Recurrence Relations

Analysis of Algorithms
Recursive Algorithms

• How do you analyze a recursive algorithm?
• No loops – cannot just “look at the code”
• Notice that the code in a recursive method contains a recurring call to itself
• Thus, you must use a recurrence relation to analyze a recursive algorithm
• The analysis depends on:
  – The preparation work to divide the input
  – The number of recursive calls
  – The concluding work to combine the results of the recursive calls
Recurrence Relations

- A recurrence relation is an equation or inequality that describes a function in terms of smaller inputs
  - $T(1) = 1$ (base case)
  - $T(N) = f(T(M))$ for $N > 0$ and $M < N$ (recursive case)
- The goal is to find a relationship between $T(N)$ and $T(M)$
Recurrence Relations

• Example:

```java
public int recursiveMethod (int N)
    int a;
    int b;
    if ( N ≤ 1) return 1;
    else
        a = recursiveMethod( N/2 );
        b = recursiveMethod( N/2 );
    return a + b;
```
Recurrence Relations

- For recursiveMethod
  - \( T(1) = 1 \) (1 statement in base case)
  - \( T(N) = 2T(N/2) + 1 \) (2 recursive calls and 1 addition)
- Solve using repeated substitutions
  - What is \( T(N/2) \)? \( T(N/2) = 2T(N/4) + 1 \)
  - Substitute \( 2T(N/4) + 1 \) back into original equation
  - \( T(N) = 2 \cdot (2T(N/4) + 1) + 1 = 4T(N/4) + 3 \)
  - What is \( T(N/4) \)? \( T(N/4) = 2T(N/8) + 1 \)
Recurrence Relations

\[ T(N) = 4 \left( 2T\left(\frac{N}{8}\right) + 1 \right) + 3 = 8T\left(\frac{N}{8}\right) + 7 \]

What is \( T(N/8) \)?

This could go on forever!

Try to find a pattern!

\[ T(N) = 2^k T\left(\frac{N}{2^k}\right) + (2^k - 1) \]

Let \( N = 2^k \) (let \( N \) be a power of 2 - can pad if necessary)

\[ T(N) = N \, T(1) + N - 1 \quad \text{(Recall } T(1) = 1 \text{ from base case)} \]

\[ = N + N - 1 \]

\[ = 2N - 1 \]

\[ T(N) = \Theta(N) \]
Recurrence Relations

• Example: Tree Traversals

```java
public void inorderTraversal (Tree root)
    if ( root is empty ) return;
    else
        inorderTraversal( left subtree of root );
        process( root );
        inorderTraversal( right subtree of root );
```
Recurrence Relations

\[
\begin{align*}
T(1) &= 1 \\
T(N) &= 2T(N/2) + 1 \\
T(N/2) &= 2T(N/4) + 1 \\
T(N) &= 2 \left(2T(N/4) + 1\right) + 1 \\
&= 4T(N/4) + 3 \\
\ldots \\
T(N) &= 2^k T(N/2^k) + 2^k - 1 \\
&= N \cdot T(1) + (N - 1) \\
&= 2N - 1 \\
&= O(N)
\end{align*}
\]
Recurrence Relations

Analysis of Recursive Binary Search:

\[
\begin{align*}
pos &= -1 \\
\text{if} \ (\text{start} \leq \text{end}) \\
& \quad\quad \text{mid} = (\text{start} + \text{end}) / 2 \\
& \quad\quad\text{if} \ (\text{target} < \text{list}[\text{mid}]) \\
& \quad\quad \quad\quad pos = \text{binSearch}(\text{list}, \text{target}, \text{start}, \text{mid}-1) \\
& \quad\quad\text{else if} \ (\text{target} > \text{list}[\text{mid}]) \\
& \quad\quad \quad\quad pos = \text{binSearch}(\text{list}, \text{target}, \text{start}, \text{mid}+1, \text{end}) \\
& \quad\quad\text{else} \\
& \quad\quad \quad\quad pos = \text{mid} \\
\text{return} \ pos
\end{align*}
\]
Recurrence Relations

1 base case, 1 recursive call with a list of size N/2, and 3 comparisons (at most) in the recursive case

\[ T(1) = 1 \]
\[ T(N) = T(N/2) + 3 \]
\[ T(N/2) = T(N/4) + 3 \]
\[ T(N) = [T(N/4) + 3] + 3 = T(N/4) + 6 = T(N/8) + 9 \]
\[ T(N) = T(N/2^k) + 3k, \ \text{let} \ N = 2^k, \ \text{or} \ k = \log_2 N \]
\[ T(N) = T(1) + 3 \log_2 N = 1 + 3 \log_2 N \]
\[ T(N) = O(\log_2 N) \]
Recurrence Relations

Analysis of mergesort: Recall that there are 2 recursive calls at each level of the recursion tree, there are N merges at each level, and there are k levels in the tree

\[
\begin{align*}
T(1) &= 1 \\
T(N) &= 2T(N/2) + N \\
&= 2^k \cdot T(N/2^k) + N \cdot k \\
&= N \cdot T(1) + N \log N \\
&= N + N \log N \\
&= O(N \log N)
\end{align*}
\]

N is number of merges at each level
k is equal to the total number of levels
let \( N = 2^k \), therefore \( k = \log_2 N \)
Recurrence Relations

Analysis of Strassen’s matrix multiplication: Recall that there are 7 recursive calls at each level of the recursion tree, there are $N^2$ multiplications at each level, and there are $k$ levels in the tree.

$T(1) = 1$

$T(N) = 7 \cdot T(N/2) + N^2$

$= 7^k \cdot T(N/2^k) + N^2 \cdot k$

$= 7^{\log_2 N} + N^2 \log N$

$= N^{\log_2 7} + N^2 \log N$

$= N^{2.81} + N^2 \log N$

$= O(N^{2.81})$

Assume each matrix is $N \times N = N^2$

$k$ is equal to the total number of levels

Let $N = 2^k$, therefore $k = \log_2 N$
Master Method

• Used for solving recurrences of the form
  \[ T(N) = a \ T \left( \frac{N}{b} \right) + f(N) \]
  where \( a \geq 1, \ b > 1, \ f(N) \geq 0 \)

• This form essentially divides a problem of size \( N \) into \( a \) sub-problems (\( a \) recursive calls), each of size \( \frac{N}{b} \)

• \( f(N) \) is the cost of combining the results of the recursion

• Memorize 3 cases (see next slide)
Master Method

1. If \( f(N) = O(N^{\log_b a - \varepsilon}) \) for \( \varepsilon > 0 \), then \( T(N) = \theta(N^{\log_b a}) \)

2. If \( f(N) = \theta(N^{\log_b a}) \), then \( T(N) = \theta(N^{\log_b a} \log N) \)

3. If \( f(N) = \Omega(N^{\log_b a + \varepsilon}) \) for \( \varepsilon > 0 \), and if \( a \cdot f(N/b) < \text{constant} \cdot f(N) \), then \( T(N) = \theta(f(N)) \)

The idea is to compare \( f(N) \) with \( f(N^{\log_b a}) \). The complexity is determined by the faster-growing function; i.e., the slower algorithm. (Note: the master method may not be helpful if the function does not fit one of these three cases.)
Master Method

Example: $T(N) = 9 \ T(N/3) + N$

$a = 9$, $b = 3$, $f(N) = N$

$N^{\log_b a} = N^{\log_3 9} = N^2$

Since $f(N) = O(N^{2-\varepsilon})$ where $0 < \varepsilon \leq 1$

Case 1 applies: $T(N) = \Theta(N^{\log_b a}) = \Theta(N^{\log_3 9}) = \Theta(N^2)$
Master Method

Example: \( T(N) = 4 \ T(N/2) + N + 5 \)

\( a = 4, \ b = 2, \ f(N) = N + 5 \)

\( N^{\log_b a} = N^{\log_2 4} = N^2 \)

\( f(N) = O(N^{2-\varepsilon}) \) where \( 0 < \varepsilon \leq 1 \)

Case 1 applies: \( T(N) = \Theta(N^{\log_b a}) = \Theta(N^{\log_2 4}) = \Theta(N^2) \)
Master Method

Example: \( T(N) = T(2N/3) + 1 \)
\( a = 1, \ b = 3/2, \ f(N) = 1 \)
\( N^{\log_b a} = N^{\log_{(3/2)} 1} = N^0 = 1 \)

\( f(N) = \Theta(N^0) = \Theta(1) = 1 \)
Case 2 applies: \( T(N) = \Theta(N^0 \log N) = \Theta(\log N) \)
Master Method

Example: \( T(N) = 3 \cdot T(N/4) + N \log N \)

\( a = 3, \ b = 4, \ f(N) = N \log N \)

\[ N^{\log_4 3} = N^{0.792} \]

\( f(N) = \Omega(N^{0.792+\varepsilon}) \) where \( 0 < \varepsilon < 0.208 \)

Check: \( a \cdot f(N/b) \leq \text{constant} \cdot f(N) \)

\( 3 \cdot (N/4) \log (N/4) \leq c \cdot N \log N \) ? Yes, let \( c = 3/4 \).

\( (3/4) \cdot N \log (N/4) \leq (3/4) \cdot N \log N \)

Case 3 applies: \( T(N) = \Theta(N \log N) \)
Master Method

Example: \( T(N) = 2 \ T(N/2) + N \log N \)

\[ a = 2, \ b = 2, \ f(N) = N \log N \]

\[ N^{\log_b a} = N^{\log_2 2} = N^1 = N \]

\( f(N) = \Omega(N^{1+\varepsilon}) \) where \( \varepsilon \leq \) ?

\( N \log N > N \), but not \textit{polynomially} larger

\( (N \log N) / N = \log N < N^\varepsilon \) for \( \varepsilon > 0 \), even if \( \varepsilon = 0.00001 \)

Case 3 does not apply.

Cannot use Master Method here.
Master Method

• Question: Recall Strassen’s algorithm for matrix multiplication $T(N) = 7 \ T(N/2) + N^2$. Prove using the Master Method that it is $O(N^{2.81})$. 
Simplified Master Method

• If recurrence is of the form \( T(N) = aT(N/b) + f(N) \) and \( f(N) \) is \( \Theta(N^d) \) with \( d \geq 0 \), then \( T(N) \) can be approximated by:
  1. \( T(N) = \Theta(N^d) \) if \( a < b^d \)
  2. \( T(N) = \Theta(N^d \log N) \) if \( a = b^d \)
  3. \( T(N) = \Theta(N^{\log_b a}) \) if \( a > b^d \)

• For example, let \( T(N) = 9 \ T(N/3) + N \)
  \( a = 9, \ b = 3, \ f(N) = N, \ d = 1, \ b^d = 3^1 = 3 \)
  Since \( a > b^d \) case 3 applies, and \( T(N) = \Theta(N^2) \)