Recursive Algorithms
Introduction

Applications to Numeric Computation
Complex Numbers

• Remember how to multiply 2 complex numbers?

• \((a+bi)(c+di) = [ac –bd] + [ad + bc] \ i\)

• Input: \(a,b,c,d\)   Output: \(ac-bd, ad+bc\)

• If a real multiplication costs $1 and an addition cost a penny, what is the cheapest way to obtain the output from the input?

• Can you do better than $4.02?
Gauss’ Method:

• Input: a, b, c, d  
  Output: ac - bd, ad + bc

• $m_1 = ac$
• $m_2 = bd$
• $A_1 = m_1 - m_2 = ac - bd$

• $m_3 = (a + b)(c + d) = ac + ad + bc + bd$
• $A_2 = m_3 - m_1 - m_2 = ad + bc$

Johann Carl Friedrich Gauss
(* 30. April 1777 in Braunschweig,
† 23. Februar 1855 in Göttingen)
Question

• The Gauss method saves one multiplication out of four. It requires 25% less work.

• Could there be a context where performing 3 multiplications for every 4 provides a more dramatic savings?

• Let’s back up a bit.
How to add 2 n-bit numbers.
How to add 2 n-bit numbers.
How to add 2 n-bit numbers.
How to add 2 n-bit numbers.
How to add 2 n-bit numbers.
How to add 2 n-bit numbers.
Time complexity of grade school addition

On any reasonable computer adding 3 bits can be done in constant time.

\[ T(n) \in O(n) \]
Is there a faster way to add?

• **QUESTION**: Is there an algorithm to add two n-bit numbers whose time grows sub-linearly in n?
Any algorithm for addition must read all of the input bits

- Suppose there is a mystery algorithm that does not examine each bit.
- Give the algorithm a pair of numbers. There must be some unexamined bit position $i$ in one of the numbers.
- If the algorithm returns the wrong answer, we have found a bug.
- If the algorithm is correct, flip the bit at position $i$ and give the algorithm this new input.
- The algorithm must return the same answer, which now is wrong.
So any algorithm for addition must use time at least linear in the size of the numbers.

Grade school addition is essentially as good as it can be.
How to multiply 2 n-bit numbers.

\[
\begin{array}{c}
\times \\
\hline
\end{array}
\]

\[
\begin{array}{c}
\ast \ast \ast \ast \ast \ast \\
\ast \ast \ast \ast \ast \ast \\
\ast \ast \ast \ast \ast \ast \\
\ast \ast \ast \ast \ast \ast \\
\ast \ast \ast \ast \ast \ast \\
\ast \ast \ast \ast \ast \ast \\
\ast \ast \ast \ast \ast \ast \\
\ast \ast \ast \ast \ast \ast \\
\ast \ast \ast \ast \ast \ast \\
\ast \ast \ast \ast \ast \ast \\
\ast \ast \ast \ast \ast \ast \\
\ast \ast \ast \ast \ast \ast \\
\ast \ast \ast \ast \ast \ast \\
\end{array}
\]

\[n^2\]
How to multiply 2 n-bit numbers.

I get it! The total time is bounded by $cn^2$. 
How to multiply 2 n-bit numbers: Kindergarten Algorithm

\[ a \times b = a + a + a + ... + a \]

\[ b \]

\[ T(n) = \theta(bn) = \Theta(2^n n)! \]

Fast?
Grade School Addition: Linear time
Grade School Multiplication: Quadratic time
Kindergarten Multiplication: Exponential time
End of Lecture 6

March 23, 2009
Neat! We have demonstrated that multiplication is a harder problem than addition.

Mathematical confirmation of our common sense.
Don’t jump to conclusions!

We have argued that grade school multiplication uses more time than grade school addition. This is a comparison of the complexity of two algorithms.

To argue that multiplication is an inherently harder problem than addition we would have to show that no possible multiplication algorithm runs in linear time.
Grade School Addition: $\theta(n)$ time
Grade School Multiplication: $\theta(n^2)$ time

Is there a clever algorithm to multiply two numbers in linear time?
Despite years of research, no one knows!
Is there a faster way to multiply two numbers than the way you learned in grade school?
Good question!
Recursive Divide And Conquer

- **DIVIDE** a problem into smaller subproblems
- **CONQUER** them recursively
- **GLUE** the answers together so as to obtain the answer to the larger problem
Multiplication of 2 n-bit numbers

- $X = a \cdot 2^{n/2} + b$
- $Y = c \cdot 2^{n/2} + d$
- $XY = ac \cdot 2^n + (ad+bc) \cdot 2^{n/2} + bd$
Multiplication of 2 n-bit numbers

• $X = \begin{array}{c} a \\ b \end{array}$
• $Y = \begin{array}{c} c \\ d \end{array}$
• $XY = ac \ 2^n + (ad+bc) \ 2^{n/2} + bd$

MULT(X,Y):
If $|X| = |Y| = 1$ then RETURN XY
Break $X$ into $a;b$ and $Y$ into $c;d$
RETURN

MULT(a,c) $2^n + (MULT(a,d) + MULT(b,c)) \ 2^{n/2} + MULT(b,d)$
Time required by MULT

- $T(n) = \text{time taken by MULT on two } n\text{-bit numbers}$

- What is $T(n)$?
Recurrence Relation

• $T(1) = k$ for some constant $k$
• $T(n) = 4 \cdot T(n/2) + k' \cdot n + k''$ for some constants $k'$ and $k''$

$MULT(X,Y)$:

If $|X| = |Y| = 1$ then RETURN $XY$

Break $X$ into $a;b$ and $Y$ into $c;d$

RETURN

$MULT(a,c) \cdot 2^n + (MULT(a,d) + MULT(b,c)) \cdot 2^{n/2} + MULT(b,d)$
For example

• $T(1) = 1$

• $T(n) = 4 \, T(n/2) + n$

• How do we unravel $T(n)$ so that we can determine its growth rate?
Technique 1: (Substitution)

- **Recurrence:** \( T(1) = 1 \)
  \[ T(n) = 4T(n/2) + n, \quad n = 2, 4, 8 \ldots \]

- **Guess:** 
  \( (*) \quad T(n) = 2n^2 - n \)

- **Proof:** 
  \( (*) \rightarrow T(1) = 2 - 1 = 1 \)

Now suppose \( (*) \) is satisfied for \( n/2 \).

\[ T(n/2) = 2(n/2)^2 - n/2 = n^2/2 - n/2 \]

Then by the recurrence relation,

\[ T(n) = 4T(n/2) + n = 2n^2 - 2n + n = 2n^2 - n. \]

Thus \( (*) \) is also satisfied for \( n \).
Technique 2: Recursion Tree

- \( T(1) = 1 \)
- \( T(n) = n + 4 \cdot T(n/2) \)
\[ T(n) = n \]

\[ = \]

\[ = n \]

\[ T(n/2) \]

\[ T(n/2) \]

\[ T(n/2) \]

\[ T(n/2) \]

\[ T(n/4) \]

\[ T(n/4) \]

\[ T(n/4) \]

\[ T(n/4) \]
\[ T(n) = \frac{n}{2} + 3 \cdot \frac{n}{4} \]
\[ T(n) = \frac{n}{2} \]
Level $i$ is the sum of $4^i$ copies of $n/2^i$
Level $i$ is the sum of $4^i$ copies of $n/2^i$

\[
\sum_{i=0}^{\log_2 n} 2^i = n(2^{\log n+1} - 1) = n(2n - 1) = 2n^2 - n \in \Theta(n^2)
\]
Level $i$ is the sum of $4^i$ copies of $\frac{n}{2^i}$

$$= 1 \cdot n$$

$$= 4 \cdot \frac{n}{2}$$

$$= 16 \cdot \frac{n}{4}$$

$$= 4^i \cdot \frac{n}{2^i}$$

$$= 4^{\log_2 n} \cdot \frac{n}{2^{\log_2 n}}$$

$$= 2^{2\log_2 n} \cdot 1 = n^2$$

**Geometric Increasing** - dominated by last term:

$$\sum f(n) = \Theta(f(n)) = \Theta(n^2)$$
Divide and Conquer MULT: $\theta(n^2)$ time
Grade School Multiplication: $\theta(n^2)$ time

All that work for nothing!
MULT revisited

MULT(X,Y):

If |X| = |Y| = 1 then RETURN XY

Break X into a;b and Y into c;d

RETURN

MULT(a,c) \(2^n\) + (MULT(a,d) + MULT(b,c)) \(2^{n/2}\) + MULT(b,d)

• MULT calls itself 4 times. Can you see a way to reduce the number of calls?
Gauss’ Idea: Input: \(a, b, c, d\)  
Output: \(ac, ad + bc, bd\)

- \(A_1 = ac\)
- \(A_3 = bd\)
- \(m_3 = (a+b)(c+d) = ac + ad + bc + bd\)
- \(A_2 = m_3 - A_1 - A_3 = ad + bc\)
Gaussified MULT (Karatsuba 1962)

MULT(X,Y):
If |X| = |Y| = 1 then RETURN XY
Break X into a;b and Y into c;d
e = MULT(a,c) and f =MULT(b,d)
RETURN e2^n + (MULT(a+b, c+d) - e - f) 2^{n/2} + f

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

(More precisely: T(n) = 2 T(n/2) + T(n/2 + 1) + kn)
\[ T(n) = n \]
\[
T(n) = n = T(n/2) + T(n/2) + T(n/2)
\]
\[ T(n) = \frac{n}{2} T\left(\frac{n}{4}\right) + T\left(\frac{n}{4}\right) + T\left(\frac{n}{4}\right) \]
\[ T(n) = \frac{n}{2} T\left(\frac{n}{4}\right) + \frac{n}{2} T\left(\frac{n}{4}\right) + \frac{n}{2} T\left(\frac{n}{4}\right) \]
Level $i$ is the sum of $3^i$ copies of $n/2^i$
Level $i$ is the sum of $3^i$ copies of $n/2^i$

$= 1 \cdot n$
$= 3 \cdot n/2$
$= 9 \cdot n/4$
$= 3^i \cdot n/2^i$
$= 3^{\log n} \cdot n/2^{\log n}$
$= n^{\log 3 \cdot 1}$

$\text{Geometric Increasing} - \text{dominated by last term: } \sum f(n) = \Theta(f(n)) = \Theta(n^{\log 3}) = \Theta(n^{1.58...})$
Dramatic improvement for large \( n \)

Not just a 25% savings!

\( \theta(n^2) \) vs \( \theta(n^{1.58\ldots}) \)
Grade-School Multiplication

\[ n^2 \text{ multiplies } + n^2 \text{ additions} \]
\[ \rightarrow T(n) \approx 2n^2 \text{ bit operations} \]
Gaussified MULT (Karatsuba 1962)

MULT(X,Y):

If $|X| = |Y| = 1$ then RETURN XY

Break X into a;b and Y into c;d

e = MULT(a,c) and f = MULT(b,d)

RETURN $e2^n + (MULT(a+b, c+d) - e - f) 2^{n/2} + f$

\[ T(n) \approx 3 \cdot T(n/2) + kn \]

What is $k$? e.g., $k=8$
Dramatic improvement for large $n$

Not just a 25% savings!

$\theta(2n^2)$ vs $\theta(8n^{1.58..})$

Example:

A networking simulation requires 10 million multiplications of 16-bit integers.

Suppose that each bit operation takes 4 picosec on your machine (realistic).

Grade School Multiplication Time = 2 days 9 hours (do it over the weekend!)

Karatsuba Multiplication Time = 5.4 minutes (just enough time to grab a coffee!)

MATLAB takes 0.07 seconds on my machine (don’t blink!)
## Multiplication Algorithms

<table>
<thead>
<tr>
<th>Method</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kindergarten</td>
<td>$n^{2^n}$</td>
</tr>
<tr>
<td>Grade School</td>
<td>$n^2$</td>
</tr>
<tr>
<td>Karatsuba</td>
<td>$n^{1.58...}$</td>
</tr>
<tr>
<td>Fastest Known</td>
<td>$n \log n \log \log n$</td>
</tr>
</tbody>
</table>

*Schönhage-Strassen algorithm, 1971*
What a difference a single recursive call makes!

• What are the underlying principles here?
• How can we systematically predict which recursive algorithms are going to save time, and which are not?
Recurrence Relations

$T(1) = 1$

$T(n) = a \cdot T(n/b) + f(n)$
Recurrence Relations
≈ Time of Recursive Program

procedure Eg(int n)
  if(n≤1) then
    put “Hi”
  else
    loop i=1..f(n)
      put “Hi”
    loop i=1..a
    Eg(n/b)

• Recurrence relations arise from the timing of recursive programs.

• Let $T(n)$ be the # of “Hi”s on an input of “size” $n$. 
Recurrence Relations
≈ Time of Recursive Program

procedure Eg(int n)

if(n ≤ 1) then

    put “Hi”

else

    loop i=1..f(n)

        put “Hi”

    loop i=1..a

        Eg(n/b)

Given size 1, the program outputs \( T(1) = 1 \) Hi’s.

Given size \( n \), the stackframe outputs \( f(n) \) Hi’s.

Recursing on \( a \) instances of size \( n/b \) generates \( aT(n/b) \) “Hi”s.

For a total of \( T(1) = 1; \ T(n) = a \cdot T(n/b) + f(n) \) “Hi”s.
Technique 1: (Substitution)

- **Recurrence:**
  
  \( T(1) = 1 \)
  \( T(n) = 4T(n / 2) + n, \quad n = 2, 4, 8 \ldots \)
  
- **Guess:**
  
  \( (*) \quad T(n) = 2n^2 - n \)
  
- **Proof:**
  
  \( (*) \rightarrow T(1) = 2 - 1 = 1 \)

Now suppose \( (*) \) is satisfied for \( n / 2 \).

\[ T(n / 2) = 2(n / 2)^2 - n / 2 = n^2 / 2 - n / 2 \]

Then by the recurrence relation,

\[ T(n) = 4T(n / 2) + n = 2n^2 - 2n + n = 2n^2 - n. \]

Thus \( (*) \) is also satisfied for \( n \).
More Generally, and Formally

\[ T(1) = 1 \quad \& \quad T(n) = 4T(\lfloor n/2 \rfloor) + n \]

**Hypothesis:** \( T(n) = \Theta(n^2) \quad \text{i.e., } \exists c_1, c_2, n_0 > 0 : \forall n \geq n_0, c_1 n^2 \leq T(n) \leq c_2 n^2 \)

**Step 1. Lower Bound**

Suppose that lower bound holds for \( \lfloor i/2 \rfloor \), i.e., \( c_1 \lfloor i/2 \rfloor^2 \leq T(\lfloor i/2 \rfloor) \)

Substituting,

\[
T(i) = 4T(\lfloor i/2 \rfloor) + i \geq 4c_1 \lfloor i/2 \rfloor^2 + i \\
\geq 4c_1 \left( \frac{i-1}{2} \right)^2 + i \\
= 4c_1 \left( \frac{i^2}{4} - \frac{i}{2} + \frac{1}{4} \right) + i \\
= c_1 \left[ i^2 - 2i + 1 \right] + i \\
\]

Suppose that \( c_1 = \frac{1}{2} \)

Then \( T(i) \geq \frac{1}{2} \left[ i^2 - 2i + 1 \right] + i = \frac{1}{2} i^2 + \frac{1}{2} \geq \frac{1}{2} i^2 = c_1 i^2 \) Thus lower bound holds for \( i! \)
To Summarize

If lower bound holds for $\lfloor i/2 \rfloor$, i.e., $c_1 \lfloor i/2 \rfloor^2 \leq T(\lfloor i/2 \rfloor)$
with $c_1 = \frac{1}{2}$,

Then lower bound holds for $i$, i.e., $c_1 i^2 \leq T(i)$
Base Case

Does lower bound hold for $i = 1$?

$$c_i i^2 = \frac{1}{2} (1)^2 = \frac{1}{2} \leq T(i) = 1 \text{ Yes!}$$

By induction, must also hold for $i = 2, 3, 4, 5, ...$

e.g.,

$i = 1 \rightarrow i = 2, 3$
$i = 2 \rightarrow i = 4, 5$
$i = 3 \rightarrow i = 6, 7$

Follow similar process to prove upper bound.
Solving Technique 2
Guess Form and Calculate Coefficients

• Recurrence Relation:

\[ T(1) = 1 \& T(n) = 4T(n/2) + n \]

• Guess: \( T(n) = an^2 + bn + c \)

• Verify:

<table>
<thead>
<tr>
<th>Left Hand Side</th>
<th>Right Hand Side</th>
</tr>
</thead>
<tbody>
<tr>
<td>( T(1) = a+b+c )</td>
<td>1</td>
</tr>
<tr>
<td>( T(n) = an^2 +bn+c )</td>
<td>( 4T(n/2) + n )</td>
</tr>
<tr>
<td>( = 4 \left[a \left(\frac{n}{2}\right)^2 + b \left(\frac{n}{2}\right) +c\right] + n )</td>
<td>( = an^2 +(2b+1)n + 4c )</td>
</tr>
</tbody>
</table>
Solving Technique 2

Guess Form and Calculate Coefficients

• Recurrence Relation:

\[ T(1) = 1 \quad \text{and} \quad T(n) = 4T(n/2) + n \]

• Guess: \( T(n) = an^2 + bn + c \)

• Verify:

<table>
<thead>
<tr>
<th>Left Hand Side</th>
<th>Right Hand Side</th>
</tr>
</thead>
<tbody>
<tr>
<td>( T(1) = a+b+c )</td>
<td>( 1 )</td>
</tr>
<tr>
<td>( T(n) = an^2 + bn + c )</td>
<td>( 4T(n/2) + n )</td>
</tr>
</tbody>
</table>

\[ \Rightarrow c = 4c \]

\[ \Rightarrow c = 0 \]
Solving Technique 2
Guess Form and Calculate Coefficients

- Recurrence Relation:
  \[ T(1) = 1 \text{ & } T(n) = 4T(n/2) + n \]

- Guess: \( T(n) = an^2 + bn + 0 \)

- Verify:

<table>
<thead>
<tr>
<th>Left Hand Side</th>
<th>Right Hand Side</th>
</tr>
</thead>
<tbody>
<tr>
<td>( T(1) = a+b+c )</td>
<td>1</td>
</tr>
<tr>
<td>( T(n) = an^2 + bn+c )</td>
<td>( 4T(n/2) + n )</td>
</tr>
<tr>
<td>( b = 2b+1 )</td>
<td>( = 4 \left[a \left(\frac{n}{2}\right)^2 + b \left(\frac{n}{2}\right) + c\right] + n )</td>
</tr>
<tr>
<td>( \Rightarrow b = -1 )</td>
<td>( = an^2 + (2b+1)n + 4c )</td>
</tr>
</tbody>
</table>
Solving Technique 2
Guess Form and Calculate Coefficients

• Recurrence Relation:

\[ T(1) = 1 \& T(n) = 4T(n/2) + n \]

• Guess: \( T(n) = an^2 - 1n + 0 \)

• Verify:

<table>
<thead>
<tr>
<th>Left Hand Side</th>
<th>Right Hand Side</th>
</tr>
</thead>
<tbody>
<tr>
<td>( T(1) = a+b+c )</td>
<td>1</td>
</tr>
<tr>
<td>( T(n) = an^2 +bn+c )</td>
<td>( 4T(n/2) + n )</td>
</tr>
<tr>
<td>( a=a )</td>
<td>( 4 \left[ a \left( \frac{n}{2} \right)^2 + b \left( \frac{n}{2} \right) +c \right] + n )</td>
</tr>
<tr>
<td></td>
<td>( = an^2 +(2b+1)n + 4c )</td>
</tr>
</tbody>
</table>
Solving Technique 2
Guess Form and Calculate Coefficients

- **Recurrence Relation:**
  \[ T(1) = 1 \ & T(n) = 4T(n/2) + n \]

- **Guess:** \( T(n) = an^2 - 1n + 0 \) \( \rightarrow T(n) = 2n^2 - n \)

- **Verify:**

<table>
<thead>
<tr>
<th>Left Hand Side</th>
<th>Right Hand Side</th>
</tr>
</thead>
<tbody>
<tr>
<td>( T(1) = a+b+c )</td>
<td>1</td>
</tr>
<tr>
<td>( T(n) = an^2 +bn+c )</td>
<td>( 4T(n/2) + n )</td>
</tr>
<tr>
<td>( = 4 \left( a \left( \frac{n}{2} \right) ^2 + b \left( \frac{n}{2} \right) + c \right) + n )</td>
<td>( = an^2 + (2b+1)n + 4c )</td>
</tr>
</tbody>
</table>

\[ a+b+c=1 \]
\[ a-1+0=1 \]
\[ \rightarrow a=2 \]
Solving Technique 3
Approximate Form and Calculate Exponent

- Recurrence Relation:

\[ T(1) = 1 \quad \text{and} \quad T(n) = aT(n/b) + f(n) \]

which is bigger?

Guess
Solving Technique 3
Calculate Exponent

• Recurrence Relation:
  \[ T(1) = 1 \text{ } \& \text{ } T(n) = aT(n/b) + f(n) \]
• Guess: \( aT(n/b) \ll f(n) \)
• Simplify: \( T(n) \approx f(n) \)

In this case, the answer is easy.
\[ T(n) = \Theta(f(n)) \]
Solving Technique 3
Calculate Exponent

• Recurrence Relation:

\[ T(1) = 1 & T(n) = aT(n/b) + f(n) \]

• Guess: \[ aT(n/b) \gg f(n) \]

• Simplify: \[ T(n) \approx aT(n/b) \]

In this case, the answer is harder.
### Solving Technique 3

**Calculate Exponent**

- **Recurrence Relation:**
  \[ T(1) = 1 & T(n) = aT(n/b) \]

- **Guess:** \( T(n) = cn^\alpha = cn^{(\log a / \log b)} \)

- **Verify:**

<table>
<thead>
<tr>
<th>Left Hand Side</th>
<th>Right Hand Side</th>
</tr>
</thead>
<tbody>
<tr>
<td>( T(n) )</td>
<td>( aT(n/b) )</td>
</tr>
<tr>
<td>( = cn^\alpha )</td>
<td>( = a [c (n/b)^\alpha ] )</td>
</tr>
<tr>
<td>( 1 = a b^{-\alpha} )</td>
<td>( = c a b^{-\alpha} n^\alpha )</td>
</tr>
<tr>
<td>( b^\alpha = a )</td>
<td></td>
</tr>
<tr>
<td>( \alpha \log b = \log a )</td>
<td></td>
</tr>
<tr>
<td>( \alpha = \log a / \log b )</td>
<td></td>
</tr>
</tbody>
</table>
### Recurrence Relation:

\[ T(1) = 1 \text{ and } T(n) = 4T(n/2) \]

### Guess:

\[ T(n) = cn^\alpha = cn^\left(\frac{\log a}{\log b}\right) = cn^{\log_4 a / \log_2 n} = cn^2 \]

### Verify:

<table>
<thead>
<tr>
<th>Left Hand Side</th>
<th>Right Hand Side</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ T(n) ]</td>
<td>[ aT(n/b) ]</td>
</tr>
<tr>
<td>= [ cn^\alpha ]</td>
<td>= [ a \left[ c \left(\frac{n}{b}\right)^\alpha \right] ]</td>
</tr>
<tr>
<td>1 = a \cdot b^{-\alpha}</td>
<td>[ = a \left[ c \left(\frac{n}{b}\right)^\alpha \right] ]</td>
</tr>
<tr>
<td>[ b^\alpha = a ]</td>
<td>[ = a \cdot b^{-\alpha} \cdot n^\alpha ]</td>
</tr>
<tr>
<td>[ \alpha \log b = \log a ]</td>
<td></td>
</tr>
<tr>
<td>[ \alpha = \frac{\log a}{\log b} ]</td>
<td></td>
</tr>
</tbody>
</table>
Solving Technique 3
Calculate Exponent

• Recurrence Relation:

\[ T(1) = 1 & T(n) = aT(n/b) + f(n) \]

If bigger then

\[ T(n) = \Theta(n^{\log a/\log b}) \]

If bigger then

\[ T(n) = \Theta(f(n)) \]

And if \( aT(n/b) \approx f(n) \)
what is \( T(n) \) then?
Technique 4: Recursion Tree Method

- \( T(1) = 1 \)

- \( T(n) = a \cdot T(n/b) + f(n) \)
\[ T(n) = f(n) \]

\[ a \]

\[ T(n/b) \]

\[ T(n/b) \]

\[ T(n/b) \]

\[ T(n/b) \]
\[ T(n) = f(n) \]

\[
\begin{align*}
T(n/b) & \quad T(n/b) \\
T(n/b) & \quad T(n/b) \\
T(n/b^2) & \quad T(n/b^2) \\
T(n/b^2) & \quad T(n/b^2) \\
\end{align*}
\]
\[ T(n) = \begin{cases} f(n) & \text{if } n = 1 \\ T(n/b) + aT(n/b^2) & \text{if } n > 1 \end{cases} \]
\[ T(n) = f(n) \]

The diagram illustrates the relationship between \( T(n) \) and \( f(n) \), with \( a \) and \( f(n/b) \) playing key roles in the recursively defined formula.
Evaluating: \( T(n) = aT(n/b) + f(n) \)

<table>
<thead>
<tr>
<th>Level</th>
<th># Instances</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>( i )</td>
<td></td>
</tr>
<tr>
<td>( h )</td>
<td></td>
</tr>
</tbody>
</table>
Evaluating: \( T(n) = aT(n/b) + f(n) \)

<table>
<thead>
<tr>
<th>Level</th>
<th># Instances</th>
<th>Instance size</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>i</td>
<td></td>
<td></td>
</tr>
<tr>
<td>h</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Evaluating: \( T(n) = aT(n/b) + f(n) \)

<table>
<thead>
<tr>
<th>Level</th>
<th># Instances</th>
<th>Instance size</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td>n</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>i</td>
<td></td>
<td></td>
</tr>
<tr>
<td>h</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Evaluating: \( T(n) = aT(n/b) + f(n) \)

<table>
<thead>
<tr>
<th>Level</th>
<th># Instances</th>
<th>Instance size</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>n</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>n/b</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>i</td>
<td></td>
<td></td>
</tr>
<tr>
<td>h</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Evaluating: \( T(n) = aT(n/b) + f(n) \)

<table>
<thead>
<tr>
<th>Level</th>
<th># Instances</th>
<th>Instance size</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td>( n )</td>
</tr>
<tr>
<td>1</td>
<td>( \ldots )</td>
<td>( n/b )</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>( n/b^2 )</td>
</tr>
<tr>
<td>( i )</td>
<td>( \ldots \ldots \ldots )</td>
<td></td>
</tr>
<tr>
<td>( h )</td>
<td>( \ldots \ldots \ldots )</td>
<td></td>
</tr>
</tbody>
</table>
## Evaluating:  $T(n) = aT(n/b) + f(n)$

<table>
<thead>
<tr>
<th>Level</th>
<th># Instances</th>
<th>Instance size</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td>$n$</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>$n/b$</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>$n/b^2$</td>
</tr>
<tr>
<td>$i$</td>
<td></td>
<td>$n/b^i$</td>
</tr>
<tr>
<td>$h$</td>
<td></td>
<td>$n/b^h$</td>
</tr>
</tbody>
</table>
Evaluating: \( T(n) = aT(n/b) + f(n) \)

<table>
<thead>
<tr>
<th>Level</th>
<th># Instances</th>
<th>Instance size</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td>( n )</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>( n/b )</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>( n/b^2 )</td>
</tr>
<tr>
<td>( i )</td>
<td></td>
<td>( n/b^i )</td>
</tr>
<tr>
<td>( h )</td>
<td></td>
<td>( n/b^h )</td>
</tr>
</tbody>
</table>
Evaluating: \( T(n) = aT(n/b) + f(n) \)

<table>
<thead>
<tr>
<th>Level</th>
<th># Instances</th>
<th>Instance size</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>n</td>
<td>n</td>
</tr>
<tr>
<td>1</td>
<td>n/b</td>
<td>n/b</td>
</tr>
<tr>
<td>2</td>
<td>n/b^2</td>
<td>n/b^2</td>
</tr>
<tr>
<td>i</td>
<td>n/b^i</td>
<td>n/b^i</td>
</tr>
<tr>
<td>h</td>
<td>n/b^h = 1</td>
<td>n/b^h = 1</td>
</tr>
</tbody>
</table>

**base case**
Evaluating: \( T(n) = aT(n/b) + f(n) \)

<table>
<thead>
<tr>
<th>Level</th>
<th># Instances</th>
<th>Instance size</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td>( n )</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>( n/b )</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>( n/b^2 )</td>
</tr>
<tr>
<td>( i )</td>
<td></td>
<td>( n/b^i )</td>
</tr>
<tr>
<td>( h )</td>
<td></td>
<td>( n/b^h = 1 )</td>
</tr>
</tbody>
</table>
Evaluating: \( T(n) = aT(n/b) + f(n) \)

<table>
<thead>
<tr>
<th>Level</th>
<th># Instances</th>
<th>Instance size</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td>( n )</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>( n/b )</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>( n/b^2 )</td>
</tr>
<tr>
<td>( i )</td>
<td></td>
<td>( n/b^i )</td>
</tr>
<tr>
<td>( h = \log \frac{n}{\log b} )</td>
<td></td>
<td>( n/b^h = 1 )</td>
</tr>
</tbody>
</table>

\( b^h = n \)
\( h \log b = \log n \)
\( h = \frac{\log n}{\log b} \)
Evaluating: \( T(n) = aT(n/b) + f(n) \)

<table>
<thead>
<tr>
<th>Level</th>
<th># Instances</th>
<th>Instance size</th>
<th>Work in stack frame</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>n</td>
<td>n</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>n/b</td>
<td>n/b</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>n/b^2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>i</td>
<td>n/b^i</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[ h = \log \frac{n}{\log b} \]

\[ \vdots \]

\[ 1 \]
Evaluating: \( T(n) = aT(n/b) + f(n) \)

<table>
<thead>
<tr>
<th>Level</th>
<th># Instances</th>
<th>Instance size</th>
<th>Work in stack frame</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>( n )</td>
<td>( f(n) )</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>( n/b )</td>
<td>( f(n/b) )</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>( n/b^2 )</td>
<td>( f(n/b^2) )</td>
</tr>
<tr>
<td>( h = \log \frac{n}{\log b} )</td>
<td>5</td>
<td>( n/b^i )</td>
<td>( f(n/b^i) )</td>
</tr>
</tbody>
</table>

\( h = \log \frac{n}{\log b} \)
Evaluating: \( T(n) = aT(n/b) + f(n) \)

<table>
<thead>
<tr>
<th>Level</th>
<th># Instances</th>
<th>Instance size</th>
<th>Work in stack frame</th>
<th># stack frames</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>( n )</td>
<td>( f(n) )</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>⚫⚫⚫⚫⚫⚫</td>
<td>( n/b )</td>
<td>( f(n/b) )</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>⚫⚫⚫⚫⚫⚫</td>
<td>( n/b^2 )</td>
<td>( f(n/b^2) )</td>
<td></td>
</tr>
<tr>
<td>( i )</td>
<td>⚫⚫⚫⚫⚫⚫</td>
<td>( n/b^i )</td>
<td>( f(n/b^i) )</td>
<td></td>
</tr>
<tr>
<td>( h = \frac{\log n}{\log b} )</td>
<td>⚫⚫⚫⚫⚫⚫</td>
<td>( n/b^h )</td>
<td>( T(1) )</td>
<td></td>
</tr>
</tbody>
</table>
Evaluating: $T(n) = aT(n/b) + f(n)$

<table>
<thead>
<tr>
<th>Level</th>
<th># Instances</th>
<th>Instance size</th>
<th>Work in stack frame</th>
<th># stack frames</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td>n</td>
<td>f(n)</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>n/b</td>
<td>f(n/b)</td>
<td>a</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>n/b^2</td>
<td>f(n/b^2)</td>
<td>a^2</td>
</tr>
<tr>
<td>i</td>
<td></td>
<td>n/b^i</td>
<td>f(n/b^i)</td>
<td>a^i</td>
</tr>
<tr>
<td>h = log n/log b</td>
<td>n/b^h</td>
<td>T(1)</td>
<td>a^h</td>
<td></td>
</tr>
</tbody>
</table>

h = log n/log b
Evaluating: $T(n) = aT(n/b) + f(n)$

<table>
<thead>
<tr>
<th>Level</th>
<th># Instances</th>
<th>Instance size</th>
<th>Work in stack frame</th>
<th># stack frames</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td>$n$</td>
<td>$f(n)$</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>$n/b$</td>
<td>$f(n/b)$</td>
<td>$a$</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>$n/b^2$</td>
<td>$f(n/b^2)$</td>
<td>$a^2$</td>
</tr>
<tr>
<td>$i$</td>
<td></td>
<td>$n/b^i$</td>
<td>$f(n/b^i)$</td>
<td>$a^i$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$h = \frac{\log n}{\log b}$</td>
<td></td>
<td>$n/b^h$</td>
<td>$T(1)$</td>
<td>$a^h$</td>
</tr>
</tbody>
</table>
Evaluating: $T(n) = aT(n/b)+f(n)$

<table>
<thead>
<tr>
<th>Level</th>
<th># Instances</th>
<th>Instance size</th>
<th>Work in stack frame</th>
<th># stack frames</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>n</td>
<td>f(n)</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>n/b</td>
<td>f(n/b)</td>
<td>a</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>n/b²</td>
<td>f(n/b²)</td>
<td>a²</td>
<td></td>
</tr>
<tr>
<td>i</td>
<td>n/b^i</td>
<td>f(n/b^i)</td>
<td>a^i</td>
<td></td>
</tr>
<tr>
<td>h = log n/log b</td>
<td>n/b^h</td>
<td>T(1)</td>
<td>a^h</td>
<td></td>
</tr>
</tbody>
</table>

$$a^h = a^{log n/log b} = n^{log a/log b}$$
### Evaluating: \( T(n) = aT(n/b) + f(n) \)

<table>
<thead>
<tr>
<th>Level</th>
<th># Instances</th>
<th>Instance size</th>
<th>Work in stack frame</th>
<th># stack frames</th>
<th>Work in Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>( n )</td>
<td>( f(n) )</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>( n/b )</td>
<td>( f(n/b) )</td>
<td>( a )</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>( n/b^2 )</td>
<td>( f(n/b^2) )</td>
<td>( a^2 )</td>
<td></td>
</tr>
<tr>
<td>( i )</td>
<td></td>
<td>( n/b^i )</td>
<td>( f(n/b^i) )</td>
<td>( a^i )</td>
<td></td>
</tr>
<tr>
<td>( h = \log n/\log b )</td>
<td></td>
<td>( n/b^h )</td>
<td>( T(1) )</td>
<td>( n^{\log a/\log b} )</td>
<td></td>
</tr>
</tbody>
</table>

**Note:** The table shows the number of instances, instance size, work in stack frame, number of stack frames, and work in level for each level. The final level \( h \) is determined by \( \log n/\log b \).
Evaluating: $T(n) = aT(n/b) + f(n)$

<table>
<thead>
<tr>
<th>Level</th>
<th># Instances</th>
<th>Instance size</th>
<th>Work in stack frame</th>
<th># stack frames</th>
<th>Work in Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td>$n$</td>
<td>$f(n)$</td>
<td>1</td>
<td>$1 \cdot f(n)$</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>$n/b$</td>
<td>$f(n/b)$</td>
<td>$a$</td>
<td>$a \cdot f(n/b)$</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>$n/b^2$</td>
<td>$f(n/b^2)$</td>
<td>$a^2$</td>
<td>$a^2 \cdot f(n/b^2)$</td>
</tr>
<tr>
<td>$i$</td>
<td></td>
<td>$n/b^i$</td>
<td>$f(n/b^i)$</td>
<td>$a^i$</td>
<td>$a^i \cdot f(n/b^i)$</td>
</tr>
<tr>
<td>$h = \log n / \log b$</td>
<td>$n/b^h$</td>
<td>$T(1)$</td>
<td>$n \log a / \log b$</td>
<td>$n \log a / \log b \cdot T(1)$</td>
<td></td>
</tr>
</tbody>
</table>

Total Work $T(n) = \sum_{i=0}^{h} a^i \cdot f(n/b^i)$
Evaluating: $T(n) = aT(n/b)+f(n)$

$$= \sum_{i=0}^{h} a^i \cdot f(n/b^i)$$

If a Geometric Sum

$$\sum_{i=0}^{n} x^i = \theta(\max(\text{first term, last term}))$$

1 + 1/2 + 1/4 + 1/8 + 1/16 + 1/32 + 1/64 + ... = 2

1 + 1 + 1/2 + 1/4 + 1/8 + 1/16 + 1/32 + 1/64 = 2 * 64 - 1

1 + 2 + 4 + 8 + 16 + 32 + 64 = 2 * 64 - 1
Evaluating: \( T(n) = aT(n/b) + f(n) \)

<table>
<thead>
<tr>
<th>Level</th>
<th>Instance size</th>
<th>Work in stack frame</th>
<th># stack frames</th>
<th>Work in Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>( n )</td>
<td>( f(n) )</td>
<td>1</td>
<td>( 1 \cdot f(n) )</td>
</tr>
<tr>
<td>1</td>
<td>( n/b )</td>
<td>( f(n/b) )</td>
<td>( a )</td>
<td>( a \cdot f(n/b) )</td>
</tr>
<tr>
<td>2</td>
<td>( n/b^2 )</td>
<td>( f(n/b^2) )</td>
<td>( a^2 )</td>
<td>( a^2 \cdot f(n/b^2) )</td>
</tr>
<tr>
<td>( i )</td>
<td>( n/b^i )</td>
<td>( f(n/b^i) )</td>
<td>( a^i )</td>
<td>( a^i \cdot f(n/b^i) )</td>
</tr>
<tr>
<td>( h = \log n / \log b )</td>
<td>( n/b^h )</td>
<td>( T(1) )</td>
<td>( \log a / \log b )</td>
<td>( n \cdot T(1) )</td>
</tr>
</tbody>
</table>

Dominated by Top Level or Base Cases
End of Lecture 7

March 25, 2009
**Master Theorem: Intuition**

Suppose \( T(n) = aT(n/b) + f(n), \ a \geq 1, \ b > 1 \)

Work at top level = \( f(n) \)

Work at bottom level = number of base cases = \( n^{\log_b a} = n^{\log a / \log b} \)

Running time = \( \max(f(n), n^{\log_b a}) \)

If they are equal, then all levels are important:

Running time = sum of work over all levels = \( n^{\log_b a} \log n \)
Theorem 4.1 (Master Theorem)

Suppose $T(n) = aT(n/b) + f(n)$, $a \geq 1$, $b > 1$

1. IF $\exists \epsilon > 0$ such that $f(n) \in O(n^{\log_b a - \epsilon})$
   THEN $T(n) \in \theta(n^{\log_b a})$

   Dominated by base cases

2. IF $f(n) \in \theta(n^{\log_b a})$
   THEN $T(n) \in \theta(n^{\log_b a} \log n)$

   Work at each level is comparable:
   Sum work over all levels

3. IF $\exists \epsilon > 0$ such that $f(n) \in \Omega(n^{\log_b a + \epsilon})$
   AND
   $\exists c < 1, n_0 > 0$ such that $af(n/b) \leq cf(n)$ $\forall n \geq n_0$
   THEN $T(n) \in \theta(f(n))$

   Dominated by top level work
   Additional regularity condition
Theorem 4.1 (Master Theorem)

Suppose \( T(n) = aT(n/b) + f(n) \), \( a \geq 1, \ b > 1 \)

\[ e.g., \ T(n) = 4T(n/2) + f(n) \rightarrow \log_b a = \log_2 4 = 2 \]

1. IF \( \exists \varepsilon > 0 \) such that \( f(n) \in O(n^{\log_b a - \varepsilon}) \)
   THEN \( T(n) \in \theta(n^{\log_b a}) \)
   \[ f(n) = n^2 \? \times \]
   \[ f(n) = n^{1.97} \? \checkmark \]
   \[ e.g., \ \varepsilon = 0.01 \]

2. IF \( f(n) \in \theta(n^{\log_b a}) \)
   THEN \( T(n) \in \theta(n^{\log_b a \log n}) \)

3. IF \( \exists \varepsilon > 0 \) such that \( f(n) \in \Omega(n^{\log_b a + \varepsilon}) \)
   AND
   \( \exists c < 1, n_0 > 0 \) such that \( af(n/b) \leq cf(n) \ \forall n \geq n_0 \)
   THEN \( T(n) \in \theta(f(n)) \)
Example 2: $T(n) = 4T(n/2) + 2^n$

$a = 4$

$b = 2$

$n \log_b a = n^2$

$f(n) = 2^n$

Thus $f(n) \in \Omega(n^{\log_b a+\epsilon})$ (Case 3: dominated by top level)
Theorem 4.1 (Master Theorem)

Suppose \( T(n) = aT(n/b) + f(n), \ a \geq 1, \ b > 1 \)

1. IF \( \exists \varepsilon > 0 \) such that \( f(n) \in O(n^{\log_b a - \varepsilon}) \)
   THEN \( T(n) \in \Theta(n^{\log_b a}) \)

2. IF \( f(n) \in \Theta(n^{\log_b a}) \)
   THEN \( T(n) \in \Theta(n^{\log_b a \log n}) \)

3. IF \( \exists \varepsilon > 0 \) such that \( f(n) \in \Omega(n^{\log_b a + \varepsilon}) \)
   AND
   \( \exists c < 1, n_0 > 0 \) such that \( af(n/b) \leq cf(n) \ \forall n \geq n_0 \)
   THEN \( T(n) \in \Theta(f(n)) \)

But what about this?
Example 2: $T(n) = 4T(n/2) + 2^n$

\[ a = 4 \quad b = 2 \quad n^{\log_b a} = n^2 \]

$f(n) = 2^n$

Thus $f(n) \in \Omega(n^{\log_b a + \varepsilon})$ (Case 3: dominated by top level)

Additional regularity condition:
\[ \exists c < 1, n_0 > 0 \text{ such that } af(n/b) \leq cf(n) \quad \forall n \geq n_0 \]

Thus we require that $4 \cdot 2^{n/2} \leq c2^n$

\[ \leftrightarrow c \geq 4 \cdot 2^{-n/2} \]

Let $n_0 = 6 \rightarrow c \geq \frac{1}{2}$

\[ \rightarrow \text{regularity condition holds for } n_0 = 6, \quad c = 0.5 \]

Thus $T(n) = \theta(f(n)) = \theta(2^n)$
Example 3: \( T(n) = 4T(n/2) + n\log_5 n \)

\[
\begin{aligned}
a &= 4 \\
b &= 2 \\
&\quad \text{such that } n^{\log_b a} = n^2
\end{aligned}
\]

\( f(n) = n\log_5 n \)

Thus \( f(n) \in O(n^{\log_b a}) \) (Case 1: dominated by base cases)

Thus \( T(n) = \theta(n^{\log_b a}) = \theta(n^2) \)
Theorem 4.1 (Master Theorem)

Suppose $T(n) = aT(n/b) + f(n)$, $a \geq 1$, $b > 1$

1. IF $\exists \varepsilon > 0$ such that $f(n) \in O(n^{\log_b a - \varepsilon})$
   THEN $T(n) \in \theta(n^{\log_b a})$

2. IF $f(n) \in \theta(n^{\log_b a})$
   THEN $T(n) \in \theta(n^{\log_b a} \log n)$

3. IF $\exists \varepsilon > 0$ such that $f(n) \in \Omega(n^{\log_b a + \varepsilon})$
   AND $\exists c < 1, n_0 > 0$ such that $af(n/b) \leq cf(n)$ $\forall n \geq n_0$
   THEN $T(n) \in \theta(f(n))$

Dominated by base cases

\[ n^{\log_b a} = n^2 \]
Example 4: $T(n) = 4T(n/2) + n^2$

$a = 4$
$b = 2$

Thus $f(n) = n^2$

Thus $f(n) \in \theta(n^{\log_b a})$ (Case 2: all levels significant)

Thus $T(n) = \theta(n^{\log_b a} \log n) = \theta(n^2 \log n)$
Theorem 4.1 (Master Theorem)

Suppose $T(n) = aT(n/b) + f(n), \ a \geq 1, \ b > 1$

1. IF $\exists \epsilon > 0$ such that $f(n) \in O(n^{\log_b a - \epsilon})$
   THEN $T(n) \in \theta(n^{\log_b a})$

2. IF $f(n) \in \theta(n^{\log_b a})$
   THEN $T(n) \in \theta(n^{\log_b a} \log n)$

3. IF $\exists \epsilon > 0$ such that $f(n) \in \Omega(n^{\log_b a + \epsilon})$
   AND $\exists c < 1, n_0 > 0$ such that $af(n/b) \leq cf(n) \ \forall n \geq n_0$
   THEN $T(n) \in \theta(f(n))$
Master Theorem Case 3: When the Regularity Condition Fails

e.g. \( T(n) = T(n/2) + n(1 - 0.8 \cos \pi n) \)

Here \( \log_b a = \log_2 1 = 0 \rightarrow n^{\log_b a} = n^0 = 1 \)

and \( f(n) = n(1 - 0.8 \cos \pi n) \geq 0.2 n \in \Omega(n) \)

Thus \( f(n) \in \Omega\left(n^{\log_b a + \varepsilon}\right) \), suggesting that Case 3 applies.

But does the regularity condition hold?
Master Theorem Case 3: When the Regularity Condition Fails

\[ T(n) = T(n/2) + n(1 - .8 \cos \pi n) \]

Does the regularity condition hold?

We require that \( af(n/b) \leq cf(n) \) for some constant \( c < 1, \forall n \geq n_0 \).

\[ \leftrightarrow f(n/2) \leq cf(n) \]

\[ \leftrightarrow (n/2)(1 - .8 \cos(\pi n/2)) \leq cn(1 - .8 \cos \pi n) \]

\[ \leftrightarrow (1/2)(1 - .8 \cos(\pi n/2)) \leq c(1 - .8 \cos \pi n) \]

Given arbitrary \( n_0 \), select an \( n \geq n_0 \) such that \( n \) is even and \( n/2 \) is odd.

Then we require that \( (1/2)(1 + .8) \leq c(1 - .8) \)

\[ \leftrightarrow .9 \leq .2c \leftrightarrow c \geq 4.5 \]

Thus the regularity condition does not hold.
Master Theorem Case 3: When the Regularity Condition Fails

So what is the solution?

\[ T(n) = T(n/2) + n(1 - 0.8 \cos \pi n) \]

Note that \( f(n) = n(1 - 0.8 \cos \pi n) \in \Theta(n) \)

So in this case,

\[ T(n) \in \Theta(f(n)) = \Theta(n), \text{ despite failure of the reg. condition.} \]

**Question:** Are there failures of the reg. condition that result in \( T(n) \notin \Theta(f(n)) \)?
Master Theorem Case 3: When the Regularity Condition Fails

*Question:* Are there failures of the regularity condition that result in $T(n) \notin \Theta(f(n))$?

Consider $T(n) = 2T(n/2) + f(n)$

where $f(n) = \begin{cases} 
  n^3 & \text{when } \lfloor \log_2 n \rfloor \text{ is even} \\
  n^2 & \text{when } \lfloor \log_2 n \rfloor \text{ is odd}
\end{cases}$

Think about this puzzle and ask yourself:

1. Is the first condition of Case 3 satisfied?
2. Is the second (regularity) condition of Case 3 satisfied?
3. Is $T(n) \in \Theta(f(n))$?

*Let’s sleep on it.*
Central Algorithmic Techniques

Recursion
Different Representations of Recursive Algorithms

<table>
<thead>
<tr>
<th>Views</th>
<th>Pros</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code</td>
<td>- Implement on Computer</td>
</tr>
<tr>
<td>Stack of Stack Frames</td>
<td>- Run on Computer</td>
</tr>
<tr>
<td>Tree of Stack Frames</td>
<td>- View entire computation</td>
</tr>
<tr>
<td>Friends &amp; Strong Induction</td>
<td>- Worry about one step at a time.</td>
</tr>
</tbody>
</table>
MULT(X,Y):
If |X| = |Y| = 1 then RETURN XY
Break X into a;b and Y into c;d
e = MULT(a,c) and f =MULT(b,d)
RETURN
e 10^n + (MULT(a+b, c+d) - e - f) 10^{n/2} + f

Pros and Cons?
Code
Representation of an Algorithm

Pros:
• Runs on computers
• Precise and succinct

Cons:
• I am not a computer
• I need a higher level of intuition.
• Prone to bugs
• Language dependent
Different Representations of Recursive Algorithms

**Views**

- Code
- Stack of Stack Frames
- Tree of Stack Frames
- Friends & Strong Induction

**Pros**

- Implement on Computer
- Run on Computer
- View entire computation
- Worry about one step at a time.
MULT(X,Y):

If $|X| = |Y| = 1$ then RETURN XY

Break X into a;b and Y into c;d

e = MULT(a,c) and f = MULT(b,d)

RETURN

e $10^n + (MULT(a+b, c+d) - e - f) 10^{n/2} + f$  

Stack Frame: A particular execution of one routine on one particular input instance.
MULT(X,Y):
If |X| = |Y| = 1 then RETURN XY
Break X into a;b and Y into c;d
e = MULT(a,c) and f = MULT(b,d)
RETURN
e 10^n + (MULT(a+b, c+d) - e - f) 10^n/2 + f

X = 2133
Y = 2312
ac = ?
bd =
(a+b)(c+d) =
XY =
MULT(X,Y):
If \(|X| = |Y| = 1\) then RETURN \(XY\)
Break \(X\) into \(a;b\) and \(Y\) into \(c;d\)
e = MULT(a,c) and \(f = MULT(b,d)\)
RETURN
\[e \cdot 10^n + (MULT(a+b, c+d) - e - f) \cdot 10^{n/2} + f\]

\[
\begin{align*}
X &= 2133 \\
Y &= 2312 \\
ac &= ? \\
bd &= \\
(a+b)(c+d) &= \\
XY &= \\
X &= 21 \\
Y &= 23 \\
ac &= \\
bd &= \\
(a+b)(c+d) &= \\
XY &= 
\end{align*}
\]
MULT(X,Y):
If \(|X| = |Y| = 1\) then RETURN XY
Break X into a;b and Y into c;d
\[ e = \text{MULT}(a,c) \text{ and } f = \text{MULT}(b,d) \]
RETURN
\[ e \times 10^n + (\text{MULT}(a+b, c+d) - e - f) \times 10^{n/2} + f \]

\[
\begin{align*}
X &= 2133 \\
Y &= 2312 \\
ac &= ? \\
bd &= \\
(a+b)(c+d) &= \\
XY &= \\
X &= 21 \\
Y &= 23 \\
ac &= ? \\
bd &= \\
(a+b)(c+d) &= \\
XY &=
\end{align*}
\]
MULT(X,Y):
If \(|X| = |Y| = 1\) then RETURN XY
Break X into a;b and Y into c;d
e = MULT(a,c) and f =MULT(b,d)
RETURN
e \cdot 10^n + \left(\text{MULT}(a+b, c+d) - e - f\right) \cdot 10^{n/2} + f

Stack of Stack Frames
MULT(X,Y):
If |X| = |Y| = 1 then RETURN XY
Break X into a;b and Y into c;d
e = MULT(a,c) and f =MULT(b,d)
RETURN
e 10^n + (MULT(a+b, c+d) - e - f) 10^{n/2} + f

X = 2133
Y = 2312
ac = ?
bd = (a+b)(c+d) =
XY =

X = 21
Y = 23
ac = ?
bd = (a+b)(c+d) =
XY =

X = 2
Y = 2
XY = 4
MULT(X,Y):
If |X| = |Y| = 1 then RETURN XY
Break X into a; b and Y into c; d
e = MULT(a, c) and f = MULT(b, d)
RETURN
e 10^n + (MULT(a+b, c+d) - e - f) 10^n/2 + f

Stack of Stack Frames

X = 2133
Y = 2312
ac = ?
bd =
(a+b)(c+d) =
XY =

X = 21
Y = 23
ac = 4
bd =
(a+b)(c+d) =
MULT(X,Y):
If |X| = |Y| = 1 then RETURN XY
Break X into a;b and Y into c;d
e = MULT(a,c) and f = MULT(b,d)
RETURN
\[ e \cdot 10^n + (MULT(a+b, c+d) - e - f) \cdot 10^{n/2} + f \]
Stack of Stack Frames

MULT(X,Y):
If |X| = |Y| = 1 then RETURN XY
Break X into a;b and Y into c;d
e = MULT(a,c) and f = MULT(b,d)
RETURN
\[ e \cdot 10^n + (MULT(a+b, c+d) - e - f) \cdot 10^{n/2} + f \]

X = 2133
Y = 2312
ac = ?
bd = ?
(a+b)(c+d) =
XY =

X = 21
Y = 23
ac = 4
bd = ?
(a+b)(c+d) =
XY =

X = 1
Y = 3
XY = 3
MULT(X,Y):
If |X| = |Y| = 1 then RETURN XY
Break X into a;b and Y into c;d
 e = MULT(a,c) and f =MULT(b,d)
RETURN
\[ e \cdot 10^n + (MULT(a+b, c+d) - e - f) \cdot 10^{n/2} + f \]

X = 2133
Y = 2312
ac = ?
bd =
(a+b)(c+d) =
XY =

X = 21
Y = 23
ac = 4
bd = 3
(a+b)(c+d) =
XY =
MULT(X,Y):
If |X| = |Y| = 1 then RETURN XY
Break X into a;b and Y into c;d
e = MULT(a,c) and f = MULT(b,d)
RETURN
\[ e \cdot 10^n + (MULT(a+b, c+d) - e - f) \cdot 10^{n/2} + f \]

Stack of Stack Frames

\[
\begin{align*}
X &= 2133 \\
Y &= 2312 \\
ac &= ? \\
bd &= \\
(a+b)(c+d) &= \\
XY &= \\
X &= 21 \\
Y &= 23 \\
ac &= 4 \\
bd &= 3 \\
(a+b)(c+d) &= ? \\
XY &= 
\end{align*}
\]
MULT(X,Y):
If |X| = |Y| = 1 then RETURN XY
Break X into a;b and Y into c;d
e = MULT(a,c) and f =MULT(b,d)
RETURN
e 10^n + (MULT(a+b, c+d) - e - f) 10^n/2 + f

X = 2133
Y = 2312
ac = ?
bd =
(a+b)(c+d) =
XY =
X = 21
Y = 23
ac = 4
bd = 3
(a+b)(c+d) = ?
XY =
X = 3
Y = 5
XY = 15
MULT(X,Y):
If |X| = |Y| = 1 then RETURN XY
Break X into a;b and Y into c;d
e = MULT(a,c) and f =MULT(b,d)
RETURN
e 10^n + (MULT(a+b, c+d) - e - f) 10^n/2 + f

X = 2133
Y = 2312
ac = ?
bd =
(a+b)(c+d) =
XY =
X = 21
Y = 23
ac = 4
bd = 3
(a+b)(c+d) = 15
XY = ?
MULT(X,Y):
If |X| = |Y| = 1 then RETURN XY
Break X into a;b and Y into c;d
e = MULT(a,c) and f = MULT(b,d)
RETURN
\[ e \cdot 10^n + (MULT(a+b, c+d) - e - f) \cdot 10^{n/2} + f \]
MULT(X,Y):
If |X| = |Y| = 1 then RETURN XY
Break X into a;b and Y into c;d
e = MULT(a,c) and f = MULT(b,d)
RETURN
e 10^n + (MULT(a+b, c+d) - e - f) 10^n/2 + f

X = 2133
Y = 2312
ac = 483
bd =
(a+b)(c+d) =
XY =
MULT(X,Y):
If |X| = |Y| = 1 then RETURN XY
Break X into a;b and Y into c;d
e = MULT(a,c) and f = MULT(b,d)
RETURN
e 10^n + (MULT(a+b, c+d) - e - f) 10^{n/2} + f

X = 2133
Y = 2312
ac = 483
bd = ?
(a+b)(c+d) =
XY =
MULT(X,Y):
If |X| = |Y| = 1 then RETURN XY
Break X into a;b and Y into c;d
e = MULT(a,c) and f = MULT(b,d)
RETURN
e 10^n + (MULT(a+b, c+d) - e - f) 10^n/2 + f

X = 2133
Y = 2312
ac = 483
bd = ?
(a+b)(c+d) =
XY =
X = 33
Y = 12
ac = ?
bd =
(a+b)(c+d) =
XY = 15
MULT(X,Y):
If |X| = |Y| = 1 then RETURN XY
Break X into a;b and Y into c;d
e = MULT(a,c) and f =MULT(b,d)
RETURN
\[ e \cdot 10^n + (MULT(a+b, c+d) - e - f) \cdot 10^{n/2} + f \]

\[ \begin{align*}
X &= 2133 \\
Y &= 2312 \\
ac &= 483 \\
bd &= ? \\
(a+b)(c+d) &= \\
XY &= \]

\[ \begin{align*}
X &= 33 \\
Y &= 12 \\
ac &= ? \\
bd &= \\
(a+b)(c+d) &= \\
XY &= 15
\end{align*} \]

\[ \begin{align*}
X &= 3 \\
Y &= 1 \\
XY &= 3
\end{align*} \]
MULT(X,Y):
If |X| = |Y| = 1 then RETURN XY
Break X into a;b and Y into c;d
e = MULT(a,c) and f = MULT(b,d)
RETURN
\[ e \cdot 10^n + (MULT(a+b, c+d) - e - f) \cdot 10^{n/2} + f \]
MULT(X,Y):

If |X| = |Y| = 1 then RETURN XY

Break X into a;b and Y into c;d

e = MULT(a,c) and f = MULT(b,d)

RETURN

e \cdot 10^n + (MULT(a+b, c+d) - e - f) \cdot 10^n/2 + f

---

X = 2133
Y = 2312
ac = 483
bd = ?
(a+b)(c+d) =
XY =

X = 33
Y = 12
ac = 3
bd = ?
(a+b)(c+d) =
XY = 15

X = 3
Y = 2
XY = 6
MULT(X,Y):
If $|X| = |Y| = 1$ then RETURN $XY$
Break $X$ into $a;b$ and $Y$ into $c;d$
ed $= \text{MULT}(a,c)$ and $f = \text{MULT}(b,d)$
RETURN
$e 10^n + (\text{MULT}(a+b, c+d) - e - f) 10^{n/2} + f$

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>ac</th>
<th>bd</th>
</tr>
</thead>
<tbody>
<tr>
<td>2133</td>
<td>2312</td>
<td>483</td>
<td>?</td>
</tr>
</tbody>
</table>

$(a+b)(c+d) = ?$
XY $= 15$
MULT(X,Y):
If \(|X| = |Y| = 1\) then RETURN XY
Break X into a;b and Y into c;d
\(e = MULT(a,c)\) and \(f = MULT(b,d)\)
RETURN
\(e \cdot 10^n + (MULT(a+b, c+d) - e - f) \cdot 10^{n/2} + f\)

An so on ....
Stack of Stack Frames
Representation of an Algorithm

**Pros:**
- Traces what actually occurs in the computer
- Concrete.

**Cons:**
- Described in words it is impossible to follow
- Does not explain *why* it works.
- Demonstrates for only one of many inputs.
## Different Representations of Recursive Algorithms

<table>
<thead>
<tr>
<th>Views</th>
<th>Pros</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code</td>
<td>- Implement on Computer</td>
</tr>
<tr>
<td>Stack of Stack Frames</td>
<td>- Run on Computer</td>
</tr>
<tr>
<td>Tree of Stack Frames</td>
<td>- View entire computation</td>
</tr>
<tr>
<td>Friends &amp; Strong Induction</td>
<td>- Worry about one step at a time.</td>
</tr>
</tbody>
</table>
MULT(X,Y):
If |X| = |Y| = 1 then RETURN XY
Break X into a;b and Y into c;d
e = MULT(a,c) and f = MULT(b,d)
RETURN
e 10^n + (MULT(a+b, c+d) - e - f) 10^n/2 + f

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>ac</th>
<th>bd</th>
<th>(a+b)(c+d)</th>
<th>XY</th>
</tr>
</thead>
<tbody>
<tr>
<td>21</td>
<td>23</td>
<td>4</td>
<td>3</td>
<td>15</td>
<td>483</td>
</tr>
<tr>
<td>33</td>
<td>12</td>
<td>3</td>
<td>6</td>
<td>18</td>
<td>396</td>
</tr>
<tr>
<td>54</td>
<td>35</td>
<td>15</td>
<td>20</td>
<td>72</td>
<td>1890</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>ac</th>
<th>bd</th>
<th>(a+b)(c+d)</th>
<th>XY</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2</td>
<td></td>
<td></td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td></td>
<td></td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td></td>
<td></td>
<td>15</td>
<td>15</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td></td>
<td></td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td></td>
<td></td>
<td>18</td>
<td>18</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td></td>
<td></td>
<td>15</td>
<td>15</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td></td>
<td></td>
<td>20</td>
<td>20</td>
</tr>
<tr>
<td>9</td>
<td>8</td>
<td></td>
<td></td>
<td>72</td>
<td>72</td>
</tr>
</tbody>
</table>
Stack of Stack Frames
Representation of an Algorithm

Pros:
• View the entire computation.
• Good for computing the running time.

Cons:
• Must describe entire tree.
  – For each stack frame
    • input instance
    • computation
    • solution returned
Different Representations of Recursive Algorithms

<table>
<thead>
<tr>
<th>Views</th>
<th>Pros</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code</td>
<td>- Implement on Computer</td>
</tr>
<tr>
<td>Stack of Stack Frames</td>
<td>- Run on Computer</td>
</tr>
<tr>
<td>Tree of Stack Frames</td>
<td>- View entire computation</td>
</tr>
<tr>
<td>Friends &amp; Strong Induction</td>
<td>- Worry about one step at a time.</td>
</tr>
</tbody>
</table>
MULT(X,Y):
If |X| = |Y| = 1 then RETURN XY
Break X into a;b and Y into c;d
e = MULT(a,c) and f = MULT(b,d)
RETURN
e \times 10^n + (MULT(a+b, c+d) - e - f) \times 10^{n/2} + f

X = 2133
Y = 2312
ac = 483
bd = 396
(a+b)(c+d) = 1890
XY = 4931496

X = 21
Y = 23
ac = 4
bd = 3
(a+b)(c+d) = 15
XY = 483

X = 33
Y = 12
ac = 3
bd = 6
(a+b)(c+d) = 18
XY = 396

X = 54
Y = 35
ac = 15
bd = 20
(a+b)(c+d) = 72
XY = 1890

X = 2
Y = 2
XY = 4

X = 1
Y = 3
XY = 3

X = 3
Y = 5
XY = 15

X = 3
Y = 1
XY = 3

X = 3
Y = 2
XY = 6

X = 6
Y = 3
XY = 18

X = 5
Y = 3
XY = 15

X = 4
Y = 5
XY = 20

X = 9
Y = 8
XY = 72
MULT(X,Y):
If |X| = |Y| = 1 then RETURN XY
Break X into a;b and Y into c;d
e = MULT(a,c) and f =MULT(b,d)
RETURN
e 10^n + (MULT(a+b, c+d) - e - f) 10^n/2 + f

Worry about one step at a time.
Imagine that you are one specific friend.
Friends & Strong Induction

MULT(X,Y):
If |X| = |Y| = 1 then RETURN XY
Break X into a;b and Y into c;d
e = MULT(a,c) and f = MULT(b,d)
RETURN
e \cdot 10^n + (MULT(a+b,c+d) - e - f) \cdot 10^{n/2} + f

X = 33
Y = 12
ac =
bd =
(a+b)(c+d) =
XY =

• Consider your input instance
MULT(X,Y):
If |X| = |Y| = 1 then RETURN XY
Break X into a;b and Y into c;d
e = MULT(a,c) and f = MULT(b,d)
RETURN
e 10^n + (MULT(a+b, c+d) - e - f) 10^n/2 + f

• Consider your input instance
• Allocate work
  • Construct one or more subinstances

X = 33
Y = 12
ac = ?
bd = ?
(a+b)(c+d) =
XY =
MULT(X,Y):
If |X| = |Y| = 1 then RETURN XY
Break X into a;b and Y into c;d
e = MULT(a,c) and f = MULT(b,d)
RETURN
e \times 10^n + (MULT(a+b, c+d) – e - f) \times 10^{n/2} + f

Friends & Strong Induction

• Consider your input instance
• Allocate work
  • Construct one or more subinstances
  • Assume by magic your friends give you the answer for these.

X = 33
Y = 12
ac = 3
bd = 6
(a+b)(c+d) = 18
XY =

X = 3
Y = 1
XY = 3

X = 3
Y = 2
XY = 6

X = 6
Y = 3
XY = 18
MULT(X,Y):
If |X| = |Y| = 1 then RETURN XY
Break X into a;b and Y into c;d
e = MULT(a,c) and f = MULT(b,d)
RETURN
e 10^n + (MULT(a+b, c+d) – e - f) 10^{n/2} + f

Friends & Strong Induction

• Consider your input instance
• Allocate work
  • Construct one or more subinstances
  • Assume by magic your friends give you the answer for these.
• Use this help to solve your own instance.
Friends & Strong Induction

MULT(X,Y):
If |X| = |Y| = 1 then RETURN XY
Break X into a;b and Y into c;d
e = MULT(a,c) and f =MULT(b,d)
RETURN
e 10^n + (MULT(a+b, c+d) - e - f) 10^{n/2} + f

• Consider your input instance
• Allocate work
  • Construct one or more subinstances
  • Assume by magic your friends give you the answer for these.
• Use this help to solve your own instance.
• Do not worry about anything else, e.g.,
  • Who your boss is.
  • How your friends solve their instance.
Friends & Strong Induction

MULT(X,Y):
If |X| = |Y| = 1 then RETURN XY
Break X into a;b and Y into c;d
e = MULT(a,c) and f = MULT(b,d)
RETURN
e 10^n + (MULT(a+b, c+d) - e - f) 10^{n/2} + f

This technique is often referred to as Divide and Conquer

X = 33
Y = 12
ac = 3
bd = 6
(a+b)(c+d) = 18
XY = 396

X = 3
Y = 1
XY=3

X = 3
Y = 2
XY=6

X = 6
Y = 3
XY=18
MULT(X,Y):
If |X| = |Y| = 1 then RETURN XY
Break X into a;b and Y into c;d
e = MULT(a,c) and f =MULT(b,d)
RETURN
\[ e \times 10^n + (MULT(a+b, c+d) - e - f) \times 10^n/2 + f \]

Consider generic instances.
MULT(X,Y):
If |X| = |Y| = 1 then RETURN XY
Break X into a;b and Y into c;d
e = MULT(a,c) and f = MULT(b,d)
RETURN
e 10^n + (MULT(a+b, c+d) - e - f) 10^n/2 + f

Friends & Strong Induction
This solves the problem for every possible instance.
Friends & Strong Induction

Recursive Algorithm:
• Assume you have an algorithm that works.
• Use it to write an algorithm that works.
Friends & Strong Induction

Recursive Algorithm:
- Assume you have an algorithm that works.
- Use it to write an algorithm that works.

If I could get in,
I could get the key.
Then I could unlock the door
so that I can get in.

Circular Argument!
Friends & Strong Induction

Recursive Algorithm:

• Assume you have an algorithm that works.
• Use it to write an algorithm that works.

To get into my house
I must get the key from a smaller house
Friends & Strong Induction

MULT(X,Y):
If $|X| = |Y| = 1$ then RETURN XY
Break X into a;b and Y into c;d
e = MULT(a, c) and f = MULT(b, d)
RETURN
e $10^n + (MULT(a+b, c+d) - e - f) 10^{n/2} + f$

• Allocate work
• Construct one or more subinstances

Each subinstance must be a smaller instance to the same problem.
Friends & Strong Induction

Recursive Algorithm:

• Assume you have an algorithm that works.
• Use it to write an algorithm that works.
MULT(X,Y):
If |X| = |Y| = 1 then RETURN XY
Break X into a;b and Y into c;d
e = MULT(a,c) and f = MULT(b,d)
RETURN
e \cdot 10^n + (MULT(a+b, c+d) - e - f) \cdot 10^{n/2} + f

Use brute force
to solve the base case
instances.
Friends & Strong Induction

Carefully write the specifications for the problem.

Preconditions: Set of legal instances (inputs) Why?

Postconditions: Required output
Friends & Strong Induction

Carefully write the specifications for the problem.

Preconditions: Set of legal instances
(inputs)
• To be sure that we solve the problem for every legal instance.
• So that we know
  – what we can give to a friend.

Postconditions: Required output
• So that we know
  – what is expected of us.
  – what we can expect from our friend.

Related to Loop Invariants
Applications of Recursion

Another Numerical Computation Example
The Greatest Common Divisor (GCD) Problem

- Given two integers, what is their greatest common divisor?
- e.g., gcd(56,24) = 8

**Notation:**
Given $d, a \in \mathbb{Z}$:

$d \mid a \iff d$ divides $a \iff \exists k \in \mathbb{Z} : a = kd$

**Note:** All integers divide 0: $d \mid 0 \forall d \in \mathbb{Z}$

**Important Property:**

$d \mid a$ and $d \mid b \rightarrow d \mid (ax + by) \forall x, y \in \mathbb{Z}$
Euclid’s Trick

Important Property:
\[ d \mid a \text{ and } d \mid b \rightarrow d \mid (ax + by) \quad \forall x, y \in \mathbb{Z} \]

Idea: Use this property to make the GCD problem easier!

Consequence: e.g.,
\[ \gcd(a, b) = \gcd(a - b, b) \]
\[ \gcd(56, 24) = \gcd(56 - 24, 24) = \gcd(32, 24) \quad \text{Good!} \]

\[ \gcd(a, b) = \gcd(a - 2b, b) \]
\[ \gcd(56, 24) = \gcd(56 - 2 \times 24, 24) = \gcd(8, 24) \quad \text{Better!} \]

\[ \gcd(a, b) = \gcd(a - 3b, b) \]
\[ \gcd(56, 24) = \gcd(56 - 3 \times 24, 24) = \gcd(-16, 24) \quad \text{Too Far!} \]

\[ \vdots \]

What is the optimal choice?

\[ \gcd(a, b) = \gcd(a \mod b, b) \]
\[ \gcd(56, 24) = \gcd(56 \mod 24, 24) = \gcd(8, 24) \]
Euclid’s Algorithm (circa 300 BC)

Euclid(a,b)

<Precondition: a and b are positive integers>

<Postcondition: returns gcd(a,b)>

if b = 0 then
    return(a)
else
    return(Euclid(b,a mod b))

Precondition met, since a mod b ∈ ℤ

Postcondition met, since

1. b = 0 → gcd(a,b) = gcd(a,0) = a
2. Otherwise, gcd(a,b) = gcd(b,a mod b)
3. Algorithm halts, since 0 ≤ a mod b < b
End of Lecture 8

March 30, 2009
Time Complexity

Euclid(a,b)

    if \( b = 0 \) then
        return(a)
    else
        return(Euclid(b, a \mod b))

Claim: 2nd argument drops by factor of at least 2 every 2 iterations.

Proof:

<table>
<thead>
<tr>
<th>Iteration</th>
<th>Arg 1</th>
<th>Arg 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>( i )</td>
<td>( a )</td>
<td>( b )</td>
</tr>
<tr>
<td>( i + 1 )</td>
<td>( b )</td>
<td>( a \mod b )</td>
</tr>
<tr>
<td>( i + 2 )</td>
<td>( a \mod b )</td>
<td>( b \mod(a \mod b) )</td>
</tr>
</tbody>
</table>

Case 1: \( a \mod b \leq b/2 \). Then \( b \mod(a \mod b) < a \mod b \leq b/2 \)

Case 2: \( b > a \mod b > b/2 \). Then \( b \mod(a \mod b) < b/2 \)
Time Complexity

Euclid(a,b)
    if \( b = 0 \) then
        return(a)
    else
        return(Euclid(b, a \text{mod} b))

Let \( k \) = total number of recursive calls to Euclid.
Let \( n \) = input size = number of bits used to represent \( a \) and \( b \).

Then \( 2^{k/2} \approx b \approx 2^{n/2} \rightarrow k = n \).

Each stackframe must compute \( a \mod b \), which takes more than constant time.

It can be shown that the resulting time complexity is \( T(n) \in O(n^2) \).
Applications of Recursion

Data Organization
A Simple Example:

The Tower of Hanoi
This job of mine is a bit daunting. Where do I start? And I am lazy.
At some point, the biggest disk moves. I will do that job.
To do this, the other disks must be in the middle.
How will these move? I will get a friend to do it. And another to move these. I only move the big disk.
Tower of Hanoi

Code:

\[ \text{algorithm } TowersOfHanoi(n, \text{source}, \text{destination}, \text{spare}) \]

\(\langle \text{pre-cond} \rangle\): The \(n\) smallest disks are on \(\text{pole}_{\text{source}}\).

\(\langle \text{post-cond} \rangle\): They are moved to \(\text{pole}_{\text{destination}}\).

begin

if \(n = 1\)

Move the single disk from \(\text{pole}_{\text{source}}\) to \(\text{pole}_{\text{destination}}\).
else

\(TowersOfHanoi(n - 1, \text{source}, \text{spare}, \text{destination})\)

Move the \(n^{th}\) disk from \(\text{pole}_{\text{source}}\) to \(\text{pole}_{\text{destination}}\).

\(TowersOfHanoi(n - 1, \text{spare}, \text{destination}, \text{source})\)

end if

end algorithm
Tower of Hanoi

Code:

algorithm TowersOfHanoi(n, source, destination, spare)

\( \text{pre-cond} \): The \( n \) smallest disks are on \( \text{pole}_{\text{source}} \).
\( \text{post-cond} \): They are moved to \( \text{pole}_{\text{destination}} \).

begin
  if \( n = 1 \)
    Move the single disk from \( \text{pole}_{\text{source}} \) to \( \text{pole}_{\text{destination}} \).
  else
    TowersOfHanoi(n - 1, source, spare, destination)
    Move the \( n \)th disk from \( \text{pole}_{\text{source}} \) to \( \text{pole}_{\text{destination}} \).
    TowersOfHanoi(n - 1, spare, destination, source)
  end if
end algorithm

Time:

\( T(1) = 1, \)

\( T(n) = 1 + 2T(n-1) \approx 2T(n-1) \)
\( \approx 2(2T(n-2)) \approx 4T(n-2) \)
\( \approx 4(2T(n-3)) \approx 8T(n-3) \)
\( \approx 2^i T(n-i) \)
\( \approx 2^n \)
More Data Organization Examples

Sorting
Recursive Sorts

- Given list of objects to be sorted
- Split the list into two sublists.
- Recursively have a friend sort the two sublists.
- Combine the two sorted sublists into one entirely sorted list.
Example: Merge Sort
Merge Sort

Divide and Conquer
Merge Sort

Split Set into Two (no real work)

Get one friend to sort the first half.

Get one friend to sort the second half.

25, 31, 52, 88, 98

14, 23, 30, 62, 79
Merge Sort

Merge two sorted lists into one

25, 31, 52, 88, 98
14, 23, 30, 62, 79

14, 23, 25, 30, 31, 52, 62, 79, 88, 98
Merge Sort

Time: \( T(n) = 2T(n/2) + \Theta(n) = \Theta(n \log(n)) \)
Example: Quick Sort
Quick Sort

Divide and Conquer
Quick Sort

Partition set into two using randomly chosen pivot
Quick Sort

Get one friend to sort the first half.

14, 23, 25, 30, 31

Get one friend to sort the second half.

62, 79, 98, 88

≤ 52 ≤
Quick Sort

Glue pieces together.
(No real work)
Quick Sort

Let pivot be the first element in the list?
Quick Sort

If the list is already sorted, then the list is worst case unbalanced.
Quick Sort

Best Time: \[ T(n) = 2T(n/2) + \Theta(n) \]
\[ = \Theta(n \log(n)) \]

Worst Time:

Expected Time:
Quick Sort

Best Time: \[ T(n) = 2T(n/2) + \Theta(n) = \Theta(n \log(n)) \]

Worst Time: \[ T(n) = T(1) + T(n-1) + \Theta(n) = \Theta(n^2) \]

Expected Time:
Quick Sort

Best Time: \[ T(n) = 2T(n/2) + \Theta(n) \]
\[ = \Theta(n \log(n)) \]

Worst Time: \[ T(n) = T(0) + T(n-1) + \Theta(n) \]
\[ = \Theta(n^2) \]

Expected Time: \[ T(n) = \Theta(n \log(n)) \]
(The proof is not difficult, but it’s a little long)
Expected Time Complexity for Quick Sort

Q: Why is it reasonable to expect $\Theta(n \log n)$ time complexity?

A: Because on average, the partition is not too unbalanced.

Example: Imagine a deterministic partition, in which the 2 subsets are always in fixed proportion, i.e., $p(n-1) \& q(n-1)$, where $p, q$ are constants, $p, q \in [0...1], p + q = 1$. 

$\begin{array}{c}
\text{p(n-1)} \\
\text{q(n-1)} \\
\end{array}$
Expected Time Complexity for Quick Sort

Then \( T(n) = T(p(n-1)) + T(q(n-1)) + \Theta(n) \)

wlog, suppose that \( q > p \).
Then recursion tree has depth \( k \in \Theta(\log n) \):
\( q^k n = 1 \rightarrow k = \log n / \log(1 / q) \)

\( \Theta(n) \) work done per level \( \rightarrow T(n) = \Theta(n \log n) \).
Properties of QuickSort

- In-place?  
- Stable?  
- Fast?
  - Depends.
  - Worst Case: $\Theta(n^2)$
  - Expected Case: $\Theta(n \log n)$, with small constants
Heaps, Heap Sort, & Priority Queues
Heapsort

- $O(n \log n)$ worst case – like merge sort
- Sorts in place – like insertion sort
- Combines the best of both algorithms
Heap Definition (MaxHeap)

- Balanced binary tree
- The value of each node $\geq$ each of the node's children.
- Left or right child could be next largest.

Where can 9 go?  Maximum is at root.
Where can 1 go?
Where can 8 go?
Some Additional Properties of Heaps

The height \( h(i) \) of a node \( i \) of the heap is the number of edges on the longest simple downward path from the node to a leaf.

The height \( H \) of a heap is the height of the root.
Some Additional Properties of Heaps

An $n$-element heap has height $H = \left\lfloor \log_2 n \right\rfloor$
Some Additional Properties of Heaps

A heap of height $H$ has at least $n = 2^H$ nodes.

A heap of height $H$ has at most $n = 2^{H+1} - 1$ nodes.

\[ 2^H \leq n \leq 2^{H+1} - 1 \]
Heap Data Structure

Balanced Binary Tree Implemented by an Array

- The root is stored in \( A[1] \).
- The parent of \( A[i] \) is \( A[\lfloor \frac{i}{2} \rfloor] \).
- The left child of \( A[i] \) is \( A[2 \cdot i] \).
- The right child of \( A[i] \) is \( A[2 \cdot i + 1] \).
- The node in the far right of the bottom level is stored in \( A[n] \).
- If \( 2i + 1 > n \), then the node does not have a right child.
algorithm MakeHeap()

\(\text{pre-cond}\): The input is an array of numbers, which can be viewed as a balanced binary tree of numbers.

\(\text{post-cond}\): Its values are rearranged in place to make it heap.

Get help from friends

Now we are just left with this problem
Heapify

Max-Heapify(A, i, n)

<pre-cond>: Left and right subtrees of A[i] are max heaps.

<post-cond>: Subtree rooted at i is a heap.

Where should the maximum be?

Maximum must be at root.
Max-Heapify\((A, i, n)\)

**<pre-cond>:** Left and right subtrees of \(A[i]\) are max heaps.

**<post-cond>:** Subtree rooted at \(i\) is a heap.

Find the maximum.

Put it in place

Repeat
Heapify

Max-Heapify($A, i, n$)

<pre-cond>: Left and right subtrees of $A[i]$ are max heaps.

<post-cond>: Subtree rooted at $i$ is a heap.

Running Time: $T(n) = \Theta(\text{the height of tree}) = \Theta(\log n)$.

$T(n) = 1 \cdot T(n/2) + \Theta(1) = \Theta(\log n)$. 
Max-Heapify($A, i, n$)

<pre-cond>: Left and right subtrees of $A[i]$ are max heaps.

<post-cond>: Subtree rooted at $i$ is a heap.

$l \leftarrow \text{LEFT}(i)$
$r \leftarrow \text{RIGHT}(i)$

if $l \leq n$ and $A[l] > A[i]$
then $\text{largest} \leftarrow l$
else $\text{largest} \leftarrow i$

if $r \leq n$ and $A[r] > A[\text{largest}]$
then $\text{largest} \leftarrow r$

if $\text{largest} \neq i$
then exchange $A[i] \leftrightarrow A[\text{largest}]$

$\text{MAX-HEAPIFY}(A, \text{largest}, n)$

\[ \text{e.g., Max-Heapify (A,2,10)} \Rightarrow \text{Max-Heapify (A,4,10)} \Rightarrow \text{Max-Heapify (A,9,10)} \]
End of Lecture 9

April 1, 2009
MakeHeap

- MakeHeap uses Max-Heapify to reorganize the tree from bottom to top to make it a heap.
- MakeHeap can be written concisely in either recursive or iterative form.
Recursive MakeHeap

\[ \text{MakeHeap}(A, i, n) \]

**<pre-cond>:** \( A[i \ldots n] \) is a balanced binary tree

**<post-cond>:** The subtree rooted at \( i \) is a heap

if \( i \leq \lceil n/4 \rceil \) then

\[ \text{MakeHeap}(A, \text{LEFT}(i), n) \]

\[ \text{MakeHeap}(A, \text{RIGHT}(i), n) \]

\[ \text{Max-Heapify}(A, i, n) \]

Running time:

\[ T(n) = 2T(n/2) + \log(n) \]

= \( \Theta(n) \)
Iterative MakeHeap

MakeHeap\((A, n)\)

<pre-cond>: \(A[1...n]\) is a balanced binary tree
<post-cond>: \(A[1...n]\) is a heap

for \(i \leftarrow \lfloor n/2 \rfloor\) downto 1

<LI>: All subtrees rooted at \(i + 1...n\) are heaps

Max-Heapify\((A, i, n)\)

Max-Heapify\((A, i, n)\)

<pre-cond>: Left and right subtrees of \(A[i]\) are max heaps.
<post-cond>: Subtree rooted at \(i\) is a heap.
Iterative MakeHeap

\textbf{MakeHeap}(A, n)

\textbf{<pre-cond>:} A[1\ldots n] \text{ is a balanced binary tree}

\textbf{<post-cond>:} A[1\ldots n] \text{ is a heap}

\text{for } i \leftarrow \lfloor n/2 \rfloor \text{ downto } 1

\textbf{<LI>}: All subtrees rooted at } i + 1\ldots n \text{ are heaps}

\textbf{Max-Heapify}(A, i, n)

\textbf{Max-Heapify}(A, i, n)

\textbf{<pre-cond>:} Left and right subtrees of A[i] are max heaps.

\textbf{<post-cond>:} Subtree rooted at } i \text{ is a heap.
Iterative MakeHeap

```
MakeHeap(A,n)
<pre-cond>: A[1...n] is a balanced binary tree
<post-cond>: A[1...n] is a heap

for i ← ⌊n/2⌋ downto 1
  <LI>: All subtrees rooted at i+1...n are heaps
  Max-Heapify(A,i,n)
```

```
Max-Heapify(A, i, n)
<pre-cond>: Left and right subtrees of A[i] are max heaps.
<post-cond>: Subtree rooted at i is a heap.
```
Iterative MakeHeap

MakeHeap \((A, n)\)

<pre-cond>: \(A[1 \ldots n]\) is a balanced binary tree

<post-cond>: \(A[1 \ldots n]\) is a heap

for \(i \gets \left\lfloor n/2 \right\rfloor\) downto 1

<LI>: All subtrees rooted at \(i + 1 \ldots n\) are heaps

Max-Heapify \((A, i, n)\)

Max-Heapify \((A, i, n)\)

<pre-cond>: Left and right subtrees of \(A[i]\) are max heaps.

<post-cond>: Subtree rooted at \(i\) is a heap.
Iterative MakeHeap

MakeHeap(A, n)
<pre-cond>: A[1...n] is a balanced binary tree
</pre-cond>
</pre-cond>
</pre-cond>
</pre-cond>
</pre-cond>
</pre-cond>
</pre-cond>
</pre-cond>
</pre-cond>
</pre-cond>
</pre-cond>
</pre-cond>
</pre-cond>
</pre-cond>
</pre-cond>
</pre-cond>
<post-cond>: A[1...n] is a heap
</post-cond>

for i ←⌈n/2⌉ downto 1
<LI>: All subtrees rooted at i + 1...n are heaps
Max-Heapify(A, i, n)

Max-Heapify(A, i, n)
<pre-cond>: Left and right subtrees of A[i] are max heaps.
</pre-cond>
</pre-cond>
</pre-cond>
</pre-cond>
</pre-cond>
</pre-cond>
</pre-cond>
</pre-cond>
</pre-cond>
</pre-cond>
</pre-cond>
</pre-cond>
</pre-cond>
</pre-cond>
</pre-cond>
</pre-cond>
</pre-cond>
</pre-cond>
<post-cond>: Subtree rooted at i is a heap.
</post-cond>
Iterative MakeHeap

MakeHeap(A, n)
<pre-cond>: A[1…n] is a balanced binary tree
$post-cond$: A[1…n] is a heap

for i ← ⌊n/2⌋ downto 1
    <LI>: All subtrees rooted at i + 1…n are heaps

Max-Heapify(A, i, n)

Runtime:
Height of heap = ⌈log₂ n⌉

It can be shown that the number of nodes at height h ≤ ⌈n / 2^(h+1)⌉

Time to heapify from node at height h ∈ O(h)

→ T(n) = \sum_{h=0}^{⌈log₂ n⌉} \frac{n}{2^{h+1}} O(h) = O\left(n \sum_{h=0}^{⌈log₂ n⌉} \frac{h}{2^h}\right) = O(n)
Iterative MakeHeap

- Recursive and Iterative MakeHeap do essentially the same thing: Heapify from bottom to top.

- Difference:
  - Recursive is “depth-first”
  - Iterative is “breadth-first”
Using Heaps for Sorting
Selection Sort

Largest $i$ values are sorted on the right. Remaining values are off to the left.

Max is easier to find if a heap.
Heap Sort

HeapSort(A, n)

<pre-cond>:A[1...n] is a list of keys
<post-cond>:A[1...n] is sorted in non-decreasing order

Largest i values are sorted on side.
Remaining values are in a heap.
Heap Data Structure

Heap

Array

Heap

Array

Heap Array

5 43 12

98

76

Heap Array

COSC 3101, PROF. J. ELDER 227
Heap Sort

Largest i values are sorted on side. Remaining values are in a heap

Put next value where it belongs.

Max-Heapify(A, i, n)

<pre-cond>: Left and right subtrees of A[i] are max heaps.

<post-cond>: Subtree rooted at i is a heap.
Heap Sort

Sorted
Heap Sort

HeapSort(A,n)
<pre-cond>:A[1...n] is a list of keys
</pre-cond>
<post-cond>:A[1...n] is sorted in non-decreasing order

MakeHeap(A,n)
for i ← n downto 2
  <LI>: A[1…i] is a heap
    A[i +1…n] contains the largest keys in non-decreasing order
    Max-Heapify(A,1,i −1)

Running Time:
MakeHeap takes Θ(n) time.
Heapifing a tree of size i takes log(i).

T(n) = Θ(n) + ∑_{i=1}^{n} log i. This sum is arithmetic.
T(n) = n × maximum value = Θ(n log n).
Other Applications of Heaps
Priority Queue

- Maintains dynamic set, A, of n elements, each with a key.
- Max-priority queue supports:
  1. MAXIMUM(A)
  2. EXTRACT-MAX(A, n)
  3. INCREASE-KEY(A, i, key)
  4. INSERT(A, key, n)

- Example Application: Schedule jobs on a shared computer.
Priority Queues cont’d...

- **MAXIMUM(A):**
  
  \[
  \text{HEAP-MAXIMUM}(A) \\
  \text{return } A[1] \\
  \]
  
  \textbf{Time: } \Theta(1).

- **EXTRACT-MAX(A,n):**
  
  \[
  \text{HEAP-EXTRACT-MAX}(A, n) \\
  \text{if } n < 1 \\
  \quad \text{then error “heap underflow”} \\
  \quad max \leftarrow A[1] \\
  \quad A[1] \leftarrow A[n] \\
  \quad \text{MAX-HEAPIFY}(A, 1, n - 1) \quad \triangleright \text{remakes heap} \\
  \text{return } max \\
  \]
  
  \textbf{Analysis:} constant time assignments plus time for MAX-HEAPIFY.

  \textbf{Time: } \mathcal{O}(\log n).
Priority Queue cont’d...

- **INCREASE-KEY(A, i, key):**

  ```plaintext
  HEAP-INCREASE-KEY(A, i, key)
  if key < A[i]
    then error “new key is smaller than current key”
  A[i] ← key
  while i > 1 and A[PARENT(i)] < A[i]
    do exchange A[i] ← A[PARENT(i)]
       i ← PARENT(i)
  
  Analysis: Upward path from node i has length O(lg n) in an n-element heap.
  Time: O(lg n).
  
  MAX-HEAP-INSERT(A, key, n)
  A[n + 1] ← −∞
  HEAP-INCREASE-KEY(A, n + 1, key)
  
  Analysis: constant time assignments + time for HEAP-INCREASE-KEY.
  Time: O(lg n).
  ```
Binary Search Trees

• Support many dynamic-set operations
• Basic operations take time proportional to height $h$ of tree.
  \( \Theta(\log n) \) for balanced tree
  \( \Theta(n) \) for worst-case unbalanced tree
Binary Search Tree

Left children \leq \text{Node} \leq \text{Right children}
BST Data Structure

Each node contains the fields

- *key* (and possibly other satellite data).
- *left*: points to left child.
- *right*: points to right child.
- *p*: points to parent. \( p[root[T]] = \text{NIL} \).
Insertion

Tree-Insert( \( T, z \) )

<pre-cond>: \( T \) is a BST, \( z \) a node to be inserted

<post-cond>: \( T \) is a BST with \( z \) inserted

\[
\begin{align*}
y & \leftarrow \text{NIL} \\
x & \leftarrow \text{root}[T] \\
\text{while } x \neq \text{NIL} & \quad \text{do } y \leftarrow x \\
& \quad \text{if } \text{key}[z] < \text{key}[x] \\
& \quad \quad \text{then } x \leftarrow \text{left}[x] \\
& \quad \quad \text{else } x \leftarrow \text{right}[x] \\
p[z] & \leftarrow y \\
\text{if } y = \text{NIL} & \quad \text{then } \text{root}[T] \leftarrow z \quad \triangleright \text{Tree } T \text{ was empty} \\
\text{else } & \quad \text{if } \text{key}[z] < \text{key}[y] \\
& \quad \quad \text{then } \text{left}[y] \leftarrow z \\
& \quad \quad \text{else } \text{right}[y] \leftarrow z
\end{align*}
\]
Insertion

\[ key[z] = 36 \]

\[ T(n) = \Theta(h) \]
Building a Tree

Build-BST(Z)
<pre-cond>: Z is a set of nodes
$post-cond$: Returns a BST consisting of the nodes in Z

T ← NIL
for z in Z
    Tree-Insert(T, z)

Time for each insertion = Θ(h)

For balanced tree, number of nodes inserted into tree of height $h$ is $2^h$
Thus $T(n) = \sum_{h=0}^{\lfloor \log_2 n \rfloor} 2^h \Theta(h) = \Theta(n \log n)$
Searching the Tree

- **PreConditions**
  - Key 25
  - A binary search tree.

- **PostConditions**
  - Find key in BST (if there).
Searching the Tree

- Maintain a sub-tree.
- If the key is contained in the original tree, then the key is contained in the sub-tree.
Define Step

- Cut sub-tree in half.
- Determine which half the key would be in.
- Keep that half.

If key < root, then key is in left half.
If key = root, then key is found
If key > root, then key is in right half.
Searching the Tree

Tree-Search(x, k)
<pre-cond>: x is a BST, k is a key to search for
<post-cond>: returns the node matching k if it exists or NIL otherwise
if x = NIL or k = key[x]
  then return x
if k < key[x]
  then return TREE-SEARCH(left[x], k)
else return TREE-SEARCH(right[x], k)

Runtime = Θ(h)
Why use (balanced) binary search trees?

• What is the advantage over a sorted linear array?
  – Search time is the same
  – However, maintaining (inserting, deleting, modifying) is
    • $\Theta(\log n)$ for balanced BSTs
    • $\Theta(n)$ for arrays