Exercises 17: Karatsuba multiplication, Master method

Questions

1. In Karatsuba multiplication, when you do the multiplication (x1 + x0)*(y1 + y0), the two values you are multiplying might be n/2 + 1 digits each, rather than n/2 digits, since the addition might have led to a carry e.g. 53 + 52 = 105. Does this create a problem for the argument that the recurrence is t(n) = 3 t(n/2) + cn ?

2. In the derivation of the master method, for the recurrence t(n) = a t(n/b) + c n^d, I claimed there was no loss in generality in assuming c=1. Why?

3. In the derivation of the master method, we assume t(1) = 1 to simplify the expression (see lecture slides, page 4 slide 6, i.e. bottom right). What happens if we don't assume this? That is, what happens if the work we do in the base case is some constant c' that is unrelated to the constant c in the recurrence t(n) = a t(n/b) + c n^d ? Does this mean "all bets are off" and the O( ) expressions in the three cases (r < 1, r = 1, r > 1) are affected ?

4. Recall Exercise 16 Question 1, where we had a recurrence t(n) = 2 t(n/2) + n^2/4 + c. What is the solution of this recurrence, according to the master method ?

5. In the case (3) analysis of the master method, where r > 1, I claimed that (r^{k+1} – 1) / (r – 1) < c r^k for some constant c. Why?

6. Consider the three cases of the master method:

   a) If the same work is done at each level (r = 1), then the O( ) bound doesn't depend on a. Why not?
   b) If leaves dominate (r > 1), then the O( ) performance doesn't depend explicitly on d. Why not?
   c) If root dominates (r < 1) then performance doesn't depend on a or b. Why not?

Answers

1. The recurrence would need to be written t(n) = 2 t(n/2) + t(n/2 + 1) + cn. We know that t(n) is O(n^2) and we are trying to prove a better bound, but the fact that it is O(n^2) allows us to say that t(n) <= c n^2 for some constant c and for sufficiently large n. (Recall the formal definition from COMP 250.) Thus, t(n/2 + 1) <= c (n/2 + 1)^2 = c(n/2)^2 + c1 n/2 + c for some constant c and for sufficiently large n. Thus, t(n/2 + 1) = t(n/2) + O(n), that is, t(n/2 + 1) is bigger than t(n/2) but only by an amount that grows linearly with n. Thus, we can write:

$t(n) = 2\, t(n/2) + t(n/2 + 1) + cn = 3\, t(n/2) + O(n)$.

In case the above went too fast, here is the basic idea: there is no problem that the Karatsuba trick requires multiplying two numbers of size $n/2 + 1$ instead of $n/2$. The reason it doesn't matter is that the extra work you need to do is bounded by some $O(n)$ term. You are already doing a bunch of $O(n)$ work at each node (of the call tree). So one extra $O(n)$ term won't make any difference.

2. If you go through the derivation, you'll see that the constant c will be multiplied by the n^d which gets factored out of the summation over i. So in the end, you have $c\, n^d$ instead of n^d. But such a constant has no effect on a O() analysis.

3. Nothing changes. We are doing a O( ) analysis and constants don't matter. More concretely, on page 4 slide 6 (bottom right), we have t(n) = sum of two terms. The second term really has a constant c which we earlier set to 1. The first term has a constant c'. So we have the two constants c and c'. If we just replace the smaller constant by the larger one, we'll get an upper bound on t(n) which is fine since we're only doing O( ) anyhow. And now both have the same constant so we can merge the two terms into one, as we did in class where we assumed both $c = c' = 1$.

4. $A = 2$, $b = 2$, $c = 1/4$, $d = 2$. So $r = a/b^d = 2/(2^2) = ½$. So this is the case that the root dominates and $t(n) = O(n^2) = O(n^2)$.

5. $(r^{k+1} – 1) / (r – 1) < r^{k+1} / (r – 1) = r/(r-1) * r^k$, so let $c = r/(r-1)$.

6.
   a) It does depend on a, since for $r = 1$ we must have $a = b^d$. i.e. since the O() bound depends on b and d, it also depends on a, just not explicitly.
   b) As long as $r = a/b^d > 1$, the work that is done at the leaves grows as $r^{\log_b n}$ and the work done in total is $c\, r^{\log_b n}$ (see lecture notes). So, the work done at the leaves has the same O() behavior as the work done in total. But at the leaves, the problem size is t(1) which is a constant. There is no d exponent at the leaves. All that matters for O() is the number of leaves and this is independent of d.
   c) The work done at the root is the work required to partition into subproblems and then combine the solutions to those subproblems. It doesn't include the work done in the solving the subproblems themselves. By the definition of the recurrence, the work done at the root is n^d (and this is true regardless of r). For $r < 1$, solving the root problem plus all of the subproblems is $O(n^d)$. So, although the subproblems may require a lot of work, they don't add up to more work in the O() sense than the work done at the root. In that sense, performance doesn't depend on a and b.