Archie Ross
2/1/2023

CSE 3500

1. Matrix Multiplication
    a. For an n by n matrix, in order to iterate through, you must create nested loops. You have to use these loops to iterate through each row and each column to get each individual item. Generally, this kind of algorithm will take a few operations per item in the matrix since we have to iterate a variable in the loop to check the indices. So there will be one addition for the row, one addition for each column and then there will be the multiplication itself taking place. I believe there will be 3n operations taking place.
2. Time Analysis
    a. Counting the number of time each line executes:
        i. Line 1 is a variable being set and is ran once
        ii. Line two is a for loop that is executed n times
        iii. Line three is another for loop that is executed n times
        iv. The result is edited n times
        v. This line just closes the loop
        vi. This line just closes the loop
    b. Big O running time
        i. O(1)
        ii. O(n)
        iii. O(n)
        iv. O(n)
        v. O(1)
        vi. O(1)
3. Asymptotic Notations Practice
    a. Group 1 (A being the fastest, D being the slowest): In this group I converted each to big O notation and could easily tell which was the slowest and which was the fastest.
        i. O(n) (10000000n)
        ii. O(n*log*(n)) ($n^{0,9999999}$Log n)
        iii. O(n^2) (n^2)
        iv. O(1.000001^n) (1.000001^n)
    b. Group 2 (A being the fastest, D being the slowest): I did the same thing in this group as the one above.
        i. O(1) ($2^{2^{\wedge}1000000}$)
        ii. O($n * n^{1/2}$) (n sqrt(n))
        iii. O(n^2) (n matrix 2)
        iv. O(2^n) ($2^{100000n}$)
    c. Show $8n^3\log(n) + 14n^2 = \theta(n^3\log(n))$
        i. We can start by ignoring the constants to get $n^3\log(n)+n^2$

ii. We can also ignore the n^2 since the equation already contains n^3 which has a run time that will always be larger than n^2
iii. That leaves us with $n^3\log(n)$

d. If $f(n) = O(g(n))$, can we conclude if $2^{f(n)} = O(2^{g(n)})$?
   i. We cannot conclude this statement because of this example:
      1. If $f(n) = 2\log(n)$ and $g(n) = \log(n)$ we can conclude that $2\log(n)$ is $<= c\log(n)$ which proves $f(n) = O(g(n))$ however… $2^{\log(n)\,\hat{}\,2} = f(n)$ and $2^{\log(n)} = g(n)$ do not have the same running time. $f(n)$ would be $O(n^2)$ where $g(n)$ would be $O(n)$, and these are not the same.

4. Sum of two numbers
   a. Assuming A is a sorted array (if not, then sort the array first), we can create an algorithm that iterates from the beginning of the array and the end of the array testing if the smaller element and the larger element combine to create the result given by the input. When doing so, we will test if the sum of the current iteration of elements produces a result larger or smaller than u. If the result is less than u, we will iterate the smaller side onwards and if the result is still smaller, we will iterate the larger side. If the result is greater than u, it's likely that the result cannot be found in the array of numbers that we have and we will end the algorithm.
   b. This will avoid many obvious combinations that don't result in our input u.
   c. The run time of this algorithm would be O(n) since we're not iterating through an index more than once.