Previously ...

Binary Search Trees
- insertion
- deletion
- traversals

BST’s delete

Case 1: node is a leaf
- trivial, delete node and set parent’s pointer to NULL

Case 2: node has 1 child
- trivial, set parent’s pointer to the only child and delete node

Case 3: node has 2 children
- find successor and copy successor’s data to node
- delete successor

Analysis

Tree Shape
- Depends on order of insertion

Implications
- Cost of basic operations?
  - search, insert, remove
  - worst-case: $O(h) = O(n)$
  - best-case: $O(h) = O(\log n)$
  - average-case: $O(h) = \?$

Correspondence with QuickSort

If $n$ distinct keys are inserted into a BST in random order,
- expected number of compares for a search/insert is $\approx 2 \ln n \approx 1.39 \log n$
- proof: 1-1 correspondence with quick-sort partitioning

Average Case: $O(\log n)$
Other Operations

- **Min()** — find smallest value $O(h)$
- **Max()** — find largest value $O(h)$
- **Floor(k)** — find largest value $\leq$ than $k$ $O(h)$
- **Ceiling(k)** — find smallest value $\geq$ than $k$ $O(h)$

Computational Cost

<table>
<thead>
<tr>
<th>Search</th>
<th>Sequential Search (unordered sequence)</th>
<th>Binary Search (ordered sequence)</th>
<th>Binary Search Tree</th>
</tr>
</thead>
<tbody>
<tr>
<td>search</td>
<td>$O(n)$</td>
<td>$O(\log n)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>insert</td>
<td>$O(n)$</td>
<td>$O(\log n)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>delete</td>
<td>$O(n)$</td>
<td>$O(\log n)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>min/max</td>
<td>$O(n)$</td>
<td>$O(1)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>floor/ceiling</td>
<td>$O(n)$</td>
<td>$O(\log n)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>rank</td>
<td>$O(n)$</td>
<td>$O(\log n)$</td>
<td>$O(n)$ **</td>
</tr>
</tbody>
</table>

** requires the use of 'size' at every node

Can we sort using BSTs?

Given $n$ numbers …

- what is the cost of a bad case? $O(n^2)$
- what is the cost of a best case? $O(n \log n)$