Binary Search Trees

A binary search tree labels each node in a binary tree with a single key such that for any node $x$, and nodes in the left subtree of $x$ have keys $\leq x$ and all nodes in the right subtree of $x$ have key’s $\geq x$.

Left: A binary search tree. Right: A heap but not a binary search tree.

The search tree labeling enables us to find where any key is. Start at the root - if that is not the one we want, search either left or right depending upon whether what we want is $\leq$ or $\geq$ then the root.
Searching in a Binary Tree

Dictionary search operations are easy in binary trees ...

TREE-SEARCH(x, k)
    if \( x = NIL \) and \( k = key[x] \)
        then return \( x \)
    if \( k < key[x] \)
        then return TREE-SEARCH(left[x], k)
    else return TREE-SEARCH(right[x], k)

The algorithm works because both the left and right subtrees of a binary search tree are binary search trees – recursive structure, recursive algorithm.

This takes time proportional to the height of the tree, \( O(h) \).
Maximum and Minimum

Where are the maximum and minimum elements in a binary tree?

TREE-MAXIMUM(X)
   while right[x] ≠ NIL
      do x = right[x]
   return x

TREE-MINIMUM(x)
   while left[x] ≠ NIL
      do x = left[x]
   return x

Both take time proportional to the height of the tree, $O(h)$. 
Where is the predecessor?

Where is the predecessor of a node in a tree, assuming all keys are distinct?

If $X$ has two children, its predecessor is the maximum value in its left subtree and its successor the minimum value in its right subtree.
What if a node doesn’t have children?

If it does not have a left child, a node’s predecessor is its first left ancestor.

The proof of correctness comes from looking at the in-order traversal of the tree.

Tree-Successor($x$)

if $right[x] \neq NIL$
    then return Tree-Minimum($right[x]$)
    $y \leftarrow p[x]$
    while ($y \neq NIL$) and ($x = right[y]$)
        do $x \leftarrow y$
        $y \leftarrow p[y]$
    return $y$

Tree predecessor/successor both run in time proportional to the height of the tree.
In-Order Traversal

A

F

B

G

D

C

E

H

Inorder-Tree-walk(\(x\))
if \((x \not< > NIL)\)
then Inorder-Tree-Walk(\(left[x]\))
print \(key[x]\)
Inorder-Tree-walk(\(right[x]\))

A-B-C-D-E-F-G-H