Algorithm Design and Analysis

LECTURE 12
Solving Recurrences
• Master Theorem

Adam Smith
Review Question: Exponentiation

**Problem:** Compute $a^b$, where $b \in \mathbb{N}$ is $n$ bits long.

**Question:** How many multiplications?

**Naive algorithm:** $\Theta(b) = \Theta(2^n)$ (exponential in the input length!)

**Divide-and-conquer algorithm:**

$$a^b = \begin{cases} 
    a^{b/2} \cdot a^{b/2} & \text{if } b \text{ is even}; \\
    a^{(b-1)/2} \cdot a^{(b-1)/2} \cdot a & \text{if } b \text{ is odd.}
\end{cases}$$

$$T(b) = T(b/2) + \Theta(1) \Rightarrow T(b) = \Theta(\log b) = \Theta(n).$$
So far: 2 recurrences

- Mergesort; Counting Inversions
  \[ T(n) = 2 \ T(n/2) + \Theta(n) \quad = \Theta(n \log n) \]
- Binary Search; Exponentiation
  \[ T(n) = 1 \ T(n/2) + \Theta(1) \quad = \Theta(\log n) \]

**Master Theorem:** method for solving recurrences.
The master method applies to recurrences of the form

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

where \( a \geq 1, \, b > 1, \) and \( f \) is asymptotically positive, that is \( f(n) > 0 \) for all \( n > n_0. \)
Three common cases

Compare $f(n)$ with $n^{\log_b a}$:

1. $f(n) = O(n^{\log_b a} - \varepsilon)$ for some constant $\varepsilon > 0$.
   - $f(n)$ grows polynomially slower than $n^{\log_b a}$ (by an $n^\varepsilon$ factor).

   **Solution:** $T(n) = \Theta(n^{\log_b a})$. 
Three common cases

Compare \( f(n) \) with \( n^{\log_b a} \):

1. \( f(n) = O(n^{\log_b a - \varepsilon}) \) for some constant \( \varepsilon > 0 \).
   - \( f(n) \) grows polynomially slower than \( n^{\log_b a} \) (by an \( n^\varepsilon \) factor).
   
   **Solution:** \( T(n) = \Theta(n^{\log_b a}) \).

2. \( f(n) = \Theta(n^{\log_b a \log k n}) \) for some constant \( k \geq 0 \).
   - \( f(n) \) and \( n^{\log_b a} \) grow at similar rates.
   
   **Solution:** \( T(n) = \Theta(n^{\log_b a \log^{k+1} n}) \).
Three common cases (cont.)

Compare $f(n)$ with $n^{\log_b a}$:

3. $f(n) = \Omega(n^{\log_b a} + \varepsilon)$ for some constant $\varepsilon > 0$.
   
   • $f(n)$ grows polynomially faster than $n^{\log_b a}$ (by an $n^\varepsilon$ factor),

   and $f(n)$ satisfies the **regularity condition** that $af(n/b) \leq cf(n)$ for some constant $c < 1$.

   **Solution:** $T(n) = \Theta(f(n))$.
Idea of master theorem

Recursion tree:

```
f(n)
  /\       a
 f(n/b) f(n/b) ... f(n/b)
    /\       a
 f(n/b^2) f(n/b^2) ... f(n/b^2)
      ...         ...
           /\       /
             T(1)
```
Idea of master theorem

Recursion tree:

\[ f(n) \quad a \quad f(n) \]
\[ f(n/b) \quad f(n/b) \quad \cdots \quad f(n/b) \quad af(n/b) \]
\[ f(n/b^2) \quad f(n/b^2) \quad \cdots \quad f(n/b^2) \quad a^2 f(n/b^2) \]
\[ T(1) \]
Idea of master theorem

Recursion tree:

\[ T(n) = \begin{cases} f(n) & \text{if } n < b \frac{n}{b} \\ af\left(\frac{n}{b}\right) + \text{recurrences} & \text{otherwise} \end{cases} \]

The master theorem helps determine the time complexity of recursive algorithms.

\[ T(n) = \begin{cases} \Theta(f(n)) & \text{if } f(n) = \Theta\left(n^d\right) \\ \Theta\left(n^{\log_b a}\right) & \text{if } f(n) = \Theta\left(n^{\log_b a}\right) \\ \Theta\left(n^d \log n\right) & \text{if } f(n) = \Theta(n^{\log_b a} \log n) \end{cases} \]
Idea of master theorem

Recursion tree:

\[ f(n) \]

\[ f(n/b) \]
\[ f(n/b) \]
\[ \cdots \]
\[ f(n/b) \]
\[ a \]

\[ \cdots \]
\[ a \]

\[ f(n/b^2) \]
\[ f(n/b^2) \]
\[ \cdots \]
\[ f(n/b^2) \]
\[ a^2 f(n/b^2) \]

\[ h = \log_b n \]

\[ \#leaves = a^h \]
\[ = a^{\log_b n} \]
\[ = n^{\log_b a} \]

\[ T(1) \]
\[ = n^{\log_b a} T(1) \]
Idea of master theorem

Recursion tree:

\[ T(1) \quad n^{\log_b a} T(1) \]

**CASE 1:** The weight increases geometrically from the root to the leaves. The leaves hold a constant fraction of the total weight.
Idea of master theorem

Recursion tree:

\[ T(n) = \begin{cases} a T(n/b) & \text{if } f(n) = \Theta(n^\log_b a) \\ a^2 T(n/b^2) & \text{if } f(n) = \Theta(n^\log_b a \log n) \\ n^{\log_b a} T(1) & \text{if } f(n) = \Theta(n^k) \end{cases} \]

CASE 2: \((k = 0)\) The weight is approximately the same on each of the \(\log_b n\) levels.

\[ \Theta(n^{\log_b a \log n}) \]
Idea of master theorem

Recursion tree:

\[ f(n) \]

\[ \frac{f(n)}{a} \]

\[ f\left(\frac{n}{b}\right) \quad f\left(\frac{n}{b}\right) \quad \cdots \quad f\left(\frac{n}{b}\right) \]

\[ a \]

\[ \frac{f\left(\frac{n}{b^2}\right)}{a} \]

\[ f\left(\frac{n}{b^2}\right) \quad f\left(\frac{n}{b^2}\right) \quad \cdots \quad f\left(\frac{n}{b^2}\right) \]

\[ a^2 \]

\[ \frac{f\left(\frac{n}{b^4}\right)}{a^2} \]

\[ \vdots \]

\[ T(1) \]

\[ n^{\log_b a} T(1) \]

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

CASE 3: The weight decreases geometrically from the root to the leaves. The root holds a constant fraction of the total weight.
Examples

Ex. \( T(n) = 4T(n/2) + n \)
\( a = 4, \ b = 2 \implies n^{\log_b a} = n^2; \ f(n) = n. \)

Case 1: \( f(n) = O(n^2 - \varepsilon) \) for \( \varepsilon = 1. \)
\[ \therefore T(n) = \Theta(n^2). \]
Examples

Ex. \( T(n) = 4T(n/2) + n \)
\[ a = 4, \ b = 2 \Rightarrow n^{\log_b a} = n^2; \ f(n) = n. \]
Case 1: \( f(n) = O(n^{2-\varepsilon}) \) for \( \varepsilon = 1 \).
\[ \therefore \ T(n) = \Theta(n^2). \]

Ex. \( T(n) = 4T(n/2) + n^2 \)
\[ a = 4, \ b = 2 \Rightarrow n^{\log_b a} = n^2; \ f(n) = n^2. \]
Case 2: \( f(n) = \Theta(n^2 \lg^k n) \), that is, \( k = 0 \).
\[ \therefore \ T(n) = \Theta(n^2 \lg n). \]
Examples

Ex.  \( T(n) = 4T(n/2) + n^3 \)
\[ a = 4, \ b = 2 \implies n^{\log_b a} = n^2; \ f(n) = n^3. \]

Case 3: \( f(n) = \Omega(n^2 + \varepsilon) \) for \( \varepsilon = 1 \)
and \( 4(n/2)^3 \leq cn^3 \) (reg. cond.) for \( c = 1/2. \)
\[ \therefore \ T(n) = \Theta(n^3). \]
Examples

Ex. \( T(n) = 4T(n/2) + n^3 \)
\[ a = 4, \; b = 2 \implies n^{\log_b a} = n^2; \; f(n) = n^3. \]

**Case 3:** \( f(n) = \Omega(n^2 + \varepsilon) \) for \( \varepsilon = 1 \)
and \( 4(n/2)^3 \leq cn^3 \) (reg. cond.) for \( c = 1/2. \)
\[ \therefore T(n) = \Theta(n^3). \]

Ex. \( T(n) = 4T(n/2) + n^2/\lg n \)
\[ a = 4, \; b = 2 \implies n^{\log_b a} = n^2; \; f(n) = n^2/\lg n. \]
Master method does not apply. In particular, for every constant \( \varepsilon > 0 \), we have \( n^\varepsilon = \omega(\lg n). \)
• Reference on Master Th’m to be posted on web
• Master Th’m generalized by Akra and Bazzi to cover many more recurrences:

\[ T(n) = f(n) + \sum_{i=1}^{k} a_i T(b_i n + h_i(n)) \]

where \( h_i(n) = O\left(\frac{n}{\log^2 n}\right) \)

• See http://www.dna.lth.se/home/Rolf_Karlsson/akrabazzi.ps
Multiplying large integers

- Given $n$-bit integers $a$, $b$ (in binary), compute $c = ab$
- **Naïve (grade-school) algorithm:**
  - Write $a, b$ in binary
  - Compute $n$ intermediate products
  - Do $n$ additions
  - Total work: $\Theta(n^2)$
Multiplying large integers

- **Divide and Conquer** (Attempt #1):
  - Write $a = A_1 2^{n/2} + A_0$
    $b = B_1 2^{n/2} + B_0$
  - We want $ab = A_1B_1 2^n + (A_1B_0 + B_1A_0) 2^{n/2} + A_0B_0$
  - Multiply $n/2$–bit integers recursively
  - $T(n) = 4T(n/2) + \Theta(n)$
  - Alas! this is still $\Theta(n^2)$ (Master Theorem, Case 1)
• **Divide and Conquer (Attempt #1):**

  - Write \( a = A_1 \cdot 2^{n/2} + A_0 \)
    \( b = B_1 \cdot 2^{n/2} + B_0 \)

  - We want \( ab = A_1 B_1 \cdot 2^n + (A_1 B_0 + B_1 A_0) \cdot 2^{n/2} + A_0 B_0 \)

- Karatsuba’s idea:

  \[
  (A_0 + A_1) (B_0 + B_1) = A_0 B_0 + A_1 B_1 + (A_0 B_1 + B_1 A_0)
  \]

  - We can get away with 3 multiplications! (in yellow)

  \( x = A_1 B_1 \quad y = A_0 B_0 \quad z = (A_0 + A_1)(B_0 + B_1) \)

  - Now we use \( ab = A_1 B_1 \cdot 2^n + (A_1 B_0 + B_1 A_0) \cdot 2^{n/2} + A_0 B_0 \)

  \[
  = x \cdot 2^n + (z - x - y) \cdot 2^{n/2} + y
  \]
Multiplying large integers

\textbf{MULTIPLY} \((n, a, b)\)

\begin{itemize}
\item \(a\) and \(b\) are \(n\)-bit integers
\item Assume \(n\) is a power of 2 for simplicity
\end{itemize}

1. If \(n \leq 2\) then use grade-school algorithm else

2. \(A_1 \leftarrow a \mod 2^{n/2}\); \(B_1 \leftarrow b \mod 2^{n/2}\);

3. \(A_0 \leftarrow a \mod 2^{n/2}\); \(B_0 \leftarrow b \mod 2^{n/2}\).

4. \(x \leftarrow \text{MULTIPLY} \(n/2, A_1, B_1\)\)

5. \(y \leftarrow \text{MULTIPLY} \(n/2, A_0, B_0\)\)

6. \(z \leftarrow \text{MULTIPLY} \(n/2, A_1+A_0, B_1+B_0\)\)

7. Output \(x 2^n + (z-x-y)2^{n/2} + y\)
Multiplying large integers

- The resulting recurrence
  \[ T(n) = 3T(n/2) + \Theta(n) \]

- Master Theorem, Case 1:
  \[ T(n) = \Theta(n^{\log_2 3}) = \Theta(n^{1.59\ldots}) \]

- Note: There is a \( \Theta(n \log n) \) algorithm for multiplication (more on it later in the course).

A. Smith; based on slides by E. Demaine, C. Leiserson, S. Raskhodnikova.