Elijah Chou

CS 326

Dr. Michelangelo Grigni

1/22/2022

HW 1

1. (a): Θ(lg(lg(n))  
   (b): The loop invariant i = 2^2^iterations is true initially and is maintained by each iteration. The final value of iterations is the smallest value of k ≥ 0 such that 2^2^k > n. For n ≥ 2, this means that k > lg(lg(n)). To find the smallest value of k, lg(lg(n)) < k ≤ (lg(lg(n)) + 1. This determines k exactly. Since lg(lg(n)) -> ∞ as n -> ∞, this shows that k = Θ(lg(lg(n)).
2. (a): r \* b^i = a^n  
   (b): If r = 1, b = a, and i = n at first, then r \* b^i = 1 \* a^n = a^n  
   (c): In considering the first iteration, let i, r, and b be values before this iteration that satisfy r \* b^i = a^n. Then, let i’, r’, and b’ be values after this iteration, which should satisfy the condition that r’ \* b’^i’ = a^n. If i is an integer and i ≥ 1, two cases should be considered:  
     
   When case i is even:  
   r’ = r, b’ = b^2, i’ = i/2. So r’ \* b’^i’ = r \* (b^2)^i/2 = r \* b^i = a^n.  
     
   When case is odd:  
   r’ = r \* b, b’ = b, i’ = i – 1. So r’ \* b’^i’ = r \* b \* b^i-1 = r\*b^i = a^n.  
     
   (d): When exiting the loop, i = 0. Invariant equation tells us that a^n = r\*b^i = r\*b^0 = r, which makes it return a^n.  
     
   (e): The variable i is cut in half after every 2 iterations. There will be a maximum of 2k iterations since i ≤ n/(2^k). Since i is an integer, it is finished when k = lg(n).
3. 1/n  
   42  
   lg(lg(n))  
   lg(n^2), lg(n lg n)  
   (lg n)^2  
   n^(1/3)  
   sqrt(n), (sqrt(2))^(lg(n))  
   n  
   log10(n^n), lg(n!)  
   n^2, 9^\*log3(n)  
   4^(sqrt(n))  
   (3/2)^n  
   2^n  
   2^(2n)  
   n!  
   n^n
4. (a): If f(n) = 2n and g(n) = n, then f(n) ≤ 2\*g(n), so f(n) = O(g(n)), but 2^(f(n))/2(g(n)) = 2^n -> ∞, so 2^(f(n)) is NOT O(2^(g(n))).  
     
   (b): If f(n) = 1/n, then f(n)/f^2(n) = n -> ∞, so f(n) is NOT O(f^2(n)).  
     
   (c): If f(n) = { n^2 if n is even,  
    2n^2 if n is odd }  
   Then n^2 ≤ f(n) ≤ 2n^2, so f(n) = Θ(n^2), but f(n) > f(n+1) for odd n ≥ 3, so f(n) is not asymptotically non-decreasing.
5. (a): The space allotted for merge between the L and R arrays is proportional to the time spent in the merge such that Θ(n) in which n is equal to the current subarray size. This means that A(n) is proportional to the total time of T(n), which is Θ(n lg(n)).  
     
   (b): We only need enough space for the last merge since the memory is reusable, which is because before starting the next merge the previous one is finished first. This means that S(n) = Θ(n).