

Example Problems

1. What does it mean if:

$$f(n) \neq O(g(n)) \quad \text{and} \quad g(n) \neq O(f(n)) \quad ???$$

2. Is $2^{n+1} = O(2^n)$?

Is $2^{2n} = O(2^n)$?

3. Does $f(n) = O(f(n))$?

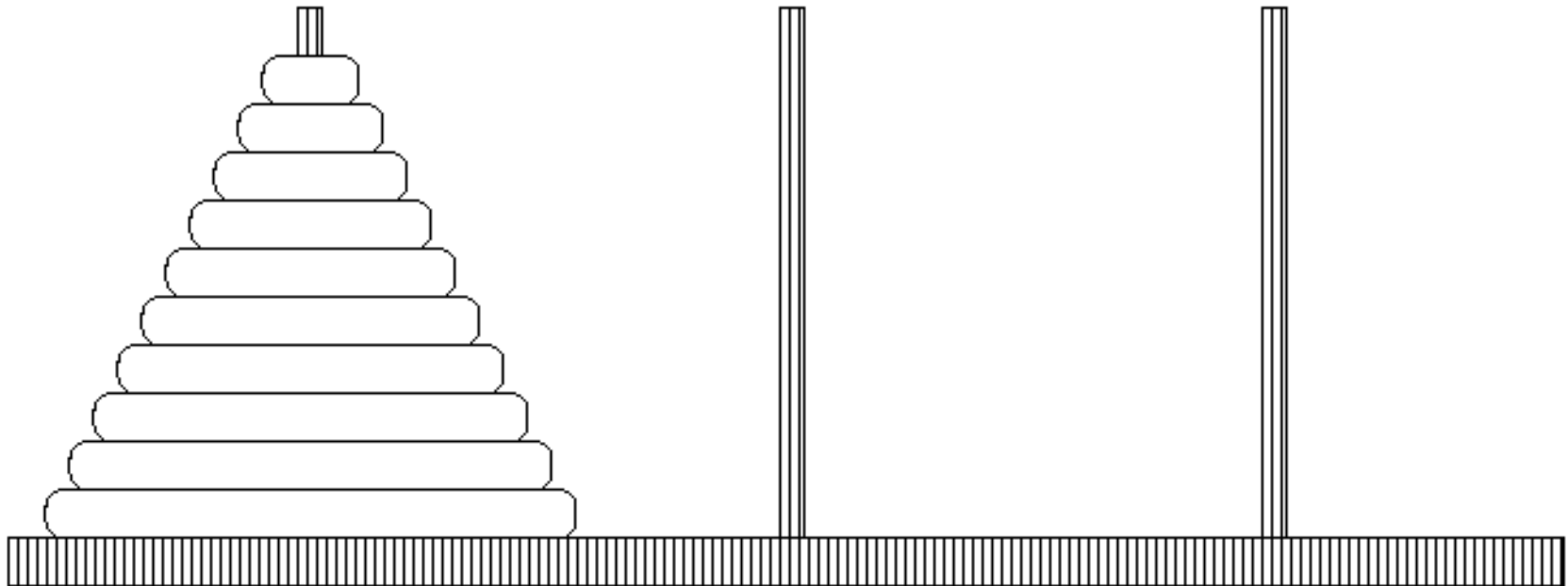
4. If $f(n) = O(g(n))$ and $g(n) = O(h(n))$,
can we say $f(n) = O(h(n))$?

Asymptotic Notation

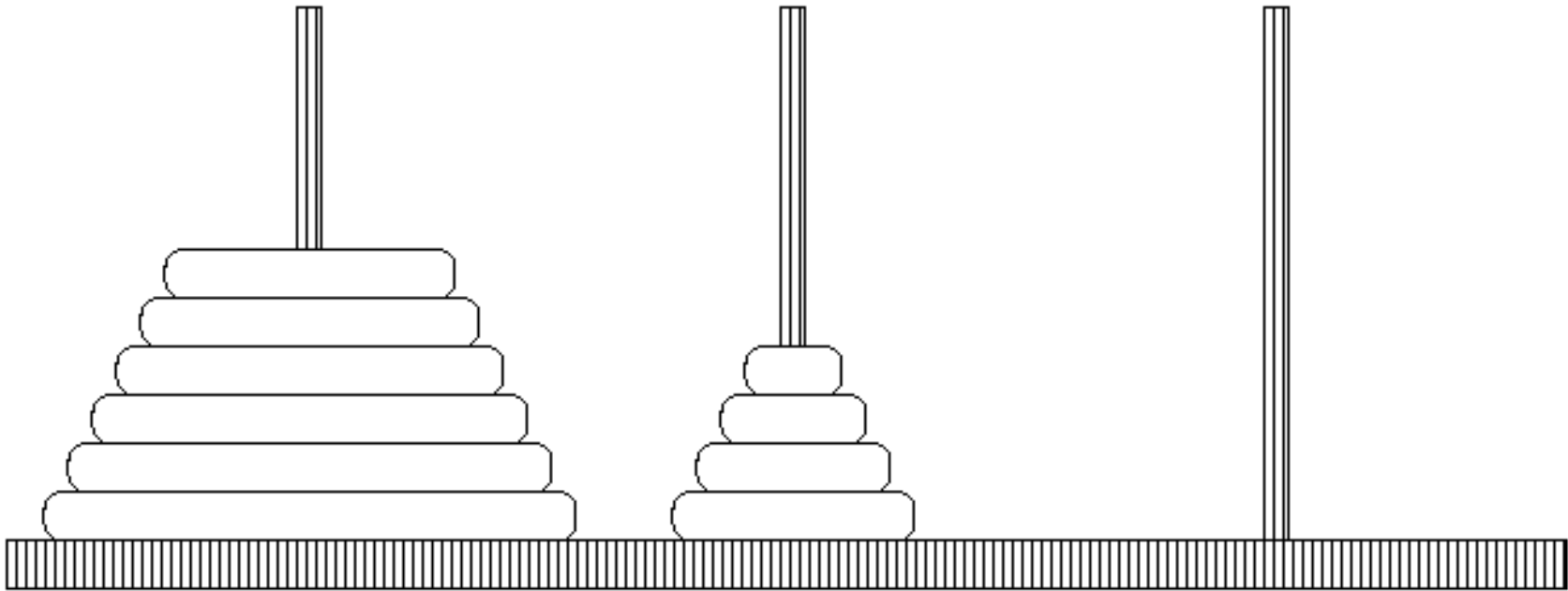
Little-oh notation indicates a bound that is *not* tight.

| <u>Notation</u> | <u>Meaning</u> |
|-----------------------|---|
| $f(x) = O(g(x))$ | $f(x) < cg(x)$, for $x > n_0$ |
| $f(x) = o(g(x))$ | $f(x)/g(x) \Rightarrow 0$ as $x \Rightarrow \infty$ |
| $f(x) = \Omega(g(x))$ | $g(x) = O(f(x))$ |
| $f(x) = \omega(g(x))$ | $g(x) = o(f(x))$ |
| $f(x) = \Theta(g(x))$ | $f(x) = O(g(x))$ AND $f(x) = \Omega(g(x))$ |

The Towers of Hanoi

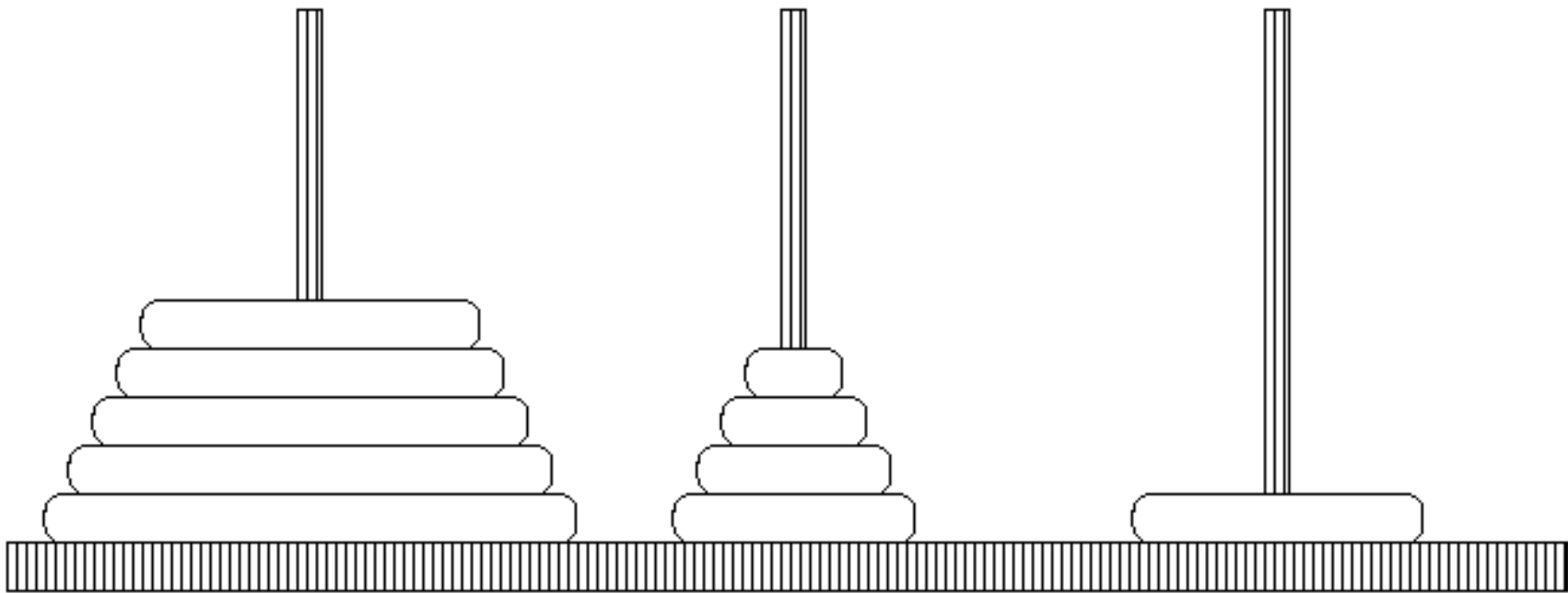


What if we knew how to solve
part of the problem?

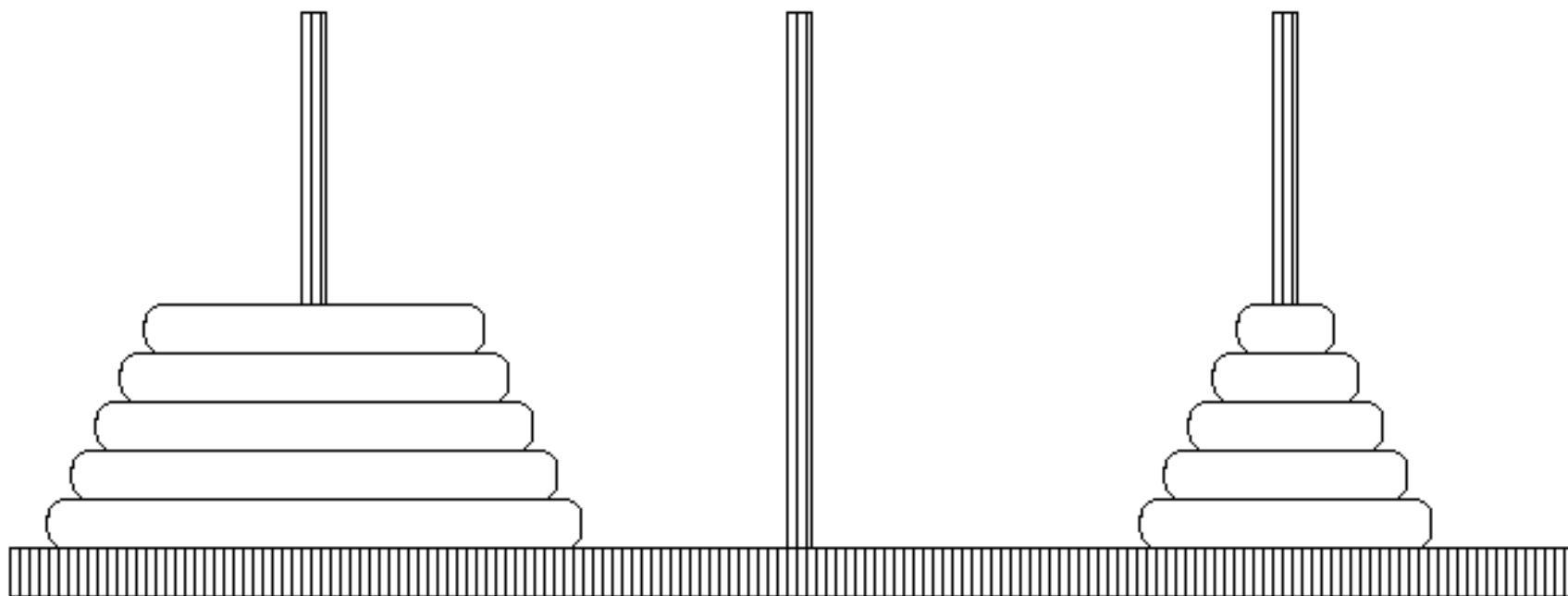


Assume we can move k (in this case, 4) different rings

Can we do one better?



Solved for one more!



Recursion *is* Mathematical Induction

In both, we have general and boundary conditions:

The **general** conditions break the problem into smaller and smaller pieces.

The *initial* or **boundary** condition terminate the recursion.

Both take a **Divide and Conquer** approach to solving mathematical problems.

Recurrence Relations

Many algorithms, particularly divide and conquer algorithms, have time complexities which are naturally modeled by recurrence relations:

$$a_n = a_{n-1} + 1; a_1 = 1 \quad \Rightarrow \quad a_n = n \quad (\textit{linear})$$

$$a_n = a_{n-1} + 2n - 1; a_1 = 1 \quad \Rightarrow \quad a_n = n^2 \quad (\textit{polynomial})$$

$$a_n = 2a_{n-1}; a_1 = 1 \quad \Rightarrow \quad a_n = 2^n \quad (\textit{exponential})$$

$$a_n = n a_{n-1}; a_1 = 1 \quad \Rightarrow \quad a_n = n! \quad (\textit{others...})$$

Solving Recurrences

We can use mathematical induction to prove that a general function solves for a recursive one.

$$T_n = 2T_{n-1} + 1 \quad ; \quad T_0 = 0$$

$$n = 0 \quad 1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6 \quad 7 \quad 8$$

$$T_n =$$

Guess what the solution is?

Solving Recurrences

Prove: $T_n = 2^n - 1$ by induction:

1. Show the base case is true: $T_0 = 2^0 - 1 = 0$

2. Now assume true for T_{n-1}

3. Substitute in T_{n-1} in recurrence for T_n

$$\begin{aligned} T_n &= 2T_{n-1} + 1 \\ &= 2(2^{n-1} - 1) + 1 \\ &= 2^n - 1 \end{aligned}$$

Solving Recurrences

Is there a general method for solving recurrences?

Consider: $a_n = 2a_{n-1} + 2a_{n-2} + 1$; $a_1 = 1$; $a_2 = 1$

1. Find the characteristic equation.
2. Solve to get roots, which appear in exponents.
3. Take care of repeated roots and inhomogeneous parts.
4. Find the constants to finish the job.

Solution: $a_n = -\frac{1}{3} + (1 - \sqrt{3})^n (1 + \sqrt{3})/3 + (1 + \sqrt{3})^n (-1 + \sqrt{3})/3$

Solving Recurrences

No general procedure for solving recurrence relations is known, which is why it is an art.

Realize that linear, finite history, constant coefficient recurrences always can be solved

For: $a_n = 2a_{n-1} + 2a_{n-2} + 1$; $a_1 = 1$; $a_2 = 1$

degree = 1

history = 2

coefficients = 2, 2, and 1

In the end, what is the best way to solve this?

-- Use Mathematica or Maple!

Why do we care?

- Recursive functions can be very powerful tools for applying divide and conquer techniques to algorithmic problems.
- Recursive algorithms can easily be converted into recurrence relations for analysis.
- Converting a recursive algorithm to the equivalent sequential algorithm can save stack space and generally improve its speed.

Techniques

- Guess a solution by looking at a few small examples and prove by induction.
- Try back-substituting until you know what is going on.
- Draw a recursion tree.

Back-substitution

Example: $T(n) = 3T(\lfloor n/4 \rfloor) + n$, $T(0) = 0$; $T(1) = 1$

$$= 3(3T(\lfloor n/16 \rfloor) + n/4) + n$$

$$= 9T(\lfloor n/16 \rfloor) + 3n/4 + n$$

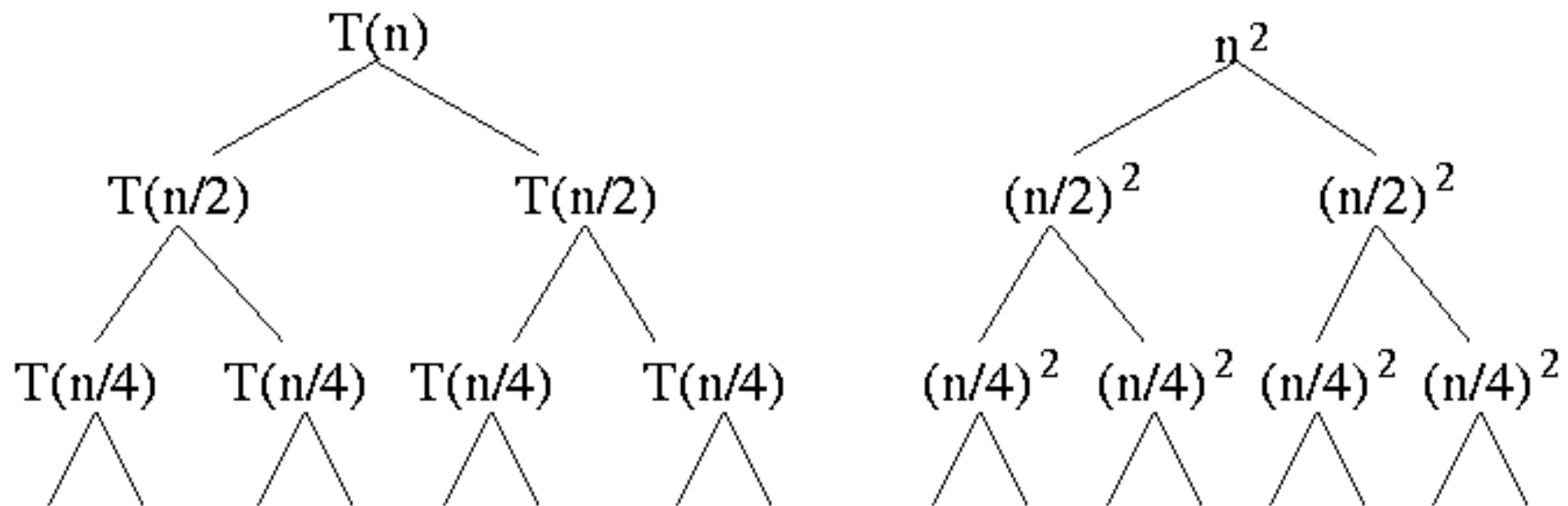
$$= 9(3T(\lfloor n/64 \rfloor) + n/16) + 3n/4 + n$$

$$= 27T(\lfloor n/64 \rfloor) + 9n/16 + 3n/4 + n$$

$$= n \cdot \sum_{i=0}^{\text{?}} \left(\frac{3}{4}\right)^i$$

Recursion Trees

$$T(n) = 2 T(n/2) + n^2, \quad T(1) = 1$$



Example Problem

Use induction to prove that MergeSort is an $O(n \log n)$ algorithm.

```
Mergesort(array)
```

```
  n = size(array)
```

```
  if ( n == 1) return array
```

```
  array1 = Mergesort(array[1 .. n/2])
```

```
  array2 = Mergesort(array[n/2 .. n])
```

```
  return Merge(array1, array2)
```

Induction Proof

Example: Prove that $T(n) = 2T(\lfloor n/2 \rfloor) + n$, $T(1) = 1$ is $O(n \log n)$.

Base case: $n=2$, $c>4$.

We need to prove that $T(n) < c n \log n$, for all n greater than some value.

Inductive step: Assume $T(n/2) \leq c (n/2) \log (n/2)$
Show that $T(n) \leq c n \log n$.

Induction Proof

Given : $T(n/2) \leq c (n/2) \log (n/2)$

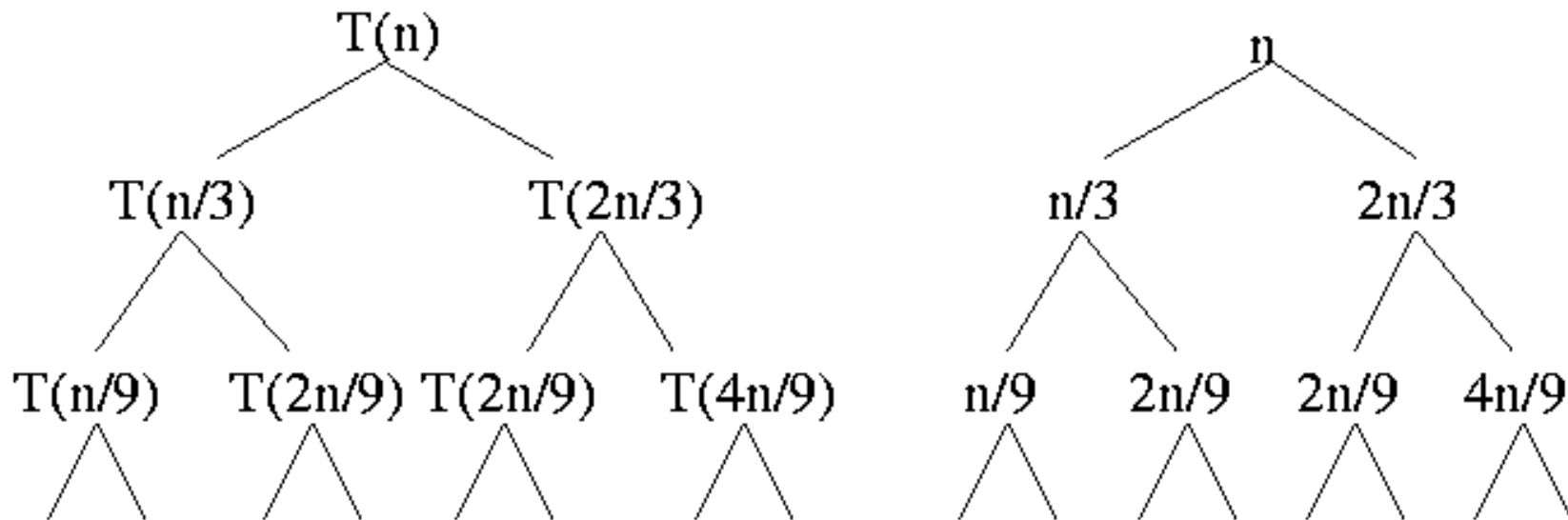
$$\begin{aligned} T(n) &= 2T(\lfloor n/2 \rfloor) + n \\ &\leq 2(c(\lfloor n/2 \rfloor) \log (\lfloor n/2 \rfloor)) + n \\ &\leq 2(c(n/2) \log (n/2)) + n \quad (\text{dropping floors makes it bigger!}) \\ &= c n \log(n/2) + n \\ &= c n (\log(n) - \log(2)) + n \\ &= c n \log(n) - c n + n \quad (\log_2 2 = 1) \\ &= c n \log(n) - (c - 1) n \\ &< c n \log(n) \quad (c > 1) \end{aligned}$$

Example Problem 2

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

Show $T(n)$ is $\Omega(n \log n)$ by appealing to the recursion tree.

Recursion Tree



Example Problem

Show that for any real constants a and b , where $b > 0$

$$(n + a)^b = \Theta(n^b)$$

What are the two things we must show to do this?

To prove $O(n^b)$, what value can should we make c_1 ?

... what is n_0 ?

To prove $\Omega(n^b)$, what value can should we make c_2 ?

... what is n_0 ?