Sorting III: Binary Search
Trees and BST Sort

Tonight: PS 1 Due, PS 2 Released
Reading: CLRS 4.3, 4.4, 12.1-3

Bruce Tidor <tidor@mit.edu> 32-212
Office Hours: by email appt (usually avail. Th/10)

Last Time: Priority Queue, Heap, Heapsort \( \mathcal{O}(n \log n) \)

• abstract data structure
• useful for task scheduling
  insert \((s, x)\)
  max \((s)\)
  extract_max \((s)\)
  increase_key \((s, x, k)\)

Heapsort
Build max heap from unordered array

\[ \rightarrow \text{Repeatedly extract max to form output (in reverse order)} \]

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

New data structure \(\rightarrow\) New sorting algorithm
Binary Search Tree (BST) \(\rightarrow\) BST Sort
BST data structure

Each node $x$ has:
- $key[x]$
- $left[x]$
- $right[x]$
- $p[x]$
- satellite data

BST Property

For any node $x$:
- all nodes $y$ in left subtree: $key[y] \leq key[x]$
- all nodes $z$ in right subtree: $key[z] \geq key[x]$

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.
Is BST unique for a given collection of keys?

The Binary Search 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, UST tree, ...

No!! Not unique

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 \( \Theta(n) \)

\[ T(0) = C, \text{ some small constant time for empty subtree} \]

\[ C > 0 \]

\[ T(n > 0) : \]

\[ T(n) \leq T(k) + T(n-k-1) + d \text{ (upper bound on time, not in recursive calls)} \]

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: \( n = 0 \) \( T(0) \leq (c+d) \cdot 0 + c = c \) ✓

Assume true for \( m < n \) and show true for \( n \):

\[ T(n) \leq T(k) + T(n-k-1) + d \]

\[ = \left[ (c+d)k + c \right] + \left[ (c+d)(n-k-1) + c \right] + d \]

\[ = (c+d)n + c - (c+d) + (c+d) \]

\[ = (c+d)n + c \] ✓
BST Sorting

Create-BST-From-Input
Inorder-Tree-Walk (Root->Tree[])

Operations on BSTs

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

running time: O(h) for any particular case
and O(n) in worst case.

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)

running time: O(h) for any particular case
and O(n) 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: \(O(h)\) 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 \(O(h)\) and worst case \(\Theta(h)\).
\(n\) insertions is \(O(n \cdot h)\) and worst case \(\Theta(n \cdot h)\)

Worst case:

\[h = n\]
So \(\Theta(n^2)\)
Where does this leave our sorting?

\[
\begin{array}{c}
\text{BST sorting} \\
\text{Create-BST-from-input} \\
\text{Inorder-Tree-Walk (root of tree)}
\end{array}
\begin{array}{cc}
\text{today} & \text{next time} \\
\Theta(n^2) & \Theta(n) \\
\Theta(n) & \Theta(n) \\
\Theta(n^2) & \Theta(n \log n)
\end{array}
\]

Next time: If our tree could remain "balanced", with \( n \approx \log n \).