CSE331 Introduction to Algorithm
Lecture 7: Solving Recurrences I

Antoine Vigneron
antoine@unist.ac.kr

Ulsan National Institute of Science and Technology

July 23, 2021

Course Organization

- Reference: Sections 4.3 and 4.4 of the textbook (p. 83–92)
  Introduction to Algorithms by Cormen, Leiserson, Rivest and Stein.

Introduction

- Some running times are given by recurrence relations:

$$\text{Merge sort: } T(n) = T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + \Theta(n) \quad (1)$$

$$\text{Binary search: } T(n) = T(\lfloor n/2 \rfloor) + \Theta(1) \quad (2)$$

$$\text{Karatsuba's algorithm: } T(n) = 3T(n/2) + \Theta(n) \quad (3)$$

- It is often going to be the case:
  - Many algorithms are recursive.
  - In particular, divide and conquer algorithms.
- In this Lecture, we will see how to solve some recurrence relations that often arise when analyzing algorithms.
- The function $T(n)$ in this lecture denotes a running time on input of size n, and thus $T(n) > 0$ for all integer $n \geqslant 1$.

The Substitution Method

The substitution method

The *substitution method* for solving recurrences comprises two steps.

1. Guess the form of the solution.
2. Use mathematical induction to find the constants and show that the solution works.

Example

We want to solve a recurrence similar to (1):

$$T(n) = 2T(\lfloor n/2 \rfloor) + n \tag{4}$$

- More precisely, we want to find a good upper bound on $T(n)$.
- We first guess that $T(n) = O(n \log n)$.

The Substitution Method

- So we want to prove by mathematical induction that $T(n) \leqslant cn \log n$ for some constant c.
- Recall that, proving by induction that a property $P(n)$ is true for all n can be done in two steps:
  - The *basis step*, where we prove that $P(1)$ is true.
  - The *inductive step*, where we prove that if $n > 1$ and $P(m)$ is true for all $m < n$, then $P(n)$ is true.
- Remark: We will often have to change the basis step to cover a few values of n, for instance $P(1)$, $P(2)$ and $P(3)$, and then the inductive step is done for all $n \geqslant 4$.
- We begin with the inductive step.

Inductive Step

- So we assume that $n > 1$ and $T(m) \leqslant cm \log m$ for all $m < n$, and we want to prove that $T(n) \leqslant cn \log n$.

$$
\begin{aligned}
T(n) &= 2T(\lfloor n/2 \rfloor) + n \\
&\leqslant 2c\lfloor n/2 \rfloor \log(\lfloor n/2 \rfloor) + n \qquad \text{by induction hypothesis (IH)} \\
&\leqslant 2c(n/2)\log(n/2) + n \\
&= cn(\log(n) - \log 2) + n \\
&= cn(\log(n) - 1) + n \\
&= cn \log n + (1 - c)n
\end{aligned}
$$

- So if we choose $c \geqslant 1$, we have $T(n) \leqslant cn \log n$, which completes the inductive step.

Base Step

- Base step: We would like to prove that $T(n) \leqslant cn \log n$ for $n = 1$.
- Problem:
  - Since $\log 1 = 0$, it means $T(1) \leqslant 0$.
  - $T(1)$ should be positive as it is a running time.
- Solution:
  - We will use $T(2)$ and $T(3)$ as base cases, because for any $n > 3$, the recurrence goes through $n = 2$ or $n = 3$ before reaching $n = 1$.
  - We want to have

$$T(2) \leqslant 2c \log 2 = 2c$$
$$T(3) \leqslant 3c \log 3$$
$$1 \leqslant c \quad \text{(from previous slide)}$$

  - So we choose

$$c = \max\left(1, \frac{T(2)}{2}, \frac{T(3)}{3 \log 3}\right)$$

The Substitution Method

- In summary, we have proved the following.
- Let c be the constant

$$c = \max\left(1, \frac{T(2)}{2}, \frac{T(3)}{3\log 3}\right).$$

- Let $P(n)$ be the property $T(n) \leqslant cn\log n$.
- Then we have proved that $P(2)$ and $P(3)$ are true, and the calculations on Slide 7 show that

$$P(\lfloor n/2 \rfloor) \text{ is true implies } P(n) \text{ is true.} \tag{5}$$

- As $P(2)$ is true, Property (5) implies that $P(4)$ and $P(5)$ are true.
- So $P(2), P(3), P(4)$ and $P(5)$ are true, which implies that $P(m)$ is true for all $2 \leqslant m \leqslant 11$.
- ...

The Substitution Method

- We have proved by induction that for all $n \geqslant 2$,

$$T(n) \leqslant cn \log n$$

  where the constant c is given by

$$c = \max\left(1, \frac{T(2)}{2}, \frac{T(3)}{3 \log 3}\right).$$

- It means that Equation (4) solves to $T(n) = O(n \log n)$.
  - Here the n_0 from the definition of the $O(\cdot)$ notation is 2.

The Substitution Method: Example 2

- We now want to find an upper bound for

$$T(n) = T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + 1 \tag{6}$$

- We guess that $T(n) = O(n)$.
- So we try to prove that $T(n) \leqslant cn$ for some constant c.
- The inductive step would be:

$$\begin{aligned} T(n) &= T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + 1 \\ &\leqslant c\lceil n/2 \rceil + c\lfloor n/2 \rfloor + 1 \\ &= cn + 1 \end{aligned}$$

- This approach failed.

The Substitution Method: Example 2

- Previous attempt was off by 1 only.
- So we make a small change: We try to prove that $T(n) \leqslant cn + d$ for some constants c, d.
- Inductive step:

$$\begin{aligned} T(n) &= T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + 1 \\ &\leqslant c(\lceil n/2 \rceil) + d + c(\lfloor n/2 \rfloor) + d + 1 \\ &= cn + 2d + 1 \end{aligned}$$

- So it works if $2d + 1 \leqslant d$, which means $d \leqslant -1$.
- We choose $d = -1$.
- As we can choose c to be as large as we want, we can handle the base case $n = 1$. For instance choose $d = -1$ and $c = T(1) + 1$.
- We just proved that $T(n) \leqslant cn - 1$ for some constant $c > 0$, and thus $T(n) = O(n)$.

Avoiding Pitfalls

- Suppose that

$$T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + n$$

and we try to prove that $T(n) \leqslant cn$ for some constant c, in the following way:

$$\begin{aligned} T(n) &\leqslant c\lfloor n/2 \rfloor + c\lceil n/2 \rceil + n \\ &= cn + n \\ &= O(n) \qquad \leftarrow \textit{wrong!} \end{aligned}$$

- Problem: In the inductive step, we should prove the *exact form* of the inductive hypothesis. So we should get $T(n) \leqslant cn$, not $T(n) \leqslant cn + n$.

The Recursion Tree Method

- The substitution method often gives short proofs.
- Difficulty: guessing the solution.
- We can use the *recursion tree* method to make a good guess.
  - We already saw this method in Lecture 4 (on MERGE SORT).
- So one approach to solve recurrences is:
  1. Guess the solution using the recursion tree method.
     - ★ We can afford to be sloppy, as the goal is to make a guess.
  2. Prove it using the substitution method.
     - ★ Needs to be rigorous (see Slide 13).

The Recursion Tree Method: Example

- We want to guess a good upper bound for

$$T(n) = 3T(\lfloor n/4 \rfloor) + \Theta(n^2) \tag{7}$$

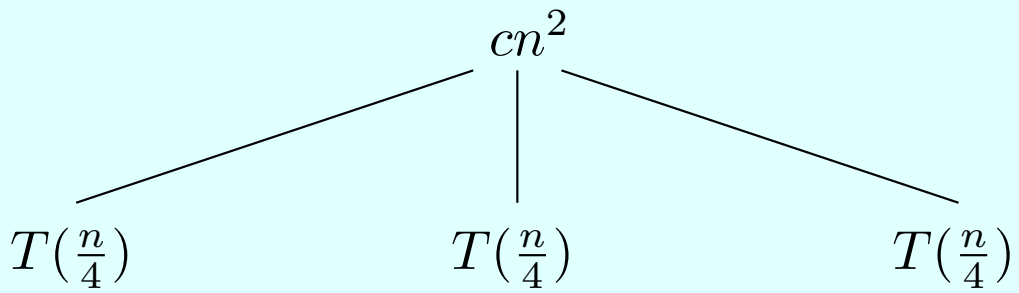
- We can afford to be sloppy. So we assume that n is a power of 4.
- It means that we can remove the floor functions.
- So we rewrite Equation 7:

$$T(n) = 3T(n/4) + cn^2 \qquad \text{for some constant } c.$$

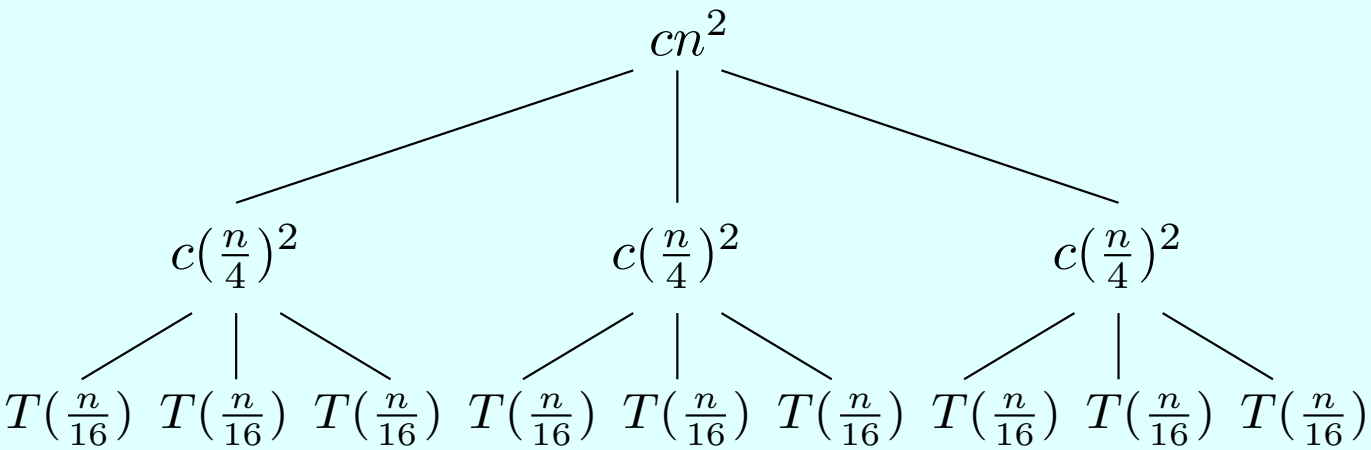
The Recursion Tree Method: Example

$$T(n)$$

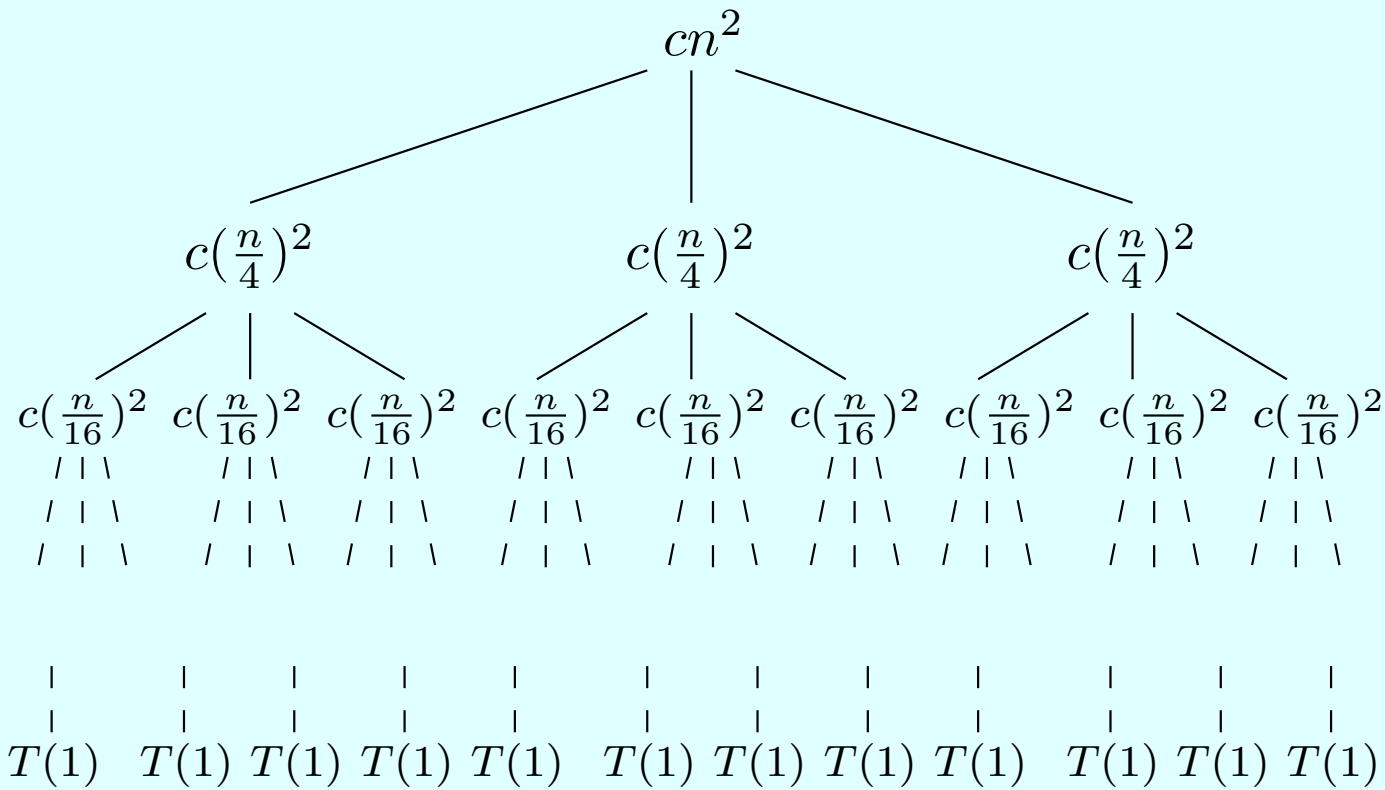
The Recursion Tree Method: Example



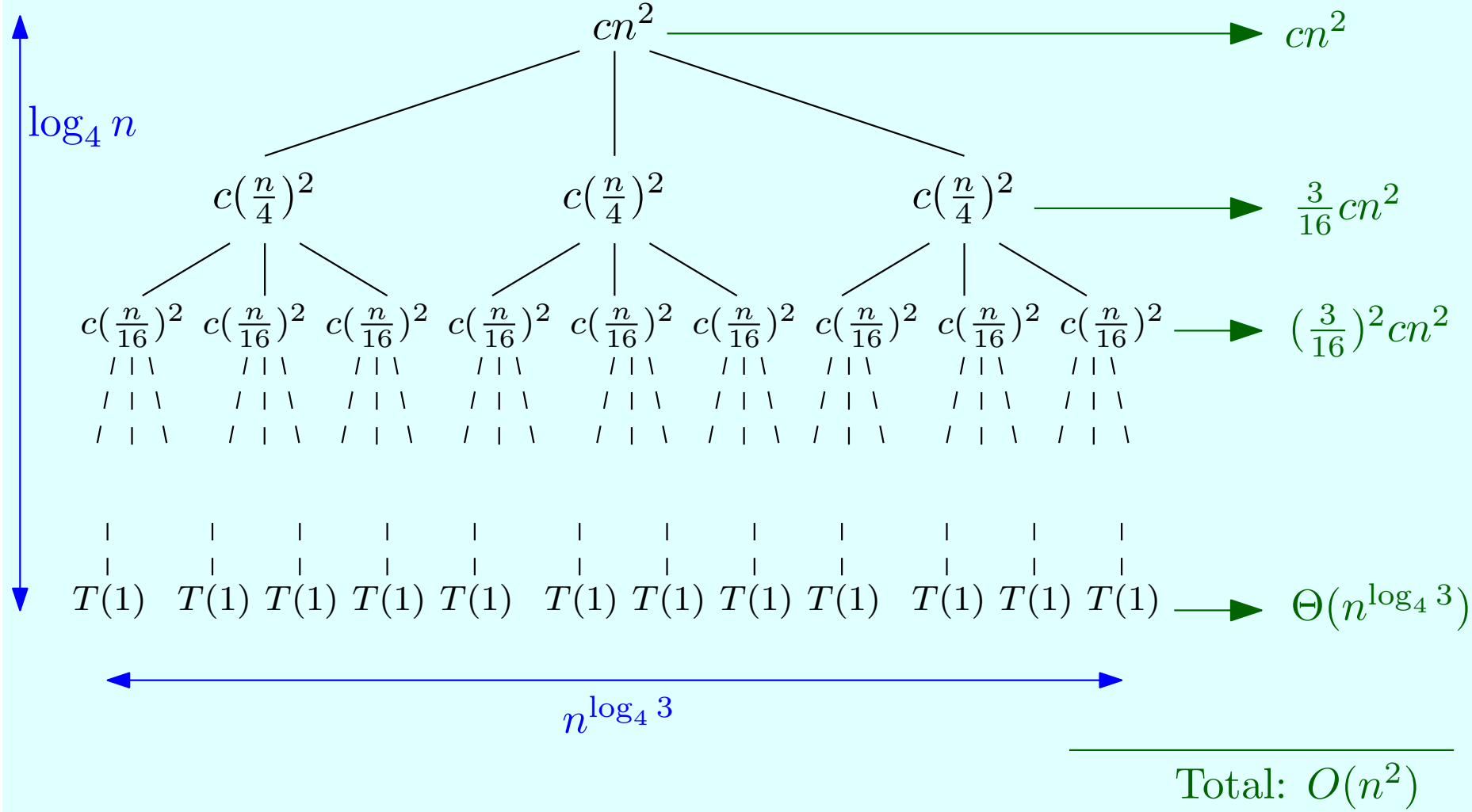
The Recursion Tree Method: Example



The Recursion Tree Method: Example



The Recursion Tree Method: Example



The Recursion Tree Method: Example

- Let h denote the height of the tree.
- The size of the subproblems decreases by a factor 4 at each level.
- So $4^h = n$, which means that $h = \log_4 n$.
- So the number of leaves is

$$3^h = 3^{\log_4 n} = 3^{\frac{\log_3 n}{\log_3 4}} = 3^{(\log_3 n)(\log_4 3)} = n^{\log_4 3} \simeq n^{0.792}$$

- At depth $i < h$:
  - The number of nodes is 3^i.
  - The size of each subproblem is $n/4^i$.
  - The total cost over all nodes at depth i is

$$3^i c \left(\frac{n}{4^i}\right)^2 = c \left(\frac{3}{16}\right)^i n^2.$$

The Recursion Tree Method: Example

- At the leaves, the total cost is $\Theta(n^{\log_4 3})$ as there are $n^{\log_4 3}$ leaves and each leaf has cost $\Theta(1)$.
- Overall

$$\begin{aligned} T(n) &= O(n^{\log_4 3}) + cn^2 \sum_{i=0}^{h-1} \left(\frac{3}{16}\right)^i \\ &\leqslant O(n^{\log_4 3}) + cn^2 \sum_{i=0}^{\infty} \left(\frac{3}{16}\right)^i \\ &= O(n^{\log_4 3}) + cn^2 \frac{16}{13} \\ &= O(n^2) \end{aligned}$$

so $T(n) = O(n^2)$.

The Recursion Tree Method: Example

- We have just guessed that Equation (7) yields $T(n) = O(n^2)$.
- We now *prove* it with the substitution method.
- We rewrite Equation (7) using an unknown constant c:

$$T(n) \leqslant 3T(\lfloor n/4 \rfloor) + cn^2.$$

- Now we prove by induction that $T(n) \leqslant dn^2$ for some constant d.

$$\begin{aligned} T(n) &\leqslant 3T(\lfloor n/4 \rfloor) + cn^2 \\ &\leqslant 3d\lfloor n/4 \rfloor^2 + cn^2 \\ &\leqslant 3d(n/4)^2 + cn^2 \\ &= \left(\frac{3}{16}d + c\right) n^2 \end{aligned}$$

The Recursion Tree Method: Example

- So we need to have $3d/16 + c \leqslant d$, that is $d \geqslant 16c/13$.
- Base cases: $T(1) \leqslant d$, $T(2) \leqslant 4d$ and $T(3) \leqslant 9d$.
- So if we choose

$$d = \max\left(\frac{16c}{13}, T(1), \frac{T(2)}{4}, \frac{T(3)}{9}\right),$$

  we have proved that $T(n) \leqslant dn^2 = O(n^2)$.
- We also have $T(n) = \Omega(n^2)$. Why?
  - We assume that the $\Theta(n^2)$ term in Equation (7) is positive, as it will be the case when we analyze algorithms.
  - We rewrite it $T(n) \geqslant 3T(\lfloor n/4 \rfloor) + c'n^2$ for some $c' > 0$.
  - Then $T(n) \geqslant c'n^2 = \Omega(n^2)$ because $T(m) > 0$ for all m.
- So we proved the tight bound $T(n) = \Theta(n^2)$.