

CSC 611: Analysis of Algorithms

Lecture 3

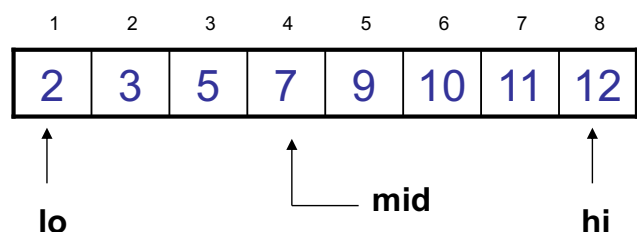
Recurrence Relations

Recurrent Algorithms: BINARY – SEARCH

- for an ordered array A , finds if x is in the array $A[\text{lo} \dots \text{hi}]$

Alg.: BINARY-SEARCH ($A, \text{lo}, \text{hi}, x$)

```
if (lo > hi)
    return FALSE
mid ← ⌊(lo+hi)/2⌋
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

- $A[8] = \{1, 2, 3, 4, 5, 7, 9, 11\}$

– $lo = 1$ $hi = 8$ $x = 7$

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

$mid = 4, lo = 5, hi = 8$

1	2	3	4	5	7	9	11
---	---	---	---	---	---	---	----

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

Example

- $A[8] = \{1, 2, 3, 4, 5, 7, 9, 11\}$

– $lo = 1$ $hi = 8$ $x = 6$

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

$mid = 4, lo = 5, hi = 8$

1	2	3	4	5	7	9	11
---	---	---	---	---	---	---	----

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

1	2	3	4	5	7	9	11
---	---	---	---	---	---	---	----

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

Analysis of BINARY-SEARCH

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

if (lo > hi)

← constant time: c_1

return FALSE

mid $\leftarrow \lfloor (lo+hi)/2 \rfloor$

← 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 $n/2$

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

- $T(n)$ – running time for an array of size n

CSC 611 - Lecture 3

5

Recurrences and Running Time

*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 (the generic term of the sequence)

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

CSC 611 - Lecture 3

7

Methods for Solving Recurrences

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

Iteration Method

Calls for expanding a recurrence as a summation of terms dependent on n and the initial conditions

The Iteration Method

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

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

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

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

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

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

Assume $n = 2^k$

$$T(n) = \underbrace{c + c + \dots + c}_{k \text{ times}} + T(1)$$

k times

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

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

Iteration Method – Example 1

$$T(n) = n + 2T(n/2) \quad \text{Assume: } n = 2^k$$

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

Iteration Method – Example 1

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

$$\begin{aligned} T(n) &= n + T(n-1) \\ &= n + (n-1) + T(n-2) \\ &= n + (n-1) + (n-2) + T(n-3) \\ \dots &= n + (n-1) + (n-2) + \dots + 2 + T(1) \\ &= n(n+1)/2 - 1 + T(1) \\ &= n^2 + T(1) = \Theta(n^2) \end{aligned}$$

Iteration Method – Example 2

- $T(n) = 3T\left(\left\lfloor \frac{n}{4} \right\rfloor\right) + n$ Can be iterated as follows:

$$T(n) = n + 3T\left(\left\lfloor \frac{n}{4} \right\rfloor\right)$$

- i^{th} term is $3^i \left\lfloor \frac{n}{4^i} \right\rfloor$

$$= n + 3\left(\left\lfloor \frac{n}{4} \right\rfloor\right) + 3T\left(\left\lfloor \frac{n}{16} \right\rfloor\right)$$

$$= n + 3T\left(\left\lfloor \frac{n}{4} \right\rfloor\right) + 3\left(\left\lfloor \frac{n}{16} \right\rfloor\right) + 3T\left(\left\lfloor \frac{n}{64} \right\rfloor\right)$$

$$= n + 3T\left(\left\lfloor \frac{n}{4} \right\rfloor\right) + 9\left\lfloor \frac{n}{16} \right\rfloor + 27T\left(\frac{n}{64}\right)$$

Iteration stops when

$$\left\lfloor \frac{n}{4^i} \right\rfloor = 1 \Rightarrow i \geq \log_4 n$$

Iteration Method – Example 2

- Thus,
 - $T(n) \leq n + 3n/4 + 9n/16 + 27n/64 + \dots + 3^{\log_4 n} \Theta(1)$

$$\leq n \sum_{i=0}^{\infty} \frac{3}{4} + \Theta(n \log_4 3)$$

$$= 4n + o(n)$$

$$= O(n)$$

$$a^{\log_b n} = n^{\log_b a}$$

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**
 - $T(n) \leq d g(n)$, 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: $T(n) \leq d \lg n$, for some d and $n \geq n_0$
 - Induction hypothesis: $T(n/2) \leq d \lg(n/2)$
- Proof of induction goal:

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

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

$$\text{if: } -d + c \leq 0, d \geq c$$

Example: Binary Search

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

- Boundary conditions:
 - Base case: $n_0 = 1 \Rightarrow T(1) = c$ has to verify condition:
 $T(1) \leq d \lg 1 \Rightarrow c \leq d \lg 1 = 0$ – contradiction
 - Choose $n_0 = 2 \Rightarrow T(2) = 2c$ has to verify condition:
 $T(2) \leq d \lg 2 \Rightarrow 2c \leq d \lg 2 = d \Rightarrow \text{choose } d \geq 2c$
- We can similarly prove that $T(n) = \Omega(\lg n)$ and therefore: $T(n) = \Theta(\lg n)$

Example 2

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

- Guess: $T(n) = O(n^2)$
 - Induction goal: $T(n) \leq c n^2$, for some c and $n \geq n_0$
 - Induction hypothesis: $T(n-1) \leq c(n-1)^2$
- Proof of induction goal:
$$T(n) = T(n-1) + n \leq c(n-1)^2 + n$$
$$= cn^2 - (2cn - c - n) \leq cn^2$$

if: $2cn - c - n \geq 0 \Rightarrow c \geq n/(2n-1) \Rightarrow c \geq 1/(2 - 1/n)$

 - For $n \geq 1 \Rightarrow 2 - 1/n \geq 1 \Rightarrow$ any $c \geq 1$ will work

Example 2

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

- Boundary conditions:
 - Base case: $n_0 = 1 \Rightarrow T(1) = 1$ has to verify condition:
$$T(1) \leq c(1)^2 \Rightarrow 1 \leq c \Rightarrow \text{OK!}$$
- We can similarly prove that $T(n) = \Omega(n^2)$ and therefore: $T(n) = \Theta(n^2)$

Example 3

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

- 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:

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

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

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

Example 3

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

- Boundary conditions:
 - Base case: $n_0 = 1 \Rightarrow T(1) = 1$ has to verify condition:
 $T(1) \leq cn_0 \lg n_0 \Rightarrow 1 \leq c * 1 * \lg 1 = 0$ – contradiction
 - Choose $n_0 = 2 \Rightarrow T(2) = 4$ has to verify condition:
 $T(2) \leq c * 2 * \lg 2 \Rightarrow 4 \leq 2c \Rightarrow$ choose $c = 2$
- We can similarly prove that $T(n) = \Omega(n \lg n)$ and therefore: $T(n) = \Omega(n \lg n)$

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)

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

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

Changing variables (cont.)

$$T(n) = T(n/2) + 1$$

– Rename: $n = 2^m$ ($n/2 = 2^{m-1}$)

$$T(2^m) = T(2^{m-1}) + 1$$

– Rename: $S(m) = T(2^m)$

$$S(m) = S(m-1) + 1$$

$$= 1 + S(m-1) = 1 + 1 + S(m-2) = \underbrace{1 + 1 + \dots + 1}_{m-1 \text{ times}} + S(1)$$

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

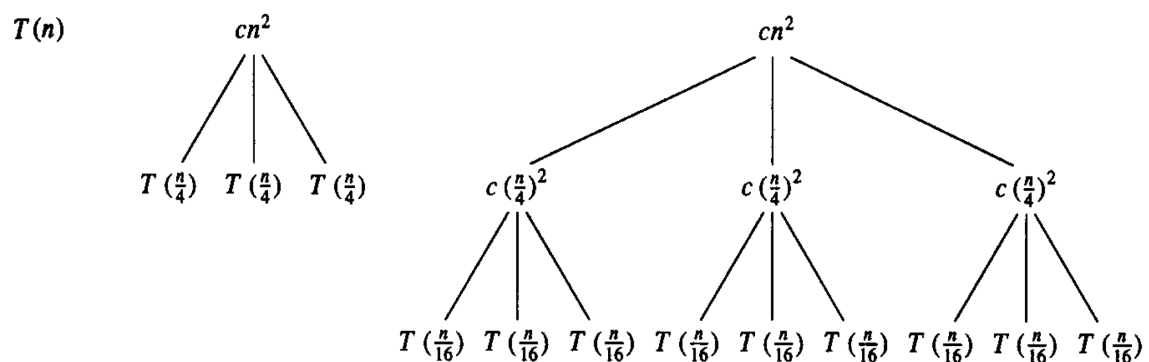
The recursion-tree method

Convert the recurrence into a tree:

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

Used to “guess” a solution for the recurrence

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



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

- 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:

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

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

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

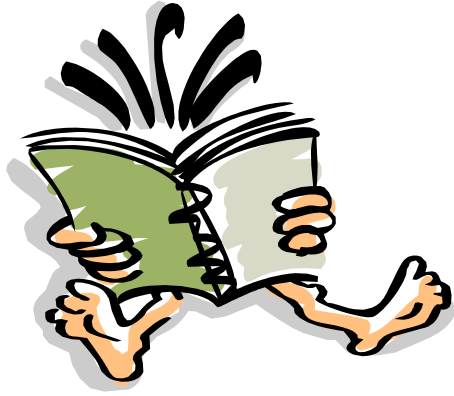
$W(n/2) = 2W(n/4) + (n/2)^2$
 $W(n/4) = 2W(n/8) + (n/4)^2$

- 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 $i = n^2/2^i$ No. of nodes at level $i = 2^i$
- h = Height of the tree $\Rightarrow n/2^h = 1 \Rightarrow h = \lg n$
- Total cost:

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

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

Readings



- Chapter 4