SORTING III: BINARY SEARCH TREES AND BST SORT

6.006
02/18/2016
L05.1

Tonight: PS 1 Due, PS 2 Released

Reading CLRS 4.3, 10.4, 12.1-3

Bruce Tidor ⟨tidor@mit.edu⟩ 32-212
Office Hours: by email appt (usually avail TH10)

Last Time: Priority Queue, Heap, Heapsort — $\Theta(n \log n)$

Priority Queue
- abstract data structure
- useful for task scheduling
  - insert (S, x)
  - max (S)
  - extract_max (S)
  - increase_key (S, x, k)

Heap
- data structure
- implements PQ
- array visualized as binary tree
- satisfies max (or min) heap property "key of node $\geq$ keys of its children"

Heapsort

Build max heap from unordered array

→ Repeatedly extract_max to form output (in reverse order)

TODAY:

Imagine wish to keep ordered list of times that airplanes en route will land at airport

New data structure → New sorting algorithm

Binary Search Tree (BST) → BST Sort

BST data structure

Each node x has

key[x], left[x], right[x], p[x], satellite data

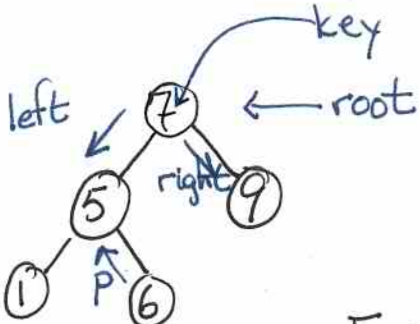
key

pointers (not an array with array operations)

parent

optional

also called "auxillary" data



BST Property

For any node x

- all nodes y in left subtree: key[y] ≤ key[x]
- all nodes z in right subtree: key[z] ≥ key[x]

Operations Supported

insert(T,x) insert node x with key key[x]

find-min(x) return node with minimum value of key
(find-max is analogous)

delete(T,x) delete the node x

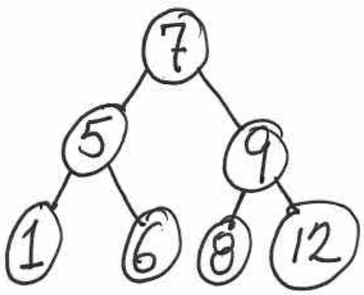
find(x,k) return the node with key k, if it exists

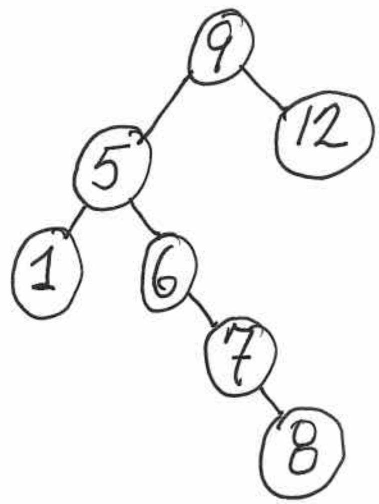
successor(x) return next node (with next highest key) after node x
(predecessor(x) is analogous)

We will talk more about these operations today and in recitation.

Same BST permits max and min, as well as successor and predecessor.

Is BST unique for a given collection of keys?





No!!
Not unique

The Binary Seach Tree is the basis for a large number of more specialized trees: (a, b) tree, 2-3 tree, 2-3-4 tree, AA tree, AVL tree, B tree, B+ tree, B* tree, Cartesian tree, Dancing tree, leftist tree, Red-black tree, Scapegoat tree, Splay tree, T tree, Tango tree, Top tree, UB tree, ...

Sorting Based on BST

Consider:
```
Inorder-Tree-Walk (x)
    if x ≠ Null
        Inorder-Tree-Walk (left [x])
        Output key [x]
        Inorder-Tree-Walk (right [x])
```

Examine example trees at top of page and see that the procedure outputs sorted keys

Inorder-Tree-Walk on BST outputs sorted keys.

Correctness: Follows by induction directly from BST property

Running Time: $\Theta(n)$ - need to "walk" entire tree and visit every node.

$\Omega(n)$ because must visit each of n nodes.

Will prove $O(n)$

$T(0) = c$, some small constant time for empty subtree (for $x \neq$ Null test)

$c > 0$

$T(n > 0)$:

$$T(n) \leq \underbrace{T(k)}_{\text{left subtree}} + \underbrace{T(n-k-1)}_{\text{right subtree}} + \underbrace{d}_{\substack{\text{upper bound on} \\ \text{time not in} \\ \text{recursive calls}}}$$

constant

induction

Solve recurrence by substitution method (CLRS §4.3)

Guess solution: $T(n) \leq (c+d)n + c$

Need to show by induction that this is correct

Base Case: $n = 0$ $\quad T(0) \leq (c+d) \cdot 0 + c = c$ ✓

Assume true for $m < n$ and show true for n:

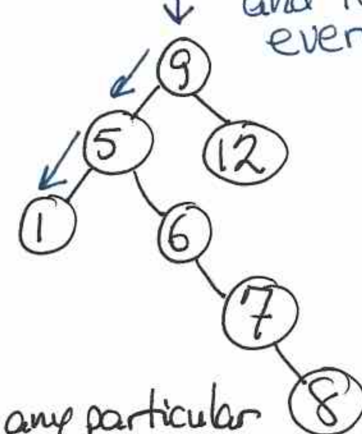
$$\begin{aligned} T(n) &\leq T(k) + T(n-k-1) + d \\ &= [(c+d)k + c] + [(c+d)(n-k-1) + c] + d \\ &= (c+d)n + c - (c+d) + (c+d) \\ &= (c+d)n + c \ ✓ \end{aligned}$$

BST Sorting

Create-BST-from-Input $\Theta(?)$
Inorder-Tree-Walk (Root [Tree]) $\Theta(n)$
+

Operations on BSTs

start at root and take every left



```
find-min (x)
  while left[x] ≠ Null
    x ← left[x]
  return x
```

running time: $O(h)$ for any particular case and $\Theta(h)$ in worst case.
height of tree

```
find (x, k)
  if x==Null or k==key[x]
    return x
  if k < key[x]
    return find (left[x], k)
  else return find (right[x], k)
```

- descend tree from top turning left if search key less than current node's key (right if greater than).
- end when find key or hit Null (key not in tree)

running time: $O(h)$ for any particular case and $\Theta(h)$ worst case.

Examples: Find 6 and then Find 4 on tree above

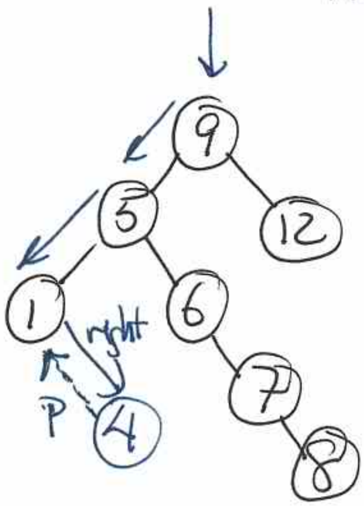
Insert (T, X)

Progress down the tree, just like find, until locate empty leaf to insert into.

Adjust pointers to accomplish insertion.

Example

Insert node containing key of 4



running time: $\Theta(h)$ for any particular case and $\Theta(h)$ worst case

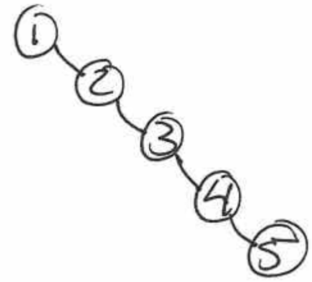
What is the worst case running time to build a BST by repeated insertion (so we can accomplish our sort)?

Each insertion is $\Theta(h)$, and worst case $\Theta(h)$,
n insertions is $\Theta(n \cdot h)$ and worst case $\Theta(n \cdot h)$

Worst case:

$h = n$

So $\Theta(n^2)$



Where does this leave our sorting?

<u>BST sorting</u>	<u>today</u>	<u>next time</u>
Create-BST-from-input	$\Theta(n^2)$	$\Theta(n \log n)$
Inorder-Tree-Walk (root[Tree])	$\Theta(n)$	$\Theta(n)$
	$\Theta(n^2)$	$\Theta(n \log n)$

Next time: If our tree could remain "balanced", with $h \approx \log n$