CS 344: Design and Analysis of Computer Algorithms **Rutgers: Spring 2022**

Lecture 5

February, 1, 2022

Instructor: Sepehr Assadi

1 Solving Recurrences: Recursion Trees

Recall that in the last lecture, we wrote a *recurrence* $T(n)$ for the worst-case running time of the merge sort algorithm. The recurrence was $T(n) \leq 2T(n/2) + O(n)$ and we briefly pointed out $T(n) = O(n \log n)$. In this lecture, we show how to "solve" this (and other similar recurrences) using a general technique called *recursion trees*.

A recursion trees is a simple and pictorial tool to solve recurrences for runtime of algorithms (or at least devise a good guess on what the runtime should be). A recursion tree is a *rooted tree* with one node for each recursive subproblem. The *value* of each node is the amount of time spent on the corresponding subproblem *excluding* recursive calls. Thus, the overall running time of the algorithm is the sum of the values of all nodes in the tree. We now solve the recurrence for merge sort using recursion trees.

Let us write the recurrence for merge sort as $T(n) \leq 2 \cdot T(n/2) + c \cdot n$ where $c > 0$ is some *constant*: it is important to change the $O(n)$ notation at this point to $c \cdot n$ (both definitions are equivalent) so that the following equations are computed correctly. The tree for this recurrence is a *binary tree* (as the algorithm has *two* recursive subproblems) and can be defined as follows:

- The root has a value $c \cdot n$ because the work we do in the first level excluding recursive calls is at most $c \cdot n$; moreover, the root has two child-nodes corresponding to the two recursive calls.
- Each child-node of the root has a value $c \cdot n/2$ (as for them we work on an array of size $n/2$ and hence spend $c \cdot n/2$ time); moreover, each of these nodes has two further child nodes (with value $c \cdot n/4$ and so on and so forth).
- The above pattern is continued the same way until we reach the nodes with value $c \cdot 1 = c$ which form the leaf-nodes of this tree.

The following is an illustration of this tree.

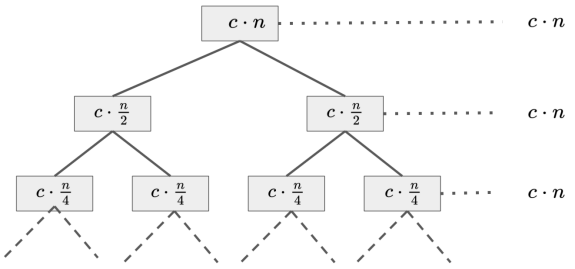


Figure 1: The first three layers of the recursion tree for merge sort. The *total time* spent for solving the subproblems at each level, excluding recursive calls, is also written on the right hand side.

By this definition, the nodes at level i of the tree (assuming the root is at level 0) have value $c \cdot n/2^i$. Moreover, there are 2^i nodes at level i of any binary tree. As such, the total time spent at each level of this tree is at most $c \cdot n$. Finally, this tree has at most $\lceil \log n \rceil$ levels because we stop the tree at a level with values c which is the layer i where $n/2^i \leq 1$ which implies $i = \lceil \log n \rceil$.

To conclude, the recurrence for merge sort takes $c \cdot n$ time per each level of the tree and there are only $\lceil \log n \rceil$ levels; hence, $T(n) \leq c \cdot n \cdot \lceil \log n \rceil = O(n \log n)$.

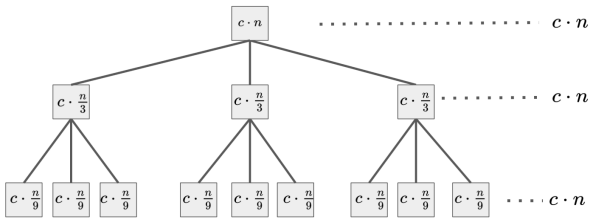
In general, using recursion trees is an easy and versatile method for solving recurrences once you get used to them. You should just remember that *always* to write the $O(f(n))$-notations as $c \cdot f(n)$ (i.e., replace $O(n^2)$ with $c \cdot n^2$). In your textbook other methods for solving recurrences beside the recursion tree method have also been discussed. For instance, induction (as always!) and *the master theorem*; we will not use these methods in this course much, however you are encouraged to take a look at them as well – moreover, even though none of your homeworks or exams will ask you a question about these methods, you are allowed to use either method instead of recursion trees when writing your solutions *unless the question explicitly asks you to use recursion trees* (just make sure you are comfortable with those techniques if you decide to do so).

1.1 More Examples

Let us solve a couple of more recurrences. A typical question for recurrences is as follows: Find the *tightest* bound for the following recurrences (you do *not* need to prove that your bound is tightest).

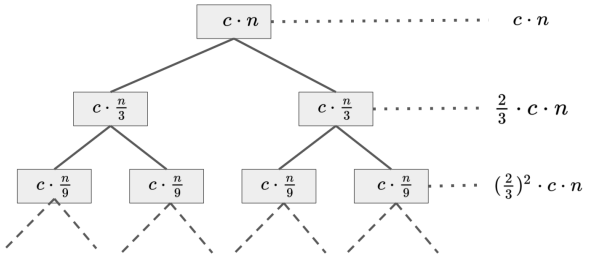
$$T(n) \leq 3T(n/3) + O(n) \qquad T(n) \leq 2T(n/3) + O(n) \qquad T(n) \leq 3T(n/3) + O(n^2).$$

- $T(n) \leq 3T(n/3) + O(n)$. We first replace $O(n)$ with $c \cdot n$ and now solve for the recurrence. The first three level of the recursion tree for this recurrence are:



  In particular, at level i of the tree, we have 3^i nodes, each with a value of $c \cdot \frac{n}{3^i}$. This means that the total value for each level is $3^i \cdot c\frac{n}{3^i} = c \cdot n$. The depth of the tree (total number of levels) is also $\lceil \log_3 n \rceil$. Hence, the total value of the tree is $T(n) \leq c \cdot n \cdot \lceil \log_3 n \rceil = O(n \log n)$.

- $T(n) \leq 2T(n/3) + O(n)$. We again replace $O(n)$ with $c \cdot n$ and solve for the recurrence using the following recursion tree:



  At level i of the tree, we have 2^i nodes each with a value of $c \cdot \frac{n}{3^i}$, hence the total value of level i is $(\frac{2}{3})^i \cdot c \cdot n$ (note that unlike the previous cases, this time the value of a level depends on its index).

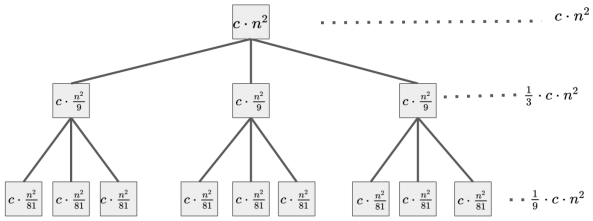
The total number of levels in this tree is also $\lceil \log_3 n \rceil$ (but as it will become evident, we actually do not need this). As such, the total value of the tree is:

$$\sum_{i=0}^{\lceil \log_3 n \rceil} (\frac{2}{3})^i \cdot c \cdot n = c \cdot n \sum_{i=0}^{\lceil \log_3 n \rceil} (\frac{2}{3})^i \leq c \cdot n \sum_{i=0}^{\infty} (\frac{2}{3})^i = c \cdot n \cdot \frac{1}{1 - 2/3} = 3 \cdot c \cdot n.$$

(the second last equality is by using the formula for sum of geometric series)

Hence, $T(n) = O(n)$ in this case.

- $T(n) \leq 3T(n/3) + O(n^2)$. We replace $O(n^2)$ term with $c \cdot n^2$ and write the following recursion tree:



At level i of the tree, there are 3^i nodes each with a value of $c \cdot (\frac{n}{3^i})^2$, hence the total value of level i is $(\frac{1}{3^i}) \cdot c \cdot n^2$. Again, there are $\lceil \log_3 n \rceil$ levels (and again this is not needed) and thus the total value of the tree is:

$$\sum_{i=0}^{\lceil \log_3 n \rceil} (\frac{1}{3^i}) \cdot c \cdot n^2 = c \cdot n^2 \cdot \sum_{i=0}^{\lceil \log_3 n \rceil} (\frac{1}{3^i}) \leq c \cdot n^2 \cdot \sum_{i=0}^{\infty} (\frac{1}{3^i}) = c \cdot n^2 \cdot \frac{1}{1 - 1/3} = \frac{3}{2} \cdot c \cdot n^2.$$

(again, the second last equality is by sum of the geometric series)

Hence, $T(n) = O(n^2)$ in this case.