Design and Analysis of Algorithms

Recurrences

Lecture 7-8

Instructor: Dr. G P Gupta

Recurrences and Running Time

- An equation or inequality that describes a function in terms of its value on smaller inputs.

  T(n) = T(n-1) + n
- Recurrences arise when an algorithm contains recursive calls to itself
- What is the actual running time of the algorithm?
- Need to solve the recurrence
  - Find an explicit formula of the expression
  - Bound the recurrence by an expression that involves n

Recurrences

•Recurrences go hand in hand with the divide-and-conquer paradigm,

- because they give us a natural way to characterize the running times of divide-and-conquer algorithms.

• A ***recurrence*** is an equation or inequality that describes a function in terms of its value on smaller inputs.

Recurrences cont..

- The expression:

$$T(n) = \begin{cases} c & n = 1 \\ 2T\left(\frac{n}{2}\right) + cn & n > 1 \end{cases}$$

- is a *recurrence*.
  - *Recurrence:* an equation that describes a function in terms of its value on smaller functions

Recurrence Examples

$$s(n) = \begin{cases} 0 & n = 0 \\ c + s(n-1) & n > 0 \end{cases}$$

$$s(n) = \begin{cases} 0 & n = 0 \\ n + s(n-1) & n > 0 \end{cases}$$

$$T(n) = \begin{cases} c & n = 1 \\ 2T\left(\frac{n}{2}\right) + c & n > 1 \end{cases}$$

$$T(n) = \begin{cases} c & n = 1 \\ aT\left(\frac{n}{b}\right) + cn & n > 1 \end{cases}$$

Methods for Solving Recurrences

- Iteration method
- Substitution method
- Recursion tree method
- Master method

The Iteration Method

- Convert the recurrence into a summation and try to bound it using known series
  - Iterate the recurrence until the initial condition is reached.
  - Use back-substitution to express the recurrence in terms of *n* and the initial (boundary) condition.

The Iteration Method

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

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

$T(n/4) = c + T(n/8)$

$$
\begin{aligned}
T(n) &= c + T(n/2)\\
&= c + c + T(n/4)\\
&= c + c + c + T(n/8)
\end{aligned}
$$

Assume $n = 2^k$

$$
\begin{aligned}
T(n) &= \underbrace{c + c + \dots + c}_{\text{k times}} + T(1)\\
&= c\lg n + T(1)\\
&= \Theta(\lg n)
\end{aligned}
$$

Iteration Method – Example

$T(n) = n + 2T(n/2)$ Assume: $n = 2^k$

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

$$
\begin{aligned}
T(n) &= n + 2T(n/2)\\
&= n + 2(n/2 + 2T(n/4))\\
&= n + n + 4T(n/4)\\
&= n + n + 4(n/4 + 2T(n/8))\\
&= n + n + n + 8T(n/8)\\
\dots &= i*n + 2^iT(n/2^i)\\
&= k*n + 2^kT(1)\\
&= n\lg n + nT(1) = \Theta(n\lg n)
\end{aligned}
$$

Substitution method

- **The substitution method (CLR 4.1)**
  - "Making a good guess" method
  - *Guess the form of the answer*, then use induction to find the constants and show that solution works
  - Examples:
    - $T(n) = 2T(n/2) + \Theta(n) \rightarrow T(n) = \Theta(n \lg n)$
    - $T(n) = 2T(\lfloor n/2 \rfloor) + n \rightarrow$???

Substitution method

1. Guess a solution
2. Use induction to prove that the solution works

Substitution method

- Guess a solution
  - $T(n) = O(g(n))$
  - Induction goal: apply the definition of the asymptotic notation
    - $T(n) \le c.\ g(n)$, for some $c > 0$ and $n \ge n_0$ (strong induction)
  - Induction hypothesis: $T(k) \le c.g(k)$ for all $k < n$
- Prove the induction goal
  - Use the **induction hypothesis** to find some values of the constants c and n_0 for which the **induction goal** holds

12

Example: Binary Search

$T(n) = d + T(n/2)$

- Guess: $T(n) = O(\lg n)$
  - ***Induction goal:*** $T(n) \le c.\ \lg n$, for some d and $n \ge n_0$
  - ***Induction hypothesis:*** $T(n/2) \le c.\ \lg(n/2)$
- ***Proof of induction goal:***

  $T(n) = T(n/2) + d \le c.\ \lg(n/2) + d$

  $= c.\lg n - c + d \le c.\lg n$

  if: $n \ge 1,\ -c + d \le 0,\ c \ge d$

Example 2

$T(n) = T(n-1) + n$

- **Guess:** $T(n) = O(n^2)$
  - **Induction goal:** $T(n) \le c.n^2$, for some c and $n \ge n_0$
  - **Induction hypothesis:** $T(k-1) \le c(k-1)^2$ for all $k < n$
- **Proof of induction goal:**

  $T(n) = T(n-1) + n \le c\,(n-1)^2 + n$

  $= cn^2 - (2cn - c - n) \le cn^2$

  if: $2cn - c - n \ge 0 \Leftrightarrow c \ge n/(2n-1) \Leftrightarrow c \ge 1/(2 - 1/n)$
  - For $n \ge 1 \Rightarrow 2 - 1/n \ge 1 \Rightarrow$ any $c \ge 1$ will work

Example 3

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

- **Guess:** $T(n) = O(n\lg n)$
  - **Induction goal:** $T(n) \le c.\ n\lg n$, for some c and $n \ge n_0$
  - **Induction hypothesis:** $T(n/2) \le c.\ n/2\ \lg(n/2)$
- **Proof of induction goal:**

  $T(n) = 2T(n/2) + n \le 2c\,(n/2)\lg(n/2) + n$

  $= c.n\lg n - cn + n \le c.n\lg n$

  if: $n \ge 1,\ -cn + n \le 0 \Rightarrow c \ge 1$

Changing variables

$T(n) = 2T(\sqrt{n}\,) + \lg n$

- Rename: $m = \lg n \Rightarrow n = 2^m$

$T(2^m) = 2T(2^{m/2}) + m$

- Rename: $S(m) = T(2^m)$

$S(m) = 2S(m/2) + m \Rightarrow S(m) = O(m\lg m)$ (demonstrated before)

$T(n) = T(2^m) = S(m) = O(m\lg m) = O(\lg n\lg\lg n)$

Idea: transform the recurrence to one that you have seen before

16

Recursion-tree method

Convert the recurrence into a tree:
- Each node represents the cost incurred at various levels of recursion
- Sum up the costs of all levels

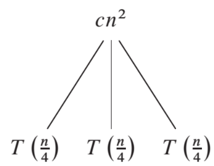
Example of recursion tree Method

$$T(n) = 3T(\lfloor n/4 \rfloor) + \Theta(n^2)$$

$T(n)$

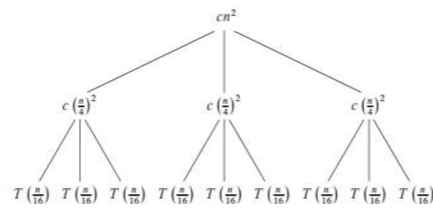
Example of recursion tree Method

$$T(n) = 3T(\lfloor n/4 \rfloor) + \Theta(n^2)$$

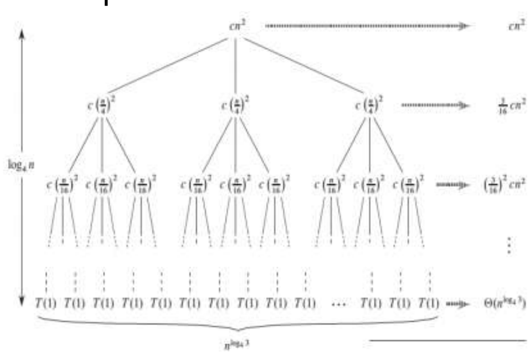


Example of recursion tree Method

$$T(n) = 3T(\lfloor n/4 \rfloor) + \Theta(n^2)$$



Example of recursion tree Method



Example of recursion tree Method

$T(n) = 3T(n/4) + cn^2$

- Subproblem size at level i is: $n/4^i$
- Subproblem size hits 1 when $1 = n/4^i \Rightarrow i = \log_4 n$
- Cost of a node at level $i = c(n/4^i)^2$
- Number of nodes at level $i = 3^i \Rightarrow$ last level has $3^{\log_4 n} = n^{\log_4 3}$ nodes
- Total cost:

$$T(n) = \sum_{i=0}^{\log_4 n-1}\left(\frac{3}{16}\right)^i cn^2 + \Theta\left(n^{\log_4 3}\right) \le \sum_{i=0}^{\infty}\left(\frac{3}{16}\right)^i cn^2 + \Theta\left(n^{\log_4 3}\right) = \frac{1}{1-\frac{3}{16}}cn^2 + \Theta\left(n^{\log_4 3}\right) = O(n^2)$$

$\Rightarrow$ **T(n) = O(n²)**

Recursion-Tree Method

- Gathering all the costs together:

$$T(n) = \sum_{i=0}^{\log_4 n-1} (3/16)^i cn^2 + \Theta(n^{\log_4 3})$$

$$T(n) \le \sum_{i=0}^{\infty} (3/16)^i cn^2 + o(n)$$

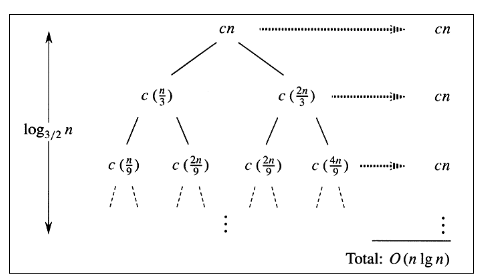
$$T(n) \le (1/(1-3/16))cn^2 + o(n)$$

$$T(n) \le (16/13)cn^2 + o(n)$$

$$T(n) = O(n^2)$$

Example 2

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



Example 2 cont..

- An overestimate of the total cost:

$$T(n) = \sum_{i=0}^{\log_{3/2} n - 1} cn + \Theta(n^{\log_{3/2} 2})$$

- Counter-indications:

$$T(n) = O(n \lg n) + \omega(n \lg n)$$

- Notwithstanding this, use as "guess":

$$T(n) = O(n \lg n)$$

Example 3

Solve $T(n) = T(n/4) + T(n/2) + n^2$:

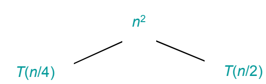
Example of recursion tree

Solve $T(n) = T(n/4) + T(n/2) + n^2$:

$T(n)$

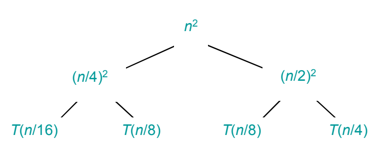
Example of recursion tree

Solve $T(n) = T(n/4) + T(n/2) + n^2$:



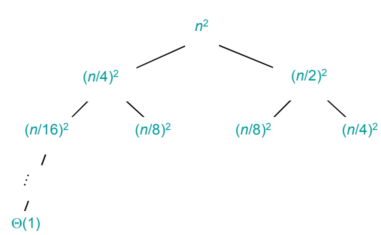
Example of recursion tree

Solve $T(n) = T(n/4) + T(n/2) + n^2$:



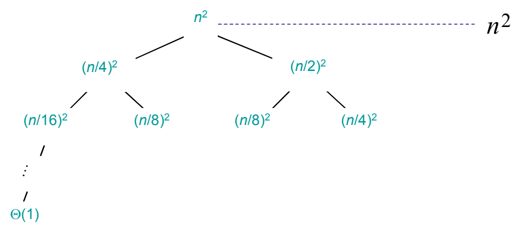
Example of recursion tree

Solve $T(n) = T(n/4) + T(n/2) + n^2$:



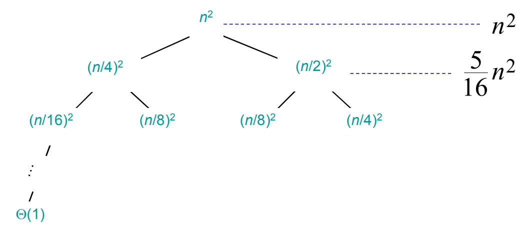
Example of recursion tree

Solve $T(n) = T(n/4) + T(n/2) + n^2$:



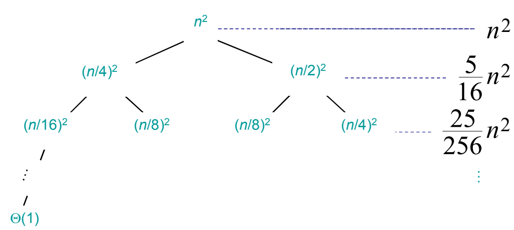
Example of recursion tree

Solve $T(n) = T(n/4) + T(n/2) + n^2$:



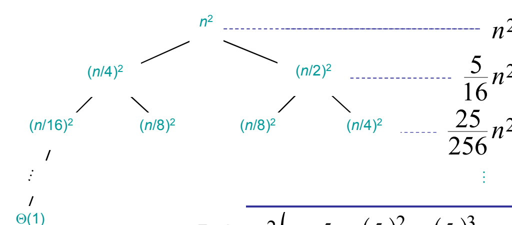
Example of recursion tree

Solve $T(n) = T(n/4) + T(n/2) + n^2$:



Example of recursion tree

Solve $T(n) = T(n/4) + T(n/2) + n^2$:



Total $= n^2\left(1+\frac{5}{16}+\left(\frac{5}{16}\right)^2+\left(\frac{5}{16}\right)^3+\cdots\right)$

$= \Theta(n^2)$ geometric series

Appendix: geometric series

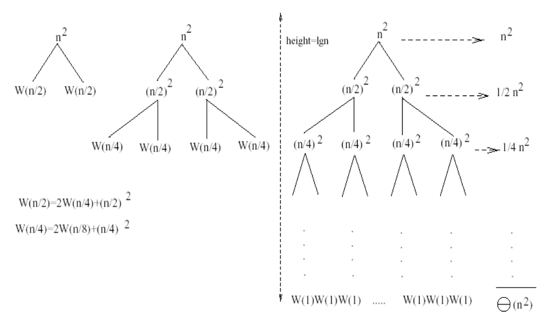
$$1+x+x^2+\cdots+x^n = \frac{1-x^{n+1}}{1-x} \quad \text{for } x \neq 1$$

$$1+x+x^2+\cdots = \frac{1}{1-x} \quad \text{for } |x| < 1$$

Return to last slide viewed.

Example 4

W(n) = 2W(n/2) + n^2



Example 4 cont..

W(n) = 2W(n/2) + n²

- Subproblem size at level i is: $n/2^i$
- Subproblem size hits 1 when $1 = n/2^i \Rightarrow i = \lg n$
- Cost of the problem at level $i = (n/2^i)^2$ No. of nodes at level $i = 2^i$
- Total cost:

$$W(n)=\sum_{i=0}^{\lg n-1}\frac{n^2}{2^i}+2^{\lg n}W(1)=n^2\sum_{i=0}^{\lg n-1}\left(\frac{1}{2}\right)^i+n\le n^2\sum_{i=0}^{\infty}\left(\frac{1}{2}\right)^i+O(n)=n^2\frac{1}{1-\frac{1}{2}}+O(n)=2n^2$$

$\Rightarrow W(n) = O(n^2)$

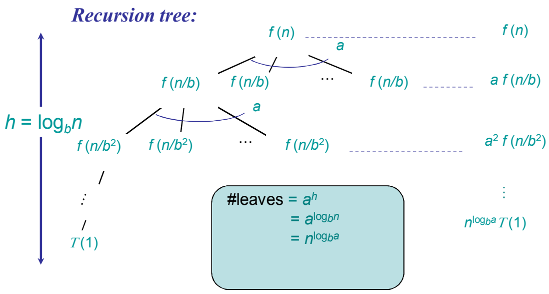
Master method

- ***"Cookbook"*** for solving recurrences of the form

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

where $a \ge 1$, $b > 1$, and $f(n)$ is asymptotically positive function.

Idea of master theorem



Master's method

$$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$

where, a ≥ 1, b > 1, and f(n) > 0

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

- f(n) is asymptotically smaller or larger than $n^{\log_b a}$ by a polynomial factor n^{ε}
- f(n) is asymptotically equal with $n^{\log_b a}$

40

Three common cases

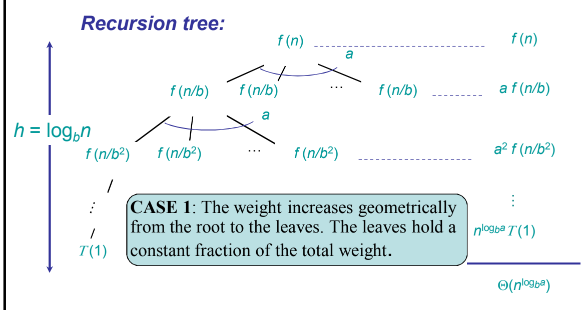
Compare $f(n)$ with $n^{\log_b a}$:

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^{ε} factor).

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

Idea of master theorem



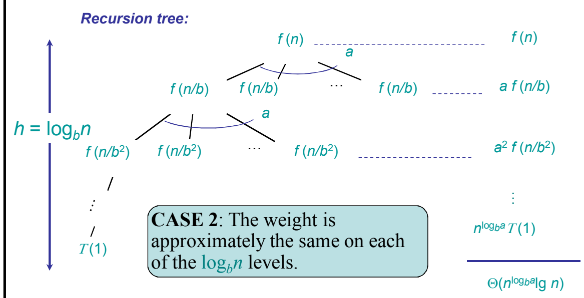
Three common cases

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

2. If $f(n) = \Theta(n^{\log_b a})$
   - $f(n)$ and $n^{\log_b a}$ grow at similar rates.

   Solution: $T(n) = \Theta(n^{\log_b a} \lg n)$.

Idea of master theorem



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^{ε} factor),

   and $f(n)$ satisfies the ***regularity condition*** that $a\, f(n/b) \le c\, f(n)$ for some constant $c < 1$.

   Solution: $T(n) = \Theta(f(n))$.

Idea of master theorem