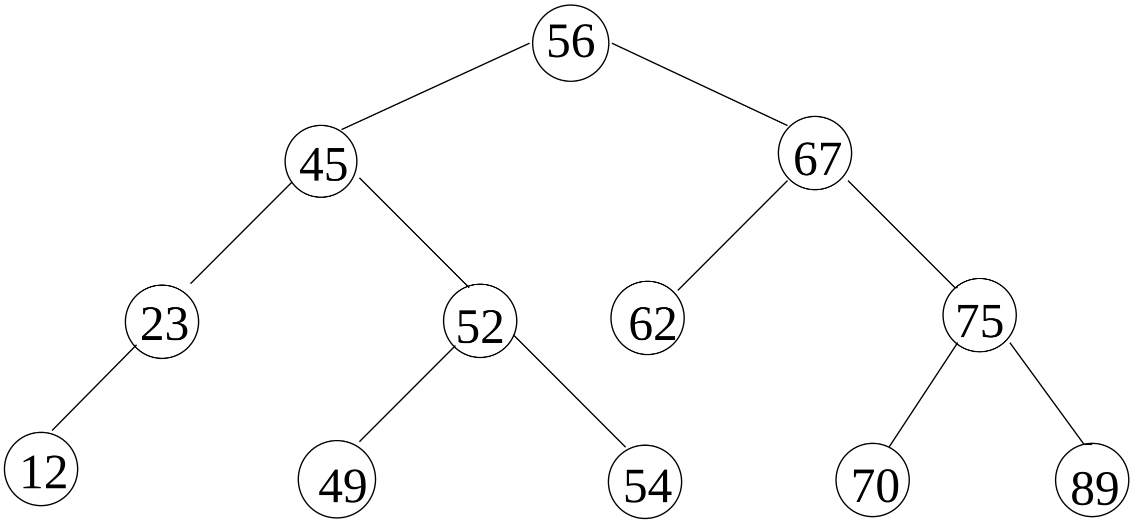
Binary search tree

- Unorganized tree may require traversal of entire tree for searching.
- Search trees are designed to support efficient searching.
- Organize elements in the tree by imposing some constraints on relative positions of neighbouring elements.
- A tree which allows efficient search and retrieval called a search tree
- Essentially a search tree store a finite set of comparable elements drawn from a totally ordered set.
- The elements satisfy **reflexive**, **transitive** and **antisymmetric** properties.

Binary search tree

- The constraints for relative placement of neighbouring elements in a search tree uses this data ordering criterion of the set.
- Mose popular search tree is a binary tree called BST.
- A binary search tree is either empty or consists of a node and two subtrees left subtree and the right subtree each of which of which is also search trees and such that the value stored at root is
  - less than the value stored at any in node belonging to the right subtree, and
  - greater than the value stored at any in node belonging to the left subtree.
  - Example of a BST is provided in the next page.

Searching a BST



Searching a BST

- The search begins at root.
- Compare the element being searched with the value stored at the current node
  - if it is less then perform recursive search with the left subtree of the current node.
  - if it is greater then perform recursive search with right subtree of the current node.
- If the element is exist in the tree than the search becomes successful.
- If the element is does not exist in the tree than the search terminates when we are about to fall off from a node in carrying out the recursive search.
- The time of search is determined by the depth of the node storing the element.

Analysis of searching a BST

- Average running time for construction a BST with n elements $a_1, a_2, \ldots, a_n$ is $O(n \log n)$, which can be proved as follows.
- Let $b_1, b_2, \ldots, b_n$ be the descending sequence of given elements.
- Let $T(n)$ be the average number of comparisons needed.
- For an empty BST, $T(0) = 0$.
- Since the sequence of elements to form the BST is random a_1 can be equal to any b_j for $1 \leq j \leq n$ with equal probability.
- a_1 becomes root then $j - 1$ elements $b_1, b_2, \ldots, b_{j-1}$ must belong to left subtree of root.
- $n-j$ elements $b_{j+1}, b_{j+2}, \ldots, b_n$ must belong to right subtree of root.

Analysis of BST searching contd.

- Recursively $T(j-1)$ comparisons necessary for left BST
- Recursively $T(n-j)$ comparisons necessary for right BST
- Each must also be compared with root. So for left BST $(j-1)+T(n-j)$ comparisons and for right BST $n-j+T(n-j)$ comparisons are necessary.
- All total $n-1+T(j-1)+T(n-j)$ comparisons are necessary.

Analysis of BST searching contd.

- Since a_1 can be any of the sequence of elements $b_1, b_2, \ldots, b_n$ with equal probability this implies average number of comparison is

$$\begin{aligned} T(n) &= \frac{1}{n}\sum_{j=1}^{n}(n-1+T(j)+T(n-j)) \\ &= n-1+\frac{2}{n}\sum_{j=1}^{n}T(j) \end{aligned}$$

- Claim is $T(n) \leq kn\log n$ for $k = 3$ and $n \geq 2$
- $T(2) = 2-1+0 = 1$, as $T(0) = 0$ and $T(1) = 0$.

Analysis of BST searching contd.

- For induction hypothesis assume

$$T(n) \leq n + \frac{2}{n} + \frac{2}{n}\sum_{j=2}^{n-1} kj \ln j$$

- Since $j \ln j$ is concave upwards we have

$$\sum_{j=2}^{n-1} j \ln j \leq \int_2^n x \ln x \leq \frac{n^2 \ln n}{2} - \frac{n^2}{4}$$

- Substituting the bound for sum we have

$$T(n) \leq n + \frac{2}{n} + kn \ln n - \frac{kn}{2}$$

- Since $n \geq 2$ and $k = 3$ it follows that

$$n + \frac{2}{n} \leq \frac{kn}{2}$$

Properties of BST

- Inorder traversal of a BST provides the sorted sequence of keys stored in the BST.
- The proof is by induction. Clearly it is true for BST of heights 0 or 1.
- Let it be true for BSTs of heights less or equal to h.
- Consider the case of BST of height $h + 1$.
- Delete the root, we have two BSTs (left and right subtrees) each of height h or less. Call these trees as T_L and T_R respectively.
- We know every element of T_L is less than any element belonging to T_R
- The element at the root of the original tree is greater than all elements of T_L and less than the every element of T_R.

Properties of BST

- By induction hypothesis inorder traversals of each of the BSTs T_L and T_R provides a sorted list. Therefore inorder traversal of T is also a sorted list.
- We can also apply induction to prove the inverse that if the inorder traversal of a binary tree T produces a sorted list, then the binary tree is a BST.
- Consider the element corresponding to root of this tree. The elements less than that obviously belong to left subtree of root. Similarly elements greater than that nodes belong to right subtree. This follows from the characteristic of inorder traversal.
- The left and right subtrees are BSTs by induction.
- Therefore the entire tree is also a BST by definition.

Deletion from BST

- A binary tree consisting 2 nodes such that its inorder traversal is a sorted list is obviously a BST.
- Assume that a binary tree of with $n-1$ nodes when traversed in inorder produces a sorted list then that tree is BST.
- Consider a binary tree of n nodes when traversed gives a sorted list.

Deletion from BST

- Deletion of a leaf node is simple.
- Deletion of an internal node x realized as follows:
  - if x has a single child say y then let the parent of x adopt y as its own child and delete x.
  - otherwise determine the inorder successor y of x, swap contents of x with that of y delete y.

Internal node with 2 children

- The inorder successor of an internal node having 2 children is a node which is reached by a traversing a sequence which begins with a single right child link and all remaining leftchild links.
- This node is either a leaf node or a node with one child.
- If it is a leaf node then delete the node otherwise collapse that node with its only child.
- More precisely, to delete an internal node with two children
  1. swap the contents with leftmost (rightmost) descendant, and
  2. remove that descendant node.