Lecture 3
Recurrence Relations and how to solve them!
Announcements!

• **Sections and office hours** have been scheduled!
  • See course website for details.

• **HW1** is posted!
  • Due Friday.
  • Follow the Piazza post on HW1 for updates.

• (STILL) sign up for **Piazza**!
  • There’s a link on the course website.
  • Course announcements will be posted on Piazza.

• OAE requests and exam conflicts:
  • [cs161-win1819-staff@lists.stanford.edu](mailto:cs161-win1819-staff@lists.stanford.edu)
  • Please send requests by Jan. 21 (one week from today)
Last time....

• Sorting: InsertionSort and MergeSort
• Analyzing correctness of iterative + recursive algs
  • Via “loop invariant” and induction
• Analyzing running time of recursive algorithms
  • By writing out a tree and adding up all the work done.

• How do we measure the runtime of an algorithm?
  • Worst-Case Analysis
  • Big-Oh Notation
Quick clarification about proving things about $O()$ notation
Example
\[ 2n^2 + 10 = O(n^2) \]

Formally:
- Choose \( c = 3 \)
- Choose \( n_0 = 4 \)
- Then:
  \[ \forall n \geq 4, \quad 0 \leq 2n^2 + 10 \leq 3 \cdot n^2 \]
In order to formally prove $2n^2 + 10 = O(n^2)$

- Choose $n_0 = 4$ and $c = 3$.
- Claim: For all $n \geq 4$, we have $2 \cdot n^2 + 10 \leq 3 \cdot n^2$.
- To prove the claim, notice that for $n \geq 4$,
  
  $2 \cdot n^2 + 10 \leq 3 \cdot n^2$
  $\iff$
  
  $10 \leq n^2$
  $\iff$
  
  $\sqrt{10} \leq n$

- This last thing is true for any $n \geq 4$, since $\sqrt{10} \approx 3.16 \leq 4$. 
Today

• Recurrence Relations!
  • How do we calculate the runtime a recursive algorithm?

• The Master Method
  • A useful theorem so we don’t have to answer this question from scratch each time.

• The Substitution Method
  • A different way to solve recurrence relations, more general than the Master Method.
Running time of MergeSort

• Let’s call this running time $T(n)$.
  • when the input has length $n$.

• We know that $T(n) = O(n \log(n))$.

• We also know that $T(n)$ satisfies:

$$T(n) \leq 2 \cdot T \left( \frac{n}{2} \right) + 11 \cdot n$$

Last time we showed that the time to run MERGE on a problem of size $n$ is at most $11n$ operations.

MERGESORT(A):
  
  n = length(A)
  if $n \leq 1$:
    return A
  L = MERGESORT(A[:n/2])
  R = MERGESORT(A[n/2:])
  return MERGE(L,R)
Recurrence Relations

• $T(n) = 2 \cdot T\left(\frac{n}{2}\right) + 11 \cdot n$ is a recurrence relation.

• It gives us a formula for $T(n)$ in terms of $T(\text{less than } n)$

• The challenge:

  Given a recurrence relation for $T(n)$, find a closed form expression for $T(n)$.

• For example, $T(n) = O(n\log(n))$
Technicalities I

Base Cases

• Formally, we should always have base cases with recurrence relations.

• $T(n) = 2 \cdot T\left(\frac{n}{2}\right) + 11 \cdot n$ with $T(1) = 1$

  *is not the same function as*

• $T(n) = 2 \cdot T\left(\frac{n}{2}\right) + 11 \cdot n$ with $T(1) = 1000000000$

• However, $T(1) = O(1)$, so sometimes we’ll just omit it.

Why does $T(1) = O(1)$?
On your pre-lecture exercise

• You played around with these examples (when $n$ is a power of 2):

1. $T(n) = T\left(\frac{n}{2}\right) + n, \quad T(1) = 1$

2. $T(n) = 2 \cdot T\left(\frac{n}{2}\right) + n, \quad T(1) = 1$

3. $T(n) = 4 \cdot T\left(\frac{n}{2}\right) + n, \quad T(1) = 1$
One approach for all of these

• The “tree” approach from last time.
• Add up all the work done at all the sub-problems.

Size 1
Solutions to pre-lecture exercise (1)

• $T_1(n) = T_1\left(\frac{n}{2}\right) + n, \quad T_1(1) = 1$.

• Adding up over all layers:

$$\log(n) \sum_{i=0}^{\log(n)} \frac{n}{2^i} = 2n - 1$$

• So $T_1(n) = O(n)$. 

Contribution at this layer:

- Size n: n
- $n/2$: $n/2$
- $n/4$: $n/4$
- ... 
- $n/2^t$: $n/2^t$
- (Size 1): 1
Solutions to pre-lecture exercise (2)

- \( T_2(n) = 4T_2\left(\frac{n}{2}\right) + n, \quad T_2(1) = 1. \)
- Adding up over all layers:
  \[
  \sum_{i=0}^{\log(n)} 4^i \cdot \frac{n}{2^i} = n \sum_{i=0}^{\log(n)} 2^i = n(2n - 1)
  \]
- So \( T_2(n) = O(n^2) \)
More examples

- **Needlessly recursive integer multiplication**
  - $T(n) = 4 \ T(n/2) + O(n)$
  - $T(n) = O( n^2 )$

- **Karatsuba integer multiplication**
  - $T(n) = 3 \ T(n/2) + O(n)$
  - $T(n) = O( n^{\log_2(3)} \approx n^{1.6} )$

- **MergeSort**
  - $T(n) = 2T(n/2) + O(n)$
  - $T(n) = O( n\log(n) )$

$T(n) =$ time to solve a problem of size $n$.  

This is similar to $T_2$ from the pre-lecture exercise.

What's the pattern?!?!?!?!
The master theorem

• A formula for many recurrence relations.
  • We’ll see an example Wednesday when it won’t work.

• Proof: “Generalized” tree method.

A useful formula it is. Know why it works you should.

Jedi master Yoda
The master theorem

• Suppose that $a, b, d$ are constants (independent of $n$).
• Suppose $T(n) = a \cdot T\left(\frac{n}{b}\right) + O(n^d)$. Then

$$T(n) = \begin{cases} 
\Theta(n^d \log(n)) & \text{if } a = b^d \\
\Theta(n^d) & \text{if } a < b^d \\
\Theta(n^{\log_b(a)}) & \text{if } a > b^d 
\end{cases}$$

Three parameters:

$a$ : number of subproblems

$b$ : factor by which input size shrinks

$d$ : need to do $n^d$ work to create all the subproblems and combine their solutions.
Technicalities II

Integer division

• If \( n \) is odd, I can’t break it up into two problems of size \( n/2 \).

\[
T(n) = T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + T\left(\left\lceil \frac{n}{2} \right\rceil\right) + O(n)
\]

• However (see CLRS, Section 4.6.2), one can show that the Master theorem works fine if you pretend that what you have is:

\[
T(n) = 2 \cdot T\left(\frac{n}{2}\right) + O(n)
\]

• Read CLRS 4.6.2; and from now on we’ll mostly ignore floors and ceilings in recurrence relations.
Examples
(details on board)

- Needlessly recursive integer mult.
  - \( T(n) = 4 \cdot T\left(\frac{n}{2}\right) + O(n) \)
  - \( T(n) = O\left( n^2 \right) \)
  - \( a = 4 \)
  - \( b = 2 \)
  - \( d = 1 \)

- Karatsuba integer multiplication
  - \( T(n) = 3 \cdot T\left(\frac{n}{2}\right) + O(n) \)
  - \( T(n) = O\left( n^{\log_2(3)} \approx n^{1.6} \right) \)
  - \( a = 3 \)
  - \( b = 2 \)
  - \( d = 1 \)

- MergeSort
  - \( T(n) = 2 \cdot T\left(\frac{n}{2}\right) + O(n) \)
  - \( T(n) = O\left( n\log(n) \right) \)
  - \( a = 2 \)
  - \( b = 2 \)
  - \( d = 1 \)

- That other one
  - \( T(n) = T\left(\frac{n}{2}\right) + O(n) \)
  - \( T(n) = O(n) \)
  - \( a = 1 \)
  - \( b = 2 \)
  - \( d = 1 \)
Proof of the master theorem

• We’ll do the same recursion tree thing we did for MergeSort, but be more careful.

• Suppose that $T(n) = a \cdot T\left(\frac{n}{b}\right) + c \cdot n^d$.

Hang on! The hypothesis of the Master Theorem was the extra work at each level was $O(n^d)$. That’s NOT the same as work $\leq cn^d$ for some constant $c$.

That’s true ... we’ll actually prove a weaker statement that uses this hypothesis instead of the hypothesis that $T(n) = a \cdot T\left(\frac{n}{b}\right) + O(n^d)$.

It’s a good exercise to make this proof work rigorously with the $O()$ notation.
Recursion tree

\[ T(n) = a \cdot T \left( \frac{n}{b} \right) + c \cdot n^d \]

<table>
<thead>
<tr>
<th>Level</th>
<th># problems</th>
<th>Size of each problem</th>
<th>Amount of work at this level</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>( n )</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>( a )</td>
<td>( \frac{n}{b} )</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>( a^2 )</td>
<td>( \frac{n}{b^2} )</td>
<td></td>
</tr>
<tr>
<td>( t )</td>
<td>( a^t )</td>
<td>( \frac{n}{b^t} )</td>
<td></td>
</tr>
</tbody>
</table>

(Size 1)

\( \log_b(n) \) \quad \( a^{\log_b(n)} \) \quad 1
Recursion tree

\[ T(n) = a \cdot T \left( \frac{n}{b} \right) + c \cdot n^d \]

<table>
<thead>
<tr>
<th>Level</th>
<th># problems</th>
<th>Size of each problem</th>
<th>Amount of work at this level</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>(n)</td>
<td>(c \cdot n^d)</td>
</tr>
<tr>
<td>1</td>
<td>(a)</td>
<td>(n/b)</td>
<td>(ac \left( \frac{n}{b} \right)^d)</td>
</tr>
<tr>
<td>2</td>
<td>(a^2)</td>
<td>(n/b^2)</td>
<td>(a^2c \left( \frac{n}{b^2} \right)^d)</td>
</tr>
<tr>
<td>(t)</td>
<td>(a^t)</td>
<td>(n/b^t)</td>
<td>(a^tc \left( \frac{n}{b^t} \right)^d)</td>
</tr>
<tr>
<td>(\log_b(n))</td>
<td>(a^{\log_b(n)})</td>
<td>1</td>
<td>(a^{\log_b(n)c})</td>
</tr>
</tbody>
</table>

Size of each problem:
- \(n\)
- \(n/b\)
- \(n/b^2\)
- \(n/b^t\)

Size of each problem:
- \(n\)
- \(n/b\)
- \(n/b^2\)
- \(n/b^t\)

Help me fill this in!
Recursion tree

\[ T(n) = a \cdot T \left( \frac{n}{b} \right) + c \cdot n^d \]

<table>
<thead>
<tr>
<th>Level</th>
<th># problems</th>
<th>Size of each problem</th>
<th>Amount of work at this level</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>n</td>
<td>( c \cdot n^d )</td>
</tr>
<tr>
<td>1</td>
<td>a</td>
<td>n/b</td>
<td>( ac \left( \frac{n}{b} \right)^d )</td>
</tr>
</tbody>
</table>

Total work (derivation on board) is at most:

\[
c \cdot n^d \cdot \sum_{t=0}^{\log_b(n)} \left( \frac{a}{b^d} \right)^t
\]
Now let’s check all the cases

$$T(n) = \begin{cases} 
\Theta(n^d \log(n)) & \text{if } a = b^d \\
\Theta(n^d) & \text{if } a < b^d \\
\Theta(n^\log_b(a)) & \text{if } a > b^d 
\end{cases}$$

Do the first one!
Case 1: $a = b^d$

$T(n) = \begin{cases} 
\Theta(n^d \log(n)) & \text{if } a = b^d \\
\Theta(n^d) & \text{if } a < b^d \\
\Theta(n^{\log_b(a)}) & \text{if } a > b^d 
\end{cases}$

$T(n) = c \cdot n^d \cdot \sum_{t=0}^{\log_b(n)} \left( \frac{a}{b^d} \right)^t$

$= c \cdot n^d \cdot \sum_{t=0}^{\log_b(n)} 1$

$= c \cdot n^d \cdot (\log_b(n) + 1)$

$= c \cdot n^d \cdot \left( \frac{\log(n)}{\log(b)} + 1 \right)$

$= \Theta(n^d \log(n))$
Case 2: $a < b^d$

- \( T(n) = c \cdot n^d \cdot \sum_{t=0}^{\log_b(n)} \left( \frac{a}{b^d} \right)^t \) Less than 1!

\[
T(n) = \begin{cases} 
\Theta(n^d \log(n)) & \text{if } a = b^d \\
\Theta(n^d) & \text{if } a < b^d \\
\Theta(n^{\log_b(a)}) & \text{if } a > b^d
\end{cases}
\]
Aside: Geometric sums

• What is $\sum_{t=0}^{N} x^t$?

• You may remember that $\sum_{t=0}^{N} x^t = \frac{x^{N+1} - 1}{x - 1}$ for $x \neq 1$.

• Morally:

\[
x^0 + x^1 + x^2 + x^3 + \ldots + x^N
\]

If $x < 1$, this term dominates. (If $x = 1$, all terms the same) If $x > 1$, this term dominates.

\[
1 \leq \frac{1 - x^{N+1}}{1 - x} \leq \frac{1}{1 - x}
\]
(Aka, doesn’t depend on $N$).

\[
x^N \leq \frac{x^{N+1} - 1}{x - 1} \leq x^N \cdot \left(\frac{x}{x - 1}\right)
\]
(Aka, $\Theta(x^N)$ if $x$ is constant and $N$ is growing).
Case 2: $a < b^d$

- $T(n) = c \cdot n^d \cdot \sum_{t=0}^{\log_b(n)} \left( \frac{a}{b^d} \right)^t$

  $= c \cdot n^d \cdot \text{[some constant]}

  = \Theta(n^d)$

  $T(n) = \begin{cases} 
  \Theta(n^d \log(n)) & \text{if } a = b^d \\
  \Theta(n^d) & \text{if } a < b^d \\
  \Theta(n^{\log_b(a)}) & \text{if } a > b^d 
  \end{cases}$
Case 3: $a > b^d$

$$T(n) = c \cdot n^d \cdot \sum_{t=0}^{\log_b(n)} \left( \frac{a}{b^d} \right)^t$$

$$= \Theta \left( n^d \left( \frac{a}{b^d} \right)^{\log_b(n)} \right)$$

$$= \Theta \left( n^{\log_b(a)} \right)$$

Larger than 1!

Convice yourself that this step is legit!

We’ll do this step on the board!

$$T(n) = \begin{cases} \Theta(n^d \log(n)) & \text{if } a = b^d \\ \Theta(n^d) & \text{if } a < b^d \\ \Theta(n^{\log_b(a)}) & \text{if } a > b^d \end{cases}$$
Now let’s check all the cases

\[ T(n) = \begin{cases} 
\Theta(n^d \log(n)) & \text{if } a = b^d \\
\Theta(n^d) & \text{if } a < b^d \\
\Theta(n^{\log_b(a)}) & \text{if } a > b^d 
\end{cases} \]
Even more generally, for $T(n) = aT(n/b) + f(n)$...

**Theorem 3.2 (Master Theorem).** Let $T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n)$ be a recurrence where $a \geq 1$, $b > 1$. Then,
- If $f(n) = O\left(n^{\log_b a - \epsilon}\right)$ for some constant $\epsilon > 0$, $T(n) = \Theta\left(n^{\log_b a}\right)$.
- If $f(n) = \Theta\left(n^{\log_b a}\right)$, $T(n) = \Theta\left(n^{\log_b a \log n}\right)$.
- If $f(n) = \Omega\left(n^{\log_b a + \epsilon}\right)$ for some constant $\epsilon > 0$ and if $af(n/b) \leq cf(n)$ for $c < 1$ and all sufficiently large $n$, then $T(n) = \Theta(f(n))$.

Figure out how to adapt the proof we gave to prove this more general version!
Understanding the Master Theorem

• Suppose \( T(n) = a \cdot T \left( \frac{n}{b} \right) + O(n^d) \). Then

\[
T(n) = \begin{cases} 
\Theta(n^d \log(n)) & \text{if } a = b^d \\
\Theta(n^d) & \text{if } a < b^d \\
\Theta(n^{\log_b(a)}) & \text{if } a > b^d 
\end{cases}
\]

• What do these three cases mean?
Branching causes the number of problems to explode!
The most work is at the bottom of the tree!

The problems lower in the tree are smaller!
The most work is at the top of the tree!
Consider our three warm-ups

1. $T(n) = T \left( \frac{n}{2} \right) + n$
2. $T(n) = 2 \cdot T \left( \frac{n}{2} \right) + n$
3. $T(n) = 4 \cdot T \left( \frac{n}{2} \right) + n$
First example: tall and skinny tree

1. \( T(n) = T\left(\frac{n}{2}\right) + n, \quad (a < b^d) \)

- The amount of work done at the top (the biggest problem) swamps the amount of work done anywhere else.

- \( T(n) = O(\text{ work at top }) = O(n) \)

WINNER

Most work at the top of the tree!
Third example: bushy tree

3. \( T(n) = 4 \cdot T\left(\frac{n}{2}\right) + n, \quad (a > b^d) \)

- There are a HUGE number of leaves, and the total work is dominated by the time to do work at these leaves.

- \( T(n) = O\left(\text{work at bottom}\right) = O\left(4^{\text{depth of tree}}\right) = O(n^2) \)
Second example: just right

2. \( T(n) = 2 \cdot T\left(\frac{n}{2}\right) + n, \quad (a = b^d) \)

- The branching **just** balances out the amount of work.
- The same amount of work is done at every level.

- \( T(n) = (\text{number of levels}) \times (\text{work per level}) \)
- \( = \log(n) \times O(n) = O(n\log(n)) \)

*TIE!*
What have we learned?

• The “Master Method” makes our lives easier.
• But it’s basically just codifying a calculation we could do from scratch if we wanted to.
The Substitution Method

• Another way to solve recurrence relations.
• More general than the master method.

• **Step 1:** Generate a guess at the correct answer.
• **Step 2:** Try to prove that your guess is correct.
• **(Step 3: Profit.)**
The Substitution Method
first example

• Let’s return to:

\[ T(n) = 2 \cdot T \left( \frac{n}{2} \right) + n, \text{ with } T(0) = 0, T(1) = 1. \]

• The Master Method says \( T(n) = O(n \log(n)) \).

• We will prove this via the Substitution Method.
Step 1: Guess the answer

- \( T(n) = 2 \cdot T\left(\frac{n}{2}\right) + n \)
- \( T(n) = 2 \cdot \left(2 \cdot T\left(\frac{n}{4}\right) + \frac{n}{2}\right) + n \)
- \( T(n) = 4 \cdot T\left(\frac{n}{4}\right) + 2n \)
- \( T(n) = 4 \cdot \left(2 \cdot T\left(\frac{n}{8}\right) + \frac{n}{4}\right) + 2n \)
- \( T(n) = 8 \cdot T\left(\frac{n}{8}\right) + 3n \)
- ...

Guessing the pattern: \( T(n) = 2^t \cdot T\left(\frac{n}{2^t}\right) + t \cdot n \)

Plug in \( t = \log(n) \), and get

\[
T(n) = n \cdot T(1) + \log(n) \cdot n = n(\log(n) + 1)
\]
Step 2: Prove the guess is correct.

- Inductive Hyp. (n): $T(j) = j(\log(j) + 1)$ for all $1 \leq j \leq n$.
- Base Case (n=1): $T(1) = 1 = 1 \cdot (\log(1) + 1)$
- Inductive Step:
  - Assume Inductive Hyp. for n-1:
    - Suppose that $T(j) = j(\log(j) + 1)$ for all $j < n$.
  - $T(n) = 2 \cdot T \left( \frac{n}{2} \right) + n$ by definition
  - $T(n) = 2 \cdot \left( \frac{n}{2} (\log \left( \frac{n}{2} \right) + 1) \right) + n$ by induction.
  - $T(n) = n(\log(n) + 1)$ by simplifying.
  - So Inductive Hyp. holds for n.
- Conclusion: For all $n \geq 1$, $T(n) = n(\log(n) + 1)$

We’re being sloppy here about floors and ceilings...what would you need to do to be less sloppy?
Step 3: Profit

• Pretend like you never did Step 1, and just write down:

  • **Theorem:** $T(n) = O(n \log(n))$
  • **Proof:** [Whatever you wrote in Step 2]
What have we learned?

• The substitution method is a different way of solving recurrence relations.

• Step 1: Guess the answer.
• Step 2: Prove your guess is correct.
• Step 3: Profit.

• We’ll get more practice with the substitution method next lecture!
Another example (if time)
(If not time, that’s okay; we’ll see these ideas in Lecture 4)

• $T(n) = 2 \cdot T\left(\frac{n}{2}\right) + 32 \cdot n$
• $T(2) = 2$

• Step 1: Guess: $O(n \log(n))$ (divine inspiration).
• But I don’t have such a precise guess about the form for the $O(n \log(n))$ ...
  • That is, what’s the leading constant?
• Can I still do Step 2?
Step 2: Prove it, working backwards to figure out the constant

- **Guess:** $T(n) \leq C \cdot n \log(n)$ for some constant $C$ TBD.
- **Inductive Hypothesis:** $T(j) \leq C \cdot j \log(j)$ for $1 \leq j \leq n$
- **Base case:** $T(2) = 2 \leq C \cdot 2 \log(2)$ as long as $C \geq 1$
- **Inductive Step:**

\[
T(n) = 2 \cdot T\left(\frac{n}{2}\right) + 32 \cdot n
\]

$T(2) = 2$
Inductive step

• Assume that the inductive hypothesis holds for n-1.
• \( T(n) = 2T\left(\frac{n}{2}\right) + 32n \)
• \( \leq 2C \frac{n}{2} \log\left(\frac{n}{2}\right) + 32n \)
• \( = n(C \cdot \log(n) + 32 - C) \)
• \( \leq n(C \cdot \log(n)) \) as long as \( C \geq 32 \).
• Then the inductive hypothesis holds for n.
Step 2: Prove it, working backwards to figure out the constant

- **Guess:** $T(n) \leq C \cdot n \log(n)$ for some constant $C$ TBD.
- **Inductive Hypothesis:** $T(j) \leq C \cdot j \log(j)$ for $1 \leq j \leq n$
- **Base case:** $T(2) = 2 \leq C \cdot 2 \log(2)$ as long as $C \geq 1$
- **Inductive step:** Works as long as $C \geq 32$
  - So choose $C = 32$.
- **Conclusion:** $T(n) \leq 32 \cdot n \log(n)$

$$T(n) = 2 \cdot T\left(\frac{n}{2}\right) + 32 \cdot n \quad T(2) = 2$$
Step 3: Profit.

• **Theorem:** \( T(n) = O(n \log(n)) \)

• **Proof:**
  
  • **Inductive Hypothesis:** \( T(j) \leq 32 \cdot j \log(j) \) for \( 1 \leq j \leq n \)
  
  • **Base case:** \( T(2) = 2 \leq 32 \cdot 2 \log(2) \) is true.

  • **Inductive step:**
    
    • Assume Inductive Hyp. for \( n-1 \).
    
    • \( T(n) = 2T\left(\frac{n}{2}\right) + 32n \) \hspace{1cm} By the def. of \( T(n) \)
    
    • \( \leq 2 \cdot 32 \cdot \frac{n}{2} \log\left(\frac{n}{2}\right) + 32n \) \hspace{1cm} By induction
    
    • \( = n(32 \cdot \log(n) + 32 - 32) \)
    
    • \( = 32 \cdot n \log(n) \)
    
    • This establishes inductive hyp. for \( n \).

• **Conclusion:** \( T(n) \leq 32 \cdot n \log(n) \) for all \( n \geq 1 \).
Why two methods?

• Sometimes the Substitution Method works where the Master Method does not.
• More on this next time!
Next Time

• What happens if the sub-problems are different sizes?
• And when might that happen?

BEFORE Next Time

• Pre-Lecture Exercise 4!