CSCI 433 Midterm Exam 1 Spring 2023 February 9, 2024
Problem 1 (10 pts)

1. True False

   $$IF, f(n) \in \Theta(g(n)), then f(n) + g(n) \in \Theta(g(n))$$

   True
   Because if f(n) is an element of Theta(g(n)), it must contain the time of f(n) + g(n), because adding the two together would only result in something like 2n, which when removing the constant from the equation, is simply n.
2. True False

   $$n^p \in \Omega(logn)$$

   for any constant p > 0.
   True
   Because in cases for Sigma, we consider the best case bound for the problem. The best case for such a problem as described above would be contained within Sigma(log n) because there could be scenarios where the p is a lower number and does not take a large amount of time.
3. True False

   $$n^2 \in \Theta(nlogn)$$

   False
   Because n^2 is not going to be equal to n log n, as the second term in n log n, will necessarily be less than n. This means that the two are not equal, so n^2 is not an element of Theta(n log n). In order for it to be an element of the theta argument, it must be equal or approaching equal.
4. True False

   $$n! \in \Omega(2^n)$$

   True
   Because performing n factorial operations takes less time than performing 2 to the n operations. So n! is within the set of times covered by Sigma 2 to the n.
5. True False

   $$n^3 + n^2 logn \in \Theta(n^3)$$

   True
   Because n to the 3rd is our most impactful term, and since it is so much larger than the other terms present within the function, we can ignore them. Thus, we can consider the term as an element of Theta n to the 3rd, because the two terms are effectively equal.
6. True False
   The worst-case time complexity of Euclid's algorithm is O(log n) where n is the larger of the two input integers.
   True
   Because Euclid's algorithm will never go through all the terms in order to find it, but may go through a section of the terms equal to log n, in the worst cases. It can also take only constant time.
7. True False
   For any two functions f and g, we always have

   $$f \in \Omega(g)//or//g \in \Omega(f)$$

   False
   Because you could have something like sin and cos waves, meaning that the outcomes could be variable.
8. True False
   Asymptotically, an O(log n) algorithm is faster than an O(n) algorithm

True
Because asymptotically is for values at upper range, log n will always be faster than n, because log n is less than n. In other cases, where n is very small, this could be false.

9. True False
   The time complexity of dequeue in a queue containing n elements implemented using an array is Θ(1).
   True
   Because we can implement a circular array using modulo
10. True False
    A Θ(n) algorithm always runs faster than a Θ(n2) algorithm.
    False
    Because there may be some cases where the n^2 algorithm is very small, and the n algorithm is larger, meaning that the n^2 algorithm would run faster. Were it most cases, this would be true, but because it is always, this is False.

Problem 2 (10 pts)

Find the time complexity of the following sorting algorithm in terms of the number of comparisons. Show your work and express the final result in Θ notation.

```
Algorithm 1 ExchangeSort($A[0, \dots, n-1]$)
for $i = 0, \dots, n-2$ do
    for $j = i+1, \dots, n-1$ do
        if $A[i] > A[j]$ then
            swap($A[i], A[j]$)
        end if
    end for
end for
```

$$\sum_{i=0}^{n-2}\sum_{j=i+1}^{n-1} 1 = \sum_{i=0}^{n-2} n-1-(i+1)+1 = \sum_{i=0}^{n-2}(n-i-1) = \frac{n(n-1)}{2} \in \theta(n^2)$$

Method

Each for loop is a summation. Because it is a nested for loop we make a nested summation. The top of the summation is the top term, base case the bottom term. Set it equal to 1, because it's a Theta comparison. Top term minus bottom term the inner summation. Add +1 to the end because we count up from 0. Resolve it and get the next term. Top term in outer summation multiplies with the inner expression, divide it by 2 because it's two summations. Set i = 0. Final term, show that final term is an element of Theta n^2 because n^2 is the most significant term in our final expression.

Problem 3 (10 pts)

Buggy Cody has recently switched majors to computer science. He just learned recursion and was excited to use it whenever possible. Below is a pseudo code he wrote to find if a string ($S1$) is contained in another string ($S2$) using recursion. After turning it into a Java program, he found that it produced correct results, but ran extremely slow for long strings $S2$. Let the length of $S1$ and $S2$ be m and n, respectively, i.e., $S1$.length() = m and $S2$.length() = n. The syntaxes S(1 : end − 1) and S(2 : end) represent the strings with the last character and first character removed from S, respectively. strcmp($S1$, $S2$) returns true if the string $S1$ is identical to string $S2$, and false otherwise. Please help Mr. Cody identify his algorithm's time complexity in terms of the number of strcmp() operations.

```
Algorithm 2 isSubstring($S_1$,$S_2$)
if $S_1$.length() > $S_2$.length() then
    return false
else
    if $S_1$.length() == $S_2$.length() then
        return strcmp($S_1$, $S_2$)
    else
        return isSubstring($S_1$, $S_2$(2 : end)) OR isSubstring($S_1$, $S_2$(1 : end − 1))
    end if
end if
```

Let N = n-m

T(N) =2T(N-1)

T(0) = 1

$$T(N) = 2T(N-1) = 2^2T(N-2) = 2^3T(N-3)... = 2^NT(0) = 2^N$$

Method

Big N represents the way we go through the strings, which is n times, minus the length of m, our second string. T(N) represents overall time complexity. We can traverse our time complexity terms over, and over again, finding n-1,2,3...all the way till n-n which is 0. That cancels, and leaves us with 2^N, which represents how many comparisons we've made overall.

Bonus rewrite

Alg isSubstring(S1, S2)

if S_1.len() > S_2.len()

return false

else

if strcmp(S_1, S_2(1:len()) == true

return true

else

return isSubstring(S_1, S_2(2:end))