UNIT II DIVIDE-AND-CONQUER

Divide and Conquer Methodology – Binary Search – Merge Sort – Quick Sort – Heap Sort – Multiplication of Large Integers – Strassen's Matrix Multiplication

Binary Search

Very efficient algorithm for searching in sorted array:

K vs A[0] . . . A[m] . . . A[n-1]

If K = A[m], stop (successful search); otherwise, continue searching by the same method in A[0..m-1] if K < A[m] and in A[m+1..n-1] if K > A[m]

$l \leftarrow 0$; $r \leftarrow n-1$
while $l \leq r$ do
    $m \leftarrow \lfloor (l+r)/2 \rfloor$
    if K = A[m] return m
    else if K < A[m] $r \leftarrow m-1$
    else $l \leftarrow m+1$
return -1

Analysis of Binary Search

- Time efficiency
  - worst-case recurrence: $C_w(n) = 1 + C_w(\lfloor n/2 \rfloor)$, $C_w(1) = 1$
    solution: $C_w(n) = \lceil \log_2(n+1) \rceil$
    This is VERY fast: e.g., $C_w(10^6) = 20$

- Optimal for searching a sorted array
  Limitations: must be a sorted array (not linked list)

- Bad (degenerate) example of divide-and-conquer
  because only one of the sub-instances is solved

- Has a continuous counterpart called *bisection method* for solving equations in one unknown $f(x) = 0$

Binary Tree Algorithms

Binary tree is a divide-and-conquer ready structure!

Ex. 1: Classic traversals (preorder, inorder, postorder)

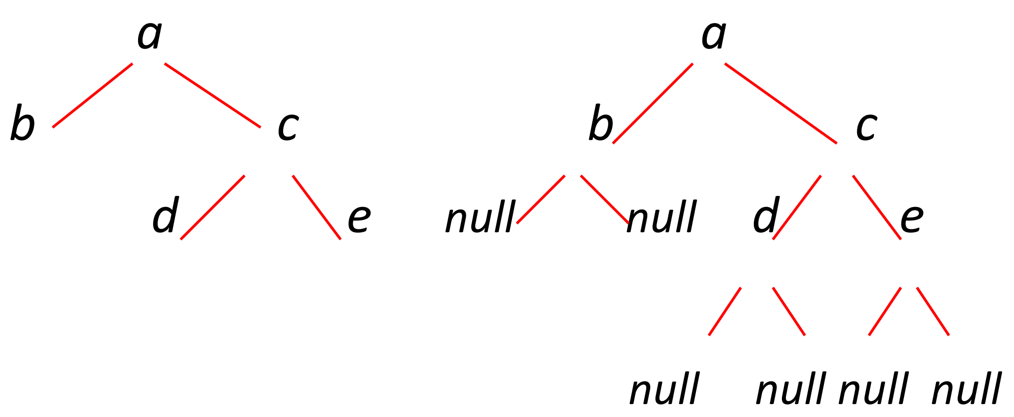
Algorithm *Inorder*(T)

if $T \neq \varnothing$

$\quad Inorder(T_{left})$

$\quad$ print(root of T)

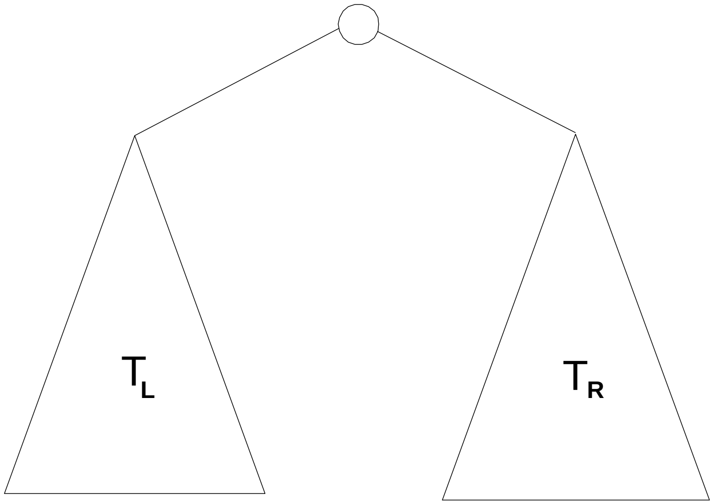
$\quad Inorder(T_{right})$



Efficiency: $\Theta(n)$. Why? Each node is visited/printed once.

Binary Tree Algorithms (cont.)

Ex. 2: Computing the height of a binary tree



$h(T) = \max\{h(T_L), h(T_R)\} + 1$ **if** $T \neq \varnothing$ **and** $h(\varnothing) = -1$

Efficiency: $\Theta(n)$. **Why?**

Thank You