Sean Hall 4-18-24

1a. 2^(n+1) = 2^n * 2^1. In big O notation, the constant gets ignored. Thus 2^(n+1) is in O(2^n).

1b. 2^2^n = 2^(2n) = (2^n)^2 = 2^n * 2^n. Thus 2^2^n is (2^n) squared which is NOT in O(2^n).

1c. n^1.01 grows at a rate slightly faster than O(n). (log^(n))^2 grows logarithmically, which grows slower than O(n). Thus, n^1.01 is NOT in O((log(n))^2).

1d. n^1.01 is in omega((log(n))^2). The omega notation describes the lower bound on growth. Because n^1.01 grows slightly faster than a linear rate, (log(n))^2 grows logarithmically which is significantly slower than linear. Even though the logarithmic function is squared, it still will never overcome linear.

1e. sqrt(n) is in O((log(n))^3). A normal logarithmic function (to the power 1), will grow slower than sqrt(n). However, because this logarithmic function is cubed, it grows faster than sqrt(n). This can be verified by graphing. Thus, log(n))^3 is above sqrt(n) as n approaches infinity.

1f. Using the same logic from 1e, log(n))^3 grows much faster than sqrt(n); Thus it CANNOT be in the lower bound of log(n))^3

2b. This function takes in an index as its argument and outputs the number at that index in the fibonacci sequence.
It does this by returning x if x is less than or equal to 1. If the index is 0, the first number in the sequence is 0. If x = 1, the 1st index of the sequence is 1.
If the index is greater than 1, two variables get created, ra and rb.
Ra contains the value of foo(x-1) which is a recursive call to the function at the index x-1.
Rb contains the value of foo(x-2) which is a recursive call to the function at the index x-2.
Then the function returns ra + rb which is the sum of the two.
Thus, the function can create the fibonacci sequence recursively.

3b. The work is O(n) because the time grows linearly with the input due to the functions iterative nature. Because the function is iterative, each iteration (excluding the first), relies on the previous one so this must be done by a single processor. Thus, the span is also O(n).

3d. If the algorithm is done sequentially, the work will be O(n*logn) and the span will be O(logn).

3e. If multiple processors are allowed to work on this algorithm, the new work is O(n) and the span will be O(logn).