A binary search tree is organized, as the name suggests, in a binary tree, as shown in Figure 12.1. We can represent such a tree by a linked data structure in which each node is an object. In addition to a key and satellite data, each node contains attributes left, right, and p that point to the nodes corresponding to its left child, its right child, and its parent, respectively. If a child or the parent is missing, the appropriate attribute contains the value NIL. The root node is the only node in the tree whose parent is NIL.

The keys in a binary search tree are always stored in such a way as to satisfy the binary-search-tree property:

Let \( x \) be a node in a binary search tree. If \( y \) is a node in the left subtree of \( x \), then \( y.key \leq x.key \). If \( y \) is a node in the right subtree of \( x \), then \( y.key \geq x.key \).
Binary Search Tree Examples

BST Property
Examples
Tree Walk
Search
Min and Max
Successor
Insert
Delete 1
Delete 2
Delete 4
Delete 3
Random Trees

CS 3343 Analysis of Algorithms
**Relationship between number of nodes and height**

\[ \frac{n}{h} \]

<table>
<thead>
<tr>
<th>( n )</th>
<th>Lower bound ( h )</th>
<th>Upper bound ( h )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>8</td>
<td>3</td>
<td>7</td>
</tr>
<tr>
<td>9</td>
<td>3</td>
<td>8</td>
</tr>
<tr>
<td>10</td>
<td>3</td>
<td>9</td>
</tr>
</tbody>
</table>

\[ n \geq \lceil \lg n \rceil \quad \Omega (\lg n) \quad O(n) \]

<table>
<thead>
<tr>
<th>( h )</th>
<th>Lower bound ( n )</th>
<th>Upper bound ( n )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>7</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>15</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>31</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>63</td>
</tr>
</tbody>
</table>

\[ h \leq h+1 \quad 2^{h-1} + 2^h = 2^{h+1} - 1 \]

\[ \Omega (h) \quad O(2^h) \]

Possible to have 1,000,000,000 nodes in a tree of height 30
Inorder Tree Walk

- Assume initial call is Inorder-Tree-Walk(T, root)

**Inorder-Tree-Walk(x)**

1. if \( x \neq \text{NIL} \)
2. \( \text{Inorder-Tree-Walk}(x, \text{left}) \) (visited left subtree)
3. print \( x, \text{key} \) (visited \( x \))
4. \( \text{Inorder-Tree-Walk}(x, \text{right}) \) (visited \( x \) and all nodes under \( x \) [visited right subtree]

**Theorem 12.1**

If \( x \) is the root of an \( n \)-node subtree, then the call Inorder-Tree-Walk\( (x) \) takes \( \Theta(n) \) time.

\[
T(n) = 2T\left(\frac{n}{2}\right) + c
\]

- **Master method**
  - \( \Theta(1) \) because \( c \) is a constant
  - \( b = 2, a = 2 \)
  - \( \Omega\left(n^\log_b^a\right) = \Omega\left(n^{\log_2^2}\right) = \Omega\left(n\right) \)
  - So by Master Theorem recurrence is \( \Theta(n) \)
\[ T(n) = 2c \left( \frac{u}{2} \right) + c \]

**Recursion tree**

\[ T(n) \rightarrow \]

\[ T \left( \frac{u}{4} \right) \rightarrow T \left( \frac{u}{8} \right) \rightarrow T \left( \frac{u}{16} \right) \rightarrow \ldots \]

\[ \text{height} = \lg u \]

\[ c 2^h = cn \]

\[ \sim 2c 2^h = 2cn \]
\( x \) is NIL — one line  
\( x \neq \text{NIL} \) — four lines  
plus recursive calls

\[
\begin{array}{c|c}
\text{\( n \)} & \text{\# of lines executed} \\
\hline
0 & 1 \\
1 & 6 \\
2 & 11 \\
3 & 16 \\
\end{array}
\]

for \( n \) guess \( S_{n+1} \)

Suppose tree size \( n \)

- Left subtree size \( n_1 \) means \( n = n_1 + n_2 + 1 \)
- Right subtree size \( n_2 \)

Suppose \( S_{n+1} \) formula works for subtrees  
then total is

\[
4 + (5n_1 + 1) + (5n_2 + 1)
\]

\[
= 5n_1 + 5n_2 + 5 + 1
\]

\[
= 5(n_1 + n_2 + 1) + 1 = S_{n+1}
\]