## Cost of Basic Operations

<table>
<thead>
<tr>
<th>Operations</th>
<th>Sequential Search</th>
<th>Binary Search</th>
<th>BST</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>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>

The height $h$ can be proportional to $O(n)$ in the worst-case or to $O(\log n)$ after random insertions.
Balanced Trees

Challenge:
provide guarantees on fast tree operations
minimum $h$ of a binary tree with $n$ nodes?
Balanced Trees

Challenge:
provide guarantees on fast tree operations
minimum $h$ of a binary tree with $n$ nodes?

**AVL Tree** (Adelson-Velskii and Landis)
BST that maintains balance using rotations
AVL Tree

BST with a **balance condition**
ensures height proportional to $O(\log n)$
AVL Tree

BST with a **balance condition**

ensures height proportional to $O(\log n)$

Simplest idea is to require that the left and right subtrees have the **same height** (for every node)

too rigid — only perfectly balanced BSTs
Relaxing the Balance Condition

An **AVL tree** is a BST wherein for every node, the **height** of the left and right subtrees can differ by at most 1. Height information is kept for each node. Height of an empty tree is defined to be \(-1\).
Balance of a Node

\[ \text{balance}(x) = \text{height}(x.\text{left}) - \text{height}(x.\text{right}) \]

An **AVL tree** is a BST wherein for every node \( x \), \( |\text{balance}(x)| \leq 1 \).
balance(x) = 1
balance(x) = 0
balance(x) = -1
AVL Tree?

height of a node

balance of a node
AVL Tree?

height of a node

balance of a node
Minimum #nodes for an AVL with h=9
**Height of an AVL Tree**

$N(h)$ — smallest number of nodes in an AVL tree of height $h$

Base cases: $N(0) = 1$ and $N(1) = 2$

For $h > 1$: $N(h) = 1 + N(h-1) + N(h-2)$
Height of an AVL Tree

\[ N(h) = N(h - 1) + N(h - 2) + 1 \]

\[ N(h) \geq \phi^h, \text{ where } \phi \approx 1.62 \text{ (cf. Fibonacci)} \]

\[ n \geq N(h) \text{ (n nodes in an AVL tree of height } h) \]

\[ n \geq \phi^h, \text{ therefore } \log_\phi n \geq h \]

\[ h \geq 1.44 \log n = O(\log n) \]
Implications

Search operations

same as BST’s search
cost $O(\log n)$

Insert and Remove operations

need to detect and fix imbalances
Implications

Search operations
- **same** as BST’s search
- cost $O(\log n)$

Insert and Remove operations
- need to **detect** and **fix** imbalances

Remove 60? Insert 73?
Insertion

Use BST’s insertion

Balance may become 2 or -2 for some node(s)
   need to check nodes along the path to the root
   update heights and calculate balance

How to fix?
   adjust the tree by applying rotation
Four Cases

Let us call the node that must be rebalanced \( x \)

1: insertion into **left** subtree of \( x \)'s **left** child

2: insertion into **right** subtree of \( x \)'s **left** child

3: insertion into **left** subtree of \( x \)'s **right** child

4: insertion into **right** subtree of \( x \)'s **right** child
Case 1: Single Rotation (R)
Case 1: Single Rotation (R)
Case 1: Single Rotation (R)
Case 4: Single Rotation (L)
Case 4: Single Rotation (L)
Case 4: Single Rotation (L)
Case 2: Double Rotation (LR)
Case 2: Double Rotation (LR)
Case 2: Double Rotation (LR)
Case 2: Double Rotation (LR)
Case 2: Double Rotation (LR)
Case 2: Double Rotation (LR)
Case 3: Double Rotation (RL)
Case 3: Double Rotation (RL)
Case 3: Double Rotation (RL)
Case 3: Double Rotation (RL)
Case 3: Double Rotation (RL)
Case 3: Double Rotation (RL)