ALGORITHMS, FALL 2019, HOMEWORK 2

– This assignment is worth 2 units.
– Due on Thursday, September 19, at noon.

1. $S(n)=3S(^{n}_{3})+b$.

(a) Use a recursion tree to show that $S(n) = \Theta(g(n))$, for some function $g(n)$.
(b) Use induction to prove that $S(n) = \Omega(g(n))$.
(c) Do not submit, this is just for practice: get a matching upper bound by induction.

2. $T(n) = T(n-1) + \log n$. Assume that there is an appropriate base case so that you

don't get an undefined function.

(a) Draw a recursion tree for T (n). Identify its depth and the amount of work per level. Use "exaggerate and simplify" to get an upper bound for T (n), and then underestimate and simplify to get a matching lower bound. (Always using Theta notation).

(b) Derive the same upper bound for T(n), by induction.
(c) Do not submit, this is just for practice: get a matching lower bound by induction.

3. Solve $T(n)=T(\sqrt{n})+1$, using a change of variables: $n=2^m$.

To warm up, consider the following question. (No need to submit an answer)

Think of an abstract recursive algorithm that operates on a n × n matrix: It does

non-recursive work proportional to the number of elements in the matrix (say just n^2

work), and then recurses on each of the four quadrants.

This could be expressed as $T(n) = 4T(\frac{n}{2}) + n^2$, meaning the parameter used for the

problem size in T is the matrix dimension. Another option is to express the time

complexity as $T(n^2) = 4T(\frac{n^2}{4}) + n^2$. Here, the parameter used for the problem size

is the total number of elements. In this case, because this parameter is a function

of n (other than just n itself), we could set $m = n^2$, define $S(m) = T(n^2)$, and get $S(m) = 4S(\frac{m}{4}) + m$.

What's the solution in each case (in terms of n)? It should be the same.