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
Implications

Cost of basic operations?
search, insert, remove

worst-case
Implications

Cost of basic operations?
search, insert, remove

worst-case $O(h) = O(n)$
Implications

Cost of basic operations?
   search, insert, remove

worst-case $O(h) = O(n)$

best-case
Implications

Cost of basic operations?
search, insert, remove

**worst-case**  \[ O(h) = O(n) \]

**best-case**  \[ O(h) = O(\log n) \]
Implications

Cost of basic operations?

search, insert, remove

**worst-case** \( O(h) = O(n) \)

**best-case** \( O(h) = O(\log n) \)

**average-case**
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
Correspondence with QuickSort

If $n$ distinct keys are inserted into a BST in random order

expected number of compares for a search/insert is

$\sim 2 \ln n \sim = 1.39 \log n$
Correspondence with QuickSort

If \( n \) distinct keys are inserted into a BST in random order

expected number of compares for a search/insert is

\(~2 \ln n \sim 1.39 \log n\)

**proof**: 1-1 correspondence with quick-sort partitioning
Correspondence with QuickSort

If \( n \) distinct keys are inserted into a BST in random order

expected number of compares for a search/insert is

\[ \sim 2 \ln n \sim = 1.39 \log n \]

proof: 1-1 correspondence with quick-sort partitioning

Average Case: \( O(\log n) \)
N = 255
max = 16
avg = 9.1
opt = 7.0
The Height of a Random Binary Search Tree

BRUCE REED

McGill University, Montreal Quebec, Canada and CNRS, Paris, France

Abstract. Let $H_n$ be the height of a random binary search tree on $n$ nodes. We show that there exist constants $\alpha = 4.311 \cdots$ and $\beta = 1.953 \cdots$ such that $E(H_n) = \alpha \ln n - \beta \ln \ln n + O(1)$, We also show that $\text{Var}(H_n) = O(1)$.

Categories and Subject Descriptors: E.1 [Data Structures]: trees; G.2 [Discrete Mathematics]; G.3 [Probability and Statistics]

General Terms: Algorithms, Theory

Additional Key Words and Phrases: Binary search tree, height, probabilistic analysis, random tree, asymptotics, second moment method
The Height of a Random Binary Search Tree

BRUCE REED

McGill University, Montreal Quebec, Canada and CNRS, Paris, France

Abstract. Let $H_n$ be the height of a random binary search tree on $n$ nodes. We show that there exist constants $\alpha = 4.311 \ldots$ and $\beta = 1.953 \ldots$ such that $\mathbb{E}(H_n) = \alpha \ln n - \beta \ln \ln n + O(1)$. We also show that $\text{Var}(H_n) = O(1)$.

Categories and Subject Descriptors: E.1 [Data Structures]: trees; G.2 [Discrete Mathematics]; G.3 [Probability and Statistics]

General Terms: Algorithms, Theory

Additional Key Words and Phrases: Binary search tree, height, probabilistic analysis, random tree, asymptotics, second moment method
The Height of a Random Binary Search Tree

BRUCE REED

McGill University, Montreal Quebec, Canada and CNRS, Paris, France

Abstract. Let $H_n$ be the height of a random binary search tree on $n$ nodes. We show that there exist constants $\alpha = 4.311 \cdots$ and $\beta = 1.953 \cdots$ such that $\mathbb{E}(H_n) = \alpha \ln n - \beta \ln \ln n + O(1)$. We also show that $\text{Var}(H_n) = O(1)$.

Categories and Subject Descriptors: E.1 [Data Structures]: trees; G.2 [Discrete Mathematics]; G.3 [Probability and Statistics]

General Terms: Algorithms, Theory

Additional Key Words and Phrases: Binary search tree, height, probabilistic analysis, random tree, asymptotics, second moment method

Expected height: $\sim 4.311 \ln n \sim = 2.99 \log n$
Other Operations

Min() — find smallest value

Max() — find largest value

Floor(k) — find largest value <= than k

Ceiling(k) — find smallest value >= than k
Other Operations

Min() — find smallest value
Max() — find largest value
Floor(k) — find largest value <= than k
Ceiling(k) — find smallest value >= than k
Other Operations

**Min()** — find smallest value \(O(h)\)

**Max()** — find largest value \(O(h)\)

**Floor\( (k)\)** — find largest value \(\leq\) than \(k\)

**Ceiling\( (k)\)** — find smallest value \(\geq\) than \(k\)
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
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></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(h)</td>
</tr>
<tr>
<td>insert</td>
<td>O(n)</td>
<td>O(n)</td>
<td>O(h)</td>
</tr>
<tr>
<td>delete</td>
<td>O(n)</td>
<td>O(n)</td>
<td>O(h)</td>
</tr>
<tr>
<td>min/max</td>
<td>O(n)</td>
<td>O(1)</td>
<td>O(h)</td>
</tr>
<tr>
<td>floor/ceiling</td>
<td>O(n)</td>
<td>O(log n)</td>
<td>O(h)</td>
</tr>
<tr>
<td>rank</td>
<td>O(n)</td>
<td>O(log n)</td>
<td>O(h) **</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?

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