Lecture 3
Recurrence Relations and the Master Theorem!
Announcements!

• **HW1** is posted!
  • Due Friday.

• **Sections** will be Tuesdays 4:30-5:20, room 380-381U!
  • They are optional.
  • But valuable!

• Sign up for **Piazza**!
  • There’s a link on the course website.
  • Course announcements will be posted on Piazza.
Register at:
Gates B08/B12/B30
October 7th (SAT) 9:00AM

2017 Stanford Local Programming Contest

Sponsored by: Google

http://cs.stanford.edu/group/acm/SLPC
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
Today

• Recurrence Relations!
  • How do we measure the runtime a recursive algorithm?
  • Like Integer Multiplication and MergeSort?

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

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

• We know that \( T(n) = O(n\log(n)) \).

• But if we didn’t know that...

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

From last time

MERGESORT(A):
  \n  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 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. $T(n) = 4 \cdot T\left(\frac{n}{2}\right) + n, \quad T(1) = 1$

• What are the closed forms?

  [on board]
One approach for all of these

• The “tree” approach from last time.

• Add up all the work done at all the sub-problems.
Another approach: Recursively apply the relationship a bunch until you see a pattern.

Formally, this should be accompanied with a proof that the pattern holds!

More next time.

- $T(n) = 2 \cdot T\left(\frac{n}{2}\right) + 11 \cdot n$
- $T(n) = 2 \cdot \left(2 \cdot T\left(\frac{n}{4}\right) + \frac{11 \cdot n}{2}\right) + 11 \cdot n$
- $T(n) = 4 \cdot T\left(\frac{n}{4}\right) + 22 \cdot n$
- $T(n) = 4 \cdot \left(2 \cdot T\left(\frac{n}{8}\right) + \frac{11 \cdot n}{4}\right) + 22 \cdot n$
- $T(n) = 8 \cdot T\left(\frac{n}{8}\right) + 33 \cdot n$

- Following the pattern...

- $T(n) = n \cdot T(1) + 11 \cdot \log(n) \cdot n = O(n \cdot \log(n))$

This slide skipped in class, provided here in case this way makes more sense to you.
More examples

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

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

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

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

What’s the pattern?!?!?!?!

These two are the same as the ones on your pre-lecture exercise.
The master theorem

- A **formula** that solves recurrences when all of the sub-problems are the same size.
  - We’ll see an example Wednesday when not all problems are the same size.

- ”Generalized” tree method.

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

Jedi master Yoda
The master theorem

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

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

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.

We can also take \( n/b \) to mean either \( \left\lfloor \frac{n}{b} \right\rfloor \) or \( \left\lceil \frac{n}{b} \right\rceil \) and the theorem is still true.
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:

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

• From now on we’ll mostly ignore floors and ceilings in recurrence relations.
Examples
(details on board)

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

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

• Needlessly recursive integer mult.
  - \( T(n) = 4 \, T(n/2) + O(n) \)
  - \( T(n) = O\left( n^2 \right) \)

• Karatsuba integer multiplication
  - \( T(n) = 3 \, T(n/2) + O(n) \)
  - \( T(n) = O\left( n^{\log_2(3)} \approx n^{1.6} \right) \)

• MergeSort
  - \( T(n) = 2T(n/2) + O(n) \)
  - \( T(n) = O\left( n \log(n) \right) \)

• That other one
  - \( T(n) = T(n/2) + O(n) \)
  - \( T(n) = O(n) \)
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

The recursion tree for the function $T(n)$ is shown below. The function is defined as:

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

**Diagram:**

- **Size $n$:**
  - $n/b$
  - $n/b$
  - $n/b$
  - $n/b$
  - $n/b$
  - $n/b$
  - $n/b$
  - $n/b$
  - $n/b$
- **Levels:***
  - **Level 0:** $n$
  - **Level 1:** $n/b$
  - **Level 2:** $n/b^2$
  - **Level $t$:** $n/b^t$
  - **Base:** $\log_b(n)$

**Table:***

<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>$a^t c \left(\frac{n}{b^t}\right)^d$</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

The base level is $\log_b(n)$, and the amount of work at this level is $a^{\log_b(n)} \cdot 1 = a^{\log_b(n)}$. 

**Formula:**

$$T(n) = a^{\log_b(n)} \cdot c^d$$
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>( \frac{n}{b} )</td>
<td>( ac \left( \frac{n}{b} \right)^d )</td>
</tr>
</tbody>
</table>

Total work (derivation on board) is at most:

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

\[
\log_b(n) \cdot a^{\log_b(n)} \cdot 1 \quad a^{\log_b(n)c}
\]
Now let’s check all the cases (on board)

\[
T(n) = \begin{cases} 
O(n^d \log(n)) & \text{if } a = b^d \\
O(n^d) & \text{if } a < b^d \\
O(n^{\log_b(a)}) & \text{if } a > b^d 
\end{cases}
\]
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! [From CLRS]
LIFE IS SO GOOD

BUT I WILL NOT BECOME COMPLACENT AND I CAN TOTALLY SOLVE RECURRENCES FROM SCRATCH IF I WANT TO
Understanding the Master Theorem

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

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

• What do these three cases mean?
The eternal struggle

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

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

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

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

• The branching **just** balances out the amount of work.

• The same amount of work is done at every level.

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

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

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

BEFORE Next Time

• Pre-Lecture Exercise 4!
  • Which should be easier if you did Pre-Lecture Exercise 3...