Data Structures and Algorithms: Lecture 4

Barbara Morawska

August 14, 2018

Solving recurrencies - substitution method

- Guess the form of the solution
- Use mathematical induction to find constants and to show that this solution works

Example

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

- Guess: $T(n) = O(n \lg n)$
- To prove: $T(n) \leq cn \lg n$ for a constant $c > 0$ and $n \geq n_0$

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

Guess: $T(n) = O(n \lg n)$
To prove: $T(n) \leq cn \lg n$ for a constant $c > 0$ and $n \geq n_0$

Let us start with induction step:

- Induction hypothesis: for all m such that $0 \leq m < n$: $T(m) \leq cm \lg m$
- Induction step:

$$
\begin{aligned}
T(n) &= 2T(\lfloor n/2 \rfloor) + n \\
&\leq_{IH} 2c\lfloor n/2 \rfloor \lg \lfloor n/2 \rfloor + n \\
&\leq cn \lg n/2 + n \\
&= cn \lg n - cn \lg 2 + n \\
&= cn \lg n - cn + n \\
&\leq cn \lg n \qquad \text{where } c \geq 1, n \geq 1
\end{aligned}
$$

- This is true if the induction hypothesis is correct for $m < n$. Check the base case!

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

Guess: $T(n) = O(n \lg n)$
To prove: $T(n) \leq cn \lg n$ for a constant $c > 0$ and $n \geq n_0$

Base case

- If $n = 1$, $T(1) = 1$, but $c \cdot 1 \lg 1 = 0$! Hence $n_0 > 1$.
- If $n = 2$, $T(2) = 2T(\lfloor 2/2 \rfloor) + 2 = 2 + 2 = 4$
  Now, $4 = T(2) \leq c2 \lg 2 = c \cdot 2$
- If $n = 3$, $T(3) = 2T(\lfloor 3/2 \rfloor) + 3 = 2 + 3 = 5$
  Now, $5 = T(3) \leq c3 \lg 3$
- Hence $c \geq 2$ and
  $4 = T(2) \leq 2 \cdot 2 \lg 2 = 4 \lg 2$
  $5 = T(3) \leq 2 \cdot 3 \lg 3 = 6 \lg 3$

Solving $T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 1$

Guess: $T(n) = O(n)$
Need to prove: $T(n) \leq cn$ for a constant $c > 0$ and $n \geq n_0$.

Induction argument

- Induction hypothesis: for all $0 \leq m < n$, $T(m) \leq cm$.
- Induction step:

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

This is not what we want! We need stronger induction hypothesis:
$T(m) \leq cm - d$ for a constant $d \geq 0$.

Obviously $cn - d \in O(n)$.

Solving $T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 1$

Guess: $T(n) = O(n)$
Need to prove: $T(n) \leq cn - d$ for constants $c > 0$, $d \geq 0$ and $n \geq n_0$.

Induction argument

- Induction hypothesis: for all $0 \leq m < n$, $T(m) \leq cm - d$.
- Induction step:

$$\begin{aligned}
T(n) &= T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 1 \\
&\leq c \cdot \lfloor n/2 \rfloor - d + c \cdot \lceil n/2 \rceil - d + 1 \\
&= cn - 2d + 1 \\
&\leq cn - d \qquad\qquad \text{for } d \geq 1
\end{aligned}$$

In the induction step we have to prove the exact form of the induction hypothesis for n.

Solving $T(n) = 2T(\lfloor\sqrt{n}\rfloor) + \lg n$

Change variables

- Let $m = \lg n$
- $T(2^m) = 2T(2^{m/2}) + m$
- Change variables: $T(2^m)$ to $S(m)$
- $S(m) = 2S(m/2) + m$
- Hence $S(m) = O(m \lg m)$
- Hence $T(n) = T(2^m) = S(m) = O(m \lg m) = O(\lg n \cdot \lg \lg n)$

Problem: how to make a good guess?

- No general advise
- Find a similar recurrence which is already solved.
  E.g. $T(n) = 2T(\lfloor n/2 \rfloor + 17) + n$
  Guess: $T(n) = O(n \lg n)$?
- Guess loose lower bound e.g. $T(n) = \Omega(n)$
  and loose upper bound e.g. $T(n) = O(n^2)$
  and try to tighten it.
- Use recursion tree to find a good guess.

Recursion tree method

How a recursion tree can help in providing a good guess for a recurrence?

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

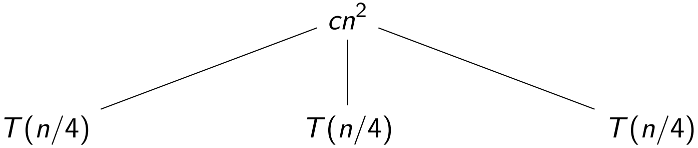
We ignore floor!

We create a recursion tree for $T(n) = 3T(n/4) + cn^2$

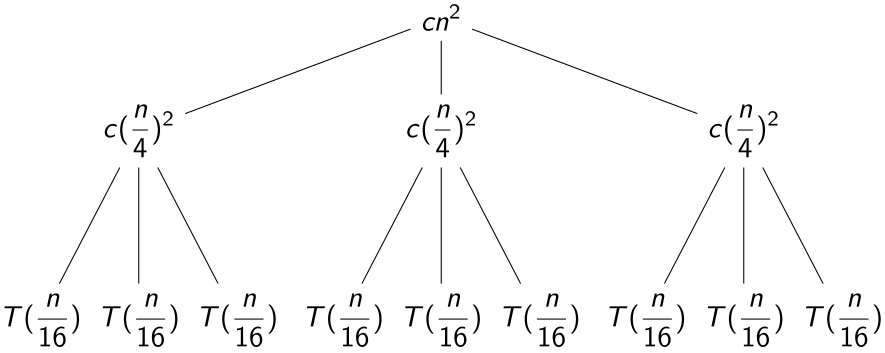
(c is a constant from $\Theta(n^2)$, $c > 0$)

We assume n is exact power of 4 (division by 4 always gives integer).

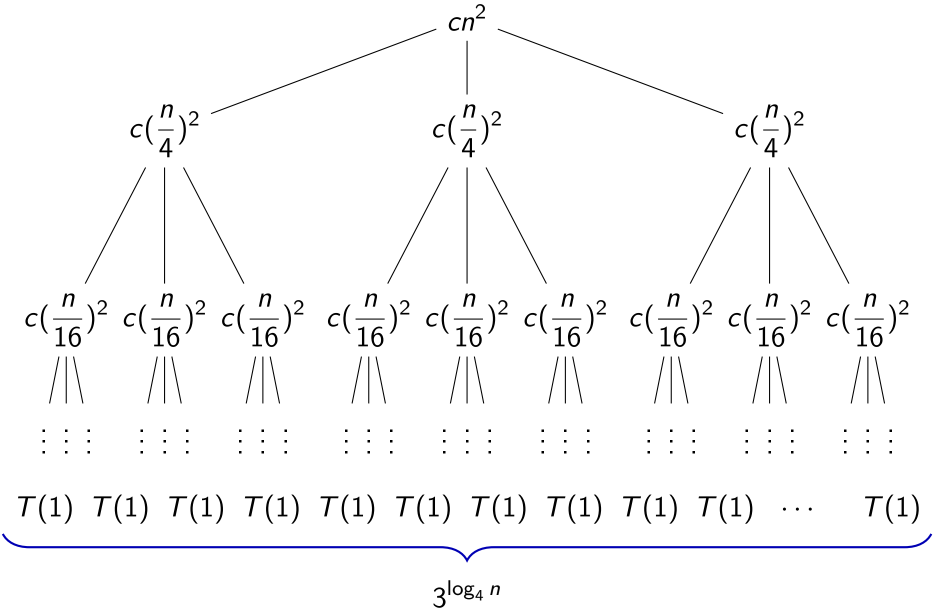
Recursion tree for $T(n) = 3T(n/4) + cn^2$



Recursion tree for $T(n) = 3T(n/4) + cn^2$



Recursion tree for $T(n) = 3T(n/4) + cn^2$



Analysis of costs: to get the good guess

- Height of the tree:
  - $n/4^x = 1$
  - Hence the height is: $x = \log_4 n$
  - Hence $\log_4 n + 1$ levels $(0, 1, 2, \ldots, \log_4 n)$
  - Compute cost at each level and add them.
  - Notice: $3^{\log_4 n} = n^{\log_4 3}$.

$$
\begin{aligned}
T(n) &= cn^2 + \frac{3}{16}cn^2 + (\frac{3}{16})^2 cn^2 + \cdots + (\frac{3}{16})^{\log_4 n - 1} cn^2 + \Theta(n^{\log_4 3}) \\
&= \Sigma_{i=0}^{\log_4 n - 1} (\frac{3}{16})^i cn^2 + \Theta(n^{\log_4 3}) \\
&< \Sigma_{i=0}^{\infty} (\frac{3}{16})^i cn^2 + \Theta(n^{\log_4 3}) \qquad \text{decreasing geometric series} \\
&= \frac{1}{1 - \frac{3}{16}} cn^2 + \Theta(n^{\log_4 3}) \\
&= \frac{16}{13} cn^2 + \Theta(n^{\log_4 3}) \\
&= O(n^2)
\end{aligned}
$$

Use substitution method to verify that $T(n) = O(n^2)$

Notice: if $O(n^2)$ is the upper bound of a recurrence, and the first call contributes $\Theta(n^2)$, then $\Omega(n^2)$ must be the lower bound and the complexity is tight.

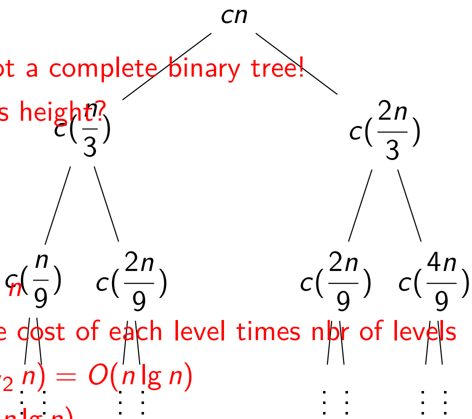
Recurrence: $T(n) = 3T(\lfloor n/4 \rfloor) + \Theta(n^2)$

Induction hypothesis: $T(m) \leq dm^2$, $0 \leq m < n$

$$\begin{aligned} T(n) &\leq 3T(\lfloor n/4 \rfloor) + cn^2 \\ &\leq 3d\lfloor n/4 \rfloor^2 + cn^2 \\ &\leq 3d(n/4)^2 + cn^2 \\ &= (3/16)dn^2 + cn^2 \\ &\leq dn^2 \qquad \text{for } d \geq (16/13)c \end{aligned}$$

Another example of recurrence

$$T(n) = T(n/3) + T(2n/3) + O(n)$$



- Notice: not a complete binary tree!
- What is its height?
- $(\frac{2}{3})^h n = 1$
- $n = (\frac{3}{2})^h$
- $h = \log_{3/2} n$
- Expect the cost of each level times nbr of levels
- $O(cn \log_{3/2} n) = O(n \lg n)$
- Guess: $O(n \lg n)$

Solving $T(n) = T(n/3) + T(2n/3) + O(n)$

$$\begin{aligned}
T(n) &= T(n/3) + T(2n/3) + cn \\
&\le d(n/3)\lg(n/3) + d(2n/3)\lg(2n/3) + cn \\
&= d(n/3)\lg n - d(n/3)\lg 3 + d(2n/3)\lg n - d(2n/3)\lg 3/2 + cn \\
&= dn\lg n - d((n/3)\lg 3 + (2n/3)\lg 3 - (2n/3)\lg 2) + cn \\
&= dn\lg n - dn(\lg 3 - 2/3) + cn \\
&\le dn\lg n
\end{aligned}$$

where $d \ge c/\lg(3 - (2/3))$

Master method for solving recurrences

Applies to the recurrences of the form:

$$T(n) = aT(n/b) + f(n)$$

where

- $a \geq 1, b > 1$,
- $f(n)$ is positive asymptotically,
- floor or ceiling of n/b is ignored.

Master theorem

Theorem

Let $a \geq 1, b > 1$ *be constants,* $f(n)$ *a function.*
$T(n) = aT(n/b) + f(n)$ *for* $n \geq 0$. *Then*

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

When Master Theorem cannot be used

There are gaps between the three cases of the theorem:

- **Gap between case 2 and case 1**: if $f(n)$ is smaller than $n^{\log_b(a)}$ but not polynomially smaller,
- **Gap between case 2 and case 3**: if $f(n)$ is bigger than $n^{\log_b(a)}$ but not polynomially bigger.
- **Regularity condition** does not hold: $af(n/b) \leq cf(n)$, $c < 1$.

Examples

$T(n) = 9T(n/3) + n$

$a = 9, b = 3, f(n) = n$

- $n^{\log_b a} = n^{\log_3 9} = n^2 = \Theta(n^2)$
- $f(n) = n = O(n^{\log_3 9 - \epsilon})$ for $\epsilon = 1$
- $n^{\log_3 9 - 1} = n^{2-1} = n^1 = n$

Hence case 1: $T(n) = \Theta(n^{\log_3 9}) = \Theta(n^2)$

Examples

$T(n) = T(2n/3) + 1$

$a = 1, b = 3/2, f(n) = 1$

- $n^{\log_b a} = n^{\log_{3/2} 1} = n^0 = 1$
- $f(n) = 1 = \Theta(1)$

Hence case 2: $T(n) = \Theta(n^{\log_{3/2} 1} \lg n) = \Theta(\lg n)$

Examples

$T(n) = 3T(n/4) + n \lg n$

$a = 3, b = 4, f(n) = n \lg n$

- $n^{\log_b a} = n^{\log_4 3} = O(n^{0.793})$
- $f(n) = \Omega(n^{\log_4 3+\epsilon})$ for $\epsilon \approx 0.2$
- Regularity condition:
  $af(n/b) = 3(n/4)\lg(n/4) \leq 3/4 n \lg n = cn \lg n$ where $c = 3/4$

Hence case 3: $T(n) = \Theta(n \lg n)$

Examples

$T(n) = 2T(n/2) + n \lg n$

$a = 2, b = 2, f(n) = n \lg n$

- $n^{\log_b a} = n^{\log_2 2} = n$
- $f(n) = n \lg n = \Omega(n)$, hence $f(n)$ is larger than $n^{\log_2 2} = n$ (case 3?). But $f(n)$ is not polynomially larger than n. For any $\epsilon > 0$, $n \lg n$ is smaller than nn^{ϵ}
- Regularity condition:
  $af(n/b) = 3(n/4)\lg(n/4) \leq 3/4n \lg n = cn \lg n$ where $c = 3/4$

Hence Master Theorem does not apply.

Examples

Recurrence for Merge-sort, Max-subarray recursive algorithms

$T(n) = 2T(n/2) + \Theta(n)$

$a = 2, b = 2, f(n) = \Theta(n)$

- $n^{\log_b a} = n^{\log_2 2} = n$
- $f(n) = \Theta(n^{\log_b a}) = \Theta(n)$

Hence case 2 applies. $T(n) = \Theta(n \lg n)$.

Examples

Recurrence for matrix-multiplication recursive algorithm

$T(n) = 8T(n/2) + \Theta(n^2)$

$a = 8, b = 2, f(n) = \Theta(n^2)$

- $n^{\log_b a} = n^{\log_2 8} = n^3$
- $f(n) = \Theta(n^2)$, hence $f(n)$ is polynomially smaller than $n^{\log_b a}$: $f(n) = O(n^{3-\epsilon})$, where $\epsilon = 1$

Hence case 1 applies. $T(n) = \Theta(n^{\log_b a}) = \Theta(n^3)$.

Examples

Recurrence for Strassen's algorithm

$T(n) = 7T(n/2) + \Theta(n^2)$

$a = 7, b = 2, f(n) = \Theta(n^2)$

- $n^{\log_b a} = n^{\log_2 7} = n^{\lg 7}$ and $2.80 < \lg 7 < 2.81$
- $f(n) = \Theta(n^2) = O(n^{\lg 7 - \epsilon})$, where $\epsilon = 0.8$

Hence case 1 applies. $T(n) = \Theta(n^{\log_b a}) = \Theta(n^{\lg 7})$.

Reaction of the Class Topper

When back benchers give the right answer