Recurrence Relations

Ira A. Fulton Schools of Engineering
Arizona State University

Recurrence Relations

- When we analyze the running time of a recursive algorithm, we often end up with a recurrence relation.
- There is no universal solutions for solving recurrence relations.
- However, there are standard methods for solving many commonly seen recurrence relations.

General Form

- Very often, the time complexity $T(n)$ of an algorithm is given by a recurrence relation. We need to solve it to get its asymptotic notation. The following is the general form:
- $T(n) = a\,T(n/b) + f(n)$, $a \geq 1$ and $b > 1$.
- Example: $T(n) = T(n/2) + 1$
- Example: $T(n) = 2T(n/2) + f(n)$, where $f(n) \in \Theta(n)$
- Example: $T(n) = 8T(n/2) + f(n)$, where $f(n) \in \Theta(n^2)$
- Example: $T(n) = 7T(n/2) + f(n)$, where $f(n) \in \Theta(n^2)$

Example

- Let $T(n)$ be the time to search a sorted array of n elements, using binary search. Then we have
- $T(n) = T(n/2) + 1$
- The parameters in the general form are
  - $a=1$
  - $b=2$
  - $f(n)=1$.

Example

- Let A be an N-by-N matrix and B be an N-by-N matrix. We can use block multiplication to get the product of A with B. https://en.wikipedia.org/wiki/Matrix_multiplication
- http://mathworld.wolfram.com/BlockMatrix.html
- Assume that we partition A into A_{11}, A_{12}, A_{21}, A_{22}. We partition B into B_{11}, B_{12}, B_{21}, B_{22}. Let C be the product of A with B. Then $C_{ij} = A_{i1} B_{1j} + A_{i2} B_{2j}$
- Therefore, the time complexity of multiplying two N-by-N matrices is T(N), where
- $T(n) = 8T(n/2) + n^2$, $a=8$, $b=2$, $f(n)=n^2$.

Example

- Volker Strassen discovered a very smart algorithm for block matrix multiplication.
- https://en.wikipedia.org/wiki/Strassen_
- The time complexity of multiplying two N-by-N matrices using Strassen's algorithm is T(n), where
- $T(n) = 7T(n/2) + c \times n^2$, $a=7$, $b=2$, $f(n)= c \times n^2$.

Example

- **Example: T(n) = T(n/2) + 1**
- **T(1) is a constant**
- **T(2) = 1 + T(1)**
- **T(4) = 1 + T(2) = 2 + T(1)**
- **T(8) = 1 + T(4) = 3 + T(1)**
- **T(16) = 1 + T(8) = 4 + T(1)**
- **T(32) = 1 + T(16) = 5 + T(1)**

- **It *looks like* T(n)** $\in \boldsymbol{\Theta}(\log(\mathbf{n}))$

Example

- Example: $T(n) = T(0.7n) + T(0.2n) + n^2$
- $T(n) = n^2 + T(0.7n) + T(0.2n) = n^2 + (0.7n)^2 + (0.2n)^2 + T(0.49n) + 2T(0.14n) + T(0.04n) = ???$

- This does not seem to be easy.

- We will present a very general method, known as the Master Method.

The Master Method, Case 2

- If $f(n) \in \Theta(\mathrm{n}^{\log_\mathrm{b} \mathrm{a}})$, then $T(n) \in \Theta(\mathrm{n}^{\log_\mathrm{b} \mathrm{a}} \log \mathrm{n})$
- We call the above case 2.

The Master Method, Case 1

- If $f(n) \in \Theta(n^{\log_b a})$, then $T(n) \in \Theta(n^{\log_b a} \log n)$
- We call the above case 2.

- Consider the case where $f(n)$ grows asymptotically slower than $n^{\log_b a}$.

- If $f(n) \in O\left(n^{(\log_b a)-\epsilon}\right)$ for some constant $\epsilon > 0$, then $T(n) \in \Theta(n^{\log_b a})$
- We call the above case 1.

The Master Method, Case 3

- If $f(n) \in \Theta(\mathrm{n}^{\log_b a})$, then $T(n) \in \Theta(\mathrm{n}^{\log_b a} \log \mathrm{n})$
- We call the above case 2.
- Consider the case where $f(n)$ grows asymptotically faster than $\mathbf{n}^{\log_b a}$.

- If $f(n) \in \Omega\left(\mathbf{n}^{(\log_b a)+\epsilon}\right)$ for some constant $\epsilon > 0$, and $a \times f\left(\frac{n}{b}\right) \leq c \times f(n)$ for $n \geq N$, where N is a constant and $c \in (0, 1)$, then $T(n) \in \Theta\big(\mathbf{f}(\mathbf{n})\big)$.
- We call the above case 3.

How to Use the Master Method?

- We start with checking whether it is Case_2.
- Study $\lim_{n\to\infty} \frac{\mathbf{f(n)}}{\mathbf{n}^{\log_{\mathbf{b}}\mathbf{a}}}$
- If $\lim_{n\to\infty} \frac{\mathbf{f(n)}}{\mathbf{n}^{\log_{\mathbf{b}}\mathbf{a}}} = c$ for some positive constant c, then $f(n) \in \boldsymbol{\Theta}\left(\mathbf{n}^{\log_{\mathbf{b}}\mathbf{a}}\right)$
- This implies that we have Case_2.
- As a result, $\mathrm{T}(n) \in \boldsymbol{\Theta}\left(\log(n) \times \mathbf{n}^{\log_{\mathbf{b}}\mathbf{a}}\right)$

How to Use the Master Method?

- Now assume that it is NOT Case_2.
- If $\lim_{n\to\infty} \frac{\mathbf{f(n)}}{\mathbf{n^{\log_b a}}} = \mathbf{0}$, we will check for Case_1, as it cannot be Case_3.
- If $\lim_{n\to\infty} \frac{\mathbf{f(n)}}{\mathbf{n^{\log_b a}}} = \infty$, we will check for Case_3, as it cannot be Case_1.

How to Use the Master Method?

- Suppose we want to check whether it is Case_1.
- A necessary condition for Case_1 is $\lim_{n\to\infty} \frac{\mathrm{f(n)}}{\mathrm{n^{\log_b a}}} = 0$.
- But this condition is not sufficient.
- We need to find a constant $\epsilon > 0$ such that $f(n) \in O(n^{(\log_b a)-\epsilon})$.
- If $\lim_{n\to\infty} \frac{\mathrm{f(n)}}{\mathrm{n^{(\log_b a)-\epsilon}}} < \infty$, then $f(n) \in O(n^{(\log_b a)-\epsilon})$.
- This implies that we have Case_1.
- As a result, $\mathrm{T}(n) \in \Theta\left(\mathrm{n^{\log_b a}}\right)$

How to Use the Master Method?

- Suppose we want to check whether it is Case_3.
- A necessary condition for Case_3 is $\lim_{n\to\infty} \frac{\mathbf{f(n)}}{\mathbf{n^{\log_b a}}} = \infty$.
- But this condition is not sufficient.
- We need to find a constant $\epsilon > 0$ such that $f(n) \in \Omega(n^{(\log_b a)+\epsilon})$.
- If $\lim_{n\to\infty} \frac{\mathbf{f(n)}}{\mathbf{n^{(\log_b a)+\epsilon}}} > 0$, then $f(n) \in \Omega(n^{(\log_b a)+\epsilon})$.
- This alone does not imply that we have Case_3.
- We need to check the regularity condition.

How to Use the Master Method?

- We need to find a constant c, $0 < c < 1$ and an integer N such that $af\left(\frac{n}{b}\right) \leq c \times f(n), \forall n \geq N$.
- If $f(n) \in \Omega\left(n^{(\log_b a)+\epsilon}\right)$for some constant $\epsilon > 0$ and the regularity condition also holds, then we have Case_3.
- As a result, T$(n) \in \Theta(f(n))$.

Example: T(n) = 8T(n/2) + n^2

- This is the recurrence we got from the block matrix multiplication. a=8, b=2, f(n)=n^2.
- $\lim_{n\to\infty} \frac{\mathbf{f(n)}}{\mathbf{n^{\log_b a}}} = \lim_{n\to\infty} \frac{n^2}{n^3} = \mathbf{0}$. This is not Case_2.
- Check for Case_1.
- For $\epsilon = 0.5$, we have $\lim_{n\to\infty} \frac{\mathbf{f(n)}}{\mathbf{n^{\log_b a-\epsilon}}} = \lim_{n\to\infty} \frac{n^2}{n^{2.5}} = \mathbf{0}$.
- Therefore we have Case_1.
- T(n) = $\Theta(n^3)$.

Example: T(n) = 7T(n/2) + $c \times n^2$

- This is the recurrence we got from This is the recurrence for Strassen's matrix multiplication.
- a=7, b=2, f(n)= $c \times n^2$
- $\lim_{n\to\infty} \frac{\mathbf{f(n)}}{\mathbf{n^{\log_b a}}} = \lim_{n\to\infty} \frac{c\times n^2}{n^{2.807}} = \mathbf{0}$. This is not Case_2.
- Check for Case_1.
- For $\epsilon = \mathbf{0.1}$, we have $\lim_{n\to\infty} \frac{\mathbf{f(n)}}{\mathbf{n^{\log_b a-\epsilon}}} = \lim_{n\to\infty} \frac{c\times n^2}{n^{2.707}} = \mathbf{0}$.
- Therefore we have Case_1.
- T(n) = Θ($n^{2.807}$).

More Examples

4.5-2

Professor Caesar wishes to develop a matrix-multiplication algorithm that is asymptotically faster than Strassen's algorithm. His algorithm will use the divide-and-conquer method, dividing each matrix into pieces of size $n/4 \times n/4$, and the divide and combine steps together will take $\Theta(n^2)$ time. He needs to determine how many subproblems his algorithm has to create in order to beat Strassen's algorithm. If his algorithm creates a subproblems, then the recurrence for the running time $T(n)$ becomes $T(n) = aT(n/4) + \Theta(n^2)$. What is the largest integer value of a for which Professor Caesar's algorithm would be asymptotically faster than Strassen's algorithm?

- **Find the largest integer a such that** $\frac{\log_2(\boldsymbol{a})}{\log_2(\mathbf{4})} < \frac{\log_2(\mathbf{7})}{\log_2(\mathbf{2})}$
- $\log_2(\boldsymbol{a}) < \mathbf{2} \times \log_2(\mathbf{7}) = \log_2(\mathbf{49})$
- $\boldsymbol{a} = \mathbf{48}$ **is the answer to this question.**
- **Need to verify this is indeed the case.**

More Examples

- **(a): Case 1**
- **(b): Case 2**
- **(c): Case 3.**
- **(d): Case 3.**

4.5-1

Use the master method to giv
rences.

a. $T(n) = 2T(n/4) + 1$.

b. $T(n) = 2T(n/4) + \sqrt{n}$.

c. $T(n) = 2T(n/4) + n$.

d. $T(n) = 2T(n/4) + n^2$.

More Examples

More Examples

- (a): Case 3
- (b): Case 3
- (c): Case 2
- (d): Case 3
- (e): Case 1
- (f): Case 2
- (g): None of the three cases. However, we can prove that $T(n) = \Theta(\mathbf{n}^3)$

4-1 Recurrence examples

Give asymptotic upper and lowe
rences. Assume that $T(n)$ is co
possible, and justify your answe

a. $T(n) = 2T(n/2) + n^4$.

b. $T(n) = T(7n/10) + n$.

c. $T(n) = 16T(n/4) + n^2$.

d. $T(n) = 7T(n/3) + n^2$.

e. $T(n) = 7T(n/2) + n^2$.

f. $T(n) = 2T(n/4) + \sqrt{n}$.

g. $T(n) = T(n-2) + n^2$.

More Examples

More Examples

- (a): Case 1
- (b): None of the three cases
- (c): Case 3
- (d): Case 2
- (e): None of the three cases
- (f): $\boldsymbol{T(n) = \Theta(\mathbf{n})}$
- (g): $\boldsymbol{T(n) = \Theta(\log \mathbf{n})}$
- (h): $\boldsymbol{T(n) = \Theta(\mathbf{n}\log \mathbf{n})}$

4-3 More recurrence examples

Give asymptotic upper and lower bounds for $T($ rences. Assume that $T(n)$ is constant for suffici as tight as possible, and justify your answers.

a. $T(n) = 4T(n/3) + n \lg n$.

b. $T(n) = 3T(n/3) + n/\lg n$.

c. $T(n) = 4T(n/2) + n^2\sqrt{n}$.

d. $T(n) = 3T(n/3 - 2) + n/2$.

e. $T(n) = 2T(n/2) + n/\lg n$.

f. $T(n) = T(n/2) + T(n/4) + T(n/8) + n$.

g. $T(n) = T(n-1) + 1/n$.

h. $T(n) = T(n-1) + \lg n$.

i. $T(n) = T(n-2) + 1/\lg n$.

j. $T(n) = \sqrt{n}T(\sqrt{n}) + n$.

More Examples

It Does Not Work All the Time

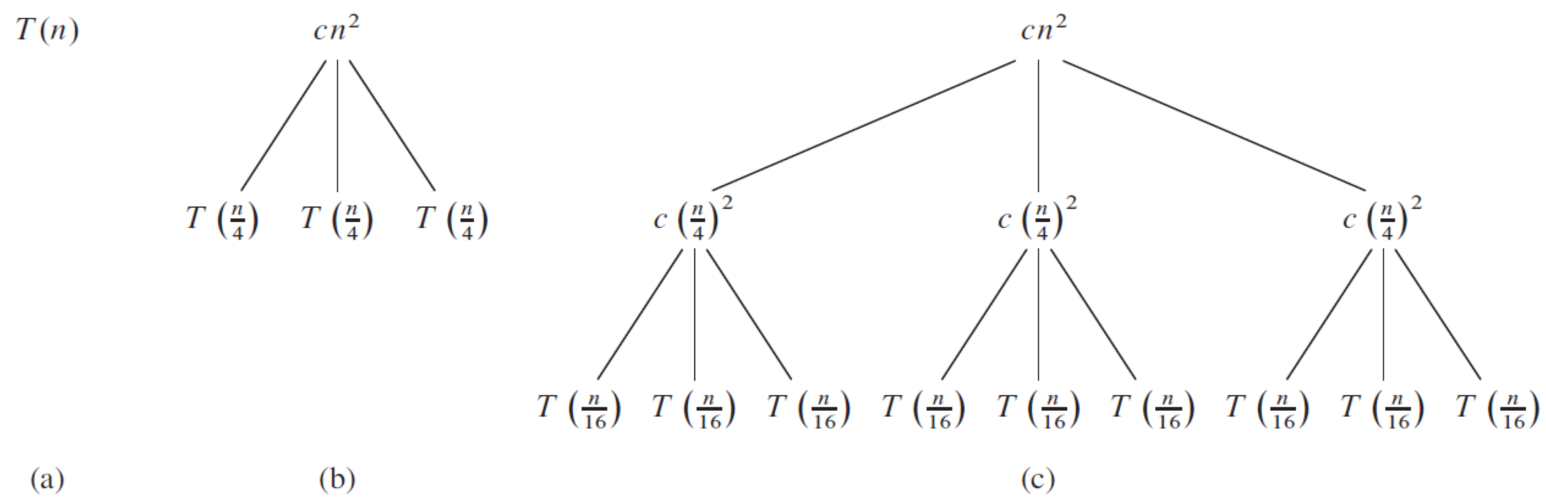
- There are some cases where the master method does not work.
- Consider the straightforward method for computing the n-th Fibonacci number.
- $T(n) = T(n-1) + T(n-2) + 1$.
- Well, the master method is not always useful.

The Substitution Method (Section 4.3)

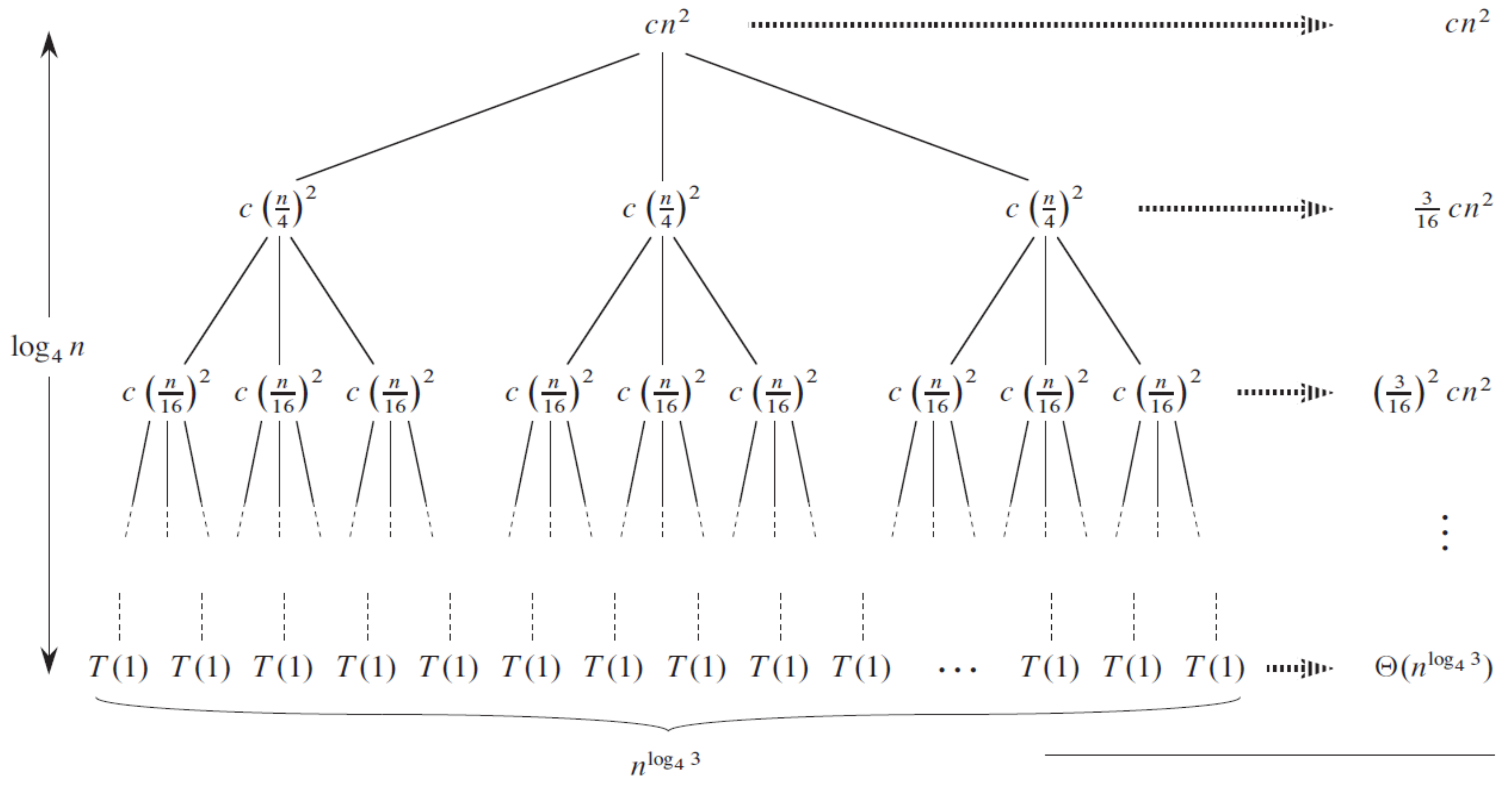
- T(n) = 2T(n/2) + 5n
- Guess: T(n) ≤ c×n lg (n) for all n ≥ N, for c≥5
- Proof: for any n s.t. N < n ≤ 2N, we have
- $T(n) = 2T\left(\frac{n}{2}\right) + 5n \leq 2\left(c\frac{n}{2}\lg\left(\frac{n}{2}\right)\right) + 5n = cn(\lg(n) - 1) + 5n = cn \times \lg(n) + (5 - c)n \leq cn \times \lg(n)$
- Take $c = \max\{T(2), T(3), 5\}$ and N=2, we have
- $T(2) \leq c \leq cn \times \lg(n), T(2) \leq c \leq cn \times \lg(n)$.
- This proves that $T(n) \in O(n \lg n)$

Recursion Tree (Section 4.4)

- $T(n) = 3T(n/4) + c\times n^2$
- $T(n) \in O(n^2)$



Recursion Tree (Section 4.4)



(d)

- $1 + x + x^2 + \cdots + x^k \leq \frac{1}{1-x}, x = \frac{3}{16}$
- $T(n) \in O(n^2)$.

ASU Ira A. Fulton Schools of Engineering

Arizona State University