Lecture 10: Binary Search Trees

Wholeness of the Lesson

Binary Search Trees arose as a natural solution to the need for combining the efficient insertion and deletion of linked lists with fast sorting and binary search provided by arrays and array lists. Expansion from a linear structure to a two-dimensional structure (a tree) makes a solution of this kind possible. Likewise, any problem becomes easier to solve if one can transcend the boundaries of the problem.

A List Wish-List

1. For many purposes, keeping data stored in memory in a sorted order is a way to optimize a variety of searches on the data.

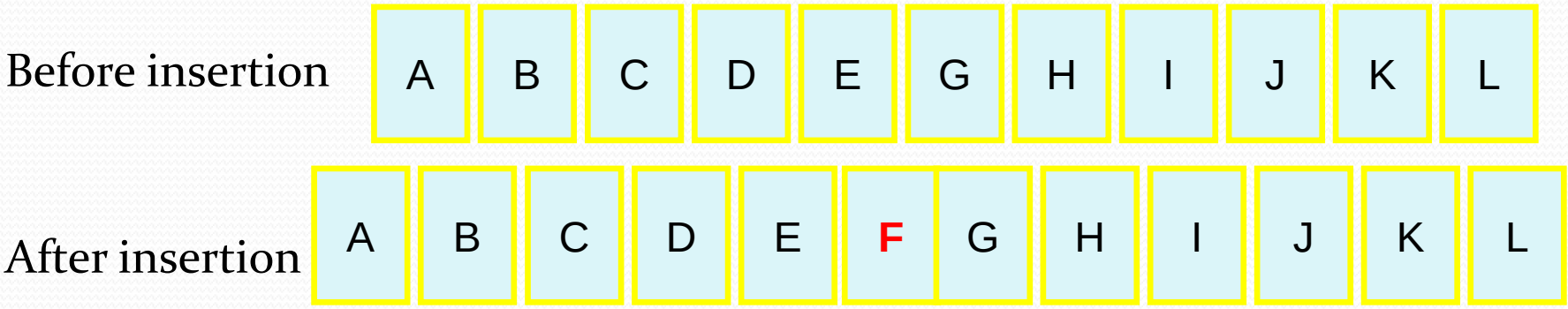
   A typical problem one might try to solve when it is possible to keep data sorted is:

   Find all Employees having a salary between $50,000 and $75,000.

2. If Employees have been maintained in sorted order – sorted by salary -- in some kind of list, then to solve the problem, we find the first Employee in the list with salary no less than 50,000 and also find the first Employee with salary bigger than 75,000. In this way we can specify the desired range of Employees.

How to Maintain Data in Memory, In Sorted Order?

- *ArrayLists.* If we are using an ArrayList, then maintaining the list in sorted order is expensive because each time we add a new element, it must be inserted into the correct spot, and this requires array copy routines. Example: suppose we need to insert F.



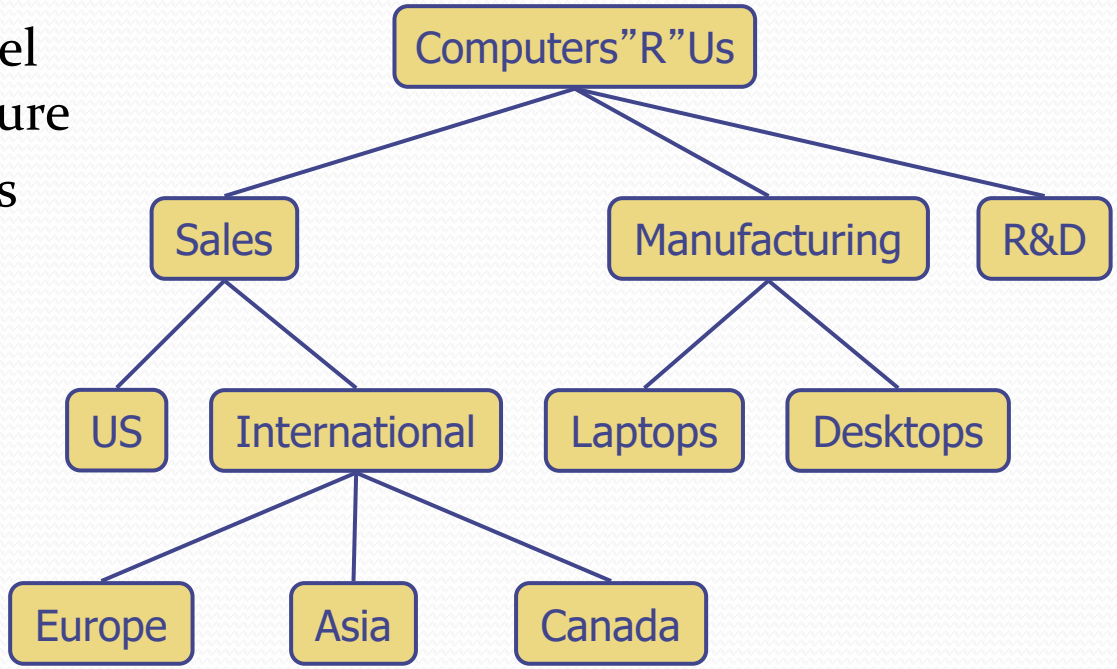
- *LinkedLists.* If we are using a LinkedList, it is easy to maintain sorted order since insertions are efficient, but searches are not very efficient.

How to Maintain Data in Memory, In Sorted Order?

- *The Need.* We need a data structure that performs insertions efficiently in order to maintain sorted order (like a linked list) but that also performs finds efficiently (binary search is highly efficient in an array list when elements are sorted)
- The solution: Binary Search Tree

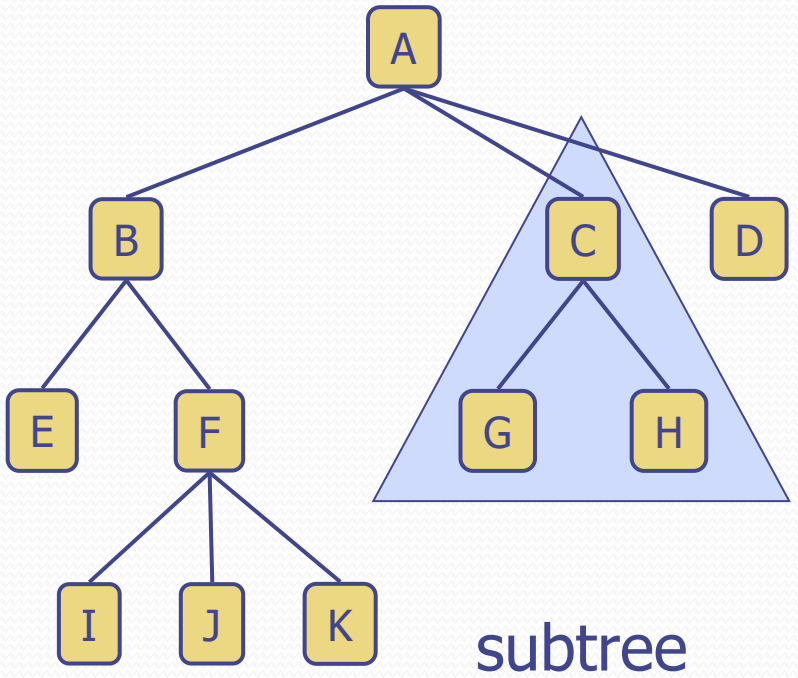
Trees

- In computer science, a tree is an abstract model of a hierarchical structure
- A tree consists of nodes with a parent-child relation
- Applications:
  - Organization charts
  - File systems



Tree Terminology

- ***Root***: node without parent (A)
- ***Internal node***: node with at least one child (A, B, C, F)
- ***Leaf*** : ***node*** is a node without children (E, I, J, K, G, H, D)
- ***Ancestors of a node***: parent, grandparent, grand-grandparent, etc. (ancestors of K: F, B, A)
- ***Descendant of a node***: child, grandchild, grand-grandchild, etc. (descendants of B are E, F, I, J, K)
- ***Subtree***: tree consisting of a node and its descendants



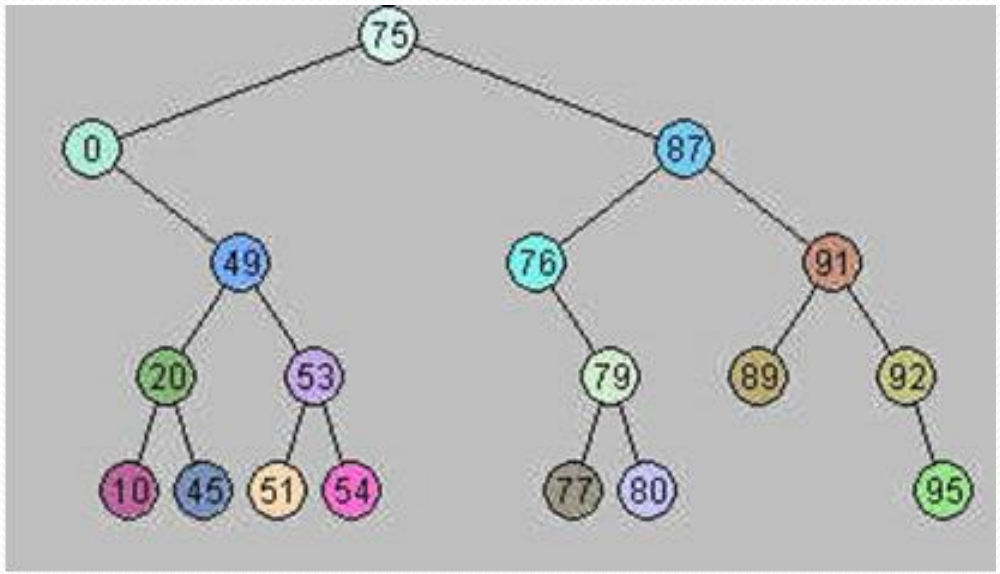
Binary Search Trees

- A *binary tree* is a tree that each node has a reference to a left and right child node (though some references may be null).
- A *binary search tree* (BST) is a binary tree in which the BST Rule is satisfied:

  BST Rule:

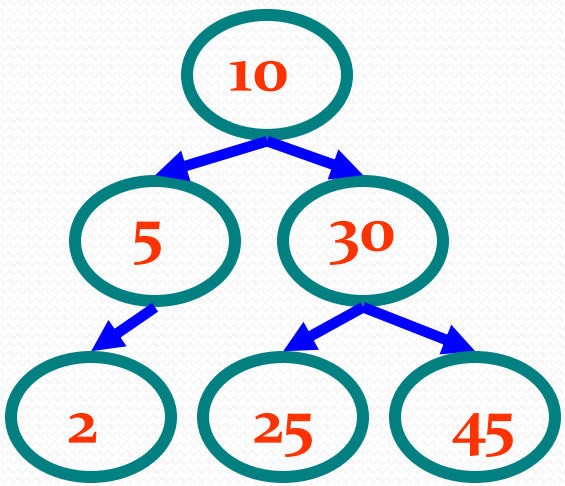
  At each node N, every value in the left subtree of N is less than the value at N, and every value in the right subtree of N is greater than the value at N.

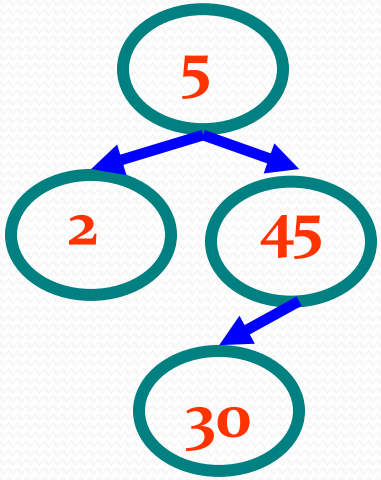
  For the moment, we assume all values are of type Integer.



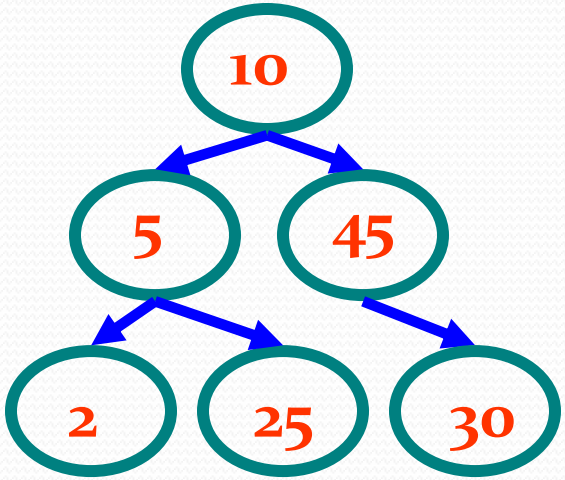
Binary Search Trees

More Examples





Binary search trees



Not a binary search tree, why?

Binary Search Trees ADT

- The fundamental operations on a BST are:

```
public boolean find(Integer val)
public void insert(Integer val)
public boolean remove(Integer val)
public void print()
```

- Demo code: `lesson10.bst`

Building Binary Search Trees

- BSTs are typically built from an *insertion sequence*, starting with an empty tree.
- We use the insertion algorithm(next slide) to insert the elements one by one to the BST.

Insertion into a BST

Recursive algorithm for insertion of Integer x [an iterative implementation is given in the demo code]

1. If the root is null, create a new root having value x
2. Otherwise, if x is less than the value in the root, insert x into the left subtree; if x is bigger than the value in the root, insert x into the right subtree

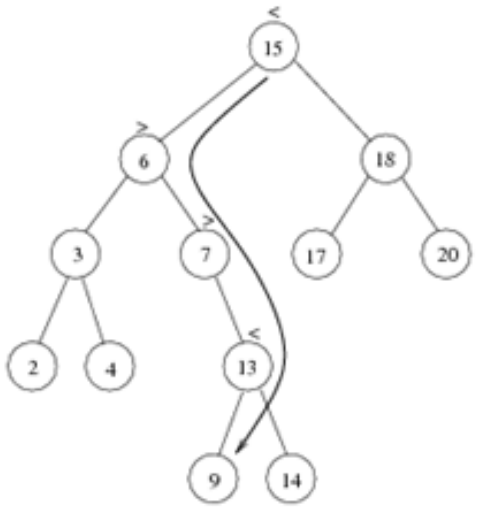
Searching a BST

Recursive algorithm for finding an Integer x

The recursive *find* operation for BSTs is in reality the binary search algorithm in the context of BSTs.

1. If the root is null, return false
2. If the value in the root equals x, return true
3. If x is less than the value in the root, return the result of searching the left subtree
4. If x is greater than the value in the root, return the result of seraching the right subtree

Example: search for 9



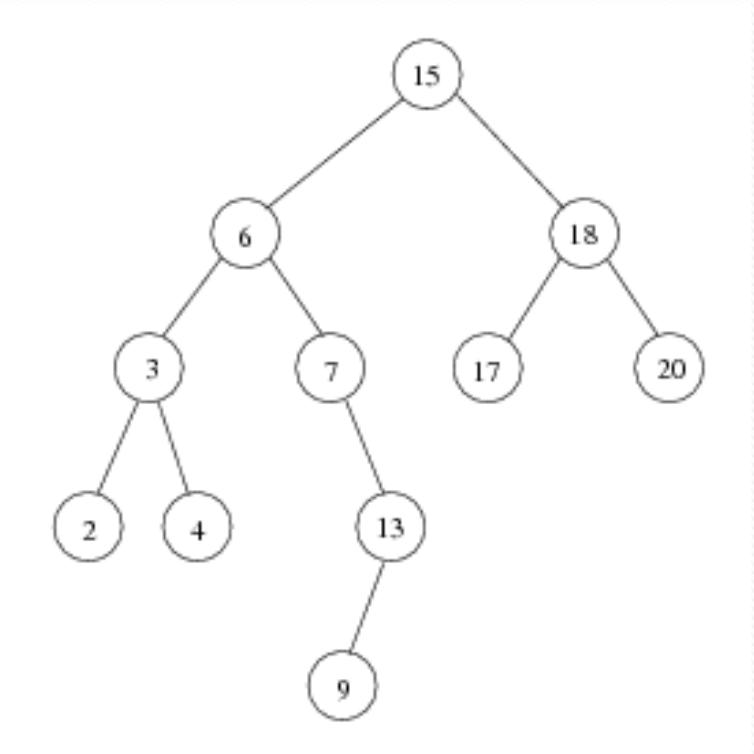
BST In-Order Traversal

Recursive print algorithm – outputs values in every node in sorted order

1. If the root is null, return
2. Print the left subtree of the root, in sorted order
3. Print the root
4. Print the right subtree of the root, in sorted order

BST In-Order Traversal

- Example:



In-Order: 2, 3, 4, 6, 7, 9, 13, 15, 17, 18, 20

Deletions in a BST

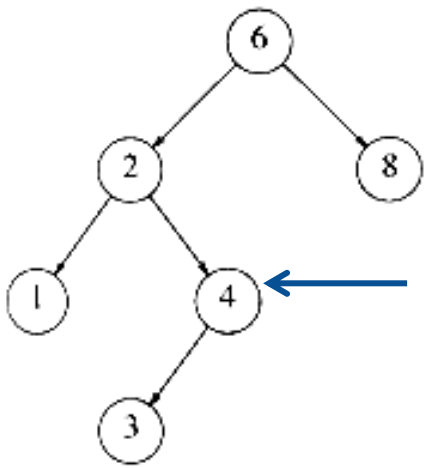
Algorithm to remove Integer x

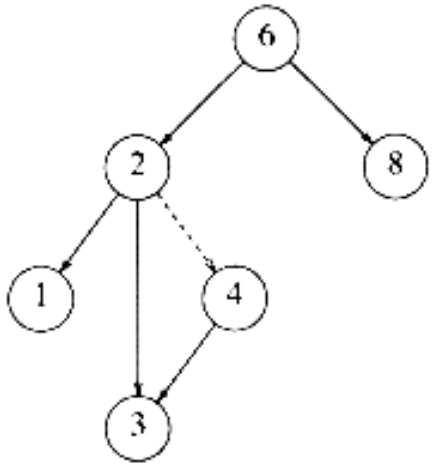
- Case I: Node to remove is a leaf node

  Find it and set to null

- Case II: Node to remove has one child

  Create new link from parent to child, and set node to be removed to null

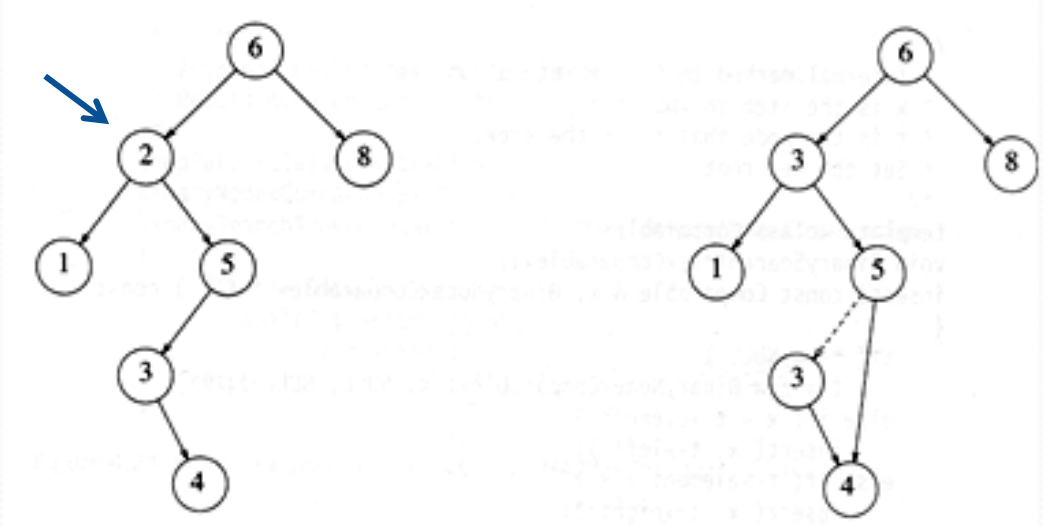




Deletions in a BST - continued

Algorithm to remove Integer x

- Case III: Node to remove has two children
  1. Find smallest node in right subtree – say it stores an Integer y. This node has at most one child
  2. Replace x with y in node to be removed. Delete node that used to store y – this is done as in one of the two cases above



Handling Duplicates in a BST

- To handle duplicate values, store in each `Node` a `List` (instead of a value); then when a value is added to a `Node`, it is instead added to the `List`.

- Example: Suppose we are storing `Employee`s in a BST, ordered by `name`. Our BST will be created so that the value in each node is a `List<Employee>` instance. Then, after insertions, all `Employee`s with the same `name` will be found in a single `List` located in a single `Node`.

- Example: Suppose we want to have a data structure that provides us with a count of how many Employees have a particular salary. To begin, we could order Employees by salary (using a `SalaryComparator`), insert into a BST using `List`s as values in each `Node`. After all `Employee`s have been inserted, we could answer the query "How many have salary 50000?" by searching for the 50,000 node and then returning the size of the list stored at that node.

Using BSTs for Sorting

- Consider the following procedure for sorting a list of Integers:
  - Insert them into a BST
  - Print the results (or modify "print" so that it puts values in a list)
- How good is this new sorting algorithm? How does it compare to `MinSort` ? The Lab asks you to implement this algorithm and compare its running time to that of `MinSort`.

BSTs in the Java Libraries

- Java uses a special kind of BST (a "balanced" tree) as a background data structure for several of their data structures – namely `TreeSet` and `TreeMap`.

Main Point

A *binary search tree* (BST) is a binary tree in which the BST Rule is satisfied:

> At each node *N*, every value in the left subtree of *N* is less than the value at *N*, and every value in the right subtree of *N* is greater than the value at *N*.

BSTs provide efficient search, insert, and remove operations on orderable data. A binary search tree is an example of the principle of Diving: Because the structure is right, the basic operations are accomplished with maximum efficiency.

Connecting the Parts of Knowledge With the Wholeness of Knowledge

Fundamental patterns of consciousness in the realm of binary trees

1. Binary search trees support insert, remove, and find operations with efficient performance, as well as efficient support for accessing elements in order, such as findMin, findMax, and finding elements in a specified range.
2. The structure of complete binary trees involves an expansion from 1 to 2 to 4 to 8, and eventually to 64 (mirroring to a large extent the significant numbers that mark the progress of unfoldment within the Ved, from A to AK to the fourfold Rishi, Devata, Chhandas, Samhita, to 8-fold prakriti through the first Richa to the 64 fundamental impulses that structure the first 192 syllables of Rk Ved).

---

3. **Transcendental Consciousness:** TC is the home of all the impulses of natural law, of creative intelligence.
4. **Wholeness moving within itself**: In Unity Consciousness, the emergence the structure of pure knowledge as appreciated as a self-referral activity of consciousness interacting with itself.