Binary Search Trees

CMSC 132

Department of Computer Science

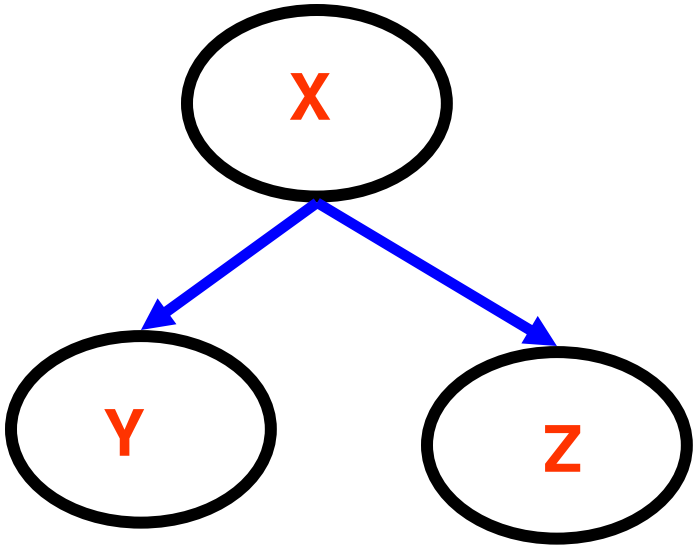
University of Maryland, College Park

Overview

- Binary Search Tree
- Search
- Properties
- Construction
- Insertion
- Deletion
- Polymorphic BST
- Polymorphic, recursive linked list

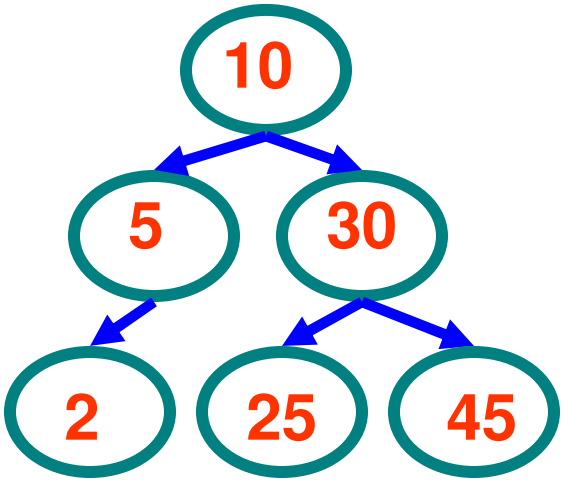
Binary Search Tree (BST)

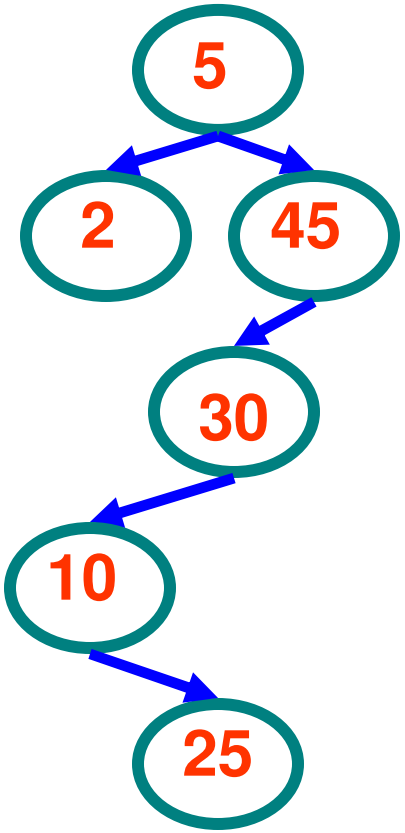
- **Key property**
  - **Value at node**
    - **Smaller values in left subtree**
    - **Larger values in right subtree**
  - **Example**
    - **X > Y**
    - **X < Z**



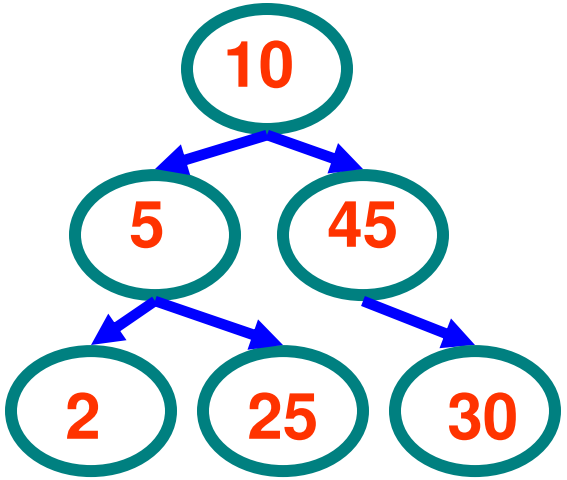
Binary Search Trees

- **Examples**





Binary search trees



Non-binary search tree

Iterative Search of BST

```
Node Find( Node n, Value key) {
    while (n != null) {
        if (n.data == key)        // Found it
            return n;
        if (n.data > key)         // In left subtree
            n = n.left;
        else                      // In right subtree
            n = n.right;
    }
    return null;
}
Find( root, keyValue );
```

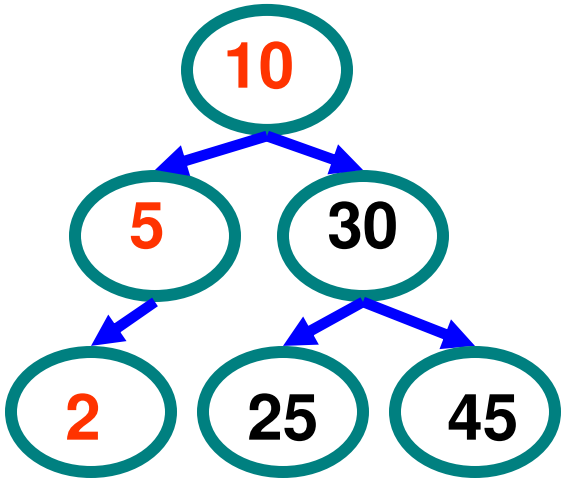
Recursive Search of Binary Tree

```
Node Find( Node n, Value key) {
    if (n == null)                  // Not found
        return( n );
    else if (n.data == key)         // Found it
        return( n );
    else if (n.data > key)          // In left subtree
        return Find( n.left, key );
    else                            // In right subtree
        return Find( n.right, key );
}

Find( root, keyValue );
```

Example Binary Searches

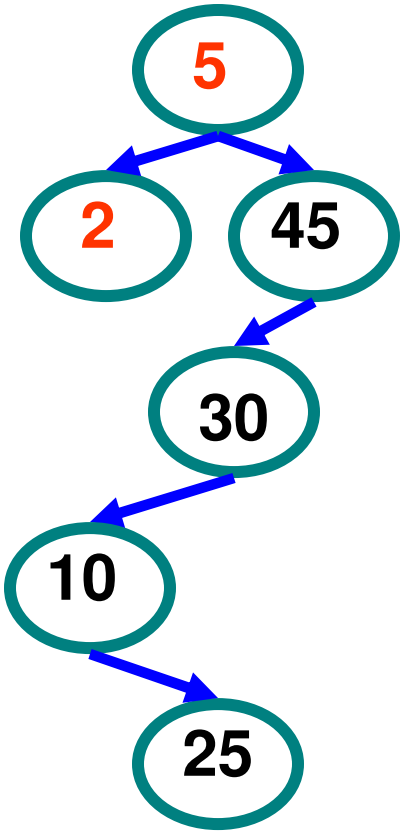
- Find (2)



10 > 2, left

5 > 2, left

2 = 2, found

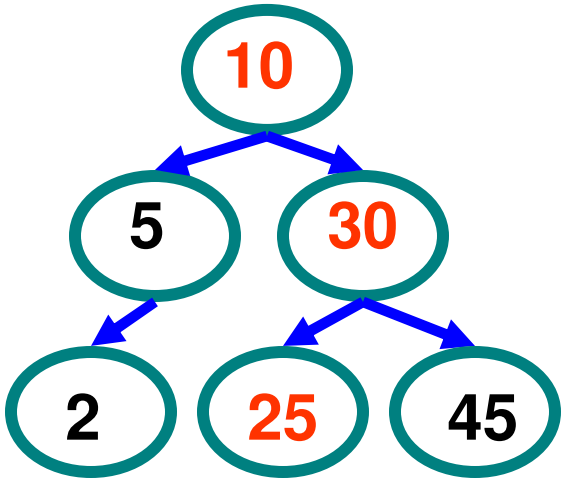


5 > 2, left

2 = 2, found

Example Binary Searches

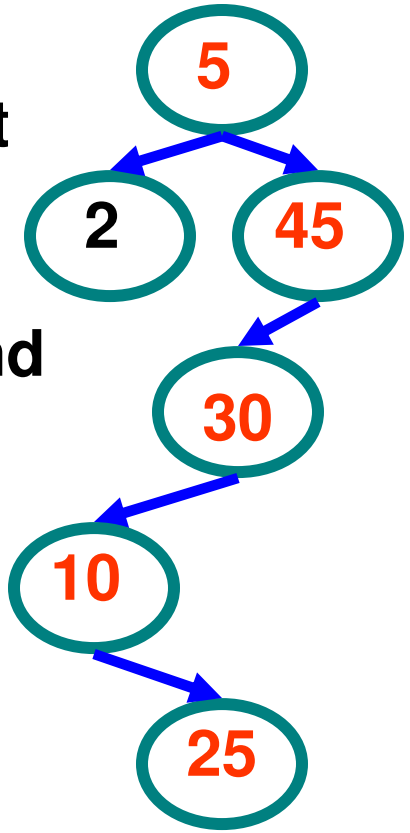
- Find (25)



10 < 25, right

30 > 25, left

25 = 25, found



5 < 25, right

45 > 25, left

30 > 25, left

10 < 25, right

25 = 25, found

Binary Search Properties

- Time of search
  - Proportional to height of tree
  - Balanced binary tree
    - O(log(n)) time
  - Degenerate tree
    - O(n) time
    - Like searching linked list / unsorted array
- Requires
  - Ability to compare key values

Binary Search Tree Construction

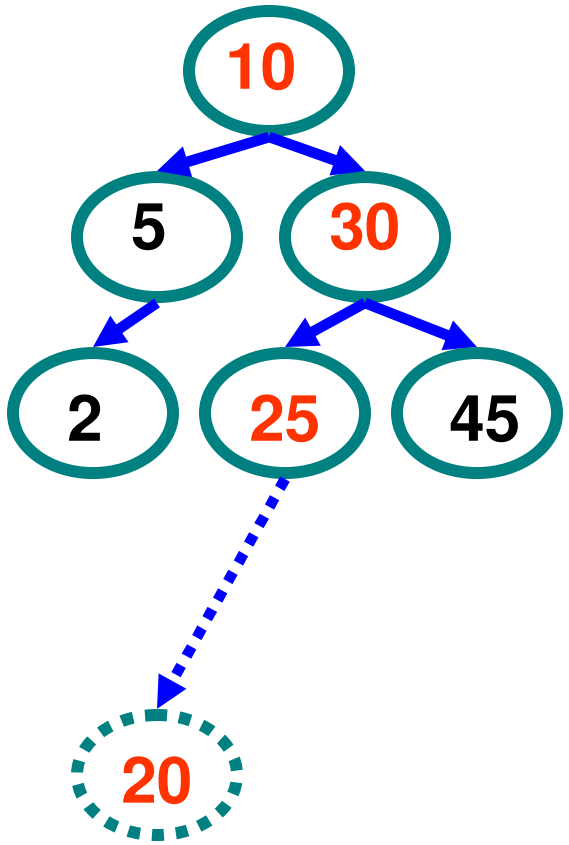
- How to build & maintain binary trees?
  - Insertion
  - Deletion
- Maintain key property (invariant)
  - Smaller values in left subtree
  - Larger values in right subtree

Binary Search Tree – Insertion

- **Algorithm**
  1. Perform search for value X
  2. Search will end at node Y (if X not in tree)
  3. If X < Y, insert new leaf X as new left subtree for Y
  4. If X > Y, insert new leaf X as new right subtree for Y
- **Observations**
  - O(log(n)) operation for balanced tree
  - Insertions may unbalance tree

Example Insertion

- **Insert (20)**



10 < 20, right

30 > 20, left

25 > 20, left

Insert 20 on left

Binary Search Tree – Deletion

- **Algorithm**
  1. Perform search for value X
  2. If X is a leaf, delete X
  3. Else // must delete internal node

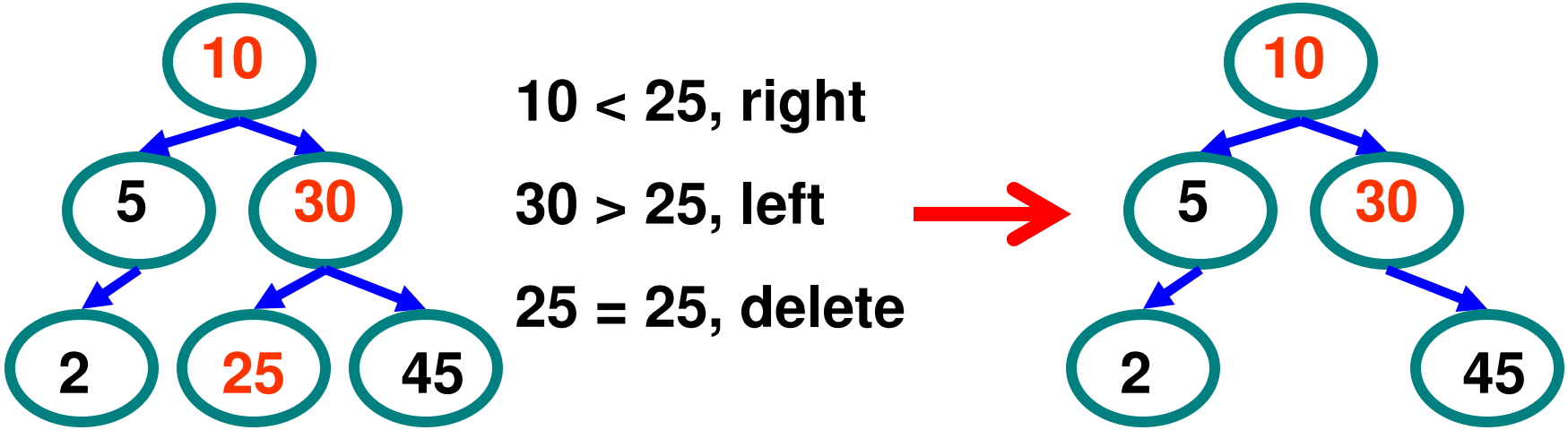
     a) Replace with largest value Y on left subtree

     OR smallest value Z on right subtree

     b) Delete replacement value (Y or Z) from subtree
- **Observation**
  - O(log(n)) operation for balanced tree
  - Deletions may unbalance tree

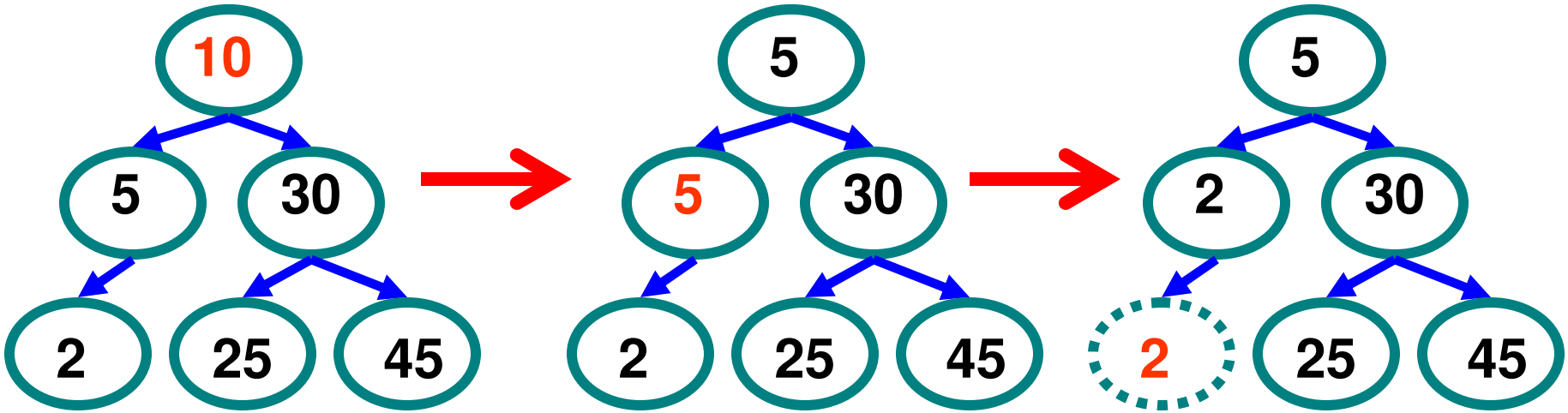
Example Deletion (Leaf)

- Delete (25)



Example Deletion (Internal Node)

- Delete (10)



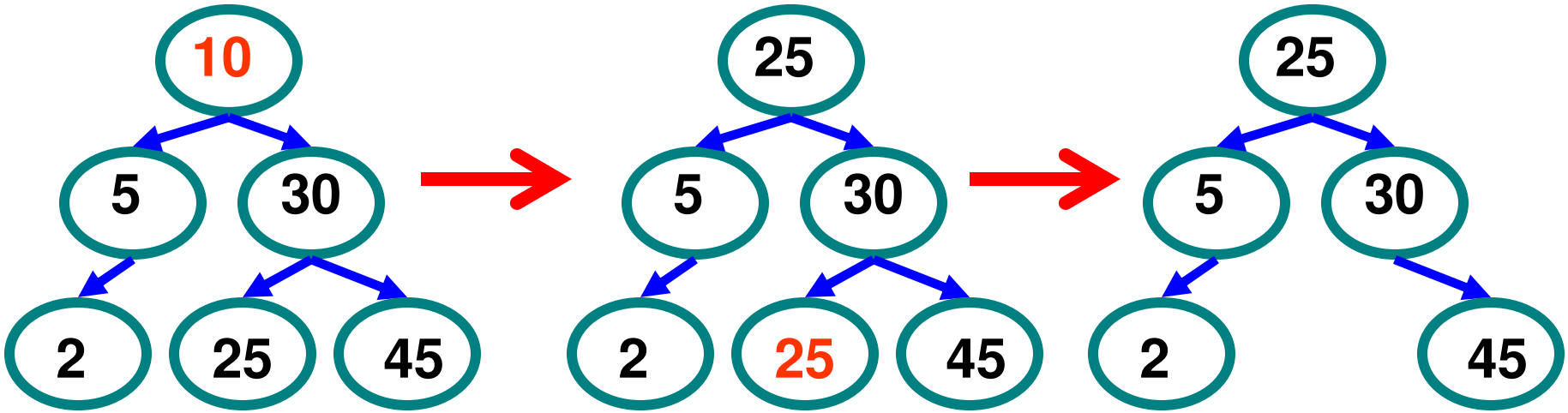
Replacing 10 with largest value in left subtree

Replacing 5 with largest value in left subtree

Deleting leaf

Example Deletion (Internal Node)

- Delete (10)



Replacing 10 with smallest value in right subtree

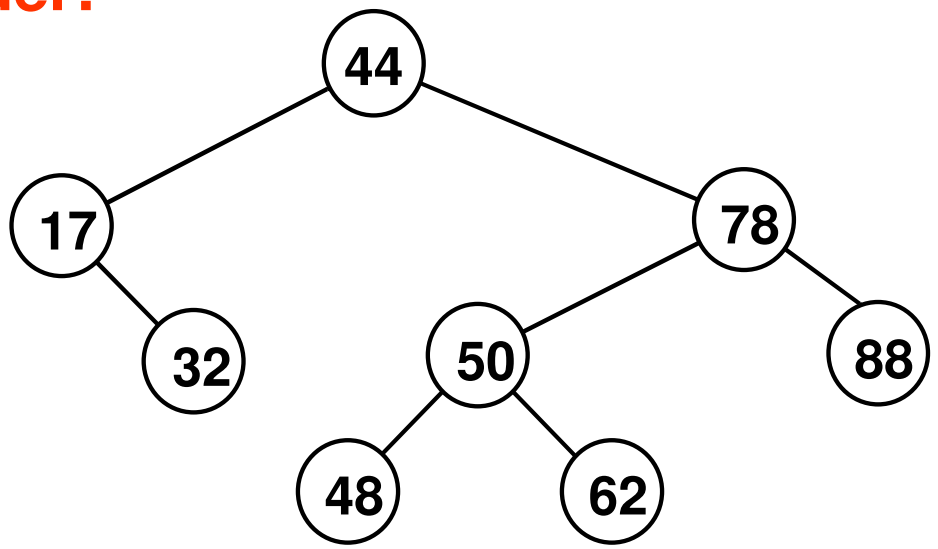
Deleting leaf

Resulting tree

BST Traversal Examples

- **Pre-order**
  - 44, 17, 32, 78, 50, 48, 62, 88
- **In-order**
  - 17, 32, 44, 48, 50, 62, 78, 88
- **Post-order**
  - 32, 17, 48, 62, 50, 88, 78, 44
- **Breadth-first**
  - 44, 17, 78, 32, 50, 88, 48, 62

Sorted order!



Binary search tree

Building Maps with Binary search Trees

- Books, etc., often talk about each node of a binary tree just containing a key
- Search trees are often used to implement maps
  - Each non-empty node contains a key, a value, and left and right subtrees
- What is the generic type
  <K extends Comparable<K>>
- Denotes any type K that can be compared to K's
  - e.g., Strings can be comparedTo'd Strings, but Strings cannot be compareTo'd Integer

Polymorphic Binary Search Trees

- What do we mean by polymorphic?
  - Implement two subtypes of Tree: EmptyTree and NonEmptyTree
  - We use EmptyTree, rather than null to represent the empty tree
  - Invoke methods on tree nodes, get empty or nonempty functionality

Recursive, Polymorphic Linked Lists

- Let's see code for linked lists that uses the same concept used for polymorphic trees
- Keys are kept in sorted order
- Have an interface List, and subtypes EmptyList and NonEmptyList
- See code distribution

compareTo

- if a.compareTo(b) = 0, then a == b
- if a.compareTo(b) < 0, then a < b
- if a.compareTo(b) > 0, then a > b