RECURRENCES

Analysis of Algorithm

Punjab University College of Information Technology (PUCIT)
University of the Punjab, Lahore, Pakistan.

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

Example Recurrences

- $T(n) = T(n-1) + n$ $\Theta(n^2)$
  - Recursive algorithm that loops through the input to eliminate one item
- $T(n) = T(n/2) + c$ $\Theta(\lg n)$
  - Recursive algorithm that halves the input in one step
- $T(n) = T(n/2) + n$ $\Theta(n)$
  - Recursive algorithm that halves the input but must examine every item in the input
- $T(n) = 2T(n/2) + 1$ $\Theta(n)$
  - Recursive algorithm that splits the input into 2 halves and does a constant amount of other work

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

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

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

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

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

Assume $n = 2^k$

$T(n) = c + c + \ldots + c + T(1)$

$= c\lg n + T(1)$

$= \Theta(\lg n)$

Substitution method

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

Example: Binary Search

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

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

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

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

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

- Base case?

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: $T(n-1) \le c(n-1)^2$ 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: $T(n) \le cn \lg n$, for some c and $n \ge n_0$
  - Induction hypothesis: $T(n/2) \le cn/2 \lg(n/2)$
- Proof of induction goal:

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

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

$$\text{if: } -cn + n \le 0 \Rightarrow c \ge 1$$

- Base case?

Changing variables

- 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

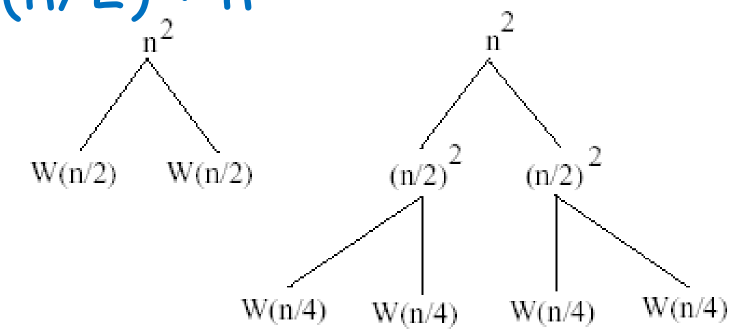
The recursion-tree method

Convert the recurrence into a tree:

- Each node represents the cost incurred at various levels of recursion
- Sum up the costs of all levels

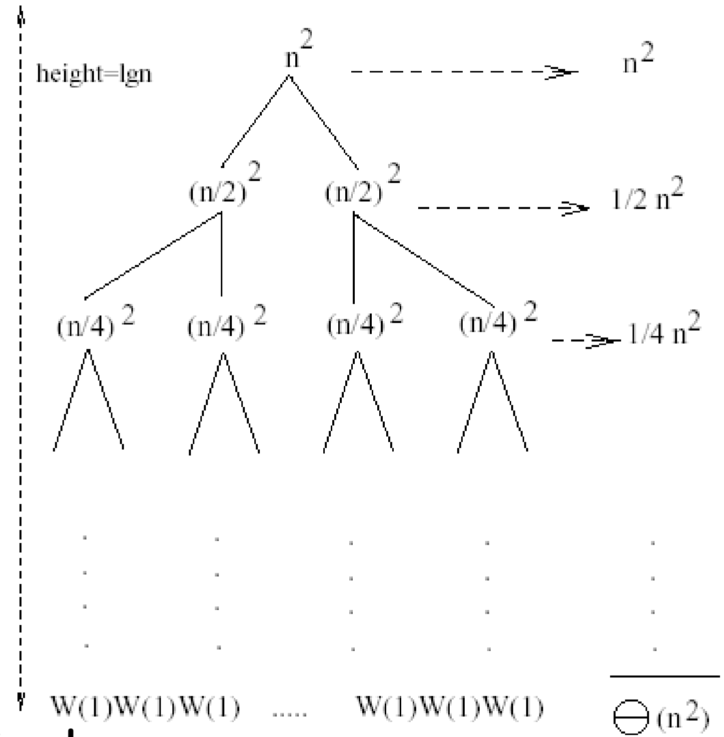
Example 1

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



W(n/2)=2W(n/4)+(n/2) 2

W(n/4)=2W(n/8)+(n/4) 2



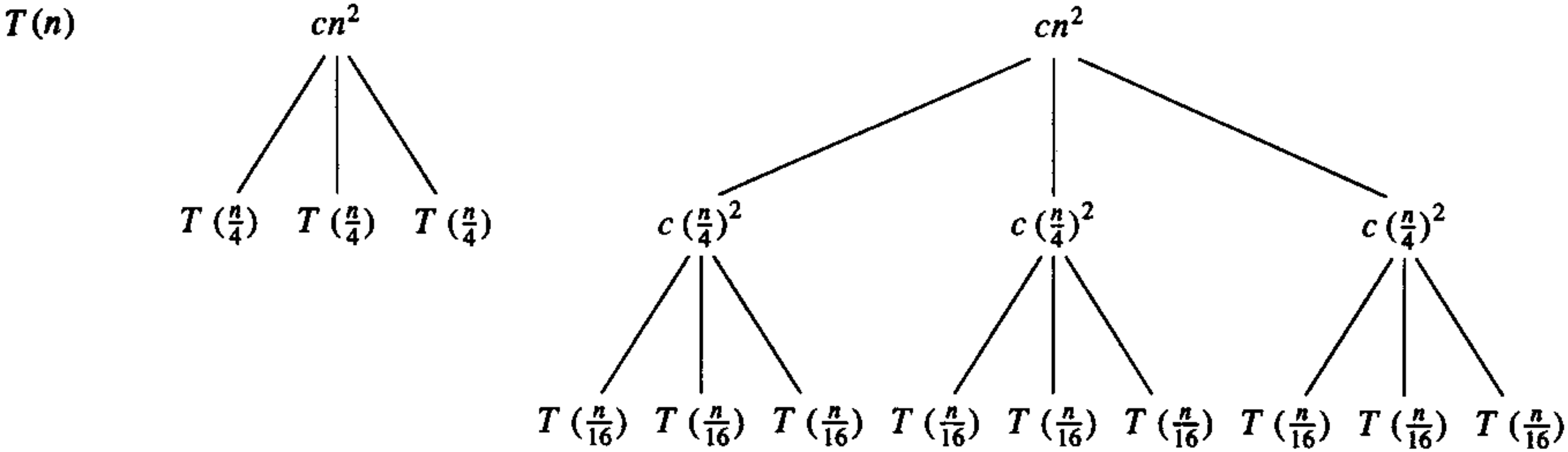
- Subproblem size at level i is: $n/2^i$
- Subproblem size hits 1 when $1 = n/2^i \Rightarrow i = \lg n$
- Cost of the problem at level i = $(n/2^i)^2$ No. of nodes at level $i = 2^i$
- Total cost:

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

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

Example 2

E.g.: $T(n) = 3T(n/4) + cn^2$



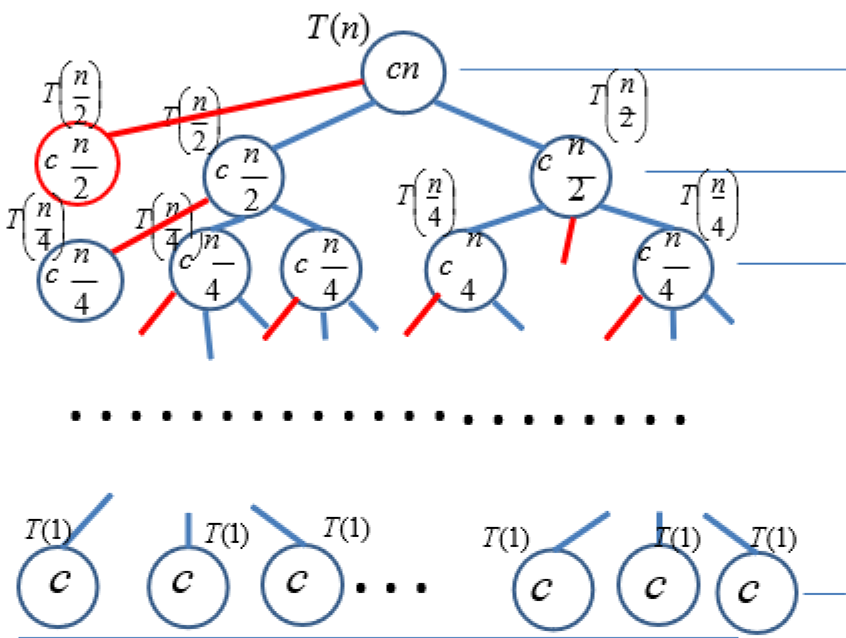
- Subproblem size at level i is: $n/4^i$
- Subproblem size hits 1 when $1 = n/4^i \Rightarrow i = \log_4 n$
- Cost of a node at level i = $c(n/4^i)^2$
- Number of nodes at level i = $3^i \Rightarrow$ last level has $3^{\log_4 n} = n^{\log_4 3}$ nodes
- Total cost:

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

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

Recursion Tree for T(n) = **3**T(n/2)+c

Base case: T(1) = c



Level	Arg/ pb size	TC of 1 node	Nodes per level	Level TC
0	n	c*n	1	c
1	n/2	c*n/2	**3**	3*c =**(3)***c
2	n/4	c*n/4	**9**	**(3)**^{2}*c
...				
i	$n/2^i$	$c*n/2^i$	**3^i**	**(3)**^{i}*c
...				
p=lgn	1 (=$n/2^p$)	c=c*1= $c*n/2^p$	**3^p** (≠n)	**(3)**^{p}*c

Stop at level p, when the subtree is T(1).
=> The problem size is 1, but the general formula for the problem size, at level p is:
$n/2^p$=> $n/2^p$=1 => 2^p=n => p = lgn

Total Tree TC for T(n) = **3**T(n/2)+cn

Closed form

$$T(n) = cn + (3/2)cn + (3/2)^2 cn + \ldots (3/2)^i cn + \ldots (3/2)^{\lg n} cn =$$

$$= cn * [1 + (3/2) + (3/2)^2 + \ldots + (3/2)^{\lg n}] = cn \sum_{i=0}^{\lg n} (3/2)^i =$$

$$= cn * \boxed{\frac{(3/2)^{\lg n + 1} - 1}{(3/2) - 1}} = 2cn[(3/2) * (3/2)^{\lg n} - 1] = 3cn * (3/2)^{\lg n} - 2cn$$

$$use: c^{\lg n} = n^{\lg c} => (3/2)^{\lg n} = n^{\lg(3/2)} = n^{\lg 3 - \lg 2} = n^{\lg 3 - 1} =>$$

$$= 3cn * n^{\lg 3 - 1} - 2cn = 3cn^{1 + \lg 3 - 1} - 2cn = 3cn^{\lg 3} - 2cn = \Theta(n^{\lg 3})$$

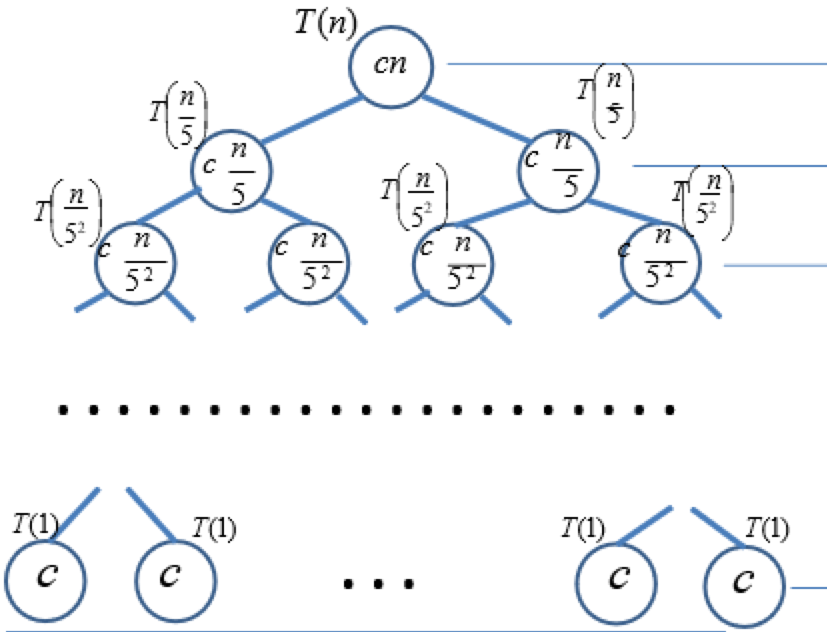
Explanation: since we need Θ, we can eliminate the constants and non-dominant terms earlier (after the closed form expression):

$$\ldots = cn * \frac{(3/2)^{\lg n + 1} - 1}{(3/2) - 1} = \Theta(n * (3/2) * (3/2)^{\lg n}) = \Theta(n * (3/2)^{\lg n})$$

$$use: c \lg n \quad = n^{\lg c} => (3/2)^{\lg n} = n^{\lg(3/2)} = n^{\lg 3 - \lg 2} = n^{\lg 3 - 1} =>$$

$$= \Theta(n * n^{\lg 3 - 1}) = \Theta(n^{\lg 3})$$

Recursion Tree for: T(n) = 2T(n/**5**)+cn



Level	Arg/ pb size	TC of 1 node	Nodes per level	Level TC
0	n	c*n	1	c*n
1	n/**5**	c*n/**5**	2	2*c*n/**5** =**(2/5)***cn
2	n/**5^2**	c*n/**5^2**	4	4*c*n/ =**$(2/5)^i$**cn
...				
i	n/**5^i**	c*n/**5^i**	2^i	2^i*c*n/5^i =**$(2/5)^i$**cn
...				
p= $\log_5 n$	1 (=n/**5^p**)	c=c*1= c*n/**5^p**	2^p (=n)	2^p*c*n/**5^p** =**$(2/5)^p$**cn

Stop at level p, when the subtree is T(1).
=> The problem size is 1, but the general formula for the problem size, at level p is:
n/**5^p**=> n/**5^p**= 1 => **5^p**=n => p = **$\log_5$**n

Tree TC
(derivation similar to TC for T(n) = 3T(n/2)+cn)

Total Tree TC for T(n) = 2T(n/5)+cn

$$T(n) = cn + (2/5)cn + (2/5)^2 cn + \ldots (2/5)^i cn + \ldots (2/5)^{\log_5 n} cn =$$

$$= cn * [1 + (2/5) + (2/5)^2 + \ldots + (2/5)^{\log_5 n}] =$$

$$= cn \sum_{i=0}^{\log_5 n} (2/5)^i \leq cn \sum_{i=0}^{\infty} (2/5)^i =$$

$$= cn * \frac{1}{1-(2/5)} = (5/3)cn = O(n)$$

Also

$$T(n) = cn + \ldots \Rightarrow T(n) \geq cn \Rightarrow T(n) = \Omega(n)$$

$$\Rightarrow T(n) = \Theta(n)$$

Master's method

- Solving recurrences of the form:

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

where, $a \geq 1$, $b > 1$, 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 $n^{\log_b a}$

Master's method

- Solving recurrences of the form:

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

where, $a \geq 1$, $b > 1$, and $f(n) > 0$

Case 1: if $f(n) = O(n^{\log_b a - \varepsilon})$ for some $\varepsilon > 0$, then: $T(n) = \Theta(n^{\log_b a})$

Case 2: if $f(n) = \Theta(n^{\log_b a})$, then: $T(n) = \Theta(n^{\log_b a} \lg n)$

Case 3: if $f(n) = \Omega(n^{\log_b a + \varepsilon})$ for some $\varepsilon > 0$, and if

$af(n/b) \leq cf(n)$ for some $c < 1$ and all sufficiently large n, then:

$$T(n) = \Theta(f(n))$$

regularity condition

Examples

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

$a = 2$, $b = 2$, $\log_2 2 = 1$

Compare $n^{\log_2 2}$ with $f(n) = n$

$\Rightarrow f(n) = \Theta(n) \Rightarrow$ Case 2

$\Rightarrow T(n) = \Theta(n\lg n)$

Examples

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

$a = 2$, $b = 2$, $\log_2 2 = 1$

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

$\Rightarrow f(n) = \Omega(n^{1+\varepsilon})$ Case 3 $\Rightarrow$ verify regularity cond.

$a\, f(n/b) \le c\, f(n)$

$\Leftrightarrow 2\, n^2/4 \le c\, n^2 \Rightarrow c = \frac{1}{2}$ is a solution $(c<1)$

$\Rightarrow T(n) = \Theta(n^2)$

Examples (cont.)

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

$a = 2$, $b = 2$, $\log_2 2 = 1$

Compare n with $f(n) = n^{1/2}$

$\Rightarrow f(n) = O(n^{1-\varepsilon})$ Case 1

$\Rightarrow T(n) = \Theta(n)$