CS 255, Spring 2014, SJSU

Recurrences

Fernando Lobo

Solving recurrences

- Iterative method
- Recursion-tree method
- Substitution method
- Master method

Recurrences

- A recurrence is a function defined in terms of:
  - 1 or more base cases.
  - itself with smaller arguments.
- They show up naturally in the analysis of recursive algorithms.
- Need to learn how to solve them.

A simple example

```
FACTORIAL(n)
  if n = 0
      return 1
  else
      return n × FACTORIAL(n-1)
```

$$T(n) = \begin{cases} \Theta(1) & \text{, se } n = 0 \\ T(n-1) + \Theta(1) & \text{, se } n > 0 \end{cases}$$

$$= \begin{cases} k_1 & \text{, se } n = 0 \\ T(n-1) + k_2 & \text{, se } n > 0 \end{cases}$$

with k_1 and k_2 constants.

Solution

$$
\begin{aligned}
T(n) &= T(n-1) + k_2 \\
&= T(n-2) + k_2 + k_2 \\
&= T(n-3) + k_2 + k_2 + k_2 \\
&= \ldots \\
&= T(n-n) + \underbrace{k_2 + k_2 + \ldots + k_2}_{\text{n times}} \\
&= k_1 + k_2 n \\
&= \Theta(n)
\end{aligned}
$$

- Simple cases can be solved like that, iterating until we hit a base case.
- For more complex cases (ex: MERGE-SORT) the iterative method gets a little messy.

Substitution method

1. Guess the form of the solution.
2. Verify it using mathematical induction.

Substitution method

Example

$$T(n) = 4T\left(\frac{n}{2}\right) + n$$

- Let's assume $T(1) = \Theta(1)$.
- Guess: $T(n) = O(n^3)$, i.e., $T(n) \leq cn^3$ for some $c > 0, n > n_0$.
- Start by assuming that $T(k) \leq ck^3$, for $k < n$
- Try to prove by induction that $T(n) \leq cn^3$.

$$
\begin{aligned}
T(n) &= 4T\left(\frac{n}{2}\right) + n \\
&\leq 4c\left(\frac{n}{2}\right)^3 + n \\
&= \frac{c}{2}n^3 + n \\
&= \underbrace{cn^3}_{\text{desired}} - \underbrace{\left(\frac{c}{2}n^3 - n\right)}_{\textit{residual}} \\
&\leq cn^3 \quad \text{if } \frac{c}{2}n^3 - n > 0.
\end{aligned}
$$

For example: $c \geq 2$ and any $n_0 \geq 1$

- Need to verify the base case to complete the proof.
- Base: $T(n) = \Theta(1)$ for $n < \underbrace{n_0}_{constant}$

For $1 \leq n < n_0$,

$\Theta(1) \leq cn^3 \Rightarrow k \leq cn^3$.

In this case it's always possible to choose some c sufficiently large (greater than k).

- $O(n^3)$ is not a tight bound.
- Let's try $O(n^2)$.
- Inductive hypothesis: $T(k) \leq ck^2$ for $k < n$

$$\begin{aligned} T(n) &= 4T\left(\frac{n}{2}\right) + n \\ &\leq 4c\left(\frac{n}{2}\right)^2 + n \\ &= cn^2 + n \\ &= \cancel{O}(n^2) \text{ WRONG! Must prove inductive hypothesis} \end{aligned}$$

$$\begin{aligned} T(n) &= cn^2 - (-n) \\ &\leq cn^2 \quad \text{if } -n < 0 \quad \boxed{\text{Doesn't work!}} \end{aligned}$$

Solution: Subtract a lower order term

- Inductive hypothesis: $T(k) \leq c_1 k^2 - c_2 k$ for $k < n$

$$\begin{aligned} T(n) &= 4T\left(\frac{n}{2}\right) + n \\ &\leq 4\left[c_1\left(\frac{n}{2}\right)^2 - c_2\frac{n}{2}\right] + n \\ &= c_1 n^2 - 2c_2 n + n \\ &= \underbrace{c_1 n^2 - c_2 n}_{desired} - \underbrace{(c_2 n - n)}_{residual} \\ &\leq c_1 n^2 - c_2 n \quad \text{if } c_2 n - n \geq 0 \end{aligned}$$

Choose $c_2 \geq 1$, and c_1 in such a way that the base case is satisfied.

Recursion-tree method

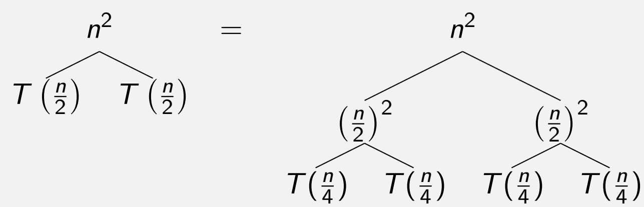
- Intuitive method (like the iterative method), but not formal.
- It's usually used to guess the form of the solution (which then we can try to prove using the substitution method.)

Recursion-tree method

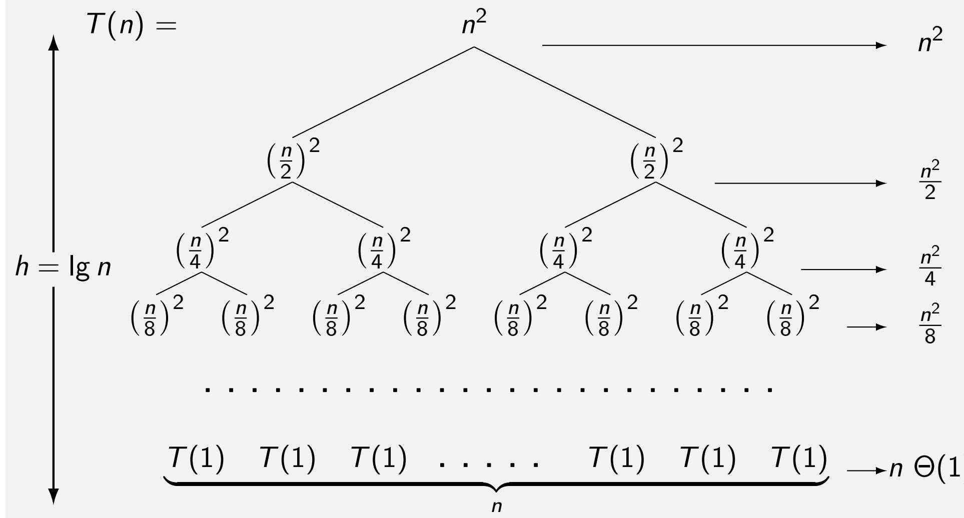
Example

$$T(n) = \begin{cases} \Theta(1) & \text{, if } n = 1 \\ 2T\left(\frac{n}{2}\right) + n^2 & \text{, if } n > 1 \end{cases}$$

$T(n) =$



Recursion-tree method



Total: $\Theta(n^2)$

Sum level by level:

$$n^2 \underbrace{\left(1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \ldots + \frac{1}{2^{h-1}}\right)}_{\text{G.P. with } \lg n \text{ terms and } r = \frac{1}{2}} + nT(1)$$

$$\begin{aligned} &= n^2 \times \frac{1 - \left(\frac{1}{2}\right)^{\lg n}}{1 - \frac{1}{2}} + n\ \Theta(1) \\ &= 2n^2\left(1 - \frac{1}{n}\right) + \Theta(n) \\ &= \Theta(n^2) \end{aligned}$$

Review:

- Sum of the first n terms of a geometric progression:

$$1 + x + x^2 + \ldots + x^{n-1} = \frac{1 - x^n}{1 - x} \quad , x \neq 1$$

- Sum of the geometric series:

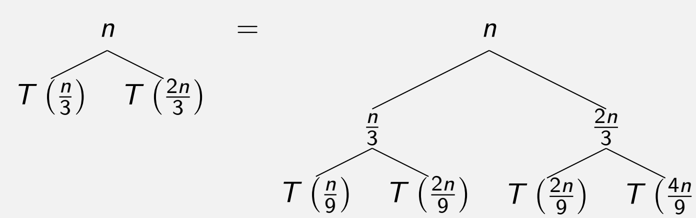
$$1 + x + x^2 + \ldots = \frac{1}{1 - x} \quad , |x| < 1$$

Recursion-tree method

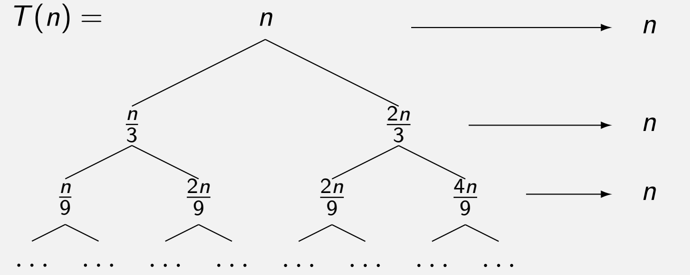
Another example

$$T(n) = T\left(\frac{n}{3}\right) + T\left(\frac{2n}{3}\right) + n$$

$T(n) =$



$T(n) =$



Sum level by level: $\leq n(1 + \log_{\frac{3}{2}} n)$

$= O(n \lg n)$

- The rightmost branch takes longer to hit the base case.
- Keeps dividing by $\frac{3}{2}$. Reaches 1 when,

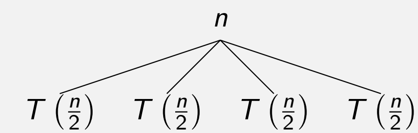
$$\left(\frac{2}{3}\right)^h n = 1 \implies h = \log_{\frac{3}{2}} n$$

Recursion-tree method

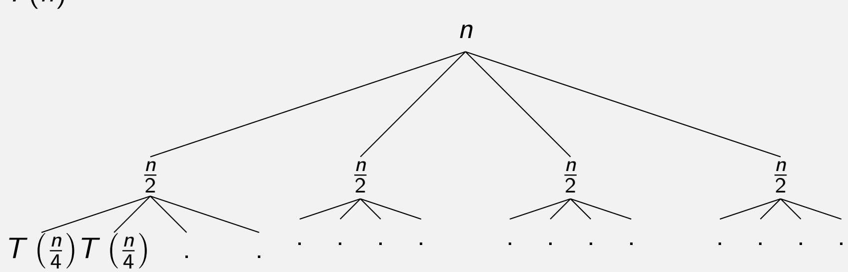
Yet another example

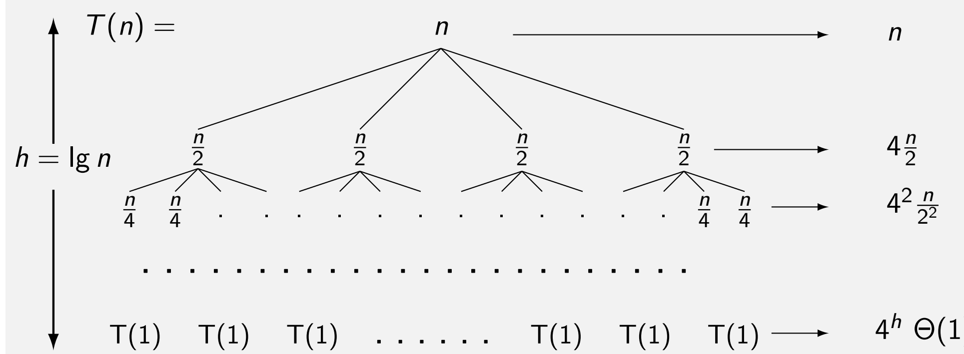
$$T(n) = 4T\left(\frac{n}{2}\right) + n$$

$T(n) =$



$T(n) =$





Sum level by level

$$\begin{aligned} &= n(\underbrace{1+2+4+8+\ldots+2^{h-1}}_{\text{G.P. with } h = \lg n \text{ terms and } r=2}) + 4^h\Theta(1) \\ &= n \times \frac{1-2^{\lg n}}{1-2} + n^2\ \Theta(1) \\ &= -n(1-n) + \Theta(n^2) \\ &= \Theta(n^2) \end{aligned}$$

Master Method

- Can be used to solve recurrences of the form:

$$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$

  with $a \geq 1, b > 1, f(n) > 0$ for sufficiently large n

- Based on the *Master Theorem*.

- 3 cases. Need to compare

$$\boxed{n^{\log_b a} \text{ with } f(n)}$$

Master Theorem: Case 1

Case 1: If $f(n) = O\left(n^{\log_b a - \epsilon}\right)$ for some $\epsilon > 0$
($f(n)$ is polynomially smaller than $n^{\log_b a}$)

Then:

$$\boxed{T(n) = \Theta(n^{\log_b a})}$$

Intuition: cost is dominated by the leaves of the recursion tree.

Master Theorem: Case 2

Case 2: If $f(n) = \Theta\left(n^{\log_b a}\right)$

Then:

$$\boxed{T(n) = \Theta(n^{\log_b a} \lg n)}$$

Intuition: cost is identical for all levels of the recursion tree.

Master Theorem: Case 3

Case 3:

- If $f(n) = \Omega\left(n^{\log_b a + \epsilon}\right)$ for some $\epsilon > 0$
  ($f(n)$ is polynomially larger than $n^{\log_b a}$)
- and $af(\frac{n}{b}) \leq cf(n)$ for some $c < 1$ and for all sufficiently large n (*Regularity Condition*)

Then:

$$\boxed{T(n) = \Theta(f(n))}$$

Intuition: cost is dominated by the root of the recursion tree.

Regularity condition is always true for $f(n) = n^k$.

Master Method

Example 1

$$T(n) = 4T(\tfrac{n}{2}) + n$$

$$a \qquad b \qquad f(n)$$

Compare $n^{log_b a}$ with $f(n)$

$$n^{log_2 4} = n^2 \qquad n$$

Case 1: $f(n) = n = O(n^{2-\epsilon})$ for $\epsilon = 1$, therefore

$$\boxed{T(n) = \Theta(n^2)}$$

Master Method

Example 2

$$T(n) = 4T(\tfrac{n}{2}) + n^2 \quad \text{f(n)}$$

$$n^{log_b a} = n^{log_2 4} = n^2 \qquad n^2$$

Case 2: $f(n) = n^2 = \Theta(n^2)$, therefore

$$\boxed{T(n) = \Theta(n^2 \lg n)}$$

Master Method

Example 3

$$T(n) = 4T(\tfrac{n}{2}) + n^3 \quad \text{f(n)}$$

$$n^{log_b a} = n^{log_2 4} = n^2 \qquad n^3$$

Case 3:

- $f(n) = n^3 = \Omega(n^{2+\epsilon})$ para $\epsilon = 1$
- Regularity Condition:

$$4\left(\frac{n}{2}\right)^3 \leq cn^3 \Leftrightarrow \frac{4n^3}{8} \leq cn^3$$

$$\Leftrightarrow \frac{1}{2}n^3 \leq cn^3 \quad \text{, can choose } c \geq \frac{1}{2}$$

- No need to show the Reg. Cond. because $f(n)$ is polynomial.

Therefore,

$$\boxed{T(n) = \Theta(n^3)}$$

Master Method

Example 4

$$T(n) = 4T(\tfrac{n}{2}) + \frac{n^2}{\lg n} \quad \text{f(n)}$$

$$n^{log_b a} = n^{log_2 4} = n^2 \qquad \frac{n^2}{\lg n}$$

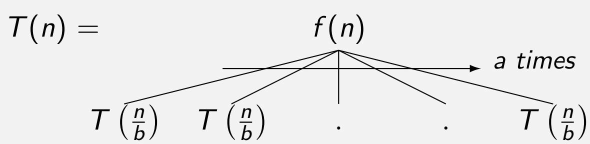
$$\boxed{\text{Master method not applicable!}}$$

Master Theorem (intuition)

- Formal proof in your textbook.
- This type of recurrence occurs in most D&C algorithms:

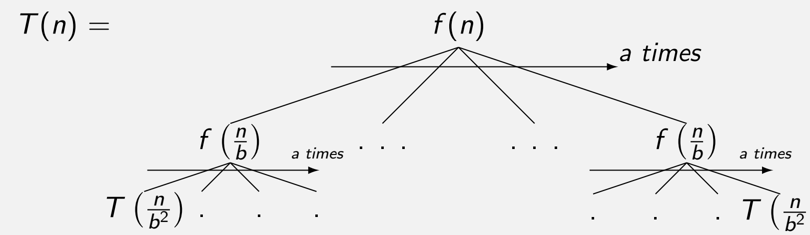
$$T(n) = \begin{cases} \Theta(1) & \text{, if } n = 1 \\ aT(\frac{n}{b}) + f(n) & \text{, if } n > 1 \end{cases} \qquad \text{with } a \geq 1, b > 1$$

number of subproblems — *size of each subproblem* — *non-recursive work*



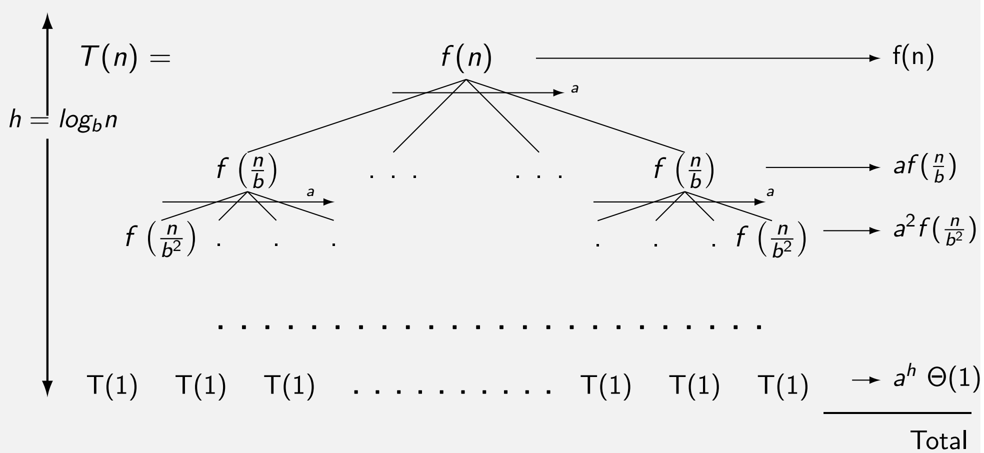
Master Theorem (intuition)

Iterating:



Master Theorem (intuition)

Iterating:



- Tree height is $h = \log_b n$
- The last level has a^h subproblems of size 1.

$$\begin{aligned} a^h\ \Theta(1) &= a^{\log_b n}\ \Theta(1) \\ &= n^{\log_b a}\ \Theta(1) \\ &= \Theta(n^{\log_b a}) \end{aligned}$$

<u>Sum level by level:</u>

$$\text{Total} = \underbrace{f(n) + af\left(\frac{n}{b}\right) + a^2 f\left(\frac{n}{b^2}\right) + .. + a^{h-1} f\left(\frac{n}{b^{h-1}}\right)} + \Theta(n^{\log_b a})$$

$$T(n) = \sum_{i=0}^{\log_b n - 1} a^i f\left(\frac{n}{b^i}\right) + \Theta(n^{\log_b a})$$

If $f(n)$ is polynomial, i.e., $f(n) = n^k$ for some fixed k, then:

$$\begin{aligned}
T(n) &= \sum_{i=0}^{\log_b n-1} a^i \left(\frac{n}{b^i}\right)^k + \Theta(n^{\log_b a}) \\
&= n^k \times \underbrace{\sum_{i=0}^{log_b n-1} \left(\frac{a}{b^k}\right)^i}_{G.P.\text{with } \log_b n \text{ terms and } r=\frac{a}{b^k}} + \Theta(n^{\log_b a})
\end{aligned}$$

From here we get:

- If $r > 1$

$$\log_b a > k \Rightarrow T(n) = \Theta(n^{\log_b a}) \qquad \text{(Case 1)}$$

- If $r = 1$

$$\log_b a = k \Rightarrow T(n) = \Theta(n^k \log_b n) \qquad \text{(Case 2)}$$

- If $r < 1$

$$\log_b a < k \Rightarrow T(n) = \Theta(n^k) \qquad \text{(Case 3)}$$