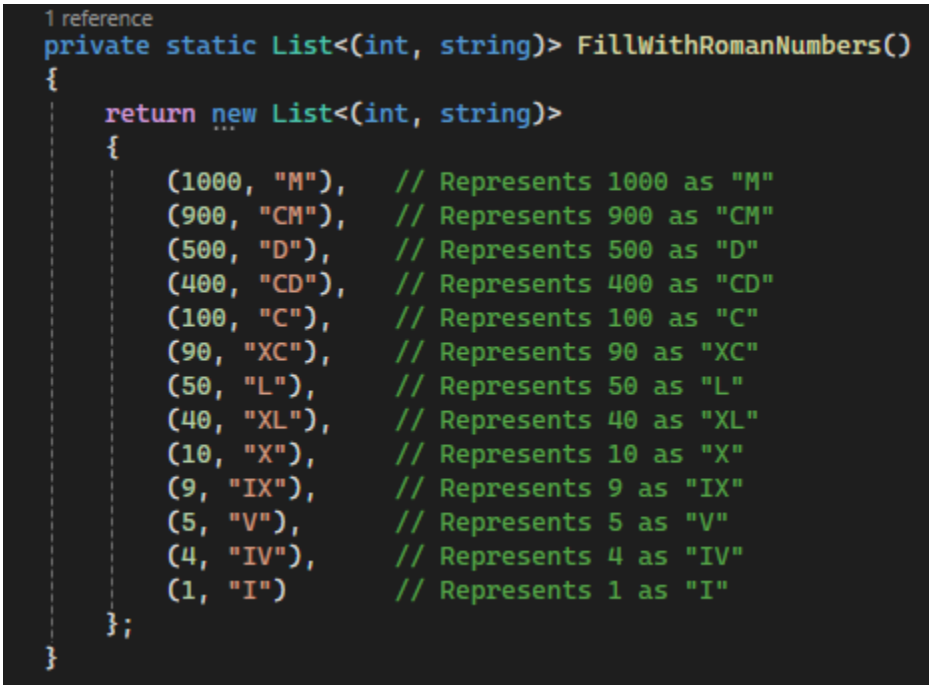
3 / 1 = 3.

Por lo tanto, añadimos "III" para representar 3 en números romanos.

Finalmente hemos terminado y si unimos todas las conversiones que hicimos nos queda que el número 3893 se representa como "MMMDCCCXCIII" en números romanos.

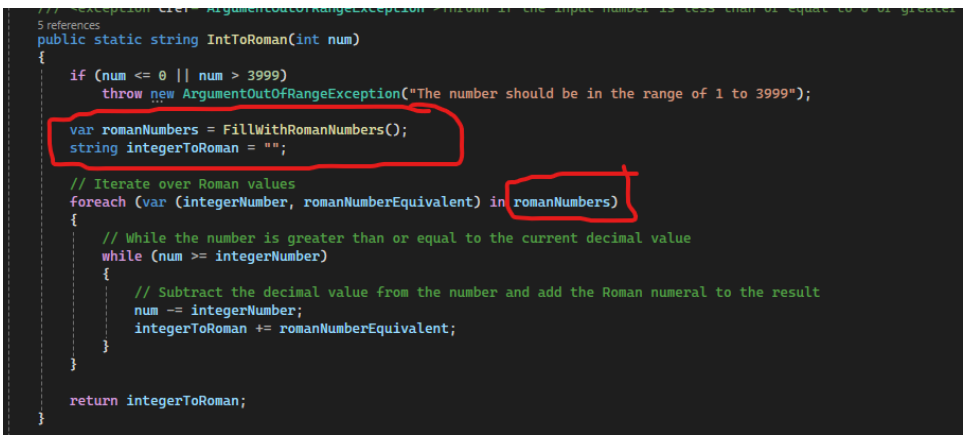
2 – Analisis de Compleidad Algortimica:

Definición de valores romanos: La definición de los valores romanos se hace a traves de una lista que almacena valores de tipo tupla (int, string) , es una lista fija de tamaño constante, que no depende del tamaño de la entrada. Por lo tanto, la complejidad de esta parte del algoritmo es O(1).



```
1 reference
private static List<(int, string)> FillWithRomanNumbers()
{
    return new List<(int, string)>
    {
        (1000, "M"),    // Represents 1000 as "M"
        (900, "CM"),    // Represents 900 as "CM"
        (500, "D"),     // Represents 500 as "D"
        (400, "CD"),    // Represents 400 as "CD"
        (100, "C"),     // Represents 100 as "C"
        (90, "XC"),     // Represents 90 as "XC"
        (50, "L"),      // Represents 50 as "L"
        (40, "XL"),     // Represents 40 as "XL"
        (10, "X"),      // Represents 10 as "X"
        (9, "IX"),      // Represents 9 as "IX"
        (5, "V"),       // Represents 5 as "V"
        (4, "IV"),      // Represents 4 as "IV"
        (1, "I")        // Represents 1 as "I"
    };
}
```

Iteración sobre los valores romanos: El algoritmo itera sobre la lista “romanNumbers” que se llena con los valores constantes “FillWithRomanNumbers” representando los numeros romanos para encontrar el numero romano correspondiente para cada dígito del número decimal. Esta iteración se realiza una cantidad constante de veces, independientemente del valor de n. Por lo tanto, la complejidad de esta parte también es O(1).



```
5 references
public static string IntToRoman(int num)
{
    if (num <= 0 || num > 3999)
        throw new ArgumentOutOfRangeException("The number should be in the range of 1 to 3999");

    var romanNumbers = FillWithRomanNumbers();
    string integerToRoman = "";

    // Iterate over Roman values
    foreach (var (integerNumber, romanNumberEquivalent) in romanNumbers)
    {
        // While the number is greater than or equal to the current decimal value
        while (num >= integerNumber)
        {
            // Subtract the decimal value from the number and add the Roman numeral to the result
            num -= integerNumber;
            integerToRoman += romanNumberEquivalent;
        }
    }

    return integerToRoman;
}
```

Bucle principal: El bucle principal itera sobre los valores romanos hasta que el número decimal se reduce a cero. En el peor caso, este bucle podría iterar hasta n veces, ya que el número decimal máximo que se puede representar en números romanos es 3999 (MMMCMXCIX). Por lo tanto, la complejidad de este bucle es O(n).

Construcción del resultado: Dentro del bucle principal, la construcción del resultado implica concatenar cadenas, lo cual puede tomar tiempo proporcional al tamaño del resultado. En el peor caso, la longitud del resultado podría ser tan grande como n (por ejemplo, para números como 3999). Por lo tanto, la complejidad de esta parte también es O(n).

En resumen, la complejidad total del algoritmo es la suma de las complejidades de las partes mencionadas anteriormente, lo que nos da una complejidad de O(1) + O(1) + O(n) + O(n) = O(n).