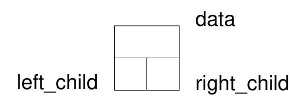
CSE 326
Lecture 7

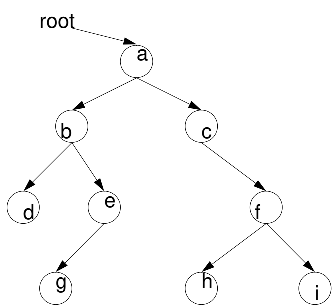
Spring Quarter 1997

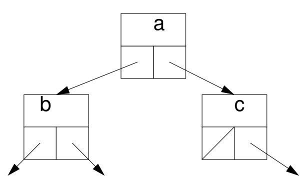
Binary Trees

- Each node has at most two children



Example of a Binary Tree





Inorder Traversal

- Inorder - Recursively visit the left subtree, then visit the node, then recursively visit the right subtree.

  inorder(n) {

  if not(n = null) then {
      inorder(left_child(n));
      Do_something;
      inorder(right_child(n)) }}

Binary Search Tree

- A binary search tree has a number at each node and the tree has the following property.

  For every node n, all the numbers in the left subtree of n are less than or equal to the number at n, and all the numbers in the right subtree of n are greater than or equal to the number at n.

  Concisely:

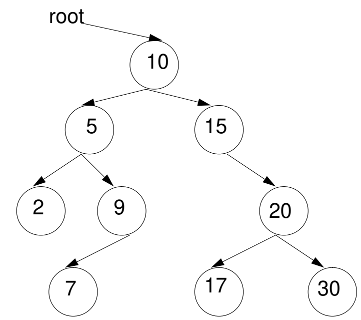
  left subtree of node < node < right subtree of node

- Why have binary search trees?

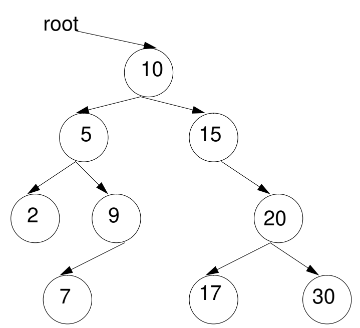
  1. Dynamic like a list - can handle inserts and deletes.

  2. Potentially efficient like a sorted array - can do fast search, O(log n) time.

Example Binary Search Tree



Example of Listing in Order



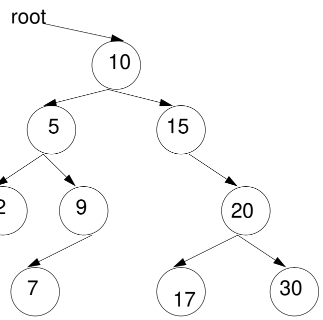
Listing the Data in Order

- List(n)

```
if not(n=null) then {
   List(left_child(n));
   print(n.data);
   List(right_child(n)) }}
```

- Call List(root) to list all the data in order.

Search Example



Search

- Find(k,n) returns a pointer to the node with data k in the subtree rooted at n or null if there is no such node.
- Find(k,n) {

```
case:
   n = null: return(null);
   n.data = k : return(n);
   n.data > k : return(Find(k,n.left_child));
   n.data < k : return(Find(k,n.right_child))
endcase}
```

- Call Find(k,root) to search the entire tree.

Iterative Searching

- Find(k) {

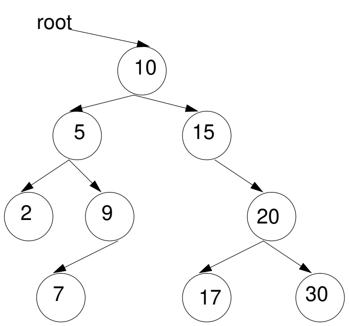
```
local node pointer n, boolean not_found;

n := root;
found := false;
while not(found) do {
   case:
      n = null: found := true;
      n.data = k : found := true;
      n.data > k : n := n.left_child;
      n.data < k : n := n.right_child;
   endcase}
return(n)}
```

- End recursion principle:

  An end recursion is a recursive call after which nothing in done by the caller. Such recursions can be eliminated by making forming a loop and making the proper assignments to the parameters.

Insert Example

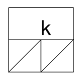


Insert

- Insert(k,n) inserts a new node in the subtree rooted at n with data equal to k. If there is already such a node nothing is done.
- Insert(k,n) {

```
case
   n = null: root := New_node(k);
   n.data > k : if n.left_child = null then n.left_child := New_node(k)
                 else Insert(k,n.left_child));
   n.data < k : if n.right_child = null then n.right_child := New_node(k)
                 else Insert(k,n.right_child));
endcase}
```

- New_node(k) returns a pointer to



Building a Binary Search Tree From Scratch

- Suppose the data $a_1, a_2, a_3, \ldots, a_n$ is inserted into an initially empty binary search tree. What can happen.

1. $a_1, a_2, a_3, \ldots, a_n$ are in increasing order.
2. $a_1, a_2, a_3, \ldots, a_n$ are in decreasing order
3. a_1 is the median of all, a_2 is the median of elements less than a_1, a_3 is the median of elements greater than a_1, etc.
4. $a_1, a_2, a_3, \ldots, a_n$ are randomly ordered.

Examples of Building from Scratch

- 1, 2, 3, 4, 5, 6, 7, 8, 9
- 5, 3, 7, 2, 4, 6, 8, 1, 9

Analysis of Building a Binary Search Tree

- Worst case is $O(n^2)$.

$$1 + 2 + 3 + \ldots + n = O(n^2)$$

- Average case is O(n log n) assuming all orderings of insertions are equally likely.