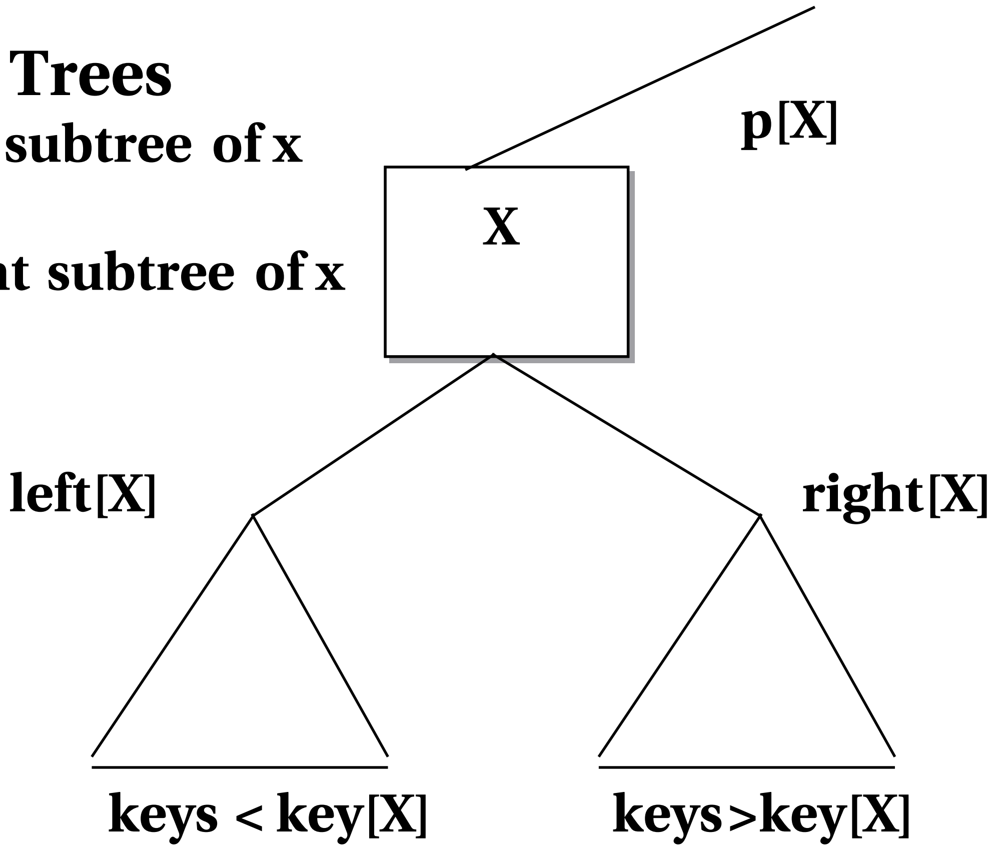
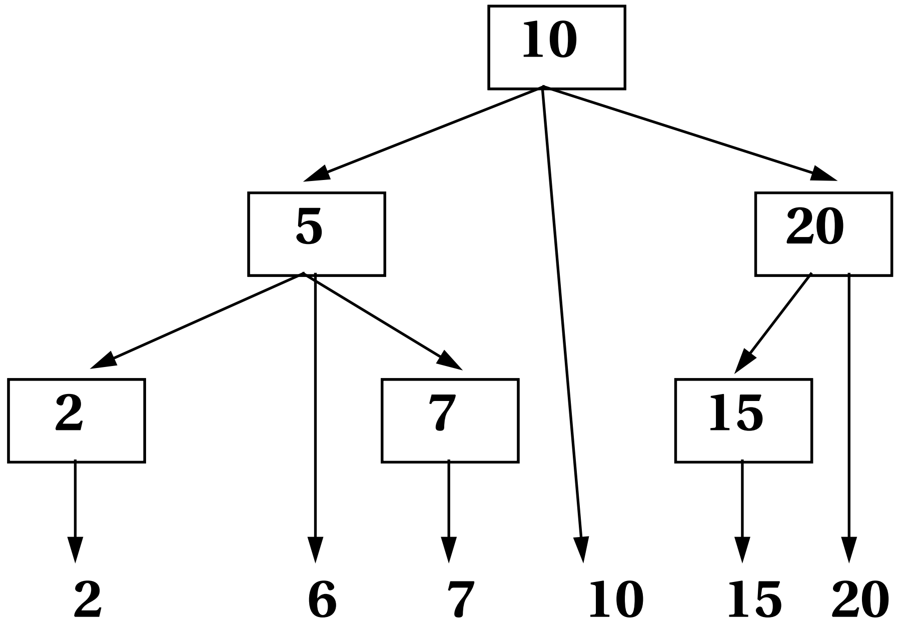
Binary Search Trees

- Subclass of Binary Trees
- for all nodes in the left subtree of x
  - key[y] < key[x]
- for all nodes in the right subtree of x
  - key[y] > key[x]

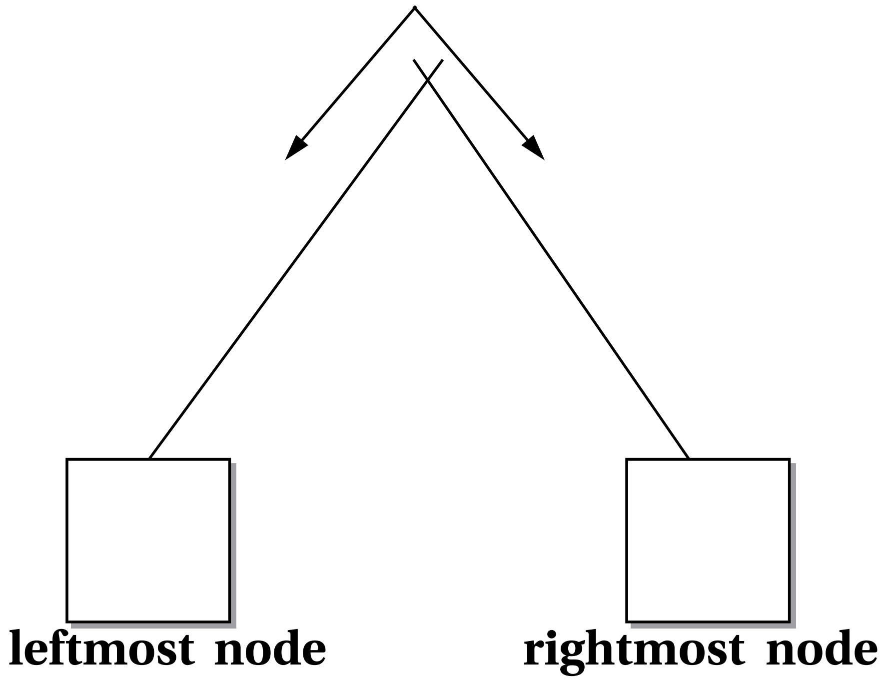


Example

- vertical projection -> sorted set of keys
- (inorder traversal)



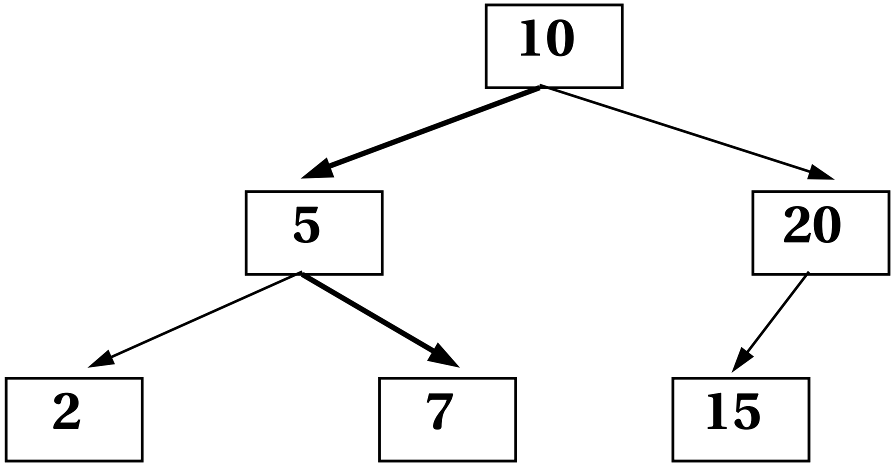
Find{Min|Max}



Search

Search for 6

Search for 7

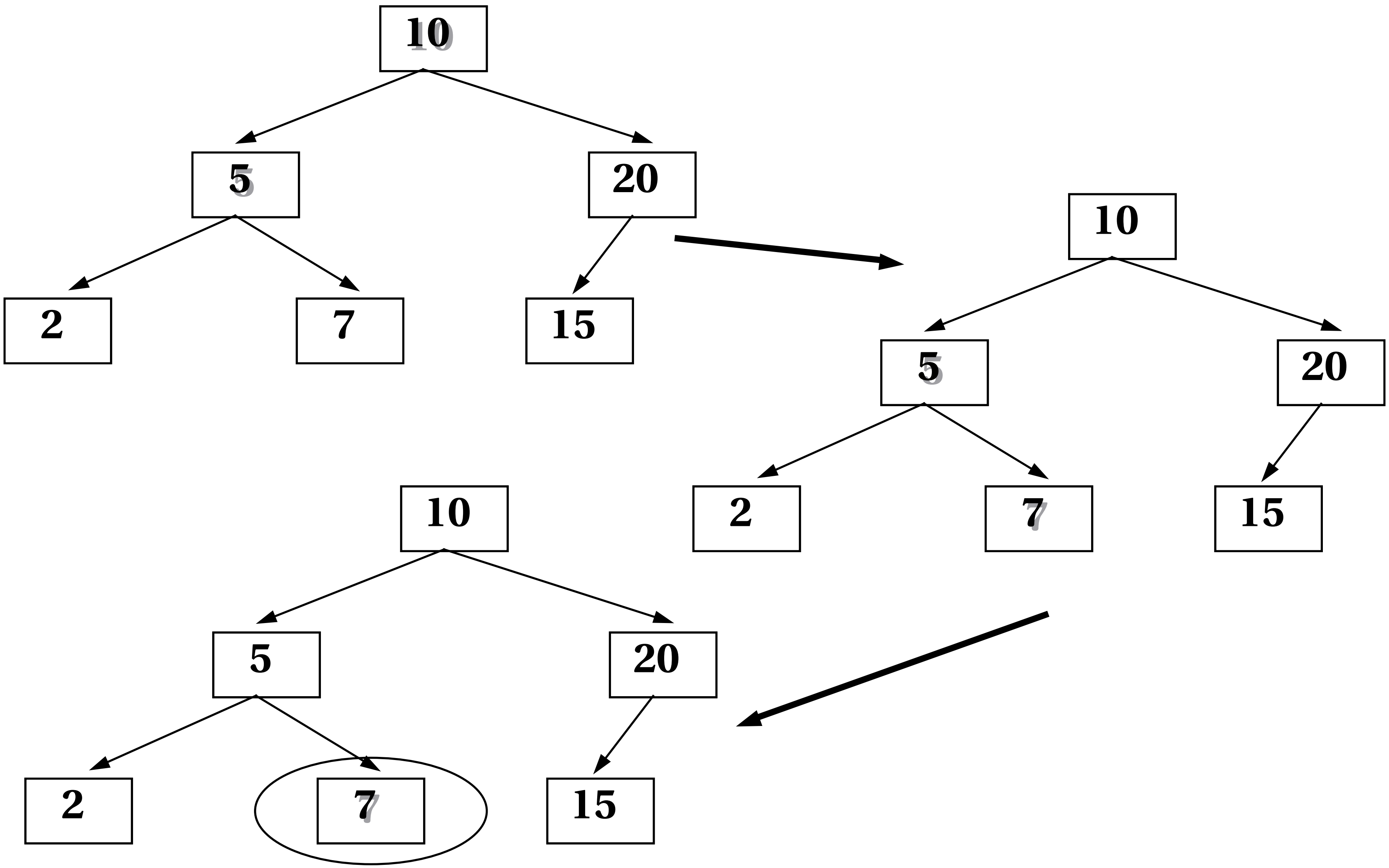


Search

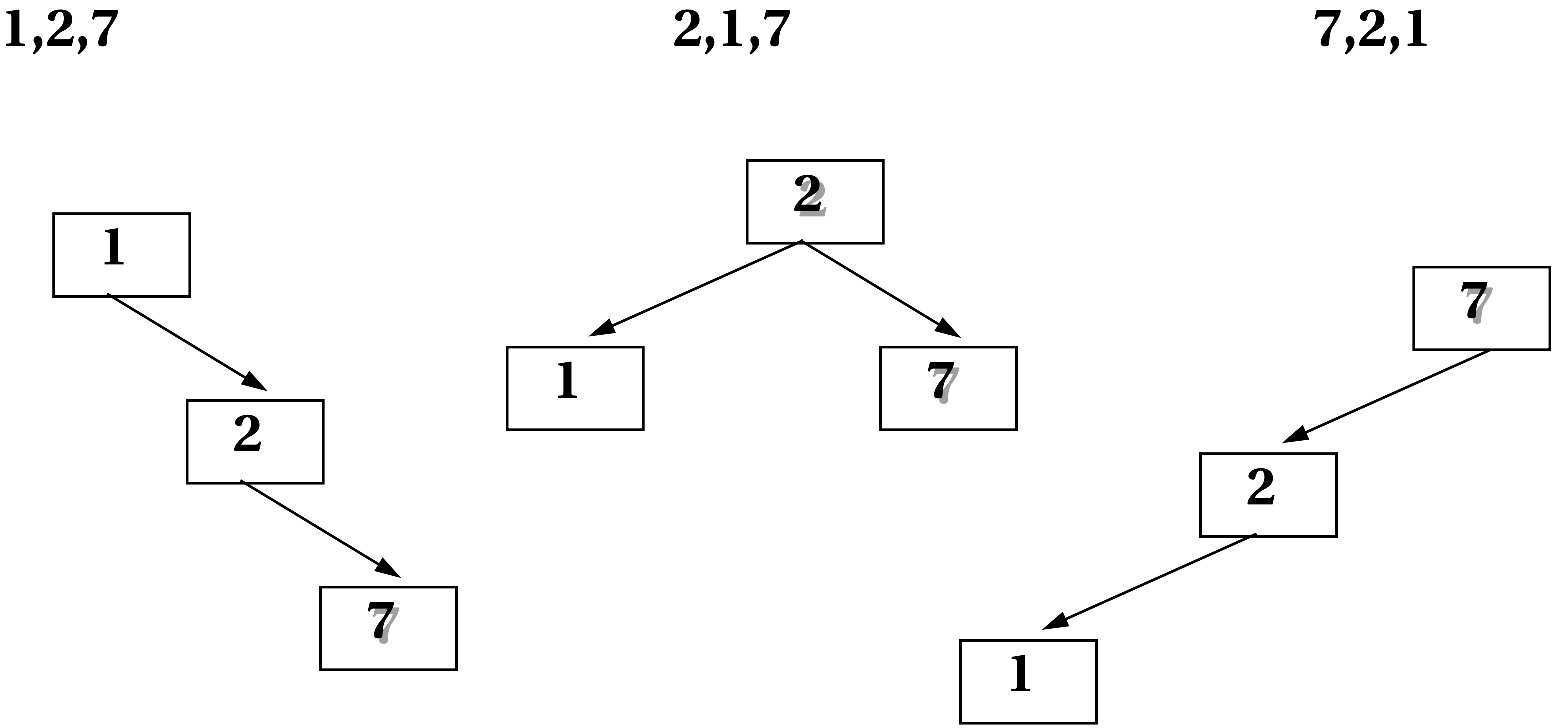
```
// Stop Condition
while (R != NULL)
 {
   P = R;
   // Advance
   if (R->key > item)
      R = R->left;
   else if (R->key < item)
      R = R->right;
   else
      return true;

 }
return NULL;
```

Insert 6



Insert

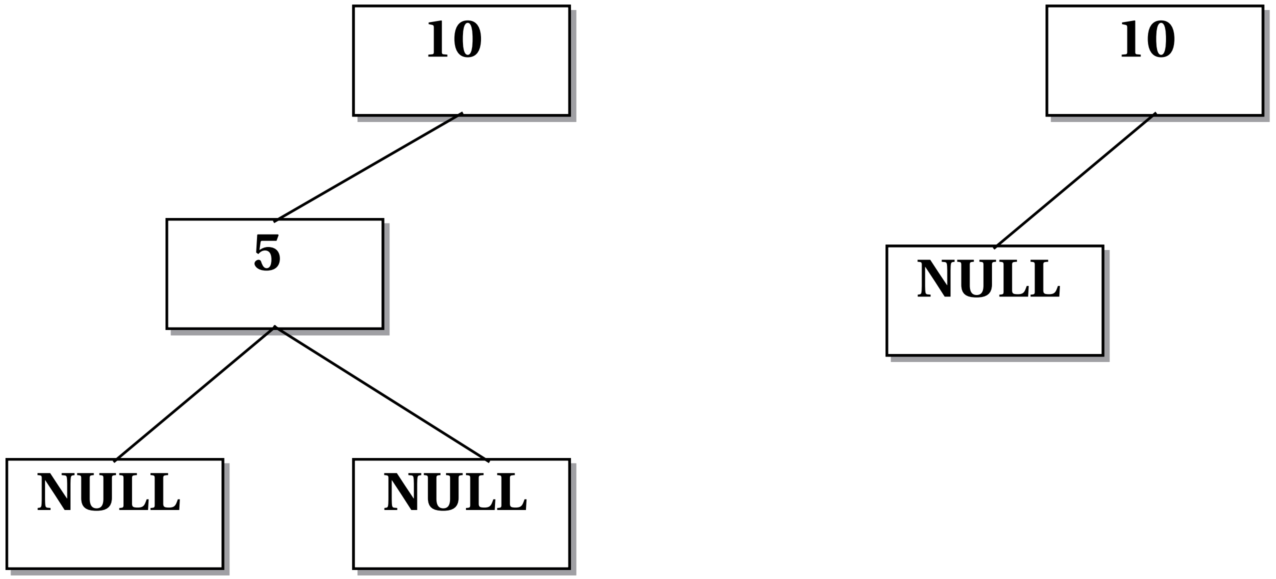


Insert - Implementation

```
template <class T>
bool BSTree<T>::Insert (TreeNode<T>*& R, T& item) {
 TreeNode<T>* P;
 if (R == NULL)
    R = new TreeNode<T> (item, NULL, NULL);
 else
   {
    // Find position
    if (Search (R, P, item) == true)
      return false;                    // key is already inserted
    // Replace
    if (P->key < item)
      P->right = new TreeNode<T> (item, P->right, NULL);
    else
      P->left = new TreeNode<T> (item, P->left, NULL);
   }
 return true;
}
```

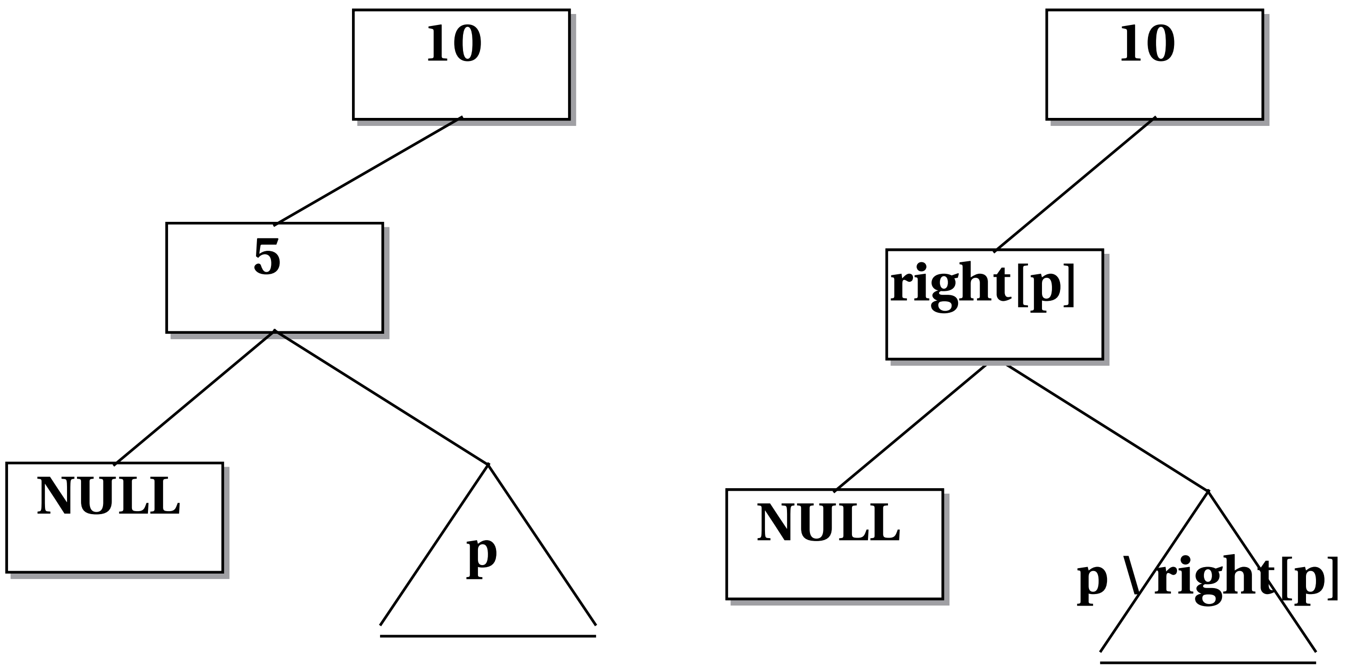
Delete - 1

- A node with no children



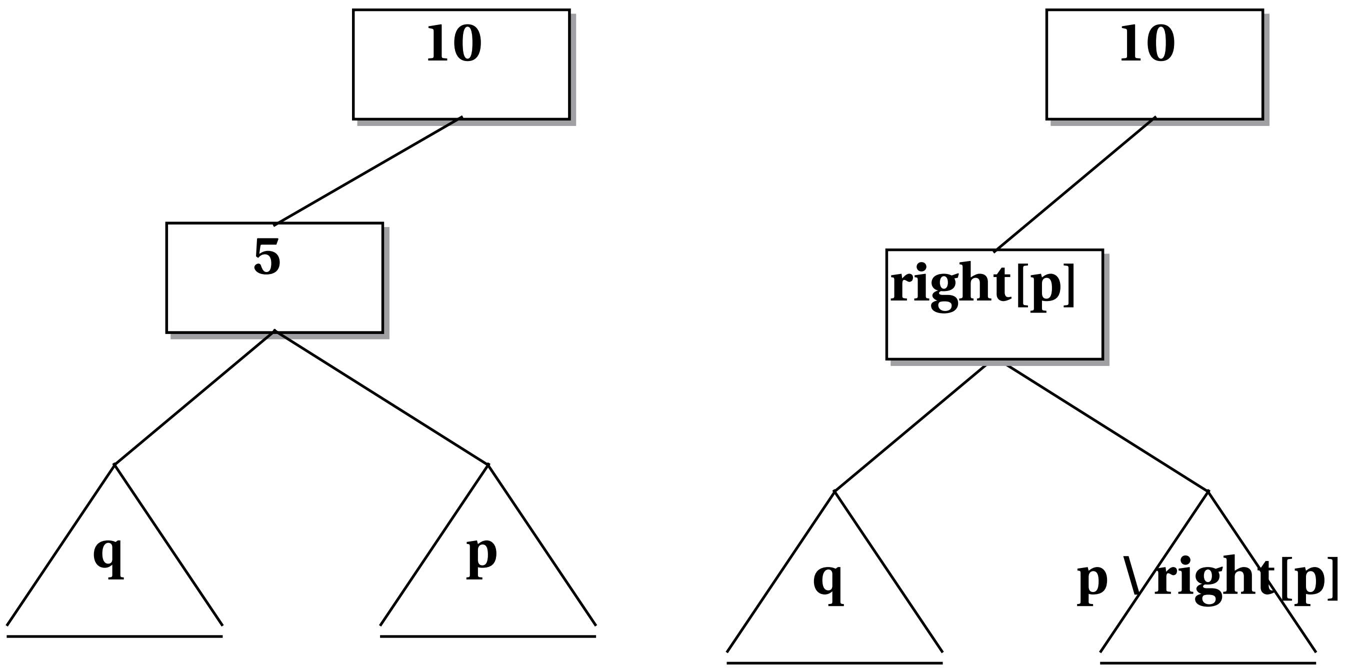
Delete - 2

- a node with one child



Delete - 3

- a node with 2 children



Efficiency

- **Maximum and Minimum size of a binary search tree**

	Average	Worst
Retrieval:	**O(log(N))**	**O(N)**
Insertion:	**O (log(N))**	**O(N)**
Deletion:	**O (log(N))**	**O (N)**
Traversal:	**O(N)**	**O(N)**

Applications: TreeSort

- create a Binary Search Tree ADT T
- Insert elements into T (O(N*log(N)) --- O(N^2))
- Traverse T in inorder (O(N))

Applications: Dictionary

- collection of items, similar to a bag
- each item has a string attached - item's key