Recurrences

Alexandre David
B2-206

Today

- Recurrences
- How to solve them:
  - Substitution method.
  - Recursion-tree method.
  - Master method.

Recurrences

- A **recurrence** is an equation or inequality that describes a function in terms of its value on smaller inputs.
  - E.g. Fibbonacci: $F_n=F_{n-1}+F_{n-2}$.
- Methods for solving recurrences:
  - Substitution method.
  - Recursion-tree method.
  - Master method.

The Substitution Method

- Two steps:
  - **Guess** the form of the solution.
  - Use **induction** to find the constants and prove that the solution works.
- Problem: To come up with a good guess.
  - Use recursion-trees.
  - Correct the guess.

The name comes from the substitution of the guessed answer for the function (solution) when the induction hypothesis is applied for smaller values (in the induction proof).

This can be used to establish lower or upper bounds.

Substitution Method - Example

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

- Guess: $T(n)=O(n \lg n)$.
- By definition of O(…) we have to prove $T(n) \leq cn \lg n$ for *some* constant c.
- Proof (by induction):
  - Induction hypothesis – prove "next".
  - Prove formula for first n – or find first n after which the formula holds.

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

substitution

$$T(n) \le 2(c\lfloor n/2 \rfloor \lg(\lfloor n/2 \rfloor)) + n$$

?

$$\le cn\lg(n/2) + n$$

$$= cn\lg n - cn\lg 2 + n$$

$$= cn\lg n - cn + n$$

$$\le cn\lg n \quad \text{for } c \ge 1$$

Substitution Method

- Important:
  - Assume the solution has some form *f(k)* up to some k.
  - Prove that it has **exactly the same form** *f(n)* for n.
- Continue the proof (boundary condition):
  - Assume $T(1)=1$ (for simplicity).
  - $T(1) \le c \lg 1$ – fails. $T(2)=4 \le 4 \lg 2$ – works for c = 4.
  - Boundary condition will often give a constraint on *c*.

Subtleties

- What if the guess is *almost* correct, i.e., it looks like it’s working but the induction hypothesis is not strong enough?
- Trick: Subtract a lower term.

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

Guess? O(n)

$$T(n) \le c(\lfloor n/2 \rfloor) + c(\lceil n/2 \rceil) + 1$$

$$= cn + 1$$

OOPS

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

Guess? O(n) but induction with *T(n)≤cn-b.*

$$T(n) \le c(\lfloor n/2 \rfloor) - b + c(\lceil n/2 \rceil) - b + 1$$

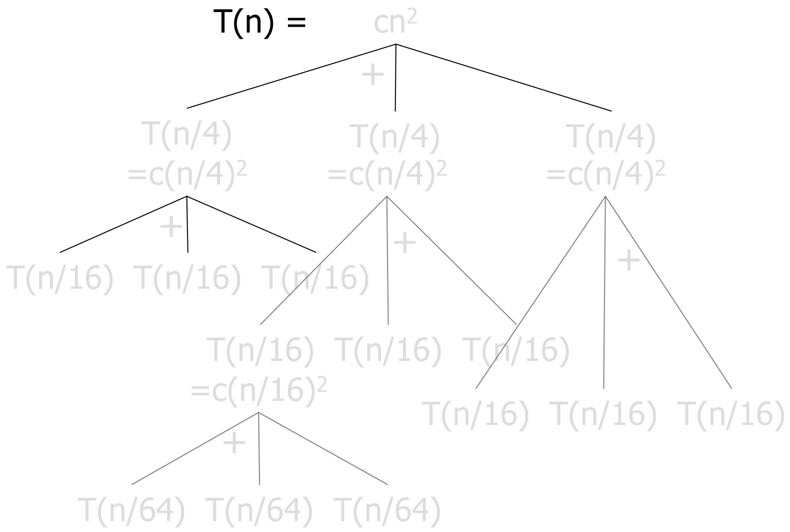
?

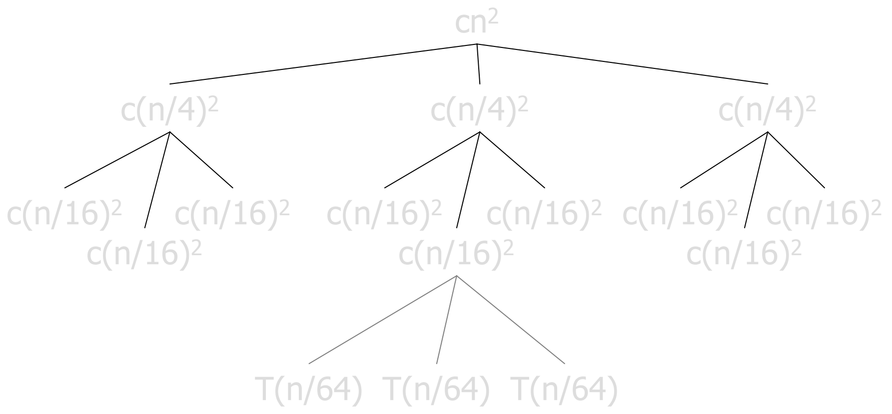
$$= cn - 2b + 1$$

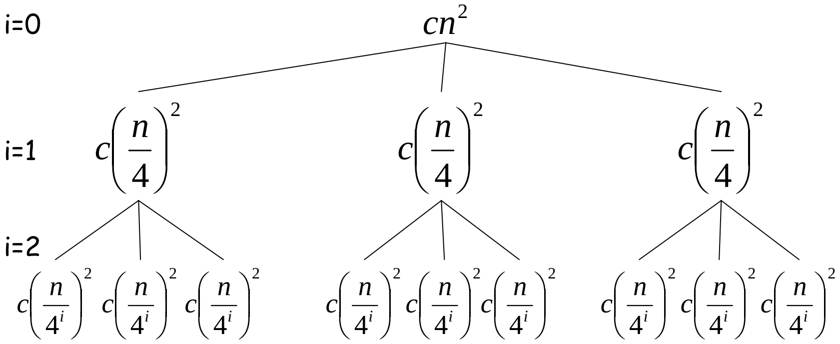
$$\le cn - b \qquad \text{for } b \ge 1$$

Recursion-Tree Method

- Each node represents the cost of a single sub-problem.
- Useful when it describes the running time of a divide-and-conquer algorithm.
- Used to generate a good guess or as a direct proof of a solution to a recurrence.
- Example: $T(n)=3T(n/4)+\Theta(n^2)$.
  - ? What does it mean?
  - Construct recursion tree to obtain a guess.
  - Use the substitution method for the proof.


T(n) =
cn2
+
T(n/4)
=c(n/4)2
T(n/4)
=c(n/4)2
T(n/4)
=c(n/4)2
T(n/16) T(n/16) T(n/16)
T(n/16) T(n/16) T(n/16)
=c(n/16)2
T(n/16) T(n/16) T(n/16)
T(n/64) T(n/64) T(n/64)


cn2
c(n/4)2 c(n/4)2 c(n/4)2
c(n/16)2 c(n/16)2 c(n/16)2 c(n/16)2 c(n/16)2 c(n/16)2
c(n/16)2 c(n/16)2 c(n/16)2
T(n/64) T(n/64) T(n/64)



$$\frac{n}{4^i} = 1 \Leftrightarrow n = 4^i \Leftrightarrow i = \log_4 n$$

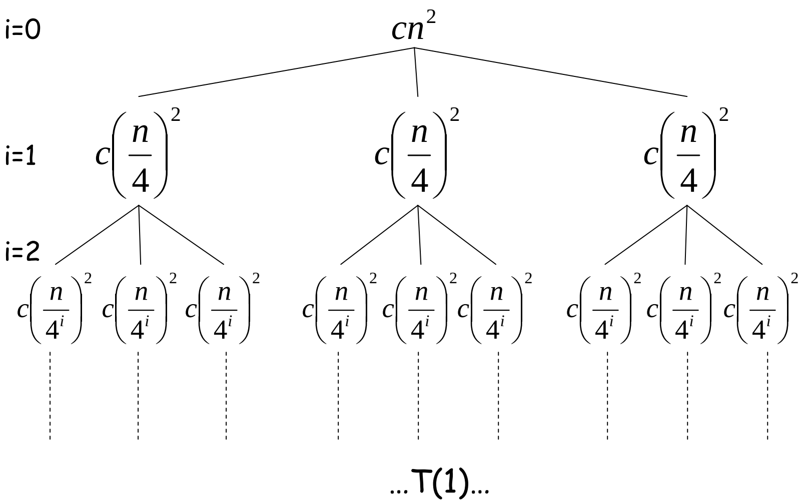
$\log_4 n+1$ levels

?

T(1)

level k has 3^k terms

last level: $\Theta(3^{\log_4 n})$



cn^2

$\frac{3}{16}cn^2$

$\left(\frac{3}{4^2}\right)^i cn^2$

$\Theta(3^{\log_4 n})$

$=Cn^2+\Theta(n^{\log_4 3})$

$=O(n^2)$

$3^{\log_4 n} = n^{\log_4 3}$:

$$\log_4(3^{\log_4 n}) = (\log_4 n)(\log_4 3) = \log_4(n^{\log_4 3})$$

The Master Method

- Apply to recurrences of the form
  $T(n)=aT(n/b)+f(n)$
  where $a \geq 1$ and $b>1$ are *constants* and $f(n)$ is asymptotically positive.
  - Don't re-invent the wheel every time.
  - General solved equations for different cases.
  - Intuition: Compare *$f(n)$* to $n^{\log_b a}$.
    - Polynomially larger/smaller (by a factor n^{ε}).

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

- ***T(n)*** is typically time to solve problem of size ***n***.
- An algorithm divides the problem of size *n* into ***a*** sub-problems of size ***n/b***, then combines the results, which costs ***f(n)***.
- Note 1: We omit the detail of floor/ceil.
- Note 2: All the cases are not covered ⇒ the master method does not solve all possible cases.

Understand what this recurrence means.

The Master Theorem

- If $f(n) = O(n^{\log_b a - \varepsilon})$ for some *ε>0* then

f(n) "smaller" than $n^{\log_b a}$ ⇒ $T(n) = \Theta(n^{\log_b a})$

- If $f(n) = \Theta(n^{\log_b a})$ then

f(n) "same" as $n^{\log_b a}$ ⇒ $T(n) = \Theta(f(n)\lg n)$

- If $f(n) = \Omega(n^{\log_b a + \varepsilon})$ for some *ε>0*
  and if $af(n/b) \leq cf(n)$ regularity condition
  for some *c<1* then

f(n) "larger" than $n^{\log_b a}$ ⇒ $T(n) = \Theta(f(n))$

Example

- *T(n)=9T(n/3)+n*.
  a=9, b=3, *f(n)=n*. $n^{\log_b a} = n^{\log_3 9} = n^2$
- Case 1 with ε=1:
  $f(n) = O(n^{\log_b a - \varepsilon}) = O(n^{2-1})$
  We conclude *T(n)=Θ(n²)*.

Example

- $T(n)=T(2n/3)+1$.
  a=1, b=2/3, $f(n)=1$. $n^{\log_b a} = n^{\log_{3/2} 1} = 1$
- Case 2:
  $f(n) = \Theta(n^{\log_b a}) = \Theta(1)$
  We conclude $T(n)=\Theta(\lg n)$.

Example

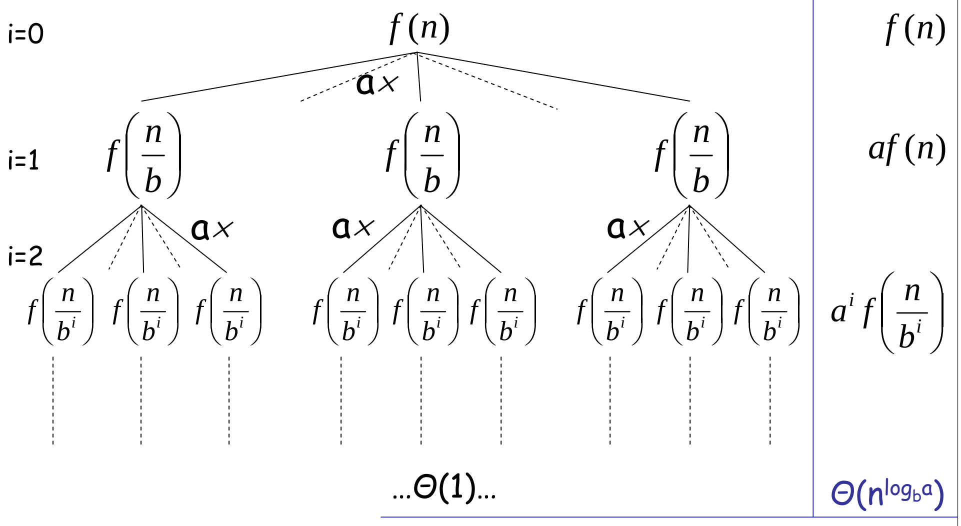
- *T(n)=3T(n/4)+n* lg*n*.
  a=3, b=4, *f(n)=n* lg*n*.
  $$n^{\log_b a} = n^{\log_4 3} = O(n^{0.793})$$
- Case 3 with ε=0.1 + check regularity:
  $f(n) = \Omega(n^{\log_b a+\varepsilon}) = \Omega(n^{0.793+0.1})$
  $af(n/b) = 3(n/4)\lg(n/4) \le cf(n) = (3/4)n\lg n$
  (c=3/4).
  We conclude *T(n)=Θ(n* lg*n)*.

Example

- $T(n)=2T(n/2)+n\lg n$.
  a=2, b=2, $f(n)=n\lg n$. $n^{\log_b a} = n$
- Case 3?
- Problem: $f(n)=n\lg(n)$ not polynomially larger than n: no $\varepsilon>0$ s.t. $n\lg n=\Omega(n^{1+\varepsilon})$. We cannot apply the theorem.

Master Theorem: Proof Idea

- Proof for a sub-domain (to simplify): $n=1,b,b^2,\ldots$
  - Compute the cost with a recursion tree (lemma 4.2):
    leaves + tree = $\Theta(n^{\log_b a}) + \sum_{j=0}^{\log_b n-1} a^j f(n/b^j)$
  - Bound the 2nd term with 3 cases (lemma 4.3).
  - Evaluate the sum asymptotically using lemma 4.3.
  - Extend the proof for any *n*.



$$\Theta(n^{\log_b a}) + \sum_{j=0}^{\log_b n-1} a^j f(n/b^j)$$

Lemma 4.3

- Bound the sum term.
- Proof not difficult, only technical.
  - Idea: Use hypothesis, substitute, and compute the sum.