Lecture 4: Divide and Conquer Paradigm Continued

MCS/CS 401 Computer Algorithms I — June 22, 2022
Instructor: Danko Adrovic

Lecture Outline

- Divide and Conquer Paradigm Continued
- Solving Recurrences
  - Substitution Method for Solving Recurrences
  - Recursion-tree Method for Solving Recurrences
  - The Master Theorem and the Master Method for Solving Recurrences

1 Divide and Conquer Paradigm Continued

As we have seen 2 lectures ago, the *divide and conquer paradigm* has the ability to reduce the problem, which we aim to solve, by systematically reducing the original problem into smaller subproblems. This kind of reduction into smaller subproblems was achieved recursively, where at each step we applied the three parts of the paradigm

1. **Divide** the problem into a number of subproblems that are smaller instances of the same problem.
2. **Conquer** the subproblems by solving them recursively.
   (a) If the subproblem sizes are small enough, just solve the subproblems in a straightforward manner.
3. **Combine** the solutions to the subproblems into the solution for the original problem.

When the subproblems are large enough to solve recursively, we call that the **recursive case**. Once the subproblems become small enough that we no longer recurs, we say that the recursion *bottoms out* and that we have gotten down to the **base case**.

Note that this is exactly how basic recursive functions work as well, functions you may have written in a language like C++, Java, or Python, just to name a few. In such cases we have a **base case**, also called *stopping condition*. Then, all other cases are recursively reduced down to the **base case**.

As an example, consider the recursively defined **Fibonacci** sequences formula.

$$\begin{cases} F(0) = 0 \\ F(1) = 1 \\ F(n) = F(n-1) + F(n-2) \end{cases}$$

In the $F(n) = F(n-1) + F(n-2)$ part, we repeatedly keep reducing input arguments of n, ie. (n-1, n-2) until they have been sufficiently reduces such that the *stopping conditions* F(0) and F(1) can be applied.

2 Solving Recurrences

There is a natural relationship between recurrences and the divide-and-conquer paradigm because they allow us to characterize the running times of the divide-and-conquer algorithms.

Definition 1. *(Recurrence) A recurrence is an equation or inequality that describes a function in terms of its value on smaller inputs.*

Definition 2. *(Recurrence for the Running Time of Divide-and-Conquer Algorithms) Let $T(n)$ be the running time of a problem of size n. When the problem size is small enough, say $n \leq c$, for some constant c, solution takes constant time, which we write $\Theta(1)$. Assume that our division of the problem yields a subproblems, each of which is $\frac{1}{b}$ the size of the original problem. Let $D(n)$ stand for the time it took to divide the problem into subproblems and $C(n)$ the time to combine the solution to the subproblems into the solution to the original problem, the recurrence relation*

$$T(n) = \begin{cases} \Theta(n) & \text{if } n \leq c \\ aT(\frac{n}{b}) + D(n) + C(n) & \text{otherwise} \end{cases}$$

describes such a running time.

Once we have analyzed a divide-and-conquer algorithm and established a recurrence relation for its running time, we have to **solve** the recurrence in order to obtain the bound on the running time of the algorithm. That is, we want to find a bound that is free of recursion and it has a closed form, such as $T(n) = \Theta(g(n))$, where $g(n)$ is an ordinary function.

For the process of solving a recurrence and finding $\Theta(g(n))$, we will use the difference methods

- Substitution method
- Recursion-tree method
- Master method

2.1 The substitution method for solving recurrences

The substitution method for solving of recurrences consists of two steps

1. Guess the form of the solution
2. Apply mathematical induction to find the constants and show that the solution is correct

Before proceeding further with examples and induction, it is worth pointing out a subtilty when working with asymptotics and induction. In general, application of induction requires us to show that the base case works. This is often omitted in the application of induction to these types of problems.

Let's say that we have T(n) is $O(g(n))$, meaning that $T(n) \leq cg(n)$. We can always choose a very large constant c (if we want), which will make $T(k_0) \leq c * g(k_0)$ hold for any initial value k_0. Such a large value for c will immediately satisfy the initial condition and, for that reason, we do not even need to check them. We say that the base case is trivially satisfied.

Example 1. *(Substitution method)*
Let $T(n) = 3T(\frac{n}{4}) + O(n^2)$ and we guess that $T(n) = O(n^2)$. Using induction, we need to show that $T(n) = O(n^2)$ is an upper bound for the recurrence, that is, $T(n) \leq dn^2$ for some constant $d > 0$.

We start by rewriting our recurrence $T(n) = 3T(\frac{n}{4}) + O(n^2)$ using $c > 0$ constant.

$$\begin{aligned} T(n) &\leq 3T(\frac{n}{4}) + cn^2 \\ &\leq 3d(\frac{n}{4})^2 + cn^2 \\ &\leq 3d\frac{n^2}{16} + cn^2 = \frac{3}{16}dn^2 + cn^2 \\ &\leq dn^2 \end{aligned}$$

We next need to link constants c and d, using the last step.

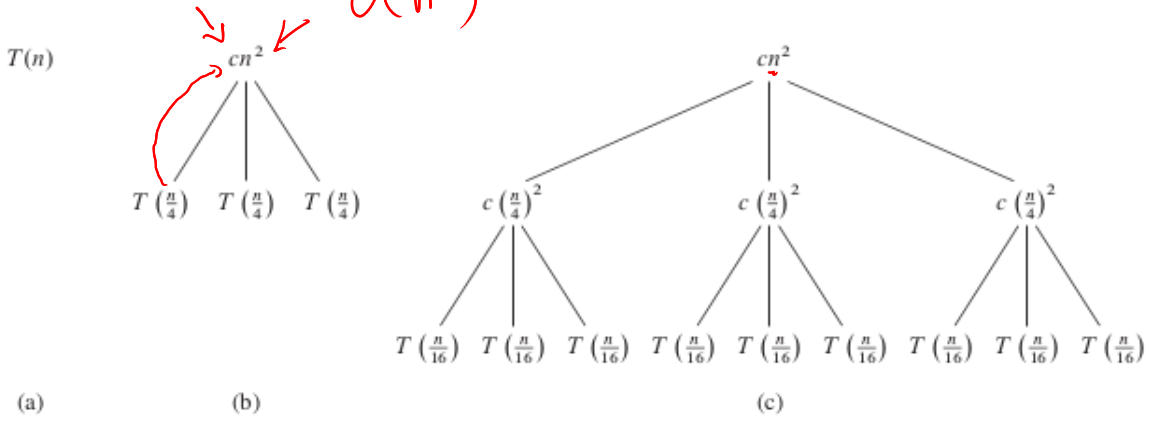
$$\begin{aligned} \frac{3}{16}dn^2 + cn^2 &\leq dn^2 \\ c &\leq (1 - \frac{3}{16})d \\ c &\leq \frac{13}{16}d \\ \frac{16}{13}c &\leq d \\ d &\geq \frac{16}{13}c \end{aligned}$$

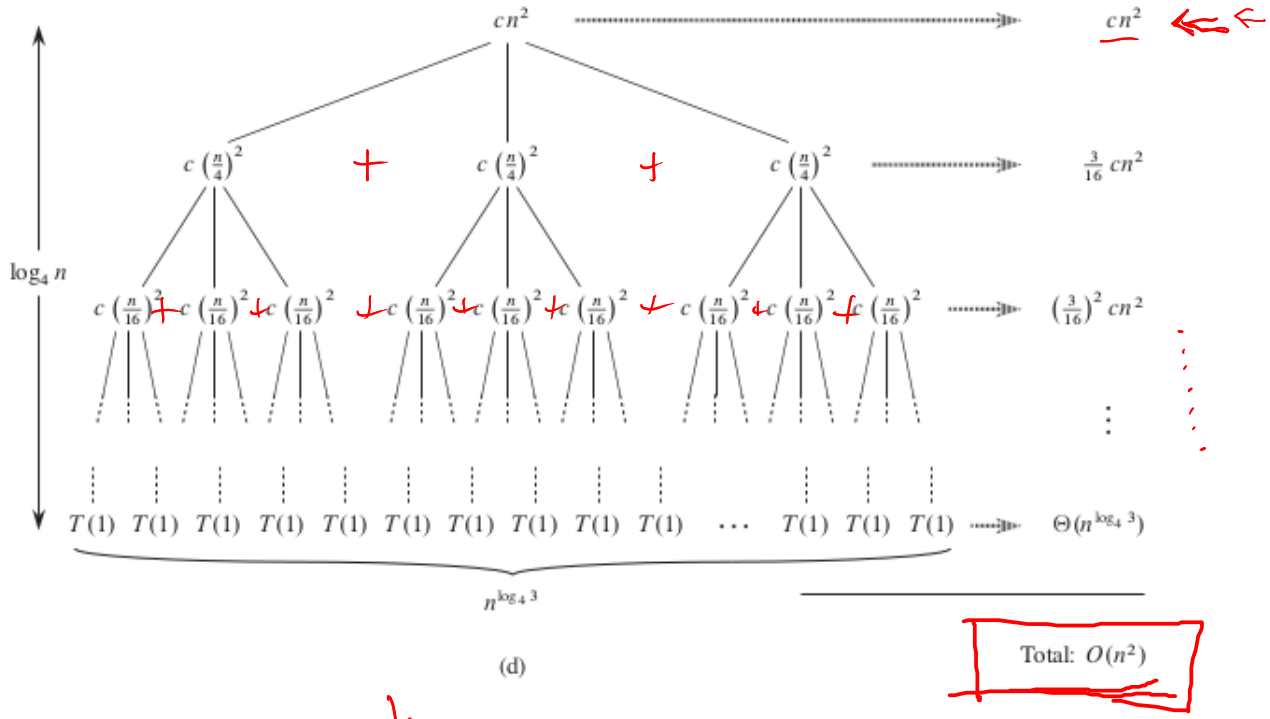
Therefore, as long as $d \geq \frac{16}{13}c$, the last step will hold and our proof is complete.

Using the substitution method is relatively easy. The hard part can be be the guessing of the correct solution! Unfortunately, there is no *one method*, which can be universally applied to obtain a good or correct guess. Instead, each case has to be analyzed separately. With this in mind, we now consider a second method to derive and prove solutions to recurrence relations.

2.2 Recursion-tree Method for Solving Recurrences

In our previous example, we considered the recurrence $T(n) = 3T(\frac{n}{4}) + O(n^2)$ and had guessed that $T(n) = O(n^2)$ was the solution for this recurrence. With some skill, we can make an educated guess of what the solutions should be but now we are going to see a method, the *Recursion-tree Method for Solving Recurrences*, which will help us zero in on the form of the recurrence solution.





Note that the height of the tree is $\log_4(n)$ and the number of levels in the tree is $\log_4(n)+1$.

The cost associated with each level contributes to the the total running time $T(n)$. Therefore, we have

$$\begin{aligned} T(n) &= cn^2 + \frac{3}{16}cn^2 + (\frac{3}{16})^2 cn^2 + \cdots + (\frac{3}{16})^{\log_4(n-1)} cn^2 + O(n^{\log_4(3)}) \\ &= \sum_{k=0}^{\log_4(n-1)} (\frac{3}{16})^k cn^2 + O(n^{\log_4(3)}) \\ &= \frac{(\frac{3}{16})^{\log_4 n - 1}}{\frac{3}{16} - 1} cn^2 + O(n^{\log_4(3)}) \end{aligned}$$

Using the decreasing geometric series, we can bound the expression by the strict inequality and infinite series given below. Note that $O(n^{\log_4(3)})$ represents the lower order terms as our tree *bottoms out* at the bottom of the tree, where all the base cases are located.

$$\begin{aligned} T(n) &= cn^2 + \frac{3}{16}cn^2 + (\frac{3}{16})^2 cn^2 + \cdots + (\frac{3}{16})^{\log_4(n-1)} cn^2 + O(n^{\log_4(3)}) \\ &= \sum_{k=0}^{\log_4(n-1)} (\frac{3}{16})^k cn^2 + O(n^{\log_4(3)}) \\ &= \frac{(\frac{3}{16})^{\log_4 n - 1}}{\frac{3}{16} - 1} cn^2 + O(n^{\log_4(3)}) \\ &< \sum_{k=0}^{\infty} (\frac{3}{16})^k cn^2 + O(n^{\log_4(3)}) \\ &= \frac{1}{1 - \frac{3}{16}} cn^2 + O(n^{\log_4(3)}) \\ &= \frac{16}{13} cn^2 + O(n^{\log_4(3)}) \\ &= \frac{16}{13} cn^2 + O(n^{0.79}) \\ &= O(n^2) \end{aligned}$$

$T(n) \leq O(n^2)$

$\leq d \cdot n^2$

$d \geq \frac{16}{13} c$

Once the "guess" has been derived, we have to use induction to prove the statement. This proof was given in the previous section.

2.3 Recursion-tree Method: a more complicated problem

Consider the recurrence relation $T(n) = T(\frac{n}{2}) + T(\frac{n}{4}) + O(n^2)$. The recursion tree would look like

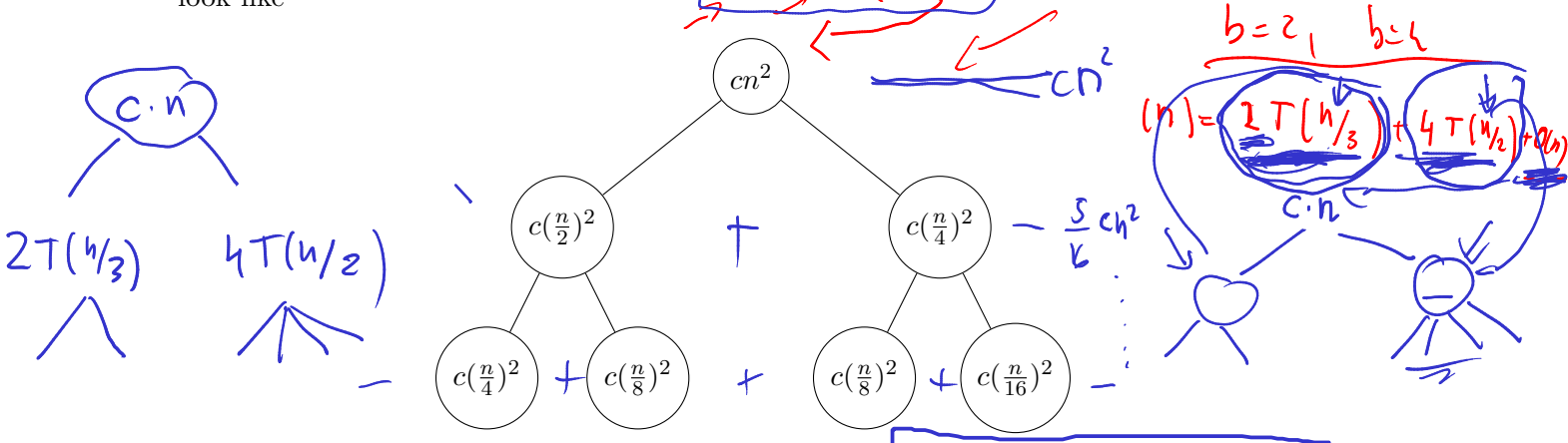


Figure 1: A partially constructed tree for the recurrence $T(n) = T(\frac{n}{2}) + T(\frac{n}{4}) + O(n^2)$

Adding up cost contribution from each level of this partially constructed tree, we see that a pattern emerges: cn^2, $\frac{5}{16}cn^2$, $\frac{25}{256}cn^2$, ..., etc. We can capture this pattern and the total running time $T(n) = cn^2(1 + \frac{5}{16} + \frac{25}{256} + \cdots)$ or $T(n) = cn^2(1 + (\frac{5}{16})^1 + (\frac{5}{16})^2 + (\frac{5}{16})^3 \cdots)$.

We recognize the last expression as the geometric series, whose value can be computed directly $cn^2 \cdot (\frac{1}{1-\frac{5}{16}}) = cn^2 \cdot \frac{16}{11}$. Hence the total cost T(n) is asymptotically bounded from above by $O(n^2)$.

2.4 The Master Theorem and the Master Method for Solving Recurrences

As we have seen in the previous two sections, guessing a solution to a recurrence relation is non-trivial. Now we are going to see a general method, called **Master method**, which can solve recurrences if they meet certain criteria.

The Master Theorem
Let $a \geq 1$ and $b > 0$ be constants, let $f(n)$ be a function, and let $T(n)$ be defined on the nonnegative integers by the recurrence $T(n) = aT(n/b) + f(n)$. Then $T(n)$ can be bounded asymptotically as follows:

1) if $f(n) = O(n^{\log_b(a)-\epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b(a)})$
2) if $f(n) = \Theta(n^{\log_b(a)})$, then $T(n) = \Theta(n^{\log_b(a)} \lg(n))$
3) if $f(n) = \Omega(n^{\log_b(a)+\epsilon})$ for some constant $\epsilon > 0$, and if $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n, then $T(n) = \Theta(f(n))$

Example 2. *$T(n) = 9T(\frac{n}{3}) + n$. In this problem $a = 9$, $b = 3$, and $f(n) = n$, with $n^{\log_3(9)} = \Theta(n^2)$. Because $f(n) = O(n^{\log_3(9)-\epsilon}) = O(n^{2-\epsilon})$, for $\epsilon = 1$, the Case 1 of the Master theorem applies and the solution is $T(n) = \Theta(n^2)$*

Example 3. $T(n) = T(\frac{2n}{3}) + 1$. *In this problem* $a = 1$ *and* $b = \frac{3}{2}$, $f(n) = 1$, *with* $n^{\log_b(a)} = n^0 = 1$). *Because* $f(n) = \Theta(n^{\log_b(a)}) = 1$, *the Case 2 of the Master theorem applies and the solution is* $T(n) = \Theta(\lg(n))$

Example 4. $T(n) = 3T(\frac{n}{4}) + n\log(n)$. *In this problem* $a = 3$ *and* $b = 4$, $f(n) = n\log(n)$, *with* $n^{\log_4(3)} = n^{0.793}$). *For* $\epsilon \approx 0.2$, *we have* $f(n) = \Omega(n^{log_4(3)+\epsilon})$. *Therefore, the Case 3 applies if we are able to show that* $af(\frac{n}{b}) \leq cf(n)$ *for some* $c < 1$ *and a sufficiently large* n. *This means then that* $\frac{3n}{4}\log(\frac{n}{4}) \leq cn\log(n)$. *Letting* $c = \frac{3}{4}$ *satisfies this condition. By Case 3,* $T(n) = \Theta(n\log(n))$.

Example 5. $T(n) = 3T(n/2) + n^2$. *In this problem* $a = 3$ *and* $b = 2$, $f(n) = n^2$, *with* $n^{\log_2(3)} = n^{1.584}$). *For* $\epsilon \approx 0.41$, *we have* $f(n) = \Omega(n^{log_2(3)+\epsilon})$. *Therefore, the Case 3 applies if we are able to show that* $af(\frac{n}{b}) \leq cf(n)$ *for some* $c < 1$ *and a sufficiently large* n. *This means then that* $3(n/2)^2 \leq cn^2$ *or* $(3/4)n^2 \leq cn^2$. *Letting* $c = \frac{3}{4}$ *satisfies this condition. By Case 3,* $T(n) = \Theta(n^2)$.