Lesson 10
Binary Search Trees:

Do Less and Accomplish More

Wholeness Statement

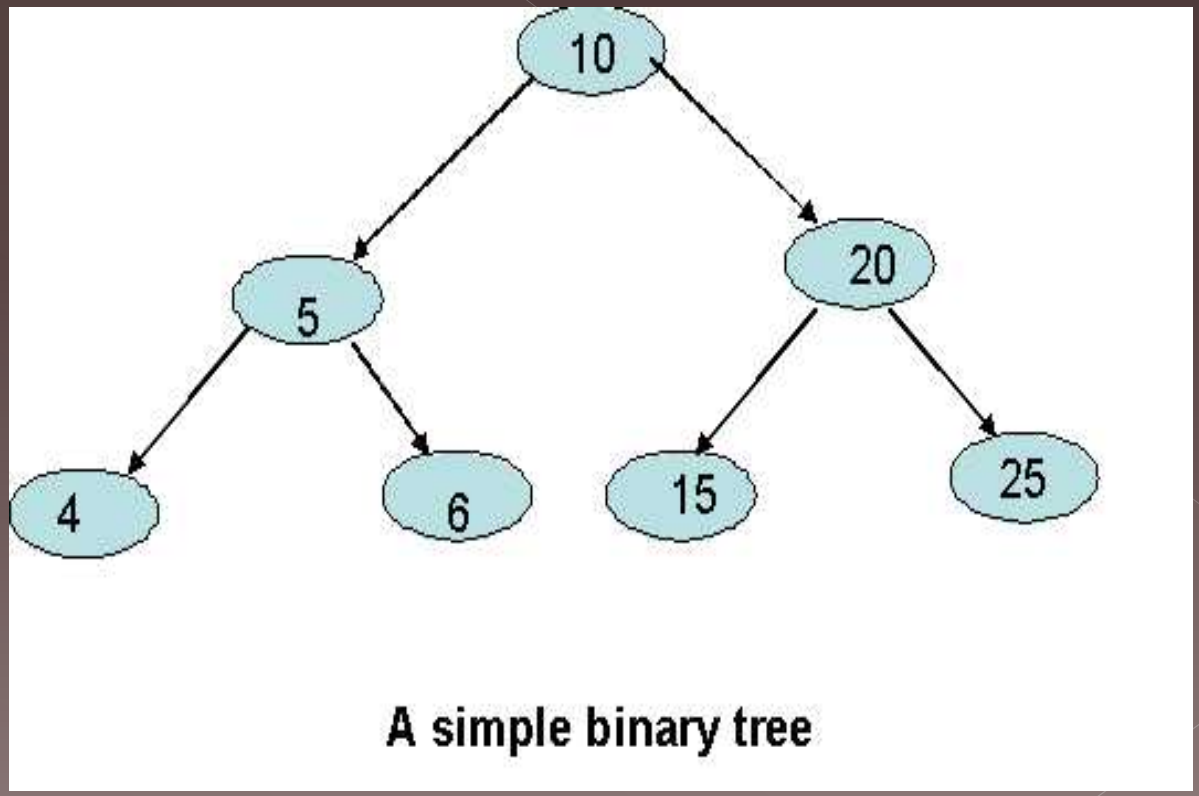
A binary search tree is an important data structure that provides a highly flexible perspective on a set of comparable objects. The whole range of space and time is open to an individual with fully developed awareness.

Set Interface

- *You can create a set using one of its three concrete classes:* **HashSet**, **LinkedHashSet**, *or* **TreeSet**.
- The concrete classes that implement **Set** must ensure that no duplicate elements can be added to the set.

Binary Trees

A list is a linear structure that consists of a sequence of elements. A binary tree is a hierarchical structure. It is either empty or consists of an element, called the *root*, and two distinct binary trees, called the *left subtree* and *right subtree*.



A simple binary tree

Binary Tree Terms

- A node without children is called a *leaf*.
- A special type of binary tree called a *binary search tree* is often useful.
- Keeping data stored in a sorted order is a way to optimize searches on the data
- A *binary search tree* (BST) is a binary tree with the following property
  - *At each node N, every value in the left sub tree of N is less than the value at N, and every value in the right sub tree of N is greater than the value at N.*
  - http://www.cs.armstrong.edu/liang/animation/

Binary Search Tree Operations

- Create an empty tree
  - new object constructor call
- Insert a new item
  - insert
- Delete the item with a given search key
  - remove
- Retrieve the item with a given search key
  - find
- Traverse the tree in preorder
  - preorderTraverse
- Traverse the tree inorder
  - inorderTraverse
- Traverse the tree in postorder
  - postorderTraverse

Advantages

- perform insertions and deletions faster than that can be done on Linked Lists
- perform any find with the same efficiency as a binary search on a sorted array
- keep all data in sorted order (eliminate the need to sort)

Algorithm for Insertion

- Start at the root of the BST
- If the node is null, create a new node with value x and attach it
- If x is less than the value in the node, recursively insert in the left subtree
- If x is greater than the value in the node, recursively insert in the right subtree

Algorithm for Searching

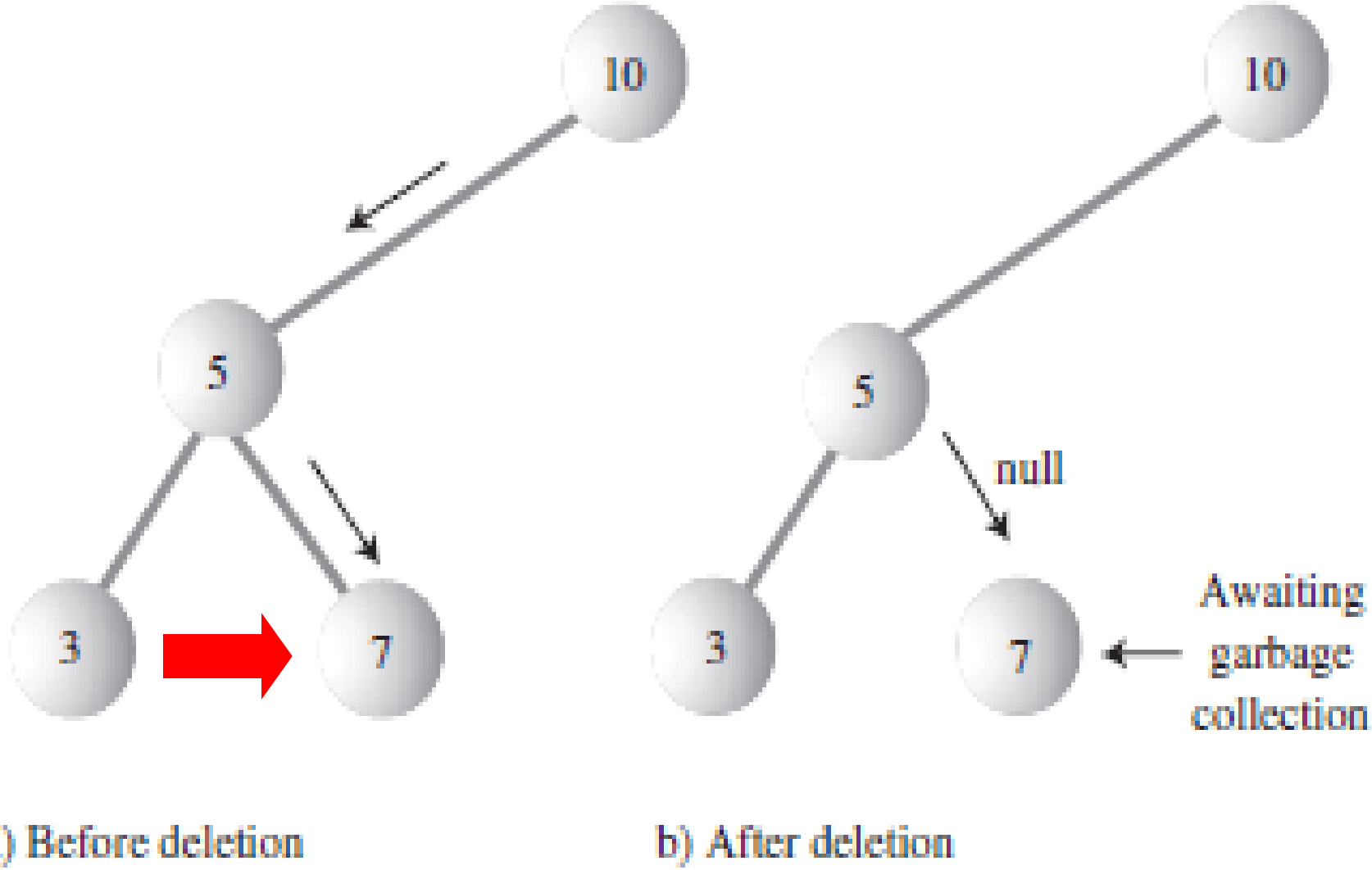
- Start at the root of the BST
- If the node is null, return false
- If x is the search value of the node, return true
- If x is less than the value in the node, recursively search the left subtree
- If x is greater than the value in the node, recursively search the right subtree

Algorithm for Removing Items

- Goal: remove the node containing the target value
  - Finding this node is done through a search
- Three cases:
  - Node to remove is a leaf node (both children are null)
  - Node to remove has one child
  - Node to remove has two children

Deletion cases: Leaf Node(Case-1)

- To delete a leaf node, simply change the appropriate child field in the node's parent to point to *null,* instead of to the node.
- The node still exists, but is no longer a part of the tree.
- Because of Java's garbage collection feature, the node need not be deleted explicitly.



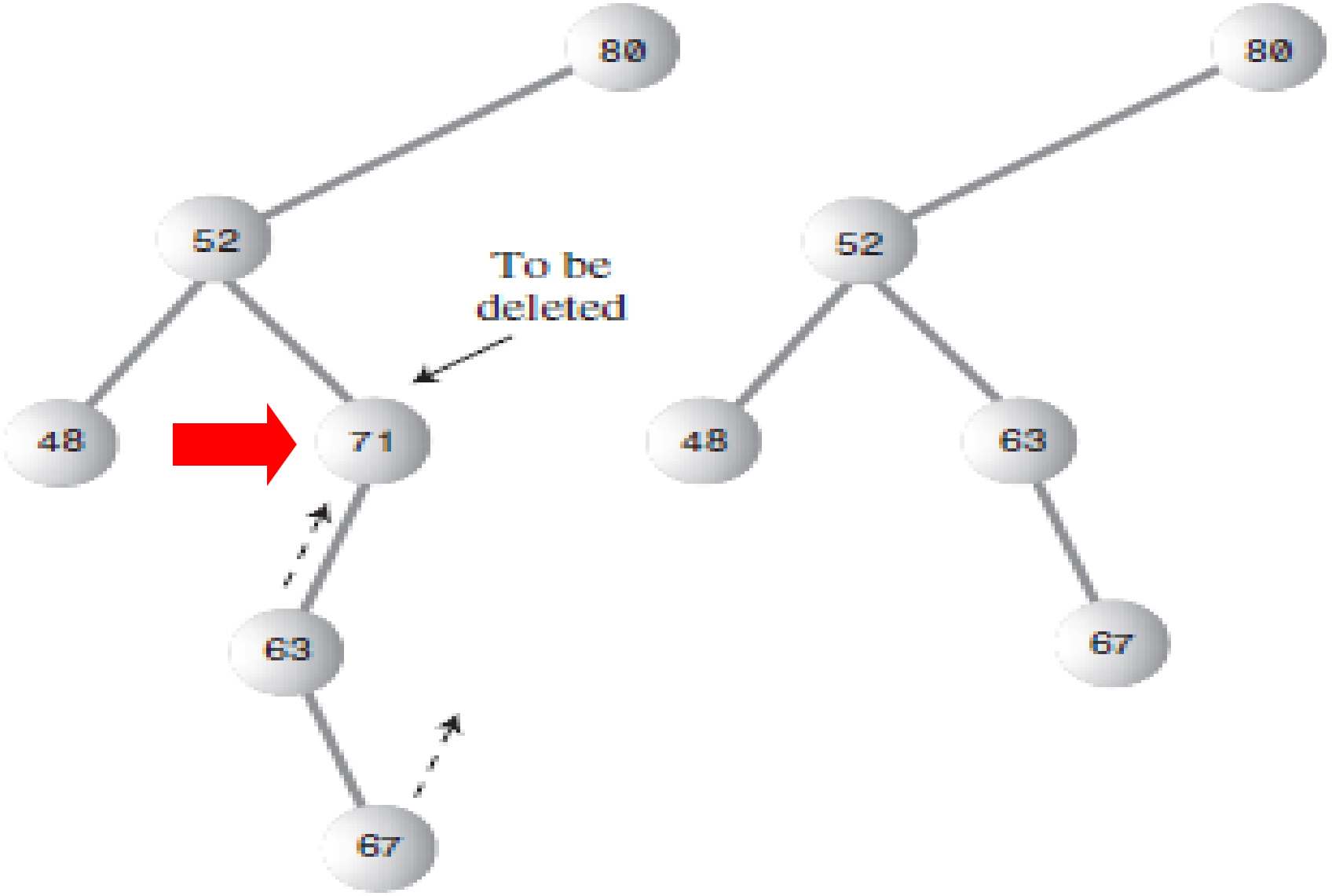
a) Before deletion

b) After deletion

Deleting a node with no children.

Deletion: One Child(Case-2)

- The node to be deleted in this case has only two connections: to its parent and to its only child.
- Connect the child of the node to the node's parent, thus cutting off the connection between the node and its child, and between the node and its parent.

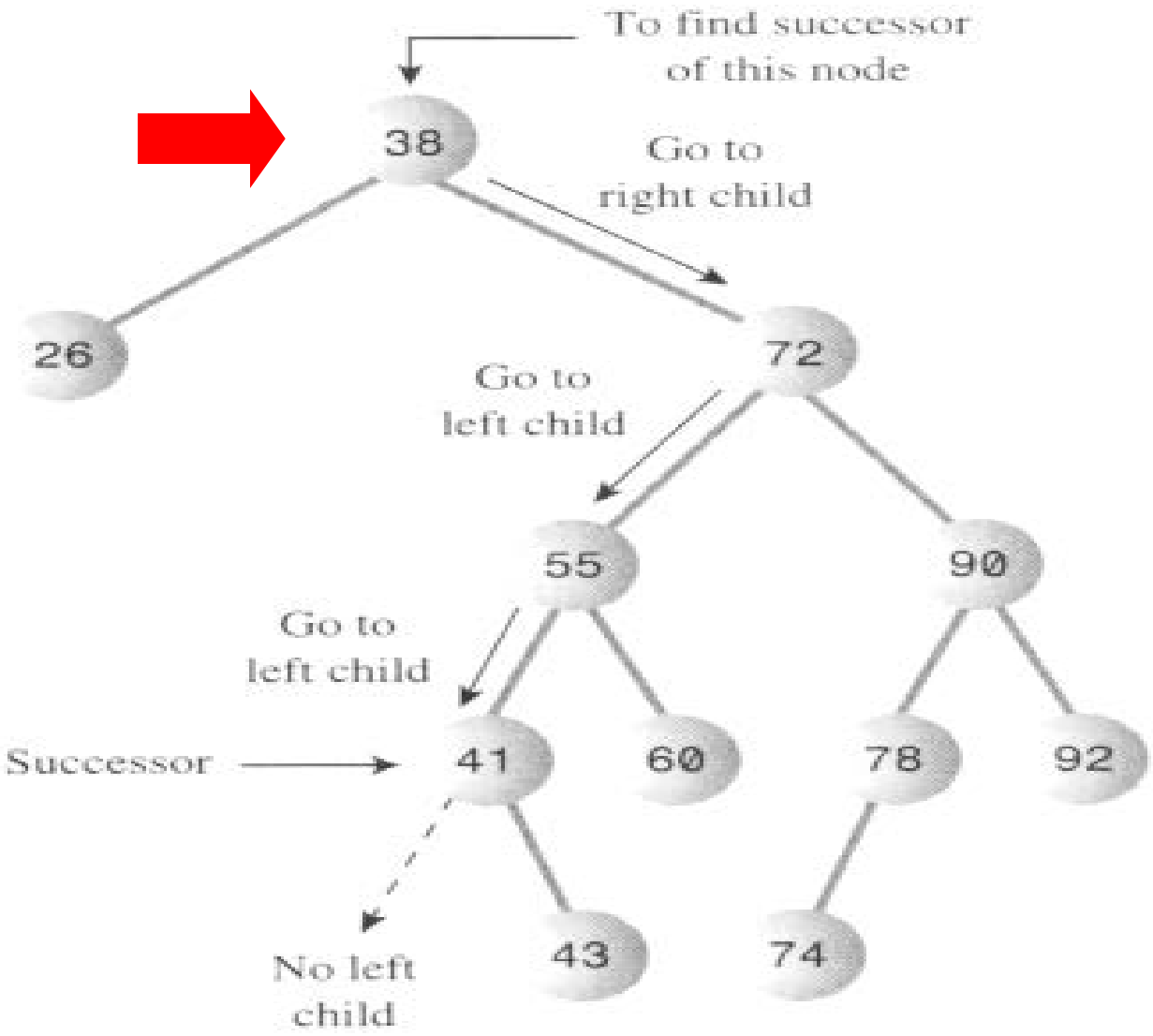


Deleting a node with one child.

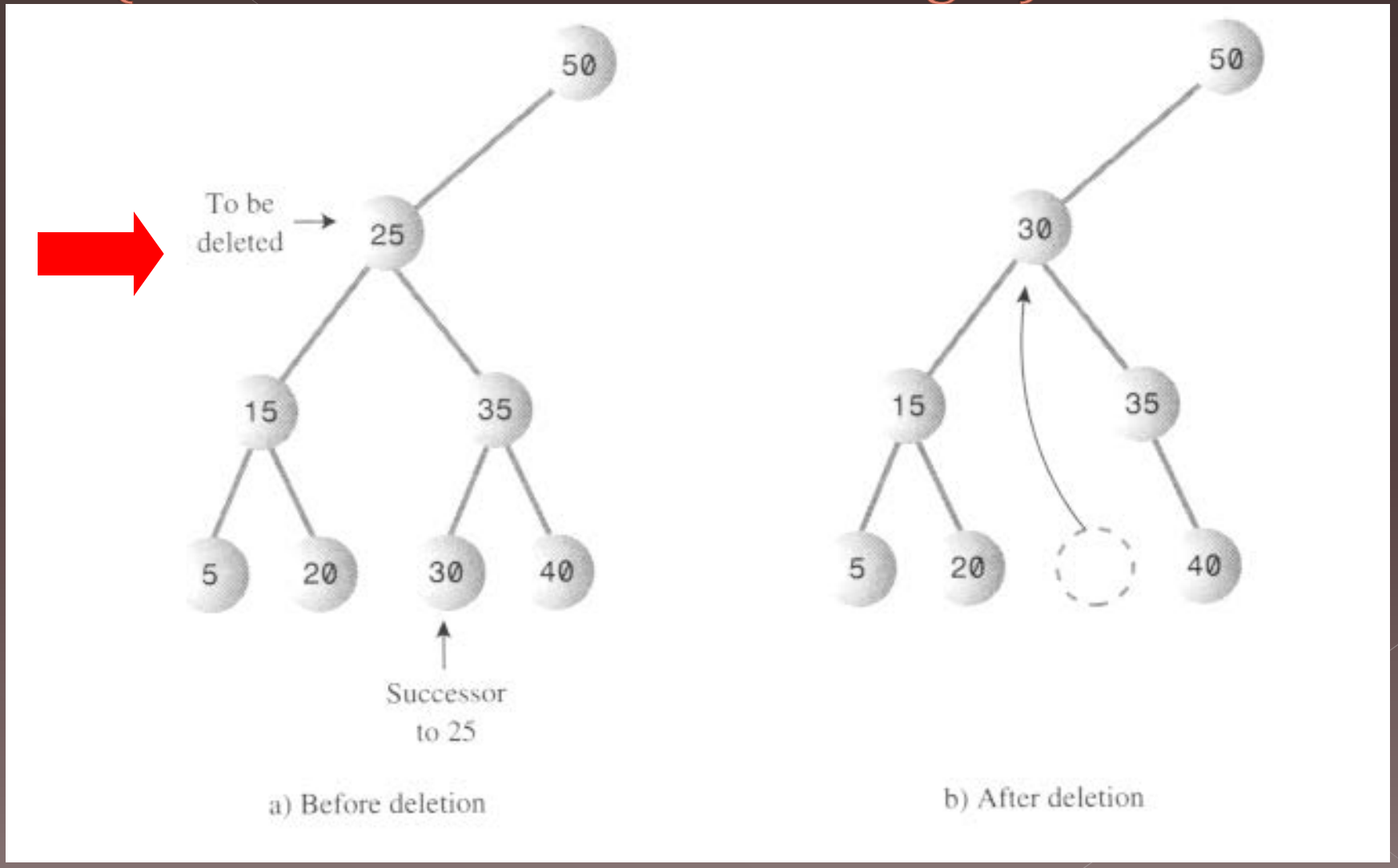
Deletion: Two Children(Case-3)

- To delete a node with two children, replace the node with its inorder successor.
- For each node, the node with the next-highest key (to the deleted node) in the subtree is called its inorder successor.
- To find the successor,
  - start with the original (deleted) node's right child.
  - Then go to this node's left child and then to its left child and so on, following down the path of left children.
  - The last left child in this path is the successor of the original node.

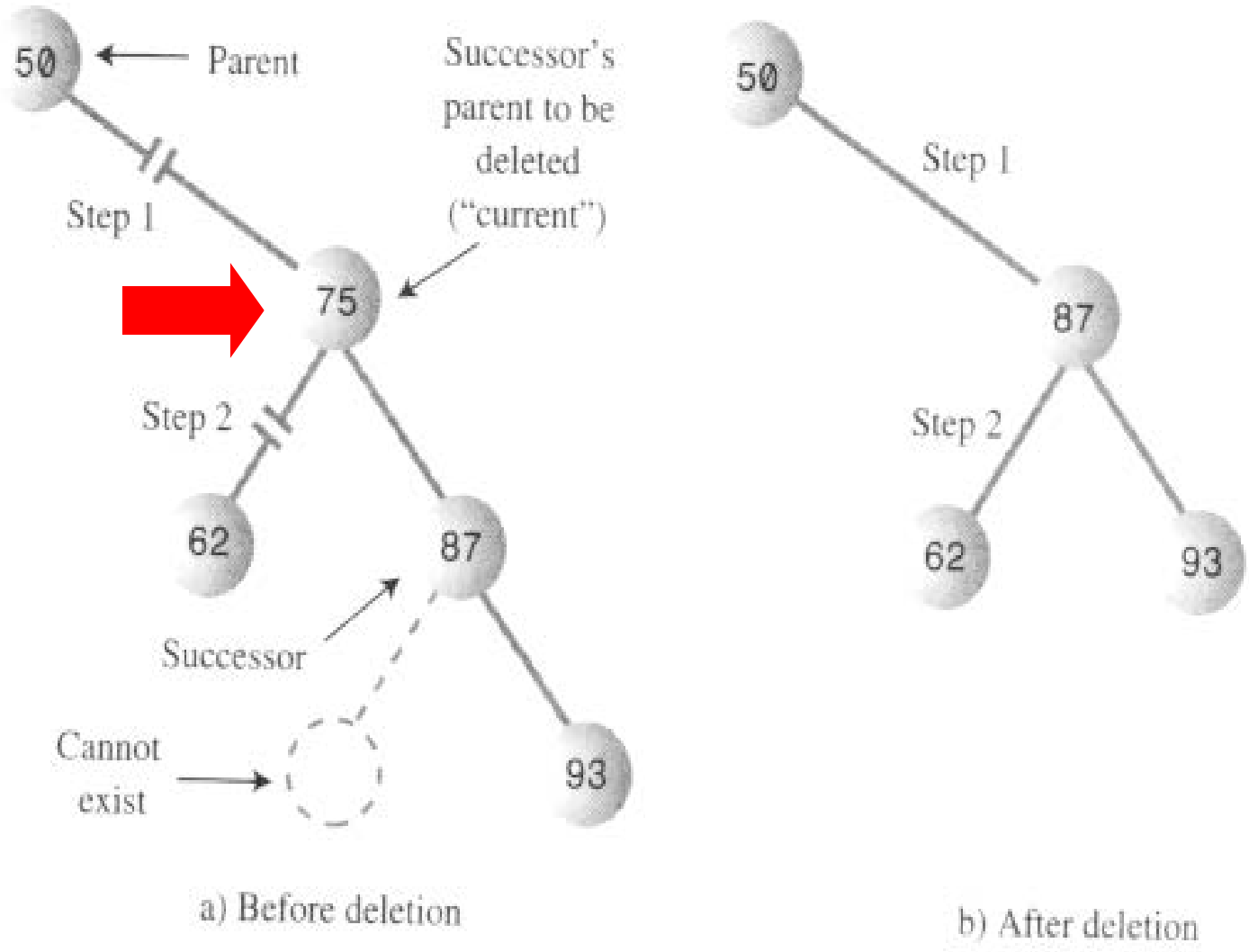
Find successor



Delete a node with subtree (case 3a) (Successor has no left and right)

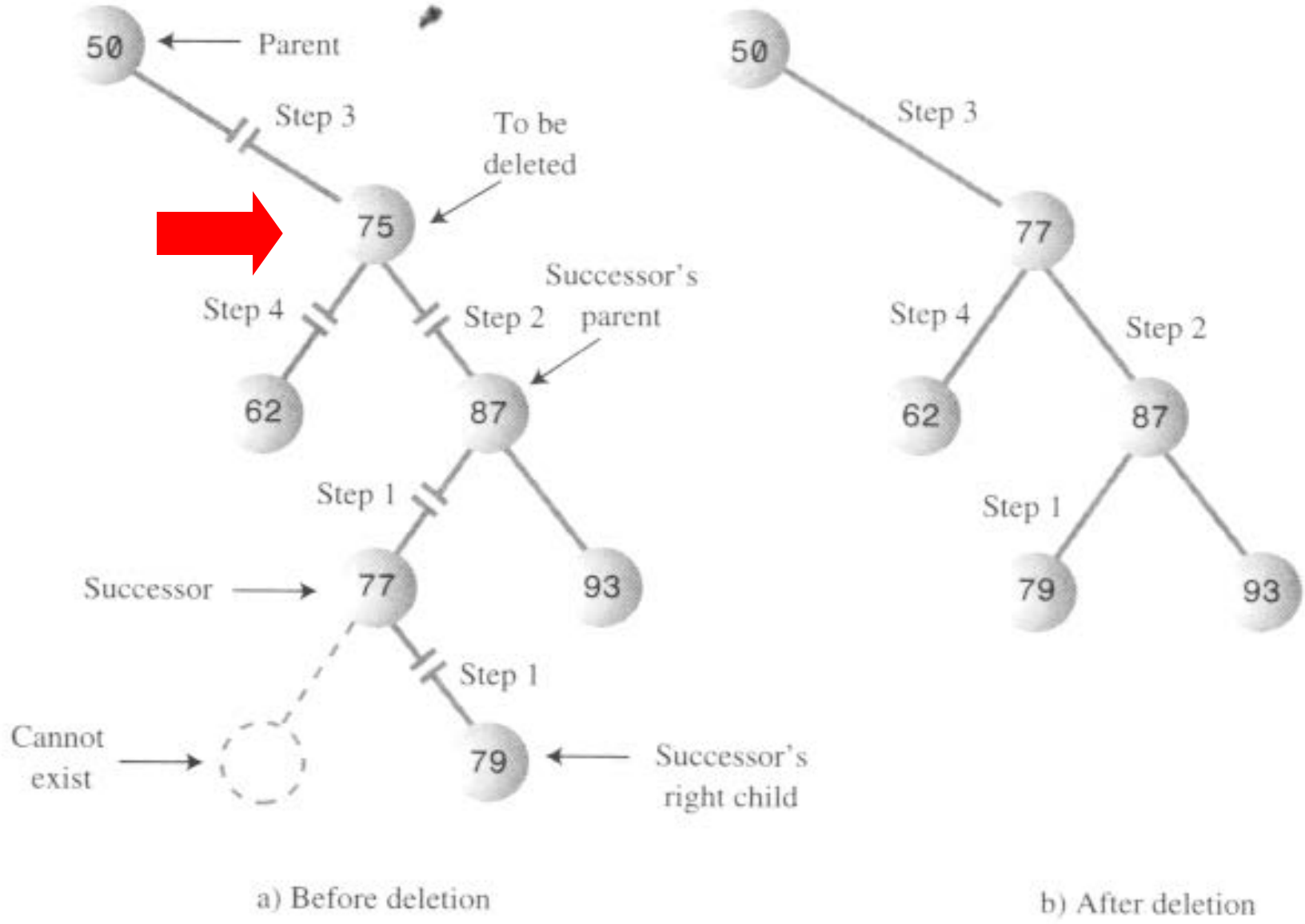


Delete a node with subtree (case 3b) Successor is the Right child of delnode



Delete a node with subtree (case 3c)

Successor is left Decendent of right child of delnode



Traversals of Binary Trees

- Preorder – Visit, Left, Right
  - Each node is visited before any of its children
- Inorder - Left, Visit, Right
  - Each node is visited after all the nodes of its left subtree and before the nodes of its right subtree
- Postorder - Left, Right, Visit
  - Each node is visited after all the nodes of its children are visited, i.e., after the nodes in the left subtree followed by the nodes of its right subtree are visited

Using BSTs For Sorting

- The following is an algorithm for sorting a list of Integers:
  - Insert them into a BST
  - Do an inorder traversal of the BST to get the sorted list.)

Main Point 1

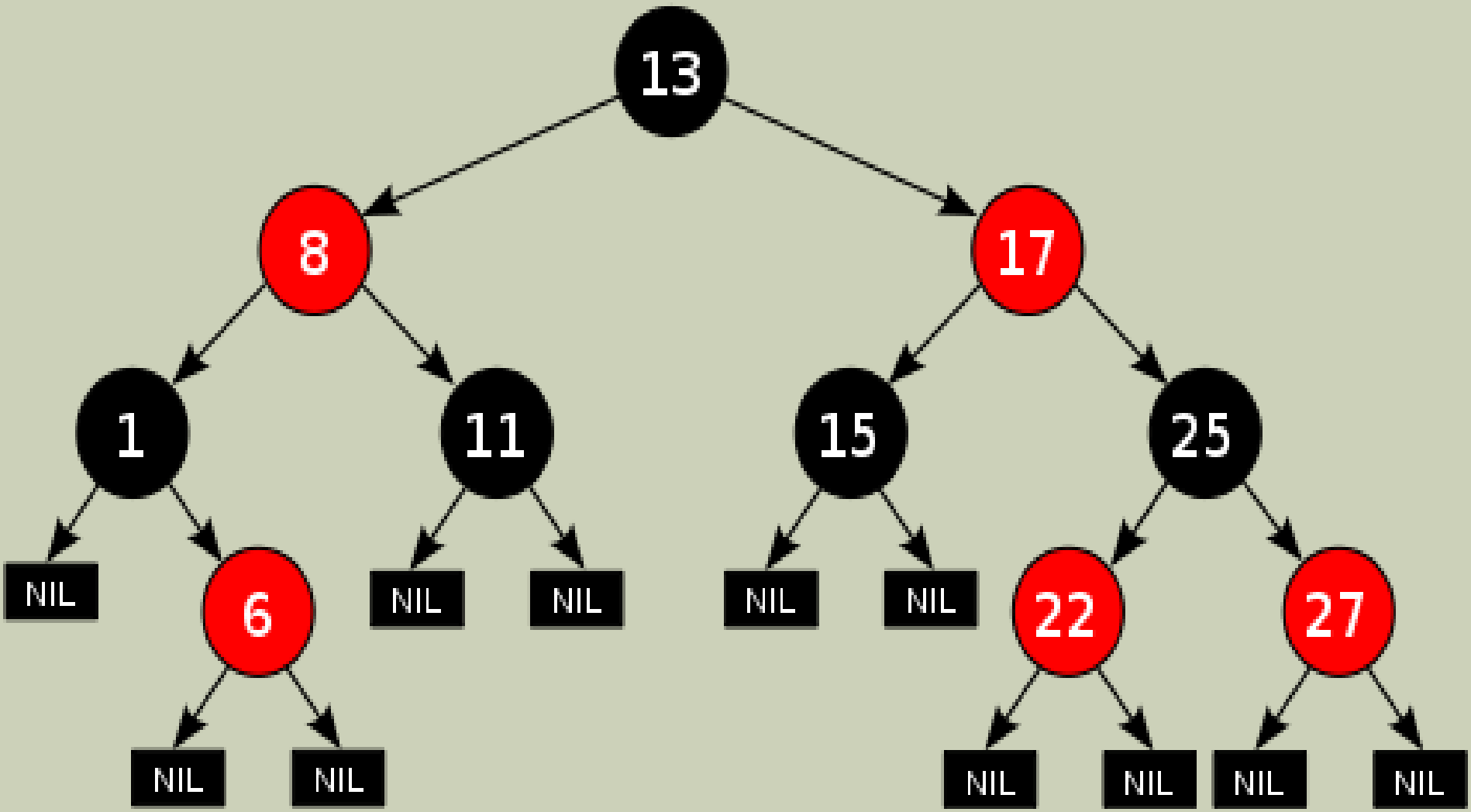
1. A binary search tree is a binary tree with the property that the value at each node is greater than the values in the nodes in its left subtree (child) and less than the values in the nodes in its right subtree.

   A binary search tree is an example of the SCI principle of diving because if the structure is right, the operations (search, insert, and remove) are accomplished with maximum efficiency.

BST from Collection Framework

- If a BST becomes unbalanced, its performance degrades dramatically
- Techniques have been developed to keep a tree from slipping into an unbalanced condition – the most popular such technique uses *red-black trees* (a type of BST, where each node has a color, red or black.)
- Java's *TreeSet* and *TreeMap* classes implement balanced trees using *red-black trees.*

Red Black Tree



TreeSet

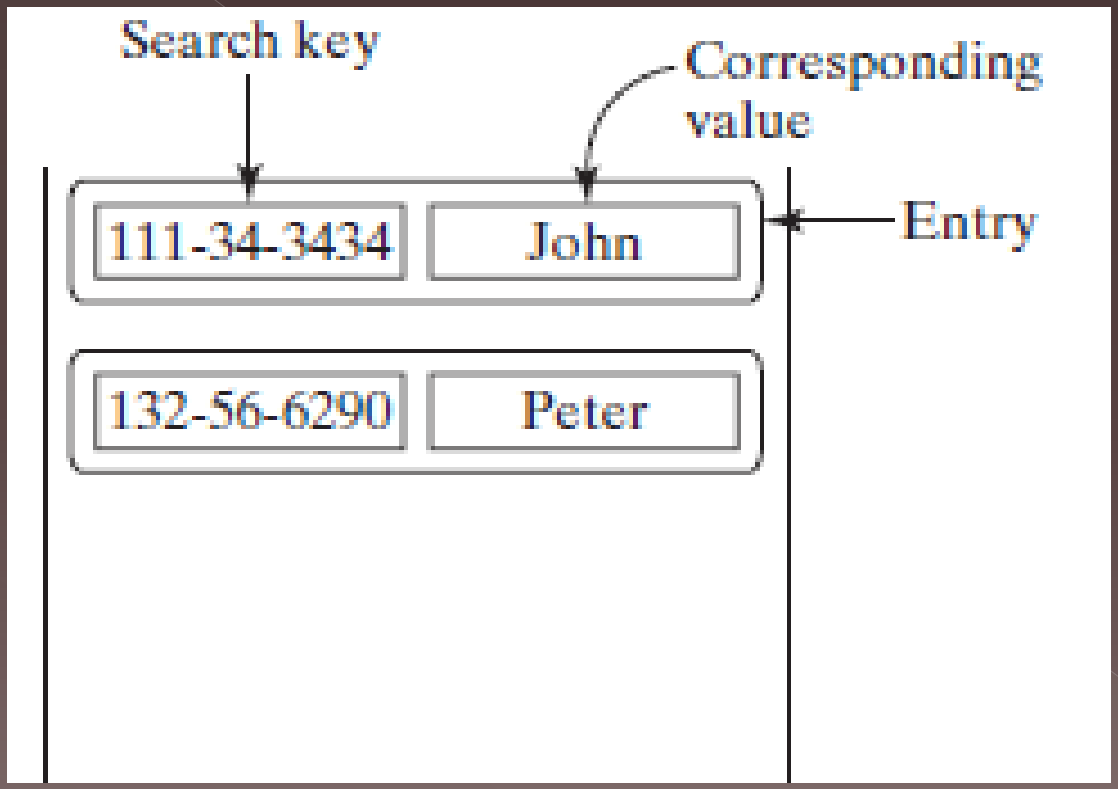
- In a **TreeSet**, elements are kept in order
- That means Java must comparing elements to decide which is “larger” and which is “smaller”
- Java done this by using the Comparator Interface
- TreeSetTest.java – demo package

Tree Map

- *You can create a map using one of its three concrete classes:* **HashMap**, **LinkedHashMap**, *or* **TreeMap**.
- A *map* is a container object that stores a collection of key/value pairs. It enables fast retrieval, deletion, and updating of the pair through the key.
- A map stores the values along with the keys.
- The keys are like indexes. In **List**, the indexes are integers. In **Map**, the keys can be any objects.
- A map cannot contain duplicate keys. Each key maps to one value. A key and its corresponding value form an entry stored in a map.

Example

A Map



Map: Basic operations

```
Object put(Object key, Object value);
Object get(Object key);
Object remove(Object key);
boolean containsKey(Object key);
boolean containsValue(Object value);
int size( );
boolean isEmpty( );
```

Demo Code

- TreeSetDemo
- TreeSetTest
- SortedTest
- MyBST
- BinaryTree
- TreeMapDemo

CONNECTING THE PARTS OF KNOWLEDGE WITH THE WHOLENESS OF KNOWLEDGE

1. Analysis of an algorithm or programming task leads to the identification of an abstract data type that will be needed.

2. Further analysis of the task and its environment provide estimates of the relative frequency and usage of method calls; on this basis, an implementation can be selected.
3. Transcendental Consciousness is the unbounded field of all the laws of nature, including specifically the law of least action.

4. Wholeness moving within itself : In Unity Consciousness, creation is seen as the frictionless flow of information and every action is appreciated in terms of the self-interacting dynamics of one's own Self.