COMP 3711H Design and Analysis of Algorithms
2015 Fall Semester
Homework 1
Handed out: Sep 16
Due: Sep 30

Assignments should be submitted to the collection box located outside room 4210A in the lab area.

Problem 1. Consider the following recurrence relation for the running time $T(n)$ of a divide and conquer algorithm:

$$\begin{aligned} T(1) &= 1 \\ T(n) &= 6\,T(n/2) + n^2 \qquad \text{if } n > 1. \end{aligned}$$

Assume that n is a power of 2. Answer the following questions regarding the recursion tree for this recurrence.

(a) Recall that there is a subproblem associated with each node of the recursion tree. How many subproblems are there at level i of the recursion tree? (Recall that the *root* is assumed to be at level 0.)

(b) What is the size of each subproblem at level i of the recursion tree?

(c) How much work is needed for the *combine* part for each subproblem at level i (*note: you must ignore the work done during the recursive calls*)?

(d) What is the work done summed over all the subproblems at level i (*again, you must ignore the work done during the recursive calls*)?

(e) How many levels are there in the recursion tree?

(f) Give a good asymptotic upper bound on the total work done summed over all the subproblems in the recursion tree. (*In other words, you need to give a good upper bound on $T(n)$. You should try to express your answer in the form of n raised to a suitable power.*)

Problem 2. Do parts (a)-(f) of Problem 1, for the following recurrence relation:

$$\begin{aligned} T(1) &= 1 \\ T(n) &= 3\,T(n/2) + n^2 \qquad \text{if } n > 1. \end{aligned}$$

Assume that n is a power of 2.

Problem 3. Consider again the recurrence relations given in Problems 1 and 2. In each case, establish an asymptotic upper bound for $T(n)$, using the *method of mathematical induction*. Make your bounds as tight as possible. You may assume that n is a power of 2.

Problem 4. Assume you have non-negative functions f and g such that $f(n)$ is $O(g(n))$. For each of the following statements, decide whether you think it is true or false and give a proof or counterexample.

(a) $2^{f(n)}$ is $O(2^{g(n)})$.

(b) $(f(n))^3$ is $O((g(n))^3)$.

Problem 5. Give asymptotic upper bounds for $T(n)$. Make your bounds as tight as possible. In each case, you may assume that $T(1) = 1$ and n is a power of 2. *Just give the answers; no explanation is needed.*

(a) $T(n) = 4\,T(n/2) + n, \qquad \text{if } n > 1.$

(b) $T(n) = 7\,T(n/2) + n^2, \qquad \text{if } n > 1.$

(c) $T(n) = T(n-1) + n^2, \qquad \text{if } n > 1.$

(d) $T(n) = T(n/2) + 10, \qquad \text{if } n > 1.$

(e) $T(n) = 3\,T(n-1) + 1, \qquad \text{if } n > 1.$

Problem 6. Arrange the following running times in order of increasing asymptotic complexity:

$$n^{4.5},\ \ \sqrt{2n},\ \ n^2+10,\ \ \log^2 n,\ \ 11^n,\ \ 5^n,\ \ n^2\log n$$

Note that you must write function $f(n)$ *before* function $g(n)$ if $f(n) = O(g(n))$. *Just give the answer; no explanation is needed.*

Problem 7. Arrange the following running times in order of increasing asymptotic complexity:

$$2^{n^2},\ \ 3^n,\ \ n^{4/3},\ \ n\log n,\ \ n^{\log n},\ \ 2^{2^n},\ \ 2^{\sqrt{\log n}}$$

Note that you must write function $f(n)$ *before* function $g(n)$ if $f(n) = O(g(n))$. *Just give the answer; no explanation is needed.*

Problem 8. You are doing some stress-testing on various models of glass jars to determine the height from which they can be dropped and still not break. The setup for this experiment, on a particular type of jar, is as follows. You have a ladder with n rungs, and you want to find the highest rung from which you can drop a copy of the jar and not have it break. We call this the *highest safe rung*.

It might be natural to try binary search: drop a jar from the middle rung, see if it breaks, and then recursively try from rung $n/4$ or $3n/4$ depending on the outcome. But this has the drawback that you could break a lot of jars in finding the answer.

If your primary goal were to conserve jars, on the other hand, you could try the following strategy. Start by dropping a jar from the first rung, then the second rung, and so forth, climbing one higher each time until the jar breaks. In this way, you only need a single jar—at the moment it breaks, you have the correct answer—but you may have to drop it n times (rather than $\log n$ as in the binary search solution).

So here is the tradeoff: it seems you can perform fewer drops if you are willing to break more jars. In this problem, we will try to understand how this tradeoff works at a quantitative level.

(a) Suppose you are given a budget of 2 jars. Describe a strategy for finding the highest safe rung that requires you to drop a jar at most $f(n)$ times. Your goal is to minimize $f(n)$, while breaking at most 2 jars. Give your upper bound on $f(n)$ using asymptotic notation.

(b) Same problem as in part (a), but this time you are given a budget of 3 jars.