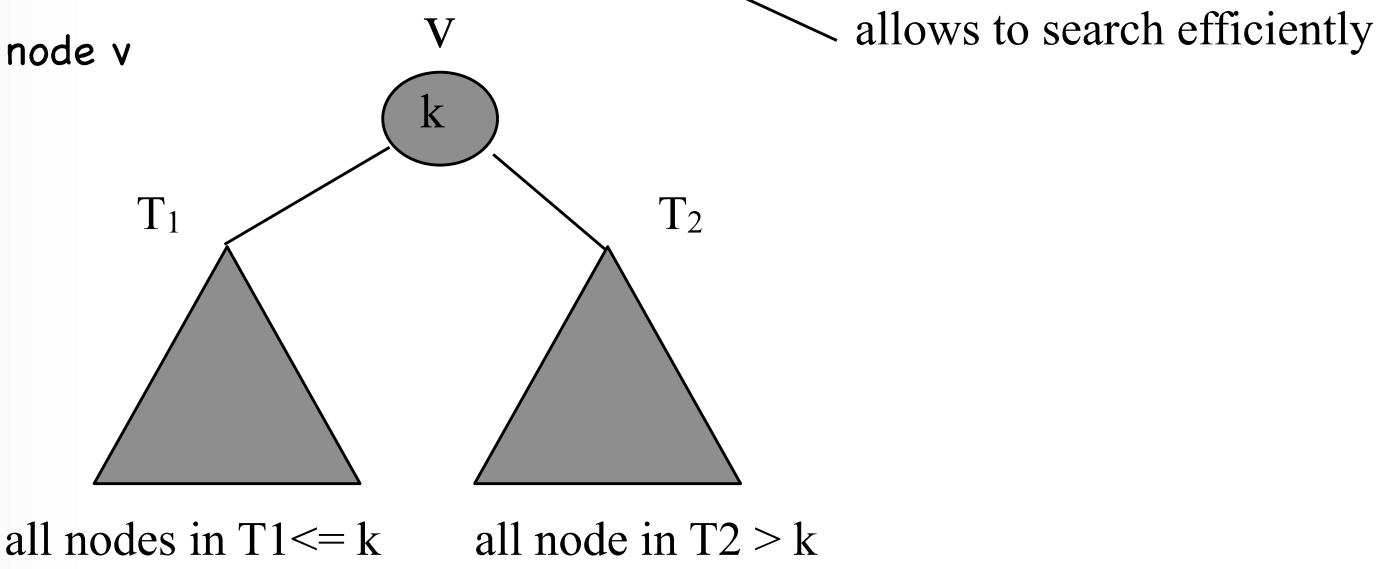
Binary Search Trees (BST)

- Motivation:
  - want a structure that can search fast
  - arrays: search fast, updates slow
  - linked lists: search slow, updates fast
- Intuition:
  - tree combines the advantages of arrays and linked lists
- Definition:
  - a BST is a binary tree with the following "search" property

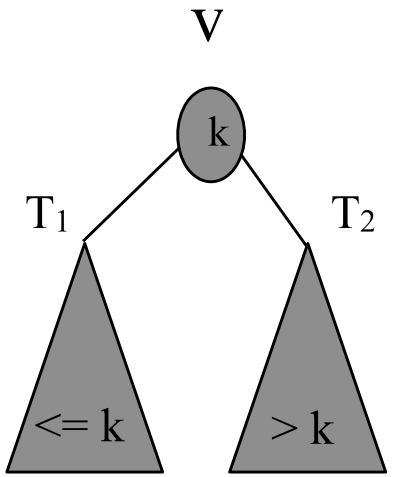
allows to search efficiently

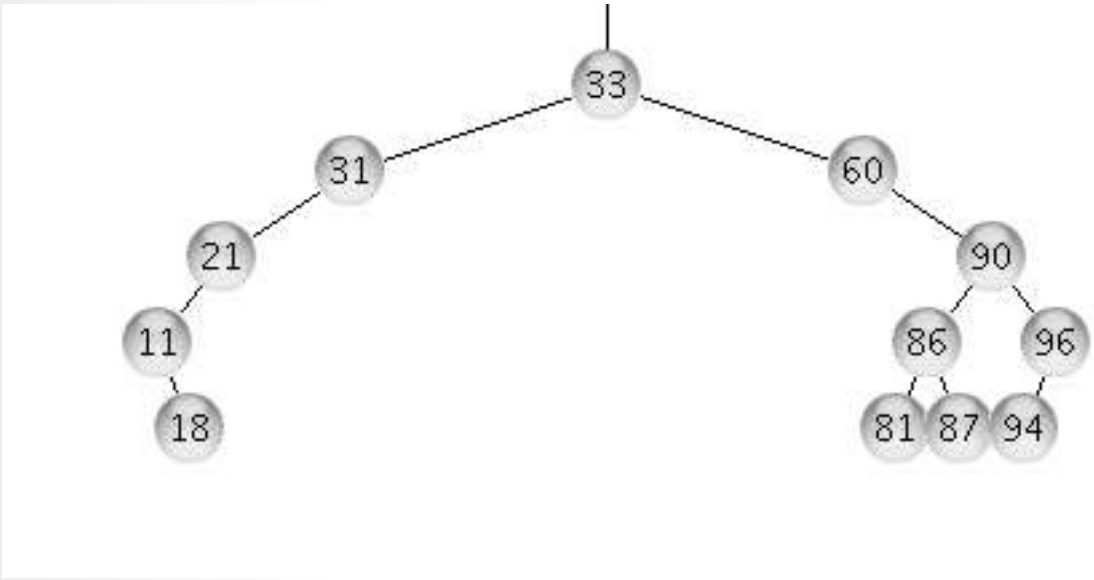
- for any node v



BST

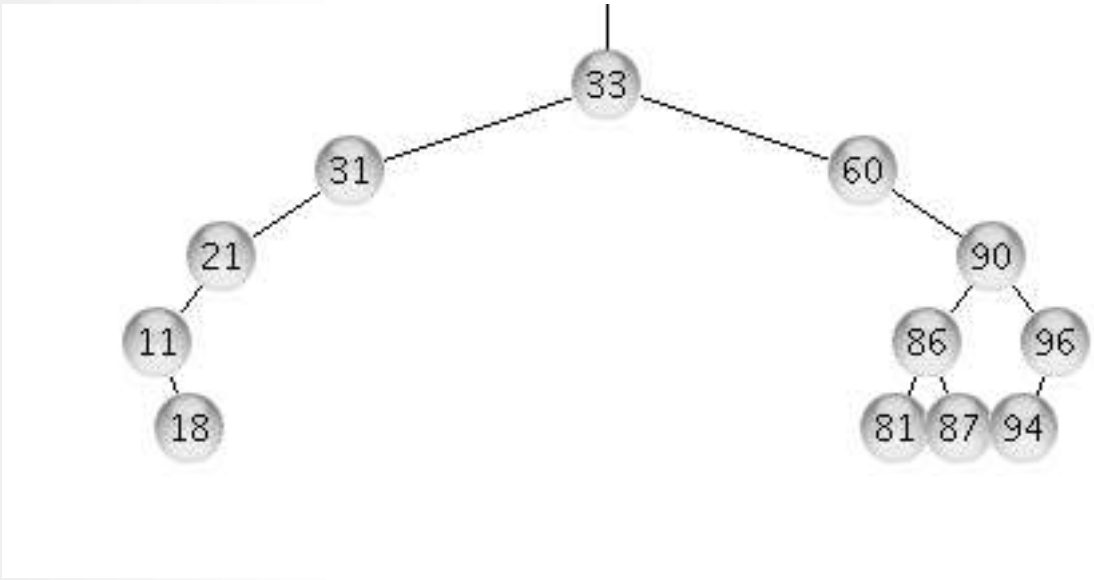
- Example





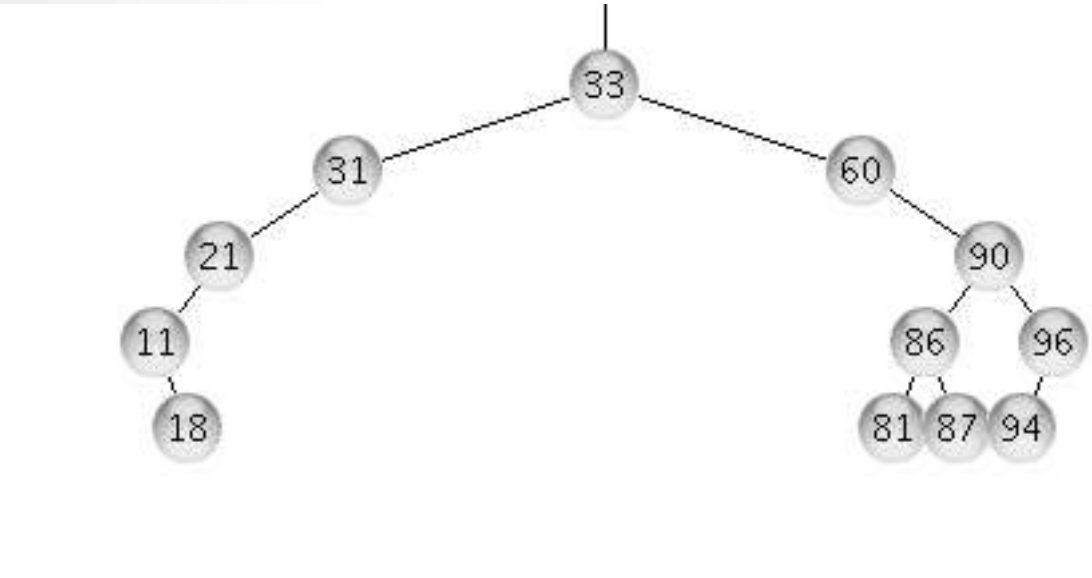
Sorting a BST

- Print the elements in the BST in sorted order



Sorting a BST

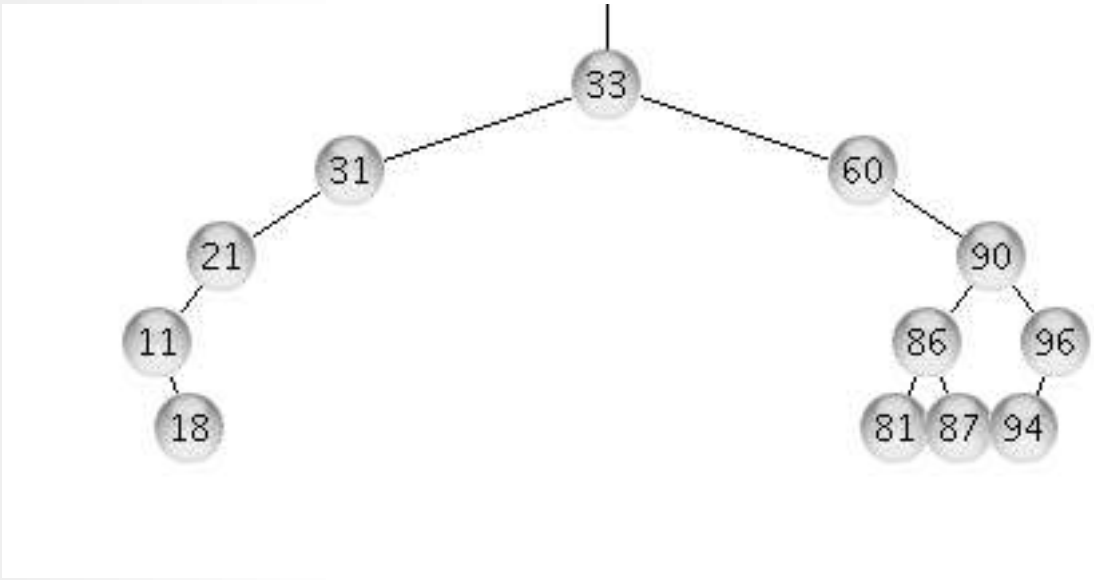
- Print the elements in the BST in sorted order.




```
//print the elements in tree of v in order
sort(BSTNode v)
    if  (v == null) return;
    sort(v.left());
    print v.getData();
    sort(v.right());
```

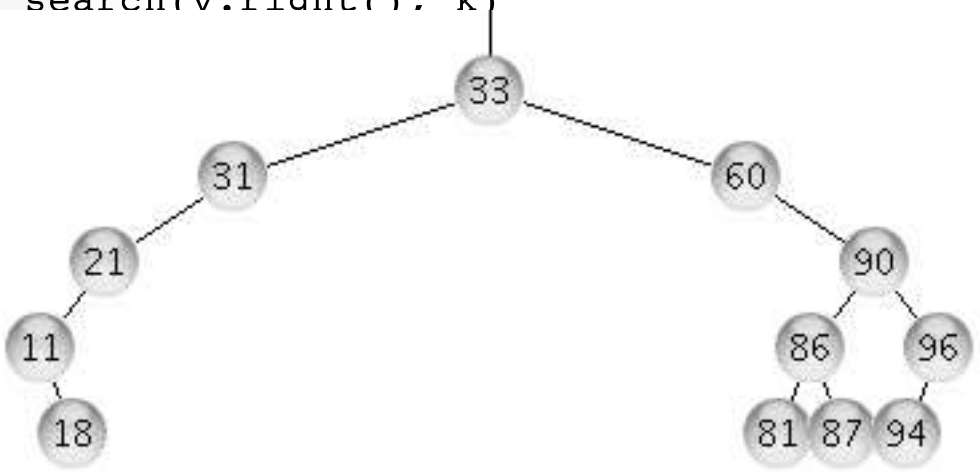
- in-order traversal: left -node-right
- Analysis: O(n)

Searching in a BST



Searching in a BST

```
//return the node w such that w.getData() == k or null if such a node
//does not exist
BSTNode search (v, k)   {
    if (v == null) return null;
    if (v.getData() == k) return v;
    if (k < v.getData()) return search(v.left(), k);
    else return search(v.right(), k)
}
```



- Analysis:
  - search traverses (only) a path down from the root
  - does NOT traverse the entire tree
  - O(depth of result node) = O(h), where h is the height of the tree