CS 5413
Chapter 4

Recurrences

A **recurrence** is a function defined in terms of

- one or more base cases, and
- itself, with smaller arguments.

Examples:

- $$T(n) = \begin{cases} 1 & \text{if } n = 1\,, \\ T(n-1) + 1 & \text{if } n > 1\,. \end{cases}$$

  Solution: $T(n) = n$.

- $$T(n) = \begin{cases} 1 & \text{if } n = 1\,, \\ 2T(n/2) + n & \text{if } n \geq 1\,. \end{cases}$$

  Solution: $T(n) = n \lg n + n$.

- $$T(n) = \begin{cases} 0 & \text{if } n = 2\,, \\ T(\sqrt{n}) + 1 & \text{if } n > 2\,. \end{cases}$$

  Solution: $T(n) = \lg \lg n$.

- $$T(n) = \begin{cases} 1 & \text{if } n = 1\,, \\ T(n/3) + T(2n/3) + n & \text{if } n > 1\,. \end{cases}$$

  Solution: $T(n) = \Theta(n \lg n)$.

Many **technical issues**:

- Floors and ceilings
- Exact vs. asymptotic functions
- Boundary conditions

In algorithm analysis, we usually express both the recurrence and its solution using **asymptotic notation**.

- Example: $T(n) = 2T(n/2) + \Theta(n)$, with solution $T(n) = \Theta(n \lg n)$.
- The boundary conditions are usually expressed as "$T(n) = O(1)$ for suffi ciently small n."
- When we desire an exact, rather than an asymptotic, solution, we need to deal with boundary conditions.
- In practice, we just use asymptotics most of the time, and we ignore boundary conditions.

Substitution method

I. Guess the solution.

2. Use induction to find the constants and show that the solution works.

Example:

$$T(n) = \begin{cases} 1 & \text{if } n = 1 , \\ 2T(n/2) + n & \text{if } n > 1 . \end{cases}$$

1. *Guess:* $T(n) = n \lg n + n$.

2. *Induction:*

   Basis: $n = 1 \Rightarrow n \lg n + n = 1 = T(n)$

   Inductive step: Inductive hypothesis is that $T(k) = k \lg k + k$ for all $k < n$. We'll use this inductive hypothesis for $T(n/2)$.

$$\begin{aligned} T(n) &= 2T\left(\frac{n}{2}\right) + n \\ &= 2\left(\frac{n}{2}\lg\frac{n}{2} + \frac{n}{2}\right) + n \qquad \text{(by inductive hypothesis)} \\ &= n\lg\frac{n}{2} + n + n \\ &= n(\lg n - \lg 2) + n + n \\ &= n\lg n - n + n + n \\ &= n\lg n + n . \end{aligned}$$

■

Generally, we use **asymptotic notation**:

- We would write T (n) = 2T (n/2) + Θ(n).
- We assume T (n) = 0(1) for sufficiently small n.
- We express the solution by asymptotic notation: T (n) = Θ(n lg n).
- We don't worry about boundary cases, nor do we show base cases in the substitution proof.

    - T (n) is always constant for any constant n.
    - Since we are ultimately interested in an asymptotic solution to a recurrence, it will always be possible to choose base cases that work.
    - When we want an asymptotic solution to a recurrence, we don't worry about the base cases in our proofs.
    - When we want an exact solution, then we have to deal with base cases.

For the substitution method:

- Name the constant in the additive term.
- Show the upper (O) and lower (Ω) bounds separately. Might need to use different constants for each.

Example: $T(n) = 2T(n/2) + \Theta(n)$. If we want to show an upper bound of $T(n) = 2T(n/2) + O(n)$, we write $T(n) \le 2T(n/2) + cn$ for some positive constant c.

1. ***Upper bound:***

   Guess: $T(n) \le dn \lg n$ for some positive constant d. We are given c in the recurrence, and we get to choose d as any positive constant. It's OK for d to depend on c.

   Substitution:

$$\begin{aligned}
T(n) &\le 2T(n/2) + cn \\
&= 2\left(d\frac{n}{2}\lg\frac{n}{2}\right) + cn \\
&= dn\lg\frac{n}{2} + cn \\
&= dn\lg n - dn + cn \\
&\le dn\lg n \qquad \text{if } -dn + cn \le 0\,, \\
&\qquad\qquad\qquad\qquad\qquad d \ge c
\end{aligned}$$

   Therefore, $T(n) = O(n \lg n)$.

2. ***Lower bound:*** Write $T(n) \geq 2T(n/2) + cn$ for some positive constant c.

Guess: $T(n) \geq dn \lg n$ for some positive constant d.

Substitution:

$$
\begin{aligned}
T(n) &\geq 2T(n/2) + cn \\
&= 2\left(d\frac{n}{2}\lg\frac{n}{2}\right) + cn \\
&= dn\lg\frac{n}{2} + cn \\
&= dn\lg n - dn + cn \\
&\geq dn\lg n \qquad \text{if } -dn + cn \geq 0\,, \\
&\qquad\qquad\qquad\qquad\quad d \leq c
\end{aligned}
$$

Therefore, $T(n) = \Omega(n \lg n)$.

Therefore, $T(n) = \Theta(n \lg n)$.

■

Make sure you show the same *exact* form when doing a substitution proof.

Consider the recurrence

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

For an upper bound:

$$T(n) \le 8T(n/2) + cn^2 .$$

Guess: $T(n) \le dn^3$.

$$\begin{aligned} T(n) &\le 8d(n/2)^3 + cn^2 \\ &= 8d(n^3/8) + cn^2 \\ &= dn^3 + cn^2 \\ &\not\le dn^3 \qquad \text{doesn't work!} \end{aligned}$$

Remedy: *Subtract off* a lower-order term.

Guess: $T(n) \le dn^3 - d'n^2$.

$$
\begin{aligned}
T(n) &\le 8(d(n/2)^3 - d'(n/2)^2) + cn^2 \\
&= 8d(n^3/8) - 8d'(n^2/4) + cn^2 \\
&= dn^3 - 2d'n^2 + cn^2 \\
&= dn^3 - d'n^2 - d'n^2 + cn^2 \\
&\le dn^3 - d'n^2 \qquad \text{if } -d'n^2 + cn^2 \le 0\,, \\
&\qquad\qquad\qquad\qquad\qquad\quad d' \ge c
\end{aligned}
$$

Be careful when using asymptotic notation.

The false proof for the recurrence $T(n) = 4T(n/4) + n$, that $T(n) = O(n)$:

$$
\begin{aligned}
T(n) &\le 4(c(n/4)) + n \\
&\le cn + n \\
&= O(n) \qquad \text{wrong!}
\end{aligned}
$$

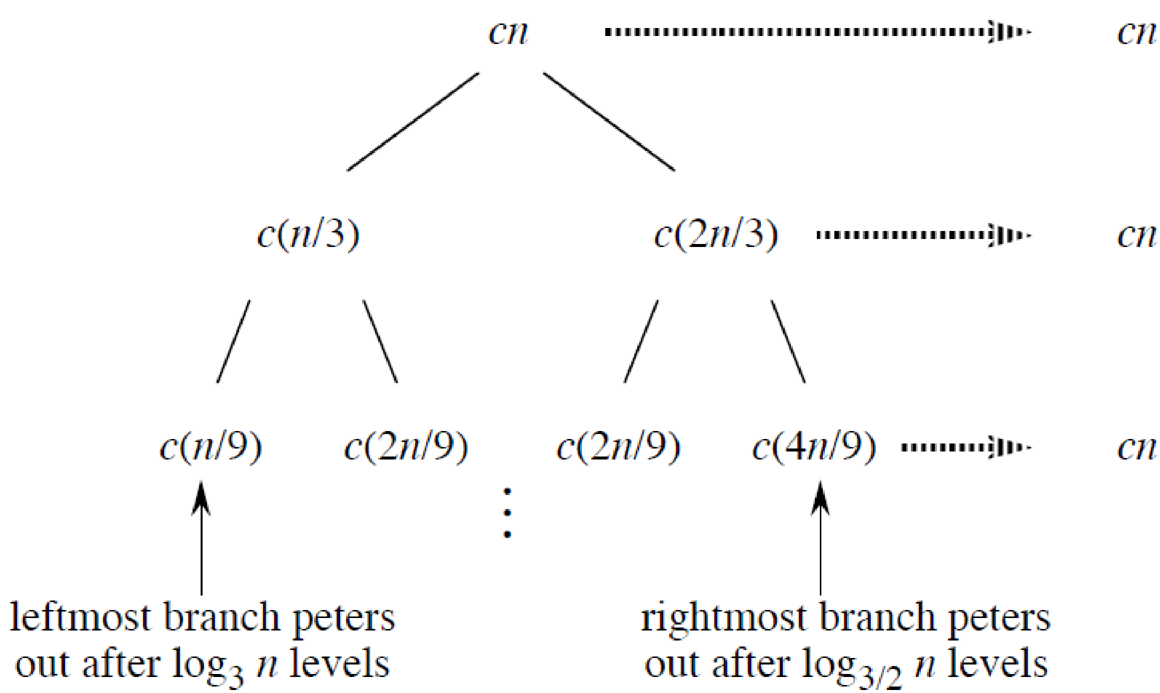
Because we haven't proven the *exact form* of our inductive hypothesis (which is that $T(n) \le cn$), this proof is false.

Recursion trees

Use to generate a guess. Then verify by substitution method.

Example: $T(n) = T(n/3) + T(2n/3) + \Theta(n)$. For upper bound, rewrite as $T(n) \le T(n/3) + T(2n/3) + cn$; for lower bound, as $T(n) \ge T(n/3) + T(2n/3) + cn$.

By summing across each level, the recursion tree shows the cost at each level of recursion (minus the costs of recursive calls, which appear in subtrees):



- There are $\log_3 n$ full levels, and after $\log_{3/2} n$ levels, the problem size is down to 1.
- Each level contributes $\leq cn$.
- Lower bound guess: $\geq dn \log_3 n = \Omega(n \lg n)$ for some positive constant d.
- Upper bound guess: $\leq dn \log_{3/2} n = O(n \lg n)$ for some positive constant d.
- Then *prove* by substitution.

1. ***Upper bound:***

Guess: $T(n) \le dn \lg n$.

Substitution:

$$
\begin{aligned}
T(n) &\le T(n/3) + T(2n/3) + cn \\
&\le d(n/3)\lg(n/3) + d(2n/3)\lg(2n/3) + cn \\
&= (d(n/3)\lg n - d(n/3)\lg 3) \\
&\qquad + (d(2n/3)\lg n - d(2n/3)\lg(3/2)) + cn \\
&= dn\lg n - d((n/3)\lg 3 + (2n/3)\lg(3/2)) + cn \\
&= dn\lg n - d((n/3)\lg 3 + (2n/3)\lg 3 - (2n/3)\lg 2) + cn \\
&= dn\lg n - dn(\lg 3 - 2/3) + cn \\
&\le dn\lg n \qquad \text{if } -dn(\lg 3 - 2/3) + cn \le 0\,, \\
&\qquad\qquad\qquad d \ge \frac{c}{\lg 3 - 2/3}\,.
\end{aligned}
$$

Therefore, $T(n) = O(n \lg n)$.

Note: Make sure that the symbolic constants used in the recurrence (e.g., c) and the guess (e.g., d) are different.

2. ***Lower bound:***

Guess: $T(n) \geq dn \lg n$.

Substitution: Same as for the upper bound, but replacing $\leq$ by $\geq$. End up needing

$$0 < d \leq \frac{c}{\lg 3 - 2/3} .$$

Therefore, $T(n) = \Omega(n \lg n)$.

Since $T(n) = O(n \lg n)$ and $T(n) = \Omega(n \lg n)$, we conclude that $T(n) = \Theta(n \lg n)$. ■

Master method

Used for many divide-and-conquer recurrences of the form

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

where $a \geq 1$, $b > 1$, and $f(n) > 0$.

Based on the ***master theorem*** (Theorem 4.1).

Compare $n^{\log_b a}$ vs. $f(n)$:

Case 1: $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$.

($f(n)$ is polynomially smaller than $n^{\log_b a}$.)

Solution: $T(n) = \Theta(n^{\log_b a})$.

(Intuitively: cost is dominated by leaves.)

Case 2: $f(n) = \Theta(n^{\log_b a} \lg^k n)$, where $k \geq 0$.

($f(n)$ is within a polylog factor of $n^{\log_b a}$, but not smaller.)
Solution: $T(n) = \Theta(n^{\log_b a} \lg^{k+1} n)$.
(Intuitively: cost is $n^{\log_b a} \lg^k n$ at each level, and there are $\Theta(\lg n)$ levels.)
Simple case: $k = 0 \Rightarrow f(n) = \Theta(n^{\log_b a}) \Rightarrow T(n) = \Theta(n^{\log_b a} \lg n)$.

Case 3: $f(n) = \Omega(n^{\log_b a+\epsilon})$ for some constant $\epsilon > 0$ and $f(n)$ satisfies the regularity condition $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n.

($f(n)$ is polynomially greater than $n^{\log_b a}$.)

Solution: $T(n) = \Theta(f(n))$.

(Intuitively: cost is dominated by root.)

What's with the Case 3 regularity condition?

- Generally not a problem.
- It always holds whenever $f(n) = n^k$ and $f(n) = \Omega(n^{\log_b a+\epsilon})$ for constant $\epsilon > 0$.

So you don't need to check it when $f(n)$ is a polynomial.

a proof that the regularity condition holds when $f(n) = n^k$ and $f(n) = \Omega(n^{\log_b a+\epsilon})$ for constant $\epsilon > 0$.

Since $f(n) = \Omega(n^{\log_b a+\epsilon})$ and $f(n) = n^k$, we have that $k > \log_b a$. Using a base of b and treating both sides as exponents, we have $b^k > b^{\log_b a} = a$, and so $a/b^k < 1$. Since a, b, and k are constants, if we let $c = a/b^k$, then c is a constant strictly less than 1. We have that $af(n/b) = a(n/b)^k = (a/b^k)n^k = cf(n)$, and so the regularity condition is satisfied.

Examples:

- $T(n) = 5T(n/2) + \Theta(n^2)$
  $n^{\log_2 5}$ vs. n^2
  Since $\log_2 5 - \epsilon = 2$ for some constant $\epsilon > 0$, use Case 1 $\Rightarrow T(n) = \Theta(n^{\lg 5})$

- $T(n) = 27T(n/3) + \Theta(n^3 \lg n)$
  $n^{\log_3 27} = n^3$ vs. $n^3 \lg n$
  Use Case 2 with $k = 1 \Rightarrow T(n) = \Theta(n^3 \lg^2 n)$

- $T(n) = 5T(n/2) + \Theta(n^3)$
  $n^{\log_2 5}$ vs. n^3
  Now $\lg 5 + \epsilon = 3$ for some constant $\epsilon > 0$
  Check regularity condition (don't really need to since $f(n)$ is a polynomial):
  $af(n/b) = 5(n/2)^3 = 5n^3/8 \le cn^3$ for $c = 5/8 < 1$
  Use Case 3 $\Rightarrow T(n) = \Theta(n^3)$

- $T(n) = 27T(n/3) + \Theta(n^3/\lg n)$
  $n^{\log_3 27} = n^3$ vs. $n^3/\lg n = n^3 \lg^{-1} n \ne \Theta(n^3 \lg^k n)$ for any $k \ge 0$.
  Cannot use the master method.

Section 4.4 of the text has the full proof of the master theorem.