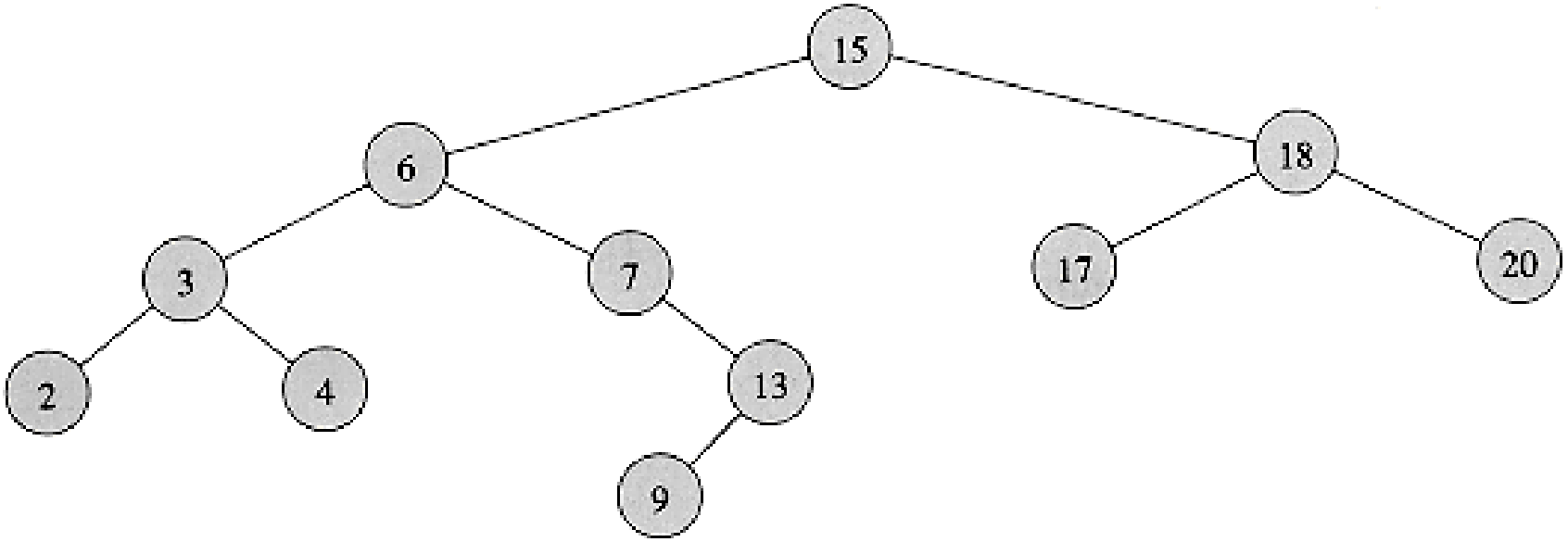
Binary search tree



Binary search tree

A sorted container

Other terms: Sorted binary tree

By default (for us)

- Elements are less than comparable

Other choices:

- Frequent: elements are identified by a key. Keys are less than comparable
- ….

-> see sorted map, priority queue

Binary search tree (BST)

BST - a binary tree that has the **BST Property**

BST Property

For each node **x** :

- if **y** is a node in the left subtree of **x** , then info(**y**) <= info(**x**)
- if **z** is a node in the right subtree of **x** , then info(**x**) <= info(**z**)

Property:

Inorder traversal → ascending order of elements

- can be used to implement a sorting algorithm. insert all the values we wish to sort into a new BST
  traverse it in order

BST – definitions **(equivalent)**

Let x be a node in a binary search tree. If y is a node in the left subtree of x, then key(y) <= key(x). If y is a node in right subtree of x, then key(x) <= key(y)

Cormen

A binary tree where every node's left subtree has keys less than the node's key, and every right subtree has keys greater than the node's key.

xlinux.nist.gov/dads/

Binary search tree

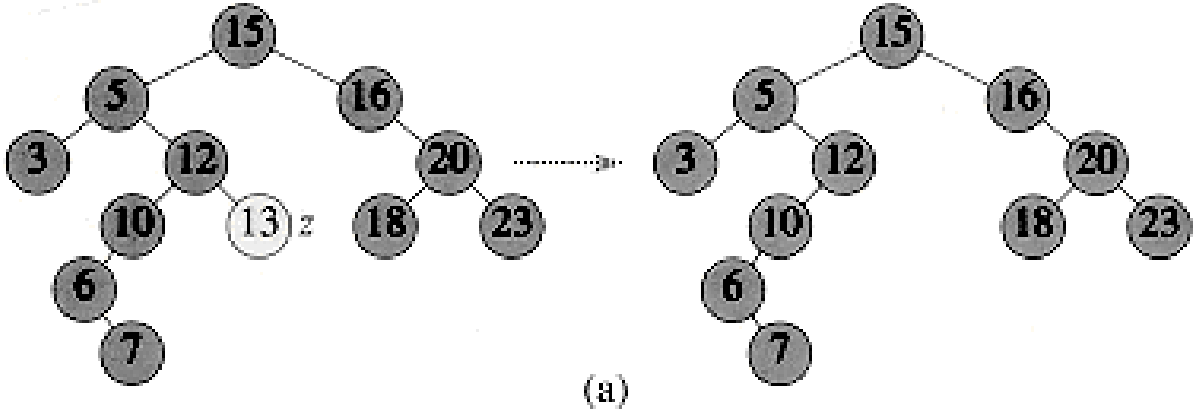
BST

	Average	Worst case
Search	O(log n)	O(n)
Insert	O(log n)	O(n)
Delete	O(log n)	O(n)

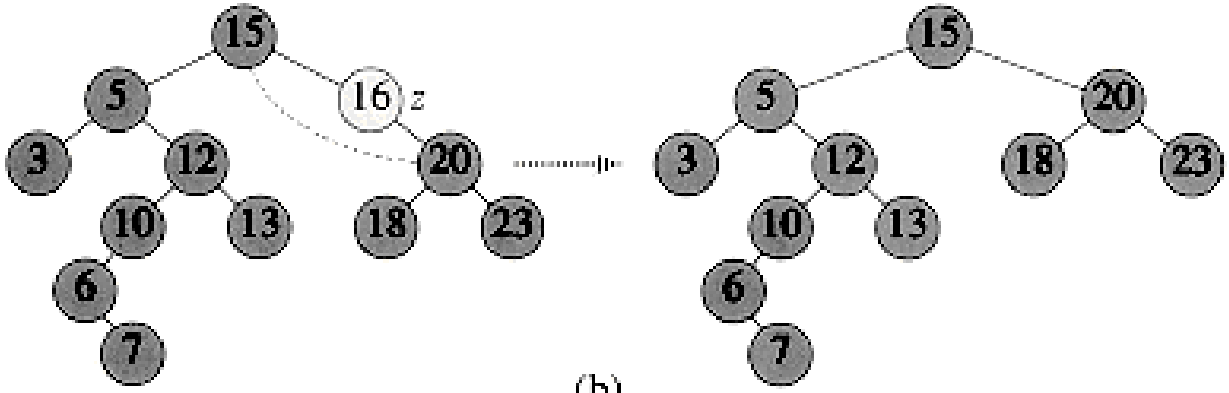
.

```
TreeNode: record
   info: TComparable
   left:      ^TreeNode
   right:     ^TreeNode
   parent:  ^TreeNode
end
```

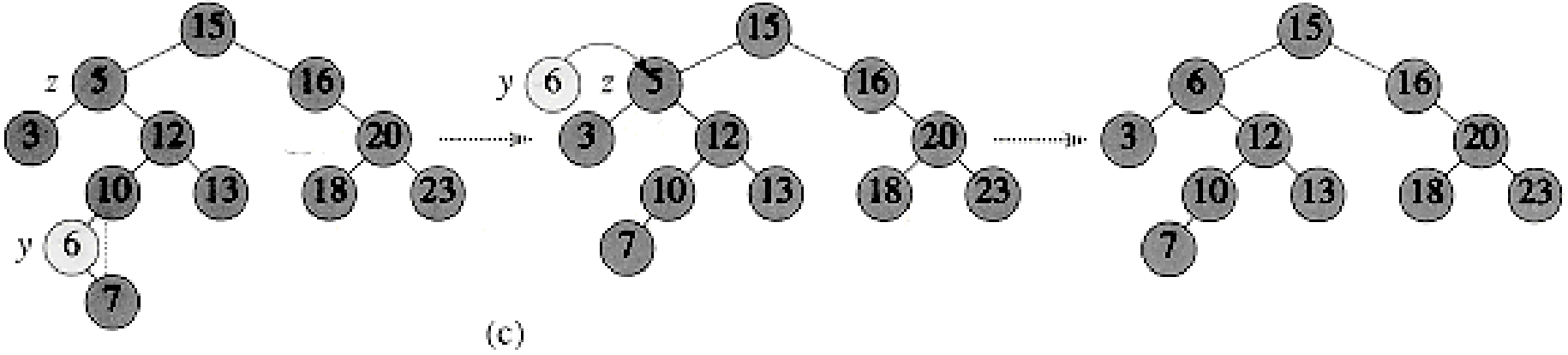
Delete


15
5
16
3
12
20
10
13
z
18
23
6
7

(a)


15
5
16
z
3
12
20
10
13
18
23
6
7

(b)


15
z
5
16
3
12
20
10
13
18
23
y
6
7

(c)

Delete

Delete a leaf

Deleting a node with one child:

delete it and replace it with its child.

Deleting a node with two children:

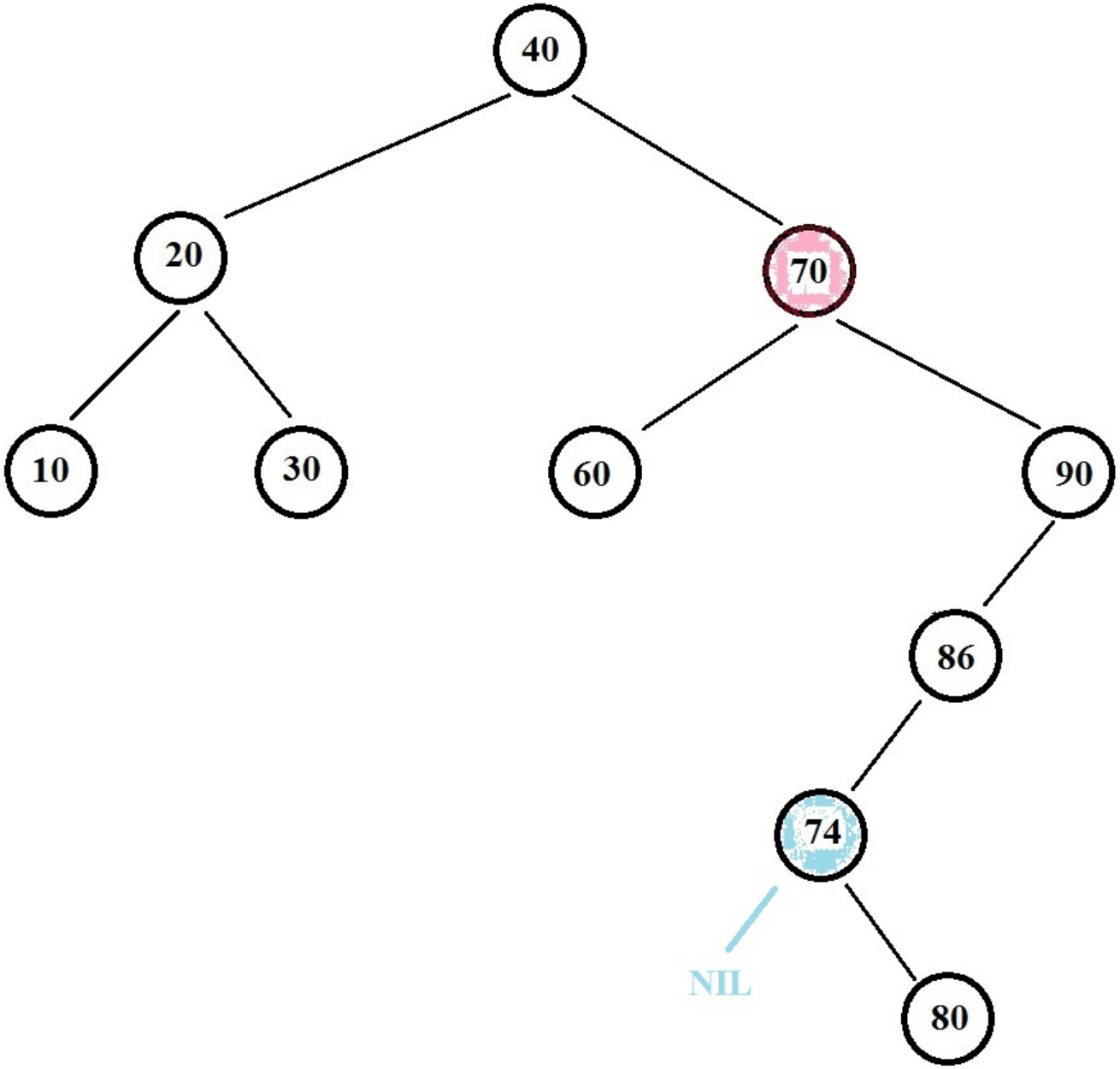
replace value with (either)

- its in-order successor

  or

- its in-order predecessor

and then delete the succ. or pred.


40
20
70
10
30
60
90
86
74
NIL
80

Subalg. delete(*T*, *z*)

@ collect information about nodes involved
  z – node to be (logically) deleted
  get y - the node to be really deleted (z or its successor)
  get x – the child of y (NIL if no children)
  get q – the parent of y (NIL if no parent)
@ copy information from y to z
@ remake link over the node y to delete
  from child to parent parent(x) <- q (if child exists)
  from parent to child
    if parent of y does not exist: update root node value
    else link from q to x
@delete y