Today:

- Master Method
- Matrix Multiplication
- Strassen’s Alg. For Matrix Mult.

COSC 581, Algorithms

January 16, 2014

Reading Assignments

- Today's class:
  - Chapter 4.2, 4.5

- Reading assignment for next class:
  - Chapter 4.1, 15.1

Recurrence Relations

- Equation or an inequality that characterizes a function by its values on smaller inputs.
- **Solution Methods** (Chapter 4)
  - Substitution Method.
  - Recursion-tree Method.
  - Master Method.
- Recurrence relations **arise when we analyze the running time of iterative or recursive algorithms**.

The Master Method

- Based on the Master theorem.
- "Cookbook" approach for solving recurrences of the form

  $T(n) = aT(n/b) + f(n)$

  - $a \geq 1$, $b > 1$ are constants.
  - $f(n)$ is asymptotically positive.
  - n/b may not be an integer, but we ignore floors and ceilings.

  – Requires memorization of 3+ cases.

The Master Theorem

Theorem 4.1

Let $a \geq 1$ and $b > 1$ be constants, let $f(n)$ be a function, and

Let $T(n)$ be defined on nonnegative integers by the recurrence

$T(n) = aT(n/b) + f(n)$, where we can replace n/b by $\lfloor n/b \rfloor$ or $\lceil n/b \rceil$.

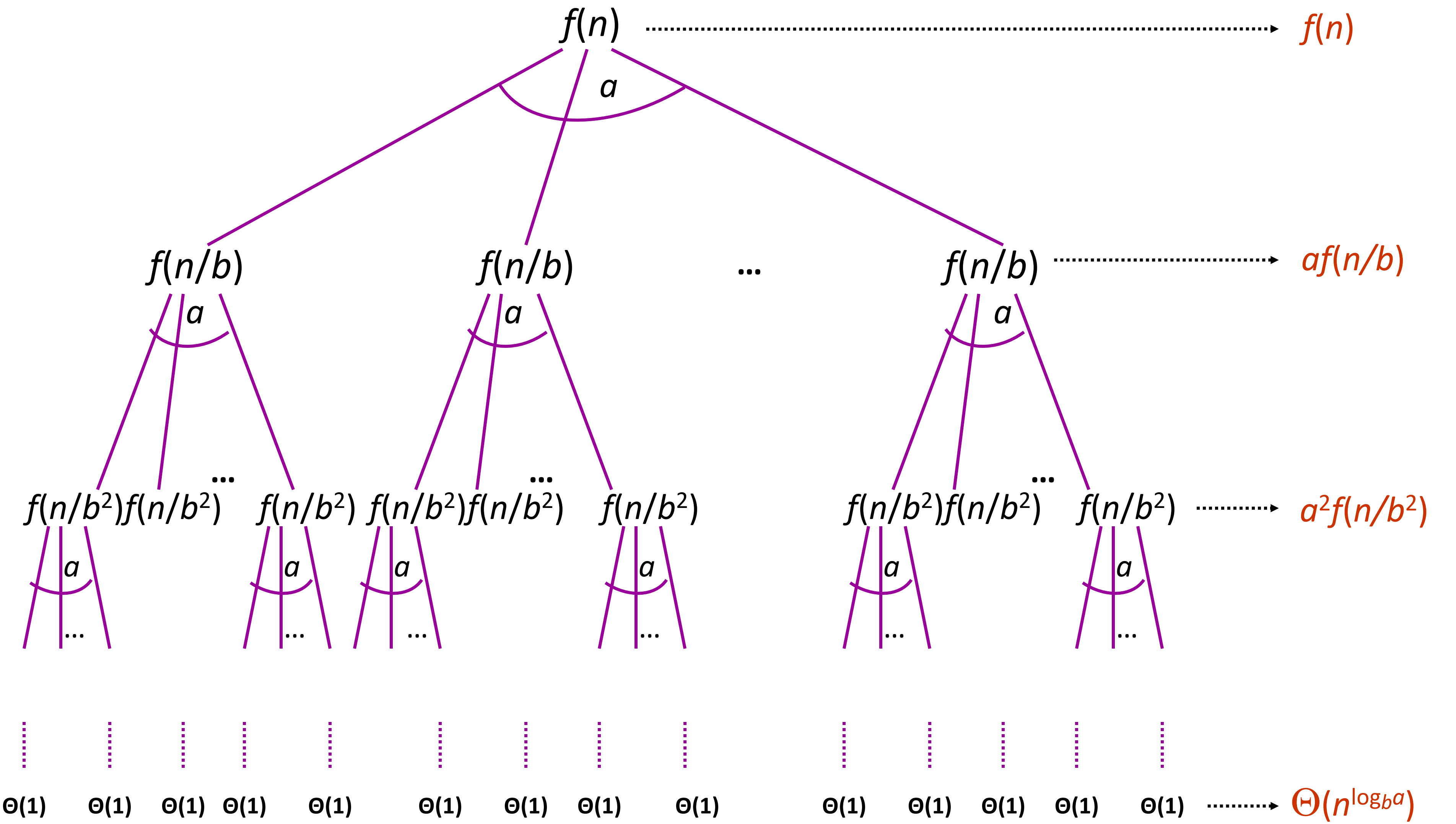
$T(n)$ can be bounded asymptotically in three cases:

1. If $f(n) = O(n^{\log_b a - \varepsilon})$ for some constant $\varepsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.
2. If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \lg n)$.
3. If $f(n) = \Omega(n^{\log_b a + \varepsilon})$ for some constant $\varepsilon > 0$,

   and if, for some constant $c < 1$ and all sufficiently large n,

   we have $a \cdot f(n/b) \leq c\, f(n)$, then $T(n) = \Theta(f(n))$.

Recursion tree view



Total: $T(n) = \Theta(n^{\log_b a}) + \sum_{j=0}^{\log_b n - 1} a^j f(n/b^j)$

The Master Theorem

Theorem 4.1

Let $a \geq 1$ and $b > 1$ be constants, let $f(n)$ be a function, and

Let $T(n)$ be defined on nonnegative integers by the recurrence

$T(n) = aT(n/b) + f(n)$, where we can replace n/b by $\lfloor n/b \rfloor$ or $\lceil n/b \rceil$.

$T(n)$ can be bounded asymptotically in three cases:

1. If $f(n) = O(n^{\log_b a - \varepsilon})$ for some constant $\varepsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.
2. If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \lg n)$.
3. If $f(n) = \Omega(n^{\log_b a + \varepsilon})$ for some constant $\varepsilon > 0$,
   and if, for some constant $c < 1$ and all sufficiently large n,
   we have $a \cdot f(n/b) \leq c\, f(n)$, then $T(n) = \Theta(f(n))$.

Recurrences

- Three basic behaviors
  - Dominated by initial case
  - Dominated by base case
  - All cases equal – we care about the depth

Gaining intuition on recurrences

Work per level changes geometrically with the level

- Geometrically increasing (dominated by leaves)
  - The bottom level wins
- Balanced (sum of internal nodes equal to leaves)
  - Equal contribution
- Geometrically decreasing (dominated by root)
  - The top level wins

Master Theorem – Case 1

- **Case 1:** If $f(n) = O(n^{\log_b a - \varepsilon})$ for some constant $\varepsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.
  - $n^{\log_b a} = a^{\log_b n}$: Number of leaves in the recursion tree.
  - $f(n) = O(n^{\log_b a - \varepsilon}) \Rightarrow$ Sum of the cost of the nodes at each internal level asymptotically smaller than the cost of leaves by a *polynomial* factor.
  - Cost of the problem dominated by leaves, hence cost is $\Theta(n^{\log_b a})$.

Master Theorem – Case 2

- **Case 2:** If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \lg n)$.
  - $n^{\log_b a} = a^{\log_b n}$: Number of leaves in the recursion tree.
  - $f(n) = \Theta(n^{\log_b a}) \Rightarrow$ Sum of the cost of the nodes at each level asymptotically the same as the cost of leaves.
  - There are $\Theta(\lg n)$ levels.
  - Hence, total cost is $\Theta(n^{\log_b a} \lg n)$.

Master Theorem – Case 3

- **Case 3:** If $f(n) = \Omega(n^{\log_b a+\varepsilon})$ for some constant $\varepsilon > 0$, and if, for some constant $c < 1$ and all sufficiently large n, we have $a \cdot f(n/b) \leq c\, f(n)$, then $\boldsymbol{T(n) = \Theta(f(n))}$.

  – $n^{\log_b a} = a^{\log_b n}$: Number of leaves in the recursion tree.

  – $f(n) = \Omega(n^{\log_b a+\varepsilon}) \Rightarrow$ Cost is dominated by the root. Cost of the root is asymptotically larger than the sum of the cost of the leaves by a polynomial factor.

  – Hence, cost is $\Theta(f(n))$.

Master Theorem – Case 3

- **Case 3:** If $f(n) = \Omega(n^{\log_b a+\varepsilon})$ for some constant $\varepsilon > 0$,
  and if, for some constant $c < 1$ and all sufficiently large n,
  we have $a \cdot f(n/b) \leq c\, f(n)$,
  then $T(n) = \Theta(f(n))$.

  Regularity condition

- Regularity condition means that total work increases as you go to larger problems
  - Examples that obey regularity condition:
    - Polynomials (n^k)
    - Polylogarithmic functions $(\lg^2 n)$
    - Exponentials (2^n)
    - Factorial functions $(n!)$
  - Example that doesn't obey regularity:
    - Functions that include trigonometrics $(n^{1+\sin n})$

Master Method – Examples

- $T(n) = 16T(n/4)+n$

- $T(n) = T(3n/7) + 1$

Master Method – Examples

- $T(n) = 16T(n/4)+n$

- $T(n) = T(3n/7) + 1$

The Master Theorem

Theorem 4.1

Let $a \geq 1$ and $b > 1$ be constants, let $f(n)$ be a function, and

Let $T(n)$ be defined on nonnegative integers by the recurrence

$T(n) = aT(n/b) + f(n)$, where we can replace n/b by $\lfloor n/b \rfloor$ or $\lceil n/b \rceil$.

$T(n)$ can be bounded asymptotically in three cases:

1. If $f(n) = O(n^{\log_b a - \varepsilon})$ for some constant $\varepsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.
2. If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \lg n)$.
3. If $f(n) = \Omega(n^{\log_b a + \varepsilon})$ for some constant $\varepsilon > 0$,

   and if, for some constant $c < 1$ and all sufficiently large n,

   we have $a \cdot f(n/b) \leq c\, f(n)$, then $T(n) = \Theta(f(n))$.

Gaps where Master Method doesn't apply

Master Recurrence Special Case

If $f(n) = \Theta(n^{\log_b a} \lg^k n)$ for $k \geq 0$, then recurrence has solution:

$$T(n) = \Theta(n^{\log_b a} \lg^{k+1} n)$$

- Previous Example: $T(n) = 2T(n/2) + n \lg n$
  - $a = 2$, $b=2$, $n^{\log_b a} = n^{\log_2 2} = n$, $f(n) = n \lg n$
  - Master Method doesn't apply
  - But, Special case applies, where $k = 1$

    Solution:
    - $T(n) = \Theta(n^{\log_b a} \lg^{k+1} n)$
    - $T(n) = \Theta(n \lg^2 n)$

Back to Divide and Conquer…

- Matrix multiplication

Basic Matrix Multiplication

```
void matrix_mult (){
  for (i = 1; i <= n; i++) {
        for (j = 1; j <= n; j++) {
             compute c_{i,j};
  }}}
```

Standard matrix multiplication algorithm

Time analysis:

$$c_{i,j} = \sum_{k=1}^{n} a_{ik} b_{kj}$$

$\Theta(n^2)$ entries, each of which requires $\Theta(n)$ work to calculate ➔ runtime = $\Theta(n^3)$

Matrix Multiplication using Divide and Conquer

- Basic divide and conquer method:

To multiply two *n* x *n* matrices, *A* x *B = C*, divide into sub-matrices:

$$\begin{vmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{vmatrix} \cdot \begin{vmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{vmatrix} = \begin{vmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{vmatrix}$$

$C_{11} = A_{11}B_{11} + A_{12}B_{21}$

$C_{12} = A_{11}B_{12} + A_{12}B_{22}$

$C_{21} = A_{21}B_{11} + A_{22}B_{21}$

$C_{22} = A_{21}B_{12} + A_{22}B_{22}$

2x2 matrix multiplication can be accomplished in 8 multiplications and 4 additions.

Runtime of Divide & Conquer Matrix Multiplication

- Recurrence:

$$T(n) = 8T\left(\frac{n}{2}\right) + \Theta(n^2)$$

- Solution:

Runtime of Divide & Conquer Matrix Multiplication

- Recurrence:

$$T(n) = 8T\left(\frac{n}{2}\right) + \Theta(n^2)$$

- Solution:

$$f(n) = n^2$$
$$n^{\log_b a} = n^{\log_2 8} = n^3$$

Case 1 of Master Method ➔ solution = $\Theta(n^3)$.

- No better than "ordinary" approach.
- What to do?

Strassens's Matrix Multiplication

- Strassen (1969) showed that 2x2 matrix multiplication can be accomplished in 7 multiplications and 18 additions or subtractions

$$T(n) = 7T\left(\frac{n}{2}\right) + \Theta(n^2)$$

$$f(n) = n^2$$

$$n^{\log_b a} = n^{\log_2 7} = n^{2.81}$$

Case 1 of Master Method ➔ solution = $\Theta(n^{2.81})$.

- His method uses Divide and Conquer Approach.

Strassen's Matrix Multiplication

Strassen observed [1969] that the product of two matrices can be computed in general as follows:

$$\begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} * \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}$$

$$= \begin{pmatrix} P_5 + P_4 - P_2 + P_6 & P_1 + P_2 \\ P_3 + P_4 & P_5 + P_1 - P_3 - P_7 \end{pmatrix}$$

Formulas for Strassen's Algorithm

$P_1 = A_{11} * (B_{12} - B_{22})$

$P_2 = (A_{11} + A_{12}) * B_{22}$

$P_3 = (A_{21} + A_{22}) * B_{11}$

$P_4 = A_{22} * (B_{21} - B_{11})$

$P_5 = (A_{11} + A_{22}) * (B_{11} + B_{22})$

$P_6 = (A_{12} - A_{22}) * (B_{21} + B_{22})$

$P_7 = (A_{11} - A_{21}) * (B_{11} + B_{12})$

Formulas for Strassen's Algorithm

$P_1 = A_{11} * (B_{12} - B_{22})$

$P_2 = (A_{11} + A_{12}) * B_{22}$

$P_3 = (A_{21} + A_{22}) * B_{11}$

$P_4 = A_{22} * (B_{21} - B_{11})$

$P_5 = (A_{11} + A_{22}) * (B_{11} + B_{22})$

$P_6 = (A_{12} - A_{22}) * (B_{21} + B_{22})$

$P_7 = (A_{11} - A_{21}) * (B_{11} + B_{12})$

First, create 10 matrices, each of which is $n/2$ x $n/2$.
Time = $\Theta(n^2)$

Formulas for Strassen's Algorithm

$P_1 = A_{11} * (B_{12} - B_{22})$

$P_2 = (A_{11} + A_{12}) * B_{22}$

$P_3 = (A_{21} + A_{22}) * B_{11}$

$P_4 = A_{22} * (B_{21} - B_{11})$

$P_5 = (A_{11} + A_{22}) * (B_{11} + B_{22})$

$P_6 = (A_{12} - A_{22}) * (B_{21} + B_{22})$

$P_7 = (A_{11} - A_{21}) * (B_{11} + B_{12})$

First, create 10 matrices, each of which is $n/2$ x $n/2$. Time = $\Theta(n^2)$

Then, recursively compute 7 matrix products

Then add together

$$\begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} * \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}$$

$$= \begin{pmatrix} P_5 + P_4 - P_2 + P_6 & P_1 + P_2 \\ P_3 + P_4 & P_5 + P_1 - P_3 - P_7 \end{pmatrix}$$

Time = $\Theta(n^2)$

Resulting Runtime for Strassens's Matrix Multiplication

$$T(n) = \Theta(1) + \Theta(n^2) + 7T\left(\frac{n}{2}\right) + \Theta(n^2)$$
$$= 7T\left(\frac{n}{2}\right) + \Theta(n^2)$$

$$f(n) = n^2$$
$$n^{\log_b a} = n^{\log_2 7} = n^{2.81}$$

Case 1 of Master Method ➔ solution = $\Theta(n^{2.81})$.

Practical Issues with Strassen’s

- Constant factor in Strassen > constant in naïve $\Theta(n^2)$ approach
- If matrices are sparse, then methods tailored to sparse matrices are faster
- Strassen’s isn’t quite as numerically stable
- Submatrices consume space
- Typically, use naïve approach for small matrices

How quickly can we multiply matrices?

- Strassen's algorithm: $O(n^{2.80736})$ time
- **Coppersmith–Winograd algorithm (1990)**: $O(n^{2.376})$ time.
  - Frequently used as a building block in other algorithms to prove theoretical time bounds.
  - However, not used in practice; only provides an advantage for extremely large matrices
- Best achieved to date (2011): $O(n^{2.3727})$
- Obvious lower bound = ?

How quickly can we multiply matrices?

- Strassen's algorithm: $O\ (n^{2.80736})$ time
- **Coppersmith–Winograd algorithm (1990)**: $O\ (n^{2.376})$ time.
  - Frequently used as a building block in other algorithms to prove theoretical time bounds.
  - However, not used in practice; only provides an advantage for extremely large matrices
- Best achieved to date (2011): $O\ (n^{2.3727})$
- Obvious lower bound = $\Omega(n^2)$, because you at least have to fill in the answer.

In-Class Exercise

You want to develop a matrix multiplication algorithm that is asymptotically faster than Strassen's algorithm.

Your algorithm will use the divide-and-conquer method, dividing each matrix into pieces of size $n/4 \times n/4$; the divide and combine steps together will take $\Theta(n^2)$ time.

You need to determine how many subproblems your algorithm has to create in order to beat Strassens' algorithm.

What is the largest integral (i.e., integer) number of subproblems your algorithm can have that would be asymptotically faster than Strassen's algorithm?

Reading Assignments

- Today's class:
  - Chapter 4.2, 4.5
- Reading assignment for next class:
  - Chapter 4.1, 15.1
  - (Maximum subarrays; dynamic programming)