So proving the inductive step as above, plus proving the bound works for $n = 2$ and $n = 3$, suffices for our proof that the bound works for all $n > 1$.

Plugging the numbers into the recurrence formula, we get $T(2) = 2T(1) + 2 = 4$ and $T(3) = 2T(1) + 3 = 5$. So now we just need to choose a c that satisfies those constraints on $T(2)$ and $T(3)$. We can choose $c = 2$, because $4 \leq 2 \cdot 2 \log 2$ and $5 \leq 2 \cdot 3 \log 3$.

Therefore, we have shown that $T(n) \leq 2n \log n$ for all $n \geq 2$, so $T(n) = O(n \log n)$.

1.1.2 Warnings

Warning: Using the substitution method, it is easy to prove a weaker bound than the one you're supposed to prove. For instance, if the runtime is $O(n)$, you might still be able to substitute cn^2 into the recurrence and prove that the bound is $O(n^2)$. Which is technically true, but don't let it mislead you into thinking it's the best bound on the runtime. People often get burned by this on exams!

Warning: You must prove the exact form of the induction hypothesis. For example, in the recurrence $T(n) = 2T(\lfloor n/2 \rfloor) + n$, we could falsely "prove" $T(n) = O(n)$ by guessing $T(n) \leq cn$ and then arguing $T(n) \leq 2(c\lfloor n/2 \rfloor) + n \leq cn + n = O(n)$. Here we needed to prove $T(n) \leq cn$, not $T(n) \leq (c + 1)n$. Accumulated over many recursive calls, those "plus ones" add up.

1.2 Recursion tree

A recursion tree is a tree where each node represents the cost of a certain recursive subproblem. Then you can sum up the numbers in each node to get the cost of the entire algorithm.

Note: We would usually use a recursion tree to generate possible guesses for the runtime, and then use the substitution method to prove them. However, if you are very careful when drawing out a recursion tree and summing the costs, you can actually use a recursion tree as a direct proof of a solution to a recurrence.

If we are only using recursion trees to generate guesses and not prove anything, we can tolerate a certain amount of "sloppiness" in our analysis. For example, we can ignore floors and ceilings when solving our recurrences, as they usually do not affect the final guess.

1.2.1 Example

Recurrence: $T(n) = 3T(\lfloor n/4 \rfloor) + \Theta(n^2)$

We drop the floors and write a recursion tree for $T(n) = 3T(n/4) + cn^2$.

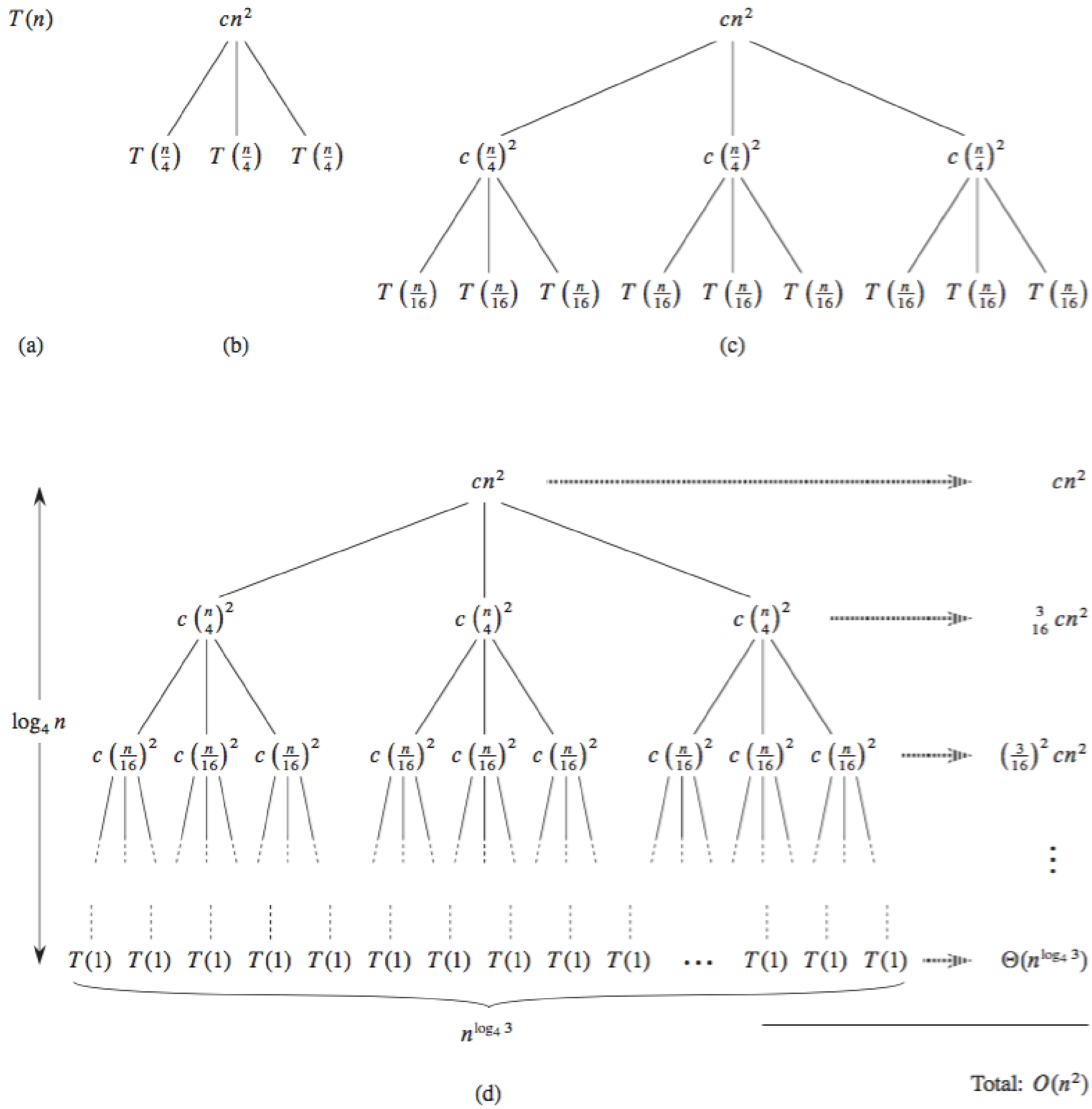

Figure 4.5 Constructing a recursion tree for the recurrence $T(n) = 3T(n/4) + cn^2$. Part **(a)** shows $T(n)$, which progressively expands in **(b)–(d)** to form the recursion tree. The fully expanded tree in part (d) has height $\log_4 n$ (it has $\log_4 n + 1$ levels).

The top node has cost cn^2, because the first call to the function does cn^2 units of work, aside from the work done inside the recursive subcalls. The nodes on the second layer all have cost $c(n/4)^2$, because the functions are now being called on problems of size $n/4$, and the functions are doing $c(n/4)^2$ units of work, aside from the work done inside their recursive subcalls, etc. The bottom layer (base case) is special because each of them contribute $T(1)$ to the cost.

Analysis: First we find the height of the recursion tree. Observe that a node at depth i reflects a subproblem of size $n/4^i$. The subproblem size hits $n = 1$ when $n/4^i = 1$, or