CSCE 3110
Data Structures and Algorithms

Recurrence Relations

Credits of some slides go to G. Bebis at UNR.

Recurrences and Running Time

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

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

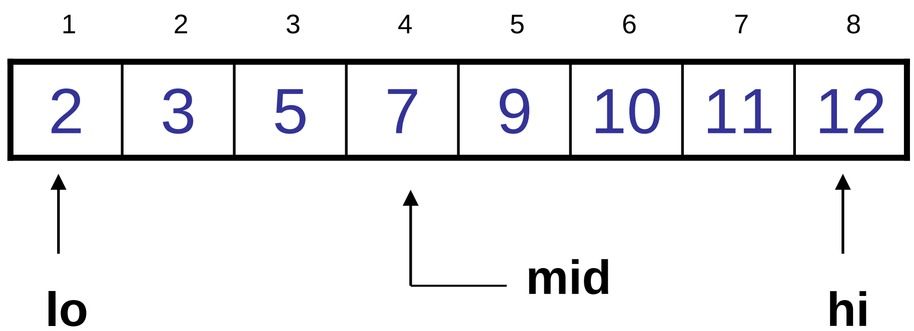
- Recurrences arise when an algorithm contains recursive calls to itself
- What is the actual running time of the algorithm?
- Need to solve the recurrence
  - Find an explicit formula of the expression
  - Bound the recurrence by an expression that involves n

Example: Binary Search

- for an ordered array A, finds if x is in the array A[lo…hi]

Alg.: BINARY-SEARCH (A, lo, hi, x)

if (lo > hi)
    return FALSE
mid ← $\lfloor (lo+hi)/2 \rfloor$
if x = A[mid]
    return TRUE
if (x < A[mid])
    BINARY-SEARCH (A, lo, mid-1, x)
if (x > A[mid])
    BINARY-SEARCH (A, mid+1, hi, x)



Example: Binary Search

- A[8] = {1, 2, 3, 4, 5, 7, 9, 11}
  - lo = 1 hi = 8 x = 7

1	2	3	4	5	6	7	8
1	2	3	4	5	7	9	11

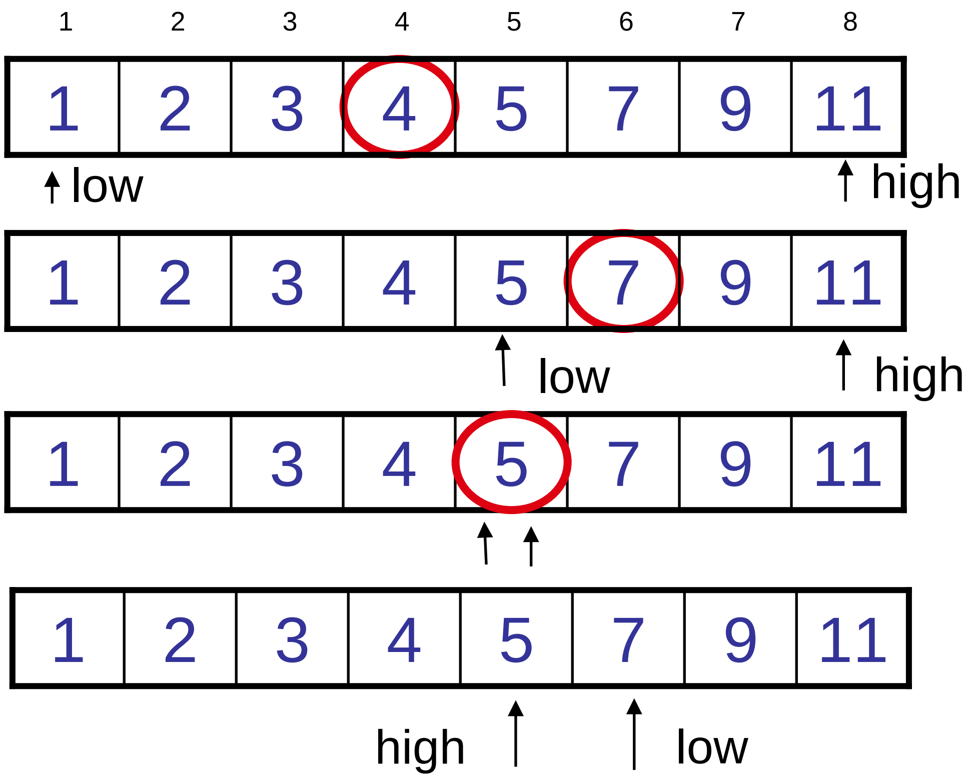
mid = 4, lo = 5, hi = 8

				5	6	7	8
1	2	3	4	5	7	9	11

mid = 6, A[mid] = x
Found!

Example: Binary Search

- A[8] = {1, 2, 3, 4, 5, 7, 9, 11}
  - lo = 1 hi = 8 x = 6



mid = 4, lo = 5, hi = 8

mid = 6, A[6] = 7, lo = 5, hi = 5

mid = 5, A[5] = 5, lo = 6, hi = 5
NOT FOUND!

Example: Binary Search

Alg.: BINARY-SEARCH (A, lo, hi, x)

if (lo > hi) ← constant time: c_1

  return FALSE

mid ← $\lfloor(lo+hi)/2\rfloor$ ← constant time: c_2

if x = A[mid] ← constant time: c_3

  return **TRUE**

if (x < A[mid])

  BINARY-SEARCH (A, lo, mid-1, x) ← same problem of size n/2

if (x > A[mid])

  BINARY-SEARCH (A, mid+1, hi, x) ← same problem of size n/2

- $T(n) = d + T(n/2)$
  - T(n) – running time for an array of size n

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 - Example

$$\mathbf{T(n) = c' + T(n/2)}$$

$$T(n) = d + T(n/2)$$
$$= d + d + T(n/4)$$
$$= d + d + d + T(n/8)$$

$$T(n/2) = d + T(n/4)$$
$$T(n/4) = d + T(n/8)$$

Assume $n = 2^k$

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

$$= d\lg n + T(1)$$
$$= \Theta(\lg n)$$

The Iteration Method - Example

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

Assume: $n = 2^k$

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

$$\begin{aligned}
T(n) &= n + 2T(n/2) \\
&= n + 2(n/2 + 2T(n/4)) \\
&= n + n + 4T(n/4) \\
&= n + n + 4(n/4 + 2T(n/8)) \\
&= n + n + n + 8T(n/8) \\
\ldots\ &= in + 2^i T(n/2^i) \\
&= kn + 2^k T(1) \\
&= n\lg n + nT(1) = \Theta(n\lg n)
\end{aligned}$$

The 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
    - $T(n) \leq c\, g(n)$, for some $c > 0$ and $n \geq n_0$
  - Induction hypothesis: $T(k) \leq c\, g(k)$ 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

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 c and $n \ge n_0$
  - Induction hypothesis: $T(n/2) \le c \lg(n/2)$
- Proof of induction goal:

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

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

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

- Base case?

Example

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

- Guess: $T(n) = O(n\lg n)$
  - Induction goal: $T(n) \leq cn \lg n$, for some c and $n \geq n_0$
  - Induction hypothesis: $T(n/2) \leq cn/2 \lg(n/2)$
- Proof of induction goal:

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

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

$$\text{if: } -cn + n \leq 0 \Rightarrow c \geq 1$$

- Base case?

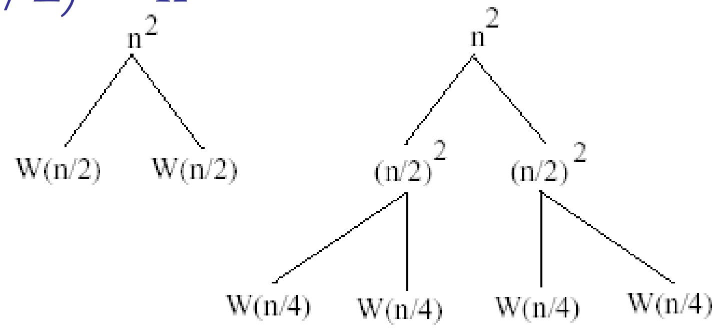
The Recursion-Tree Method

Convert the recurrence into a tree:

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

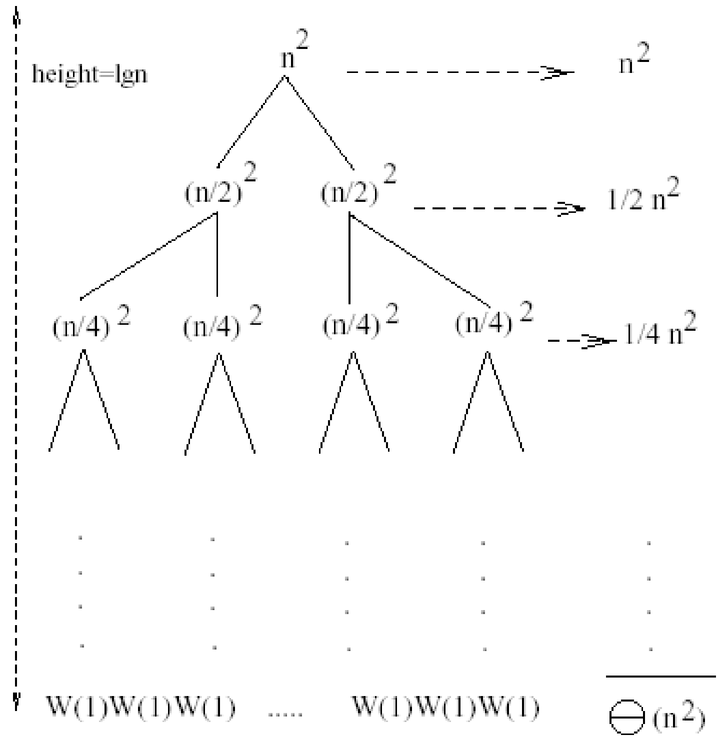
Example 1

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



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

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



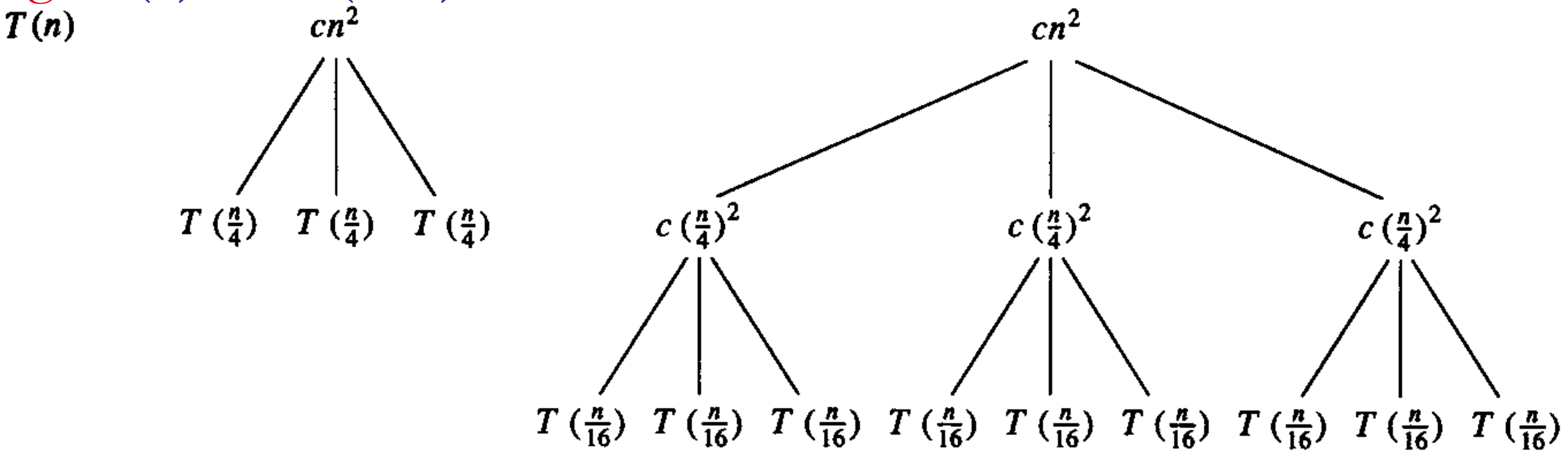
- Subproblem size at level i is: $n/2^i$
- Subproblem size hits 1 when $1 = n/2^i \Rightarrow i = \lg n$
- Cost of the problem at level i $= (n/2^i)^2$ No. of nodes at level i $= 2^i$
- Total cost:

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

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

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:

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

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

Master's Method

- "Cookbook" for 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

- "Cookbook" for 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:

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

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.

$a\ f(n/b) \leq c\ f(n)$

$\Leftrightarrow 2\ n^2/4 \leq c\ n^2 \Rightarrow c = ½$ is a solution $(c<1)$

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

Examples

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

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

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

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

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

Examples

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

$a = 3$, $b = 4$, $\log_4 3 = 0.793$

Compare $n^{0.793}$ with $f(n) = n\lg n$

$f(n) = \Omega(n^{\log_4 3+\varepsilon})$ Case 3

Check regularity condition:

$$3*(n/4)\lg(n/4) \le (3/4)n\lg n = c*f(n),\ c=3/4$$

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

Examples

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

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

- Compare n with $f(n) = n\lg n$
  - seems like case 3 should apply
- $f(n)$ must be polynomially larger by a factor of n^{ε}
- In this case it is only larger by a factor of $\lg n$

$\Rightarrow$ Master's method does NOT apply!

Next Class

Abstract Data Types, Elementary Data Structures

Reading: Weiss, chap. 3