Assignment 1

Question 1

(a) Worst-Case Number of Symbols for Encoding in Roman Numerals

To start, the best-case number of symbols for encoding is a one-to-one mapping between the Roman numeral system and the decimal (base-10) system. For example, the decimal number 1 is represented by the Roman numeral I, meaning one symbol is required. This implies that, for a positive integer n with m symbols, the lower bound of the corresponding Roman numerals is also m symbols.

Looking at the upper bound, one can prove that the encoding is bounded by $O(m^2)$. There exists some positive real $F_m \leq c \cdot m^2$ for all m in $\mathbb{N}$. Proof by construction will be used, where some concrete c is proposed and shown that it works.

Trying some values of m, such that the number of symbols needed increases:

Decimal Number	# of Symbols	Roman Numeral	# of Symbols
1	1	I	1
99	2	$XCIX$	4
999	3	$CMXCIX$	6
9999	4	$\overline{IX}CMXCIX$	9
99999	5	$\overline{XCIX}CMXCIX$	11

Seeing what c would work:

$$m = 1, F_1 = 1, 1^2 = 1 \rightarrow c = 1\ works$$
$$m = 2, F_2 = 4, 2^2 = 4 \rightarrow c = 1\ works$$
$$m = 3, F_3 = 6, 3^2 = 9 \rightarrow c = 1\ works$$
$$m = 4, F_4 = 9, 4^2 = 16 \rightarrow c = 1\ works$$
$$m = 5, F_5 = 11, 5^2 = 25 \rightarrow c = 1\ works$$

It appears that $c = 1$ works. Thus, this chosen c will be adopted and checked with a proof by induction:

- Base case: $m = 1, F_1 = 1, 1^2 = 1, 1 \leq 1 \rightarrow True$
- Step: Seek to show $F_m \leq m^2$ given that $F_k \leq k^2$ for all $k = 1, 2, \ldots, m - 1$
- Assume true for $m = k$, show true for $m = k + 1$
- $F_k = k^2, F_{k+1} = (k+1)^2, F_k \leq F_{k+1} \rightarrow Done$

Thus, the upper bound is $O(m^2)$. Furthermore, analyzing the examples used in the table above, it can be seen that the number of symbols needed for the decimal numbers, squared, are all $\leq$ the number of symbols needed for their Roman numeral equivalents. Therefore, the tight bound for this encoding is $\Theta(m^2)$.

(b) Worst-Case Number of Symbols for Encoding in Decimal System

The worst-case number of symbols to encode a positive integer in decimal notation is $\Theta(m)$. For every digit in the positive integer n, the decimal system allows each to be represented by a symbol from $1 - 9$. First prove that $F_m = \Omega(m)$:

- By trial and error: It appears that $F(m) \geq m$ for all $m \geq 1$
- To prove: For all positive integers $m \geq 1 \rightarrow F_m \geq m$
- By induction on m. Base case: $m = 1, F_m = 1, 1 \geq 1 \rightarrow True$
- Step, assume: Indeed, true that for all $m = 1, 2, \ldots, k \rightarrow F_k \geq k$
- To prove: $F_{k+1} \geq k + 1$
- Indeed: $k + 1 \geq k \rightarrow Done$

Thus, the lower bound is $\Omega(m)$. Now for the upper bound, a similar proof from part (a) can be used to prove a bound of $O(m)$. Using $c = 1$ and a proof by induction on m:

- Base case: $m = 1, F_1 = 1, 1 \leq 1 \rightarrow True$
- Step: Seek to show $F_m \leq m$ given that $F_k \leq k$ for all $k = 1, 2, \ldots, m - 1$
- Assume true for $m = k$, show true for $m = k + 1$
- $F_k = k, F_{k+1} = k + 1, F_k \leq F_{k+1} \rightarrow Done$

Thus, the upper bound is $O(m)$. Therefore, the tight bound for this encoding is $\Theta(m)$. Comparing the Roman numeral system to the decimal system, the decimal encoding is better than the Roman numeral encoding from part (a). It is more efficient with respect to the number of symbols used, as it can represent these integers with the same or less symbols.

Assignment 1

Question 2

The goal for this problem is to prove that the Harmonic Series, namely $\sum_{i=1}^{n} \frac{1}{i}$, has an upper bound of $O(\log n)$. First, let $n = 2^k$ to exponentiate and work with a proof involving log. The Harmonic Series then becomes:

$$\sum_{i=1}^{2^k} \frac{1}{i} = H_{2^k}$$

Consequently, this series can be expanded as follows:

$$H_{2^k} = 1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \frac{1}{5} + \frac{1}{6} + \frac{1}{7} + \cdots + \frac{1}{2^{k-1}} + \frac{1}{2^{k-1}+1} + \cdots + \frac{1}{2^k - 1}$$

To carry the proof through, group the terms by powers of two and compare, similar to how divergence is proven for the Harmonic Series [1]. Thus, the grouping is shown as:

$$H_{2^k} = (1) + \left(\frac{1}{2} + \frac{1}{3}\right) + \left(\frac{1}{4} + \frac{1}{5} + \frac{1}{6} + \frac{1}{7}\right) + \cdots + \left(\frac{1}{2^{k-1}} + \frac{1}{2^{k-1}+1} + \cdots + \frac{1}{2^k - 1}\right)$$

There is also one extra $\frac{1}{2^k}$ term in the series. Then, a valid upper bound can be generated by using this strategy of increasing powers of two and applying it to the grouped terms as follows:

$$U_{2^k} = (1) + \left(\frac{1}{2} + \frac{1}{2}\right) + \left(\frac{1}{4} + \frac{1}{4} + \frac{1}{4} + \frac{1}{4}\right) + \cdots + \left(\frac{1}{2^{k-1}} + \frac{1}{2^{k-1}} + \cdots + \frac{1}{2^{k-1}}\right)$$

Clearly, $H_{2^k} \leq U_{2^k}$, where U_{2^k} represents the desired upper bound in question. Now, note that each grouped set of terms adds up to 1, therefore all the grouped terms added together equals k. This, along with the extra term, leads to the following inequality:

$$H_{2^k} \leq k + \frac{1}{2^k}$$

Subbing back in $n = 2^k$ leads to the following equation for k:

$$\log n = \log 2^k = k \cdot \log 2$$

$$k = \frac{\log n}{\log 2} = \log_2 n$$

Assignment 1

Using the hint of adopting the exact form of $\log_b n + c$, together with the power of two grouping, leads naturally to use $b = 2$ and $c = 1$. Finally, this results in the following inequality:

$$H_n \leq \log_2 n + 1$$

With this result, the value of $c = 1$ is an additive constant, meaning it can be omitted from the big O notation. This simplifies the inequality:

$$H_n \leq \log_2 n$$

Therefore, the final upper bound of the Harmonic Series is $O(\log n)$. In mathematical form, this means that:

$$\sum_{i=1}^{n} \frac{1}{i} = O(\log n)$$

This concludes that the upper bound of the Harmonic Series is indeed $O(\log n)$.

Assignment 1

Question 3

(a) Proof of Recurrence Correctness

The recurrence can be proven by adopting a case analysis, meaning tackling each condition one at a time. More specifically, for every case, one can use logical deduction, and if each case individually is proven to be correct, then the entire gcd (a, b) algorithm itself is correct. There are six conditions to check, with the first two acting as the base cases. These first two cases can be invoked as follows:

$$\text{gcd } (a, 0) \text{ and gcd } (0, b)$$

In other words, if one input is a 0, it does not matter what the other input value is (of course omitting simultaneous 0), the function will always return the other value. This is correct and can be proven by looking at the definition of the greatest common denominator, which is the largest positive number that divides into both numbers without a remainder. Since no number can divide into 0, then it can only return the other input value.

The remaining four cases compose of the recursive step and can be proven separately. To prove the case if both a, b are even:

- Let $g = \text{gcd } (a/2, b/2)$
- Then, g divides both $a/2$ and $b/2$
- Therefore, $2g$ divides both a and b
- Therefore, $\text{gcd}(a, b) \geq 2g$
- Similarly, a and b must be $\geq g$
- However, a and b must not be $> 2g$
- Therefore, $\text{gcd}(a, b) \leq 2g$

Since $\text{gcd}(a, b) \geq 2g$ and $\text{gcd}(a, b) \leq 2g$ are both proven, then $\text{gcd}(a, b) = 2g$, and case three is correct. Case four is if a is odd and b is even:

- Let $g = \text{gcd } (a, b/2)$
- Then, g divides both a and $b/2$
- Therefore, $2g$ divides both $2a$ and b
- Therefore, $\text{gcd}(a, 2b) \geq \text{gcd } (a, b)$ (2 is not a common divisor)

- Similarly, $2a$ and b must be $\geq$ gcd (a, b)
- However, $2a$ and b must not be $>$ gcd (a, b)
- Therefore, $\gcd(a, 2b) \leq$ gcd (a, b)

Since $\gcd(a, 2b) \geq$ gcd (a, b) and $\gcd(a, 2b) \leq$ gcd (a, b) are both proven, then $\gcd(a, 2b) =$ gcd (a, b), and case four is correct. Similarly, case five is if a is even and b is odd:

- Let $g =$ gcd $(a/2, b)$
- Then, g divides both $a/2$ and b
- Therefore, $2g$ divides both a and $2b$
- Therefore, $\gcd(2a, b) \geq$ gcd (a, b) (2 is not a common divisor)
- Similarly, a and $2b$ must be $\geq$ gcd (a, b)
- However, a and $2b$ must not be $>$ gcd (a, b)
- Therefore, $\gcd(2a, b) \leq$ gcd (a, b)

Since $\gcd(2a, b) \geq$ gcd (a, b) and $\gcd(2a, b) \leq$ gcd (a, b) are both proven, then $\gcd(2a, b) =$ gcd (a, b), and case five is correct. The final case six is if a and b are both odd. An important distinction here is that since both a and b are odd, then the result of $a - b$ will always be even [2]. Hence, the result can be divided by 2 since all even numbers are divisible by 2, which is done to all even inputs in the other cases. The absolute value is there to ensure the algorithm stays within positive bounds, since $a, b \in \mathbb{Z}_0^+$. As such:

- If both a, b are odd, then $a - b$ is even
- Since $|a - b|$ is even, then $|a - b| <$ max (a, b)

This says that, comparing two numbers with two numbers subtracted, the maximum of the two numbers will always be greater, which is true. Similarly, the second input $\min(a, b)$ is valid because the greater of the two numbers can never be a common factor. This can be traced back again to the definition, which says that the factor must divide evenly into each of the integers. This can be proven by contradiction: if it was instead $\max(a, b)$, then it could result in the factor being larger than the two numbers, which is invalid when determining the greatest common denominator. This finishes the proof for the sixth case, and since all six cases are proven, this completes the proof for the recurrence.

(b) Worst-Case Time Efficiency

First, one must identify what n is in this case, where it is defined as the size of the input $\langle a, b \rangle$ of the recursive algorithm. More specifically, the inputs are a pair of non-negative integers a and b. So, to encode these two numbers (i.e., write them out), it would take $\Theta(\log a + \log b)$ symbols, which is the same as $\Theta(\log(\max\{a, b\}))$, so this is n. The recurrence contains six cases, however the first two are the base cases, so only the last four recursive cases will be analyzed to determine the time efficiency. In these cases, one can see the following operations present: subtraction, multiplication, and division.

In the course notes, the following was deduced for bitwise addition/subtraction:

- # digits needed to encode $x \epsilon Z^+ = \lfloor \log_{10} x \rfloor + 1$
- # bits needed to encode $x \epsilon Z^+ = \lfloor \log_2 x \rfloor + 1$

So, time efficiency to add/subtract $x, y \epsilon Z^+$ as measured by number of lookups:

- $1 + \lfloor \log_{10}(\max\{x, y\}) \rfloor$ in the best case
- $2 + \lfloor \log_{10}(\max\{x, y\}) \rfloor$ in the worst case

So, either way, $\Theta(n)$, or linear time, where n is the size of the input. This means that the subtraction operation takes $\mathrm{O}(n)$ time. As for multiplication and division, an important distinction can be made regarding its use. Across the cases, the results are either multiplied by 2 or divided by 2 only, which are equivalent to bit shifting left and bit shifting right, respectively. Thus, these operations also take $O(n)$ time in the worst-case for n-sized inputs.

However, one must also consider the number of recursive calls the algorithm may take in a worst-case scenario. In the worst-case, if n is the number of bits of the larger number, the algorithm would have to step through the entirety of its bits, which would take $O(n)$ time.

In summary, the number of recursive calls in the worst-case is $O(n)$, and for each recursion, the algorithm would have to step through n-bits of the larger number in order to determine the greatest common denominator, which would also take $O(n)$ time. Therefore, the worst-case time efficiency of this function is $O(n^2)$.

Assignment 1

Question 4

- First enunciate a correctness property for $FactTwo$ that assumes only that $lo \leq hi$ and $lo, hi \in \{1, 2, \ldots, n\}$
- But doesn't demand that $lo = 1$ nor that $hi = n$

Question 5

- We can infer things based on properties of mod
- E.g., since:

$$a \times a \ mod \ p = (a \ mod \ p) \times (a \ mod \ p) \ mod \ p$$

- Then we know that:

$$a^b \ mod \ p = (a \ mod \ p)^b \ mod \ p$$

References

[1] E. W. Weisstein, "Harmonic Series," Wolfram MathWorld, [Online]. Available: https://mathworld.wolfram.com/HarmonicSeries.html. [Accessed 23 January 2021].

[2] ProofWiki, "Odd Number minus Odd Number is Even," [Online]. Available: https://proofwiki.org/wiki/Odd_Number_minus_Odd_Number_is_Even. [Accessed 24 January 2021].