# Binary Search Trees

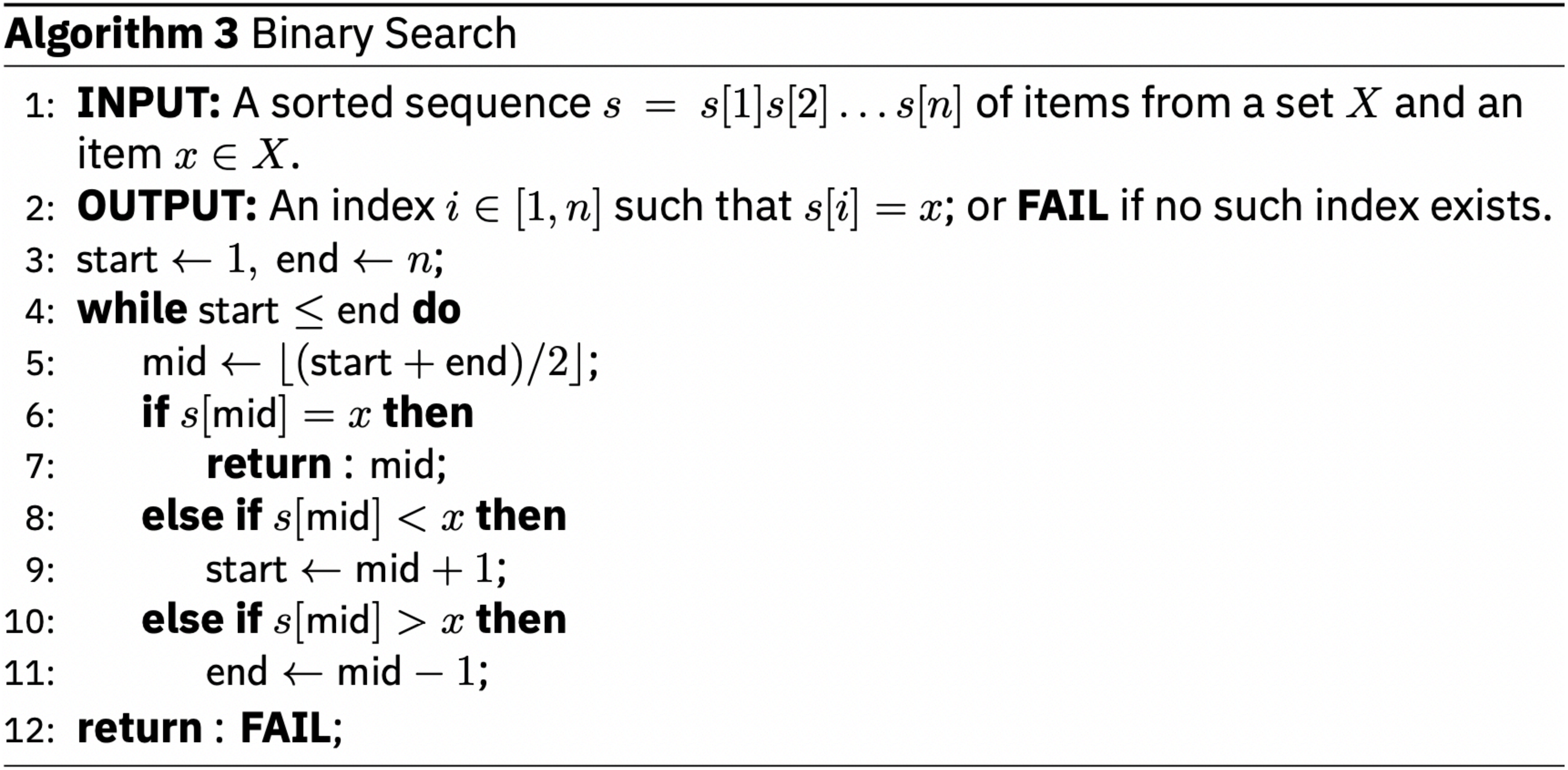
Chapter 12 of Cormen’s book

Giulia Bernardini
*giulia.bernardini@units.it*

Algorithmic Design
a.y. 2022/2023

# Binary Search

Binary search is an efficient algorithm using a divide-and-conquer strategy. Its running time is $O(\log n)$.



**Algorithm 3** Binary Search

```
INPUT: A sorted sequence $s = s[1]s[2]\ldots s[n]$ of items from a set $X$ and an item $x \in X$.
OUTPUT: An index $i \in [1, n]$ such that $s[i] = x$; or FAIL if no such index exists.
start $\leftarrow 1$, end $\leftarrow n$;
while start $\leq$ end do
    mid $\leftarrow \lfloor(\text{start} + \text{end})/2\rfloor$;
    if $s[\text{mid}] = x$ then
        return : mid;
    else if $s[\text{mid}] < x$ then
        start $\leftarrow$ mid $+ 1$;
    else if $s[\text{mid}] > x$ then
        end $\leftarrow$ mid $- 1$;
return : FAIL;
```

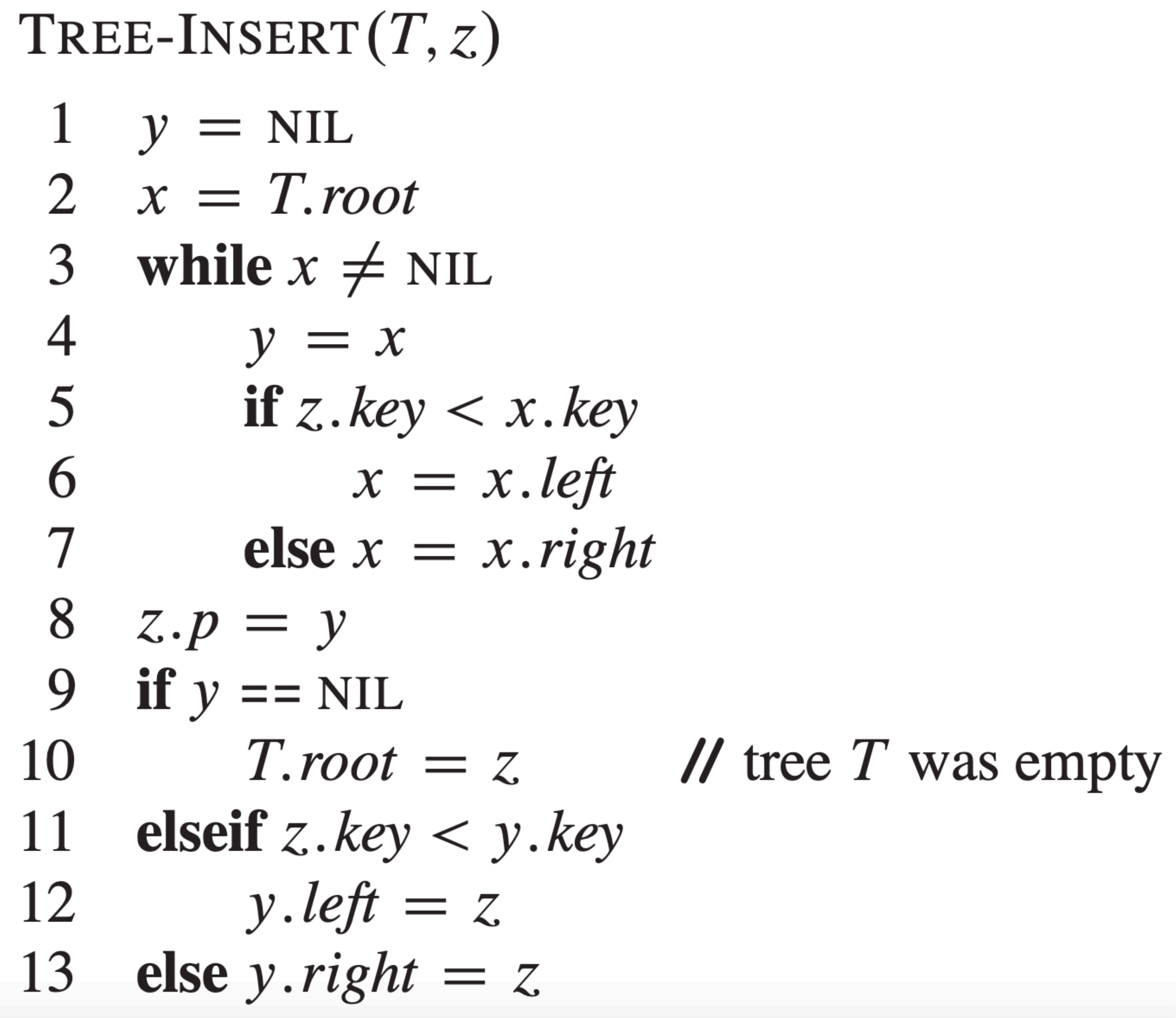
# Binary Search Trees

BSTs have the following property:

For every node $x$ in the tree, for every node $y$ in the left subtree of $x$, then $key(x) \leq key(y)$; for every node $z$ in the right subtree of $x$, $key(z) \geq key(x)$.

```
ITERATIVE-TREE-SEARCH$(x, k)$
while $x \neq$ NIL and $k \neq x.key$
    if $k < x.key$
        $x = x.left$
    else $x = x.right$
return $x$
```

# Binary Search Trees



```
TREE-INSERT($T, z$)
$y = $ NIL
$x = T.root$
while $x \neq$ NIL
    $y = x$
    if $z.key < x.key$
        $x = x.left$
    else $x = x.right$
$z.p = y$
if $y ==$ NIL
    $T.root = z$            // tree $T$ was empty
elseif $z.key < y.key$
    $y.left = z$
else $y.right = z$
```

# Binary Search Trees

```
INORDER-TREE-WALK($x$)
if $x \neq$ NIL
        INORDER-TREE-WALK($x.left$)
        print $x.key$
        INORDER-TREE-WALK($x.right$)
```

# Binary Search Trees

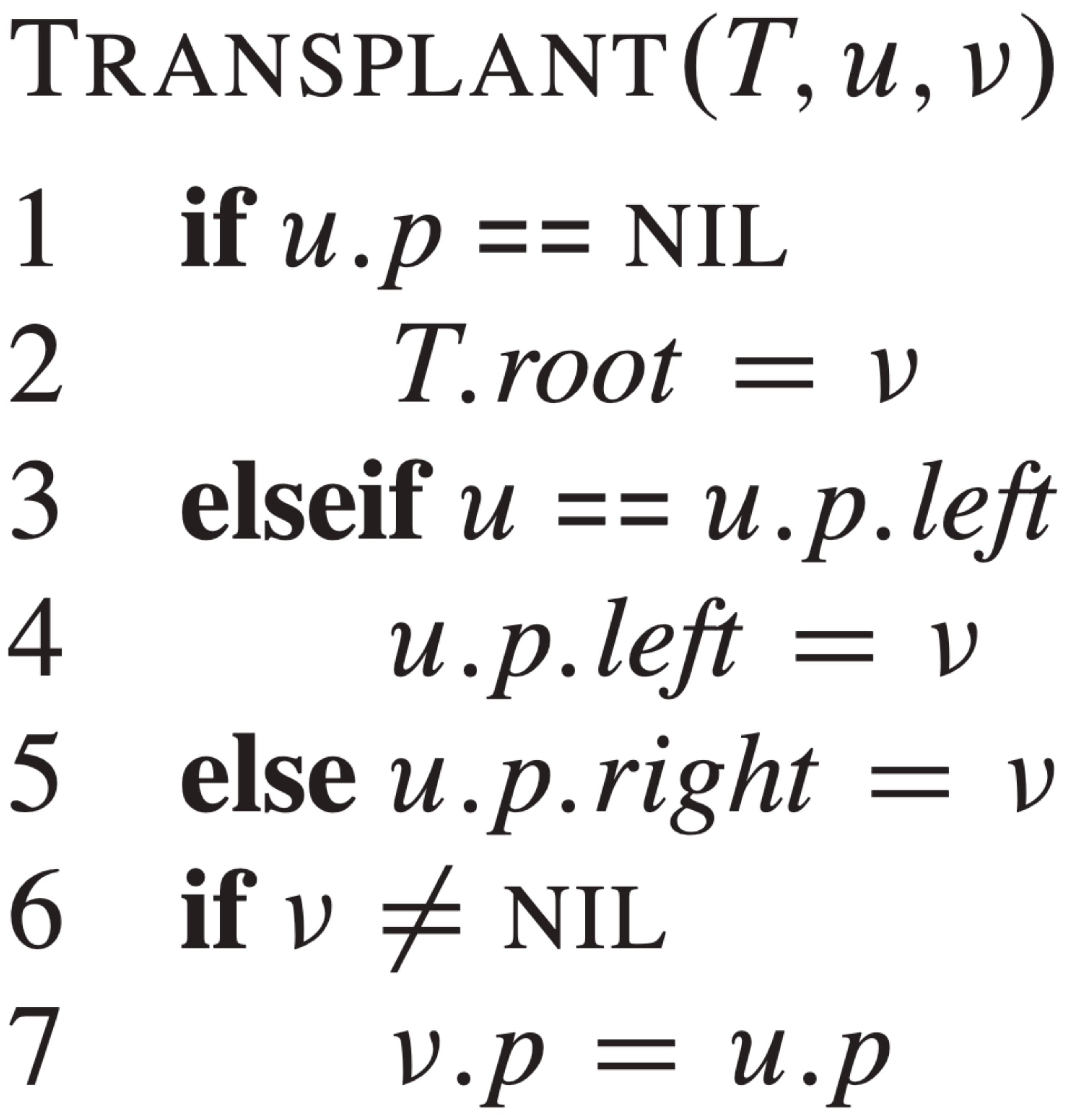
```
TREE-MINIMUM($x$)
    while $x.left \neq$ NIL
        $x = x.left$
    return $x$
```

```
TREE-MAXIMUM($x$)
    while $x.right \neq$ NIL
        $x = x.right$
    return $x$
```

# Binary Search Trees

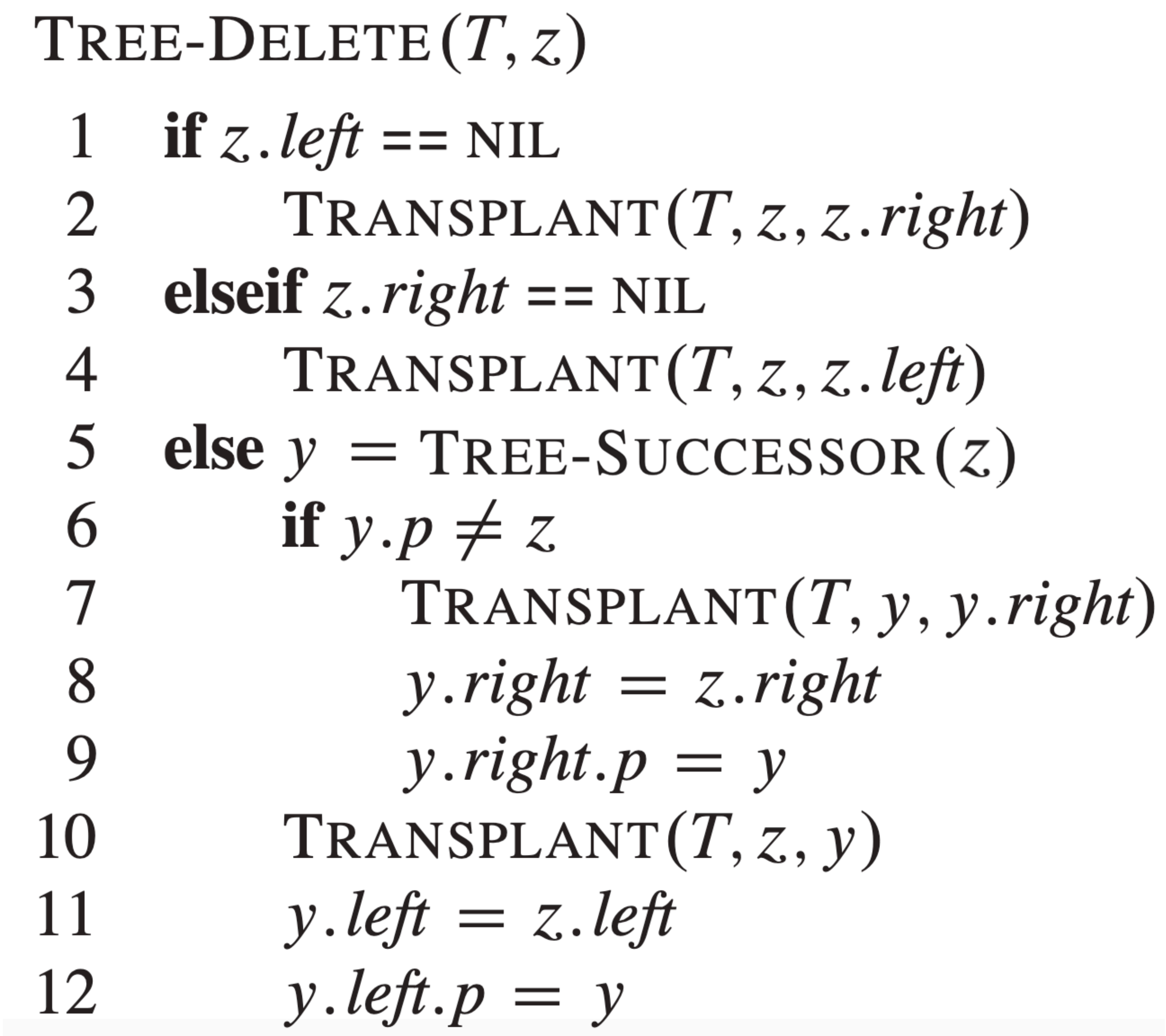
```
TREE-SUCCESSOR($x$)
if $x.right \neq$ NIL
    return TREE-MINIMUM($x.right$)
$y = x.p$
while $y \neq$ NIL and $x == y.right$
    $x = y$
    $y = y.p$
return $y$
```

# Binary Search Trees



```
TRANSPLANT$(T, u, v)$
if $u.p$ == NIL
        $T.root = v$
elseif $u == u.p.left$
        $u.p.left = v$
else $u.p.right = v$
if $v \neq$ NIL
        $v.p = u.p$
```

# Binary Search Trees



```
TREE-DELETE($T, z$)
if $z.left$ == NIL
    TRANSPLANT($T, z, z.right$)
elseif $z.right$ == NIL
    TRANSPLANT($T, z, z.left$)
else $y = $ TREE-SUCCESSOR($z$)
    if $y.p \neq z$
        TRANSPLANT($T, y, y.right$)
        $y.right = z.right$
        $y.right.p = y$
    TRANSPLANT($T, z, y$)
    $y.left = z.left$
    $y.left.p = y$
```

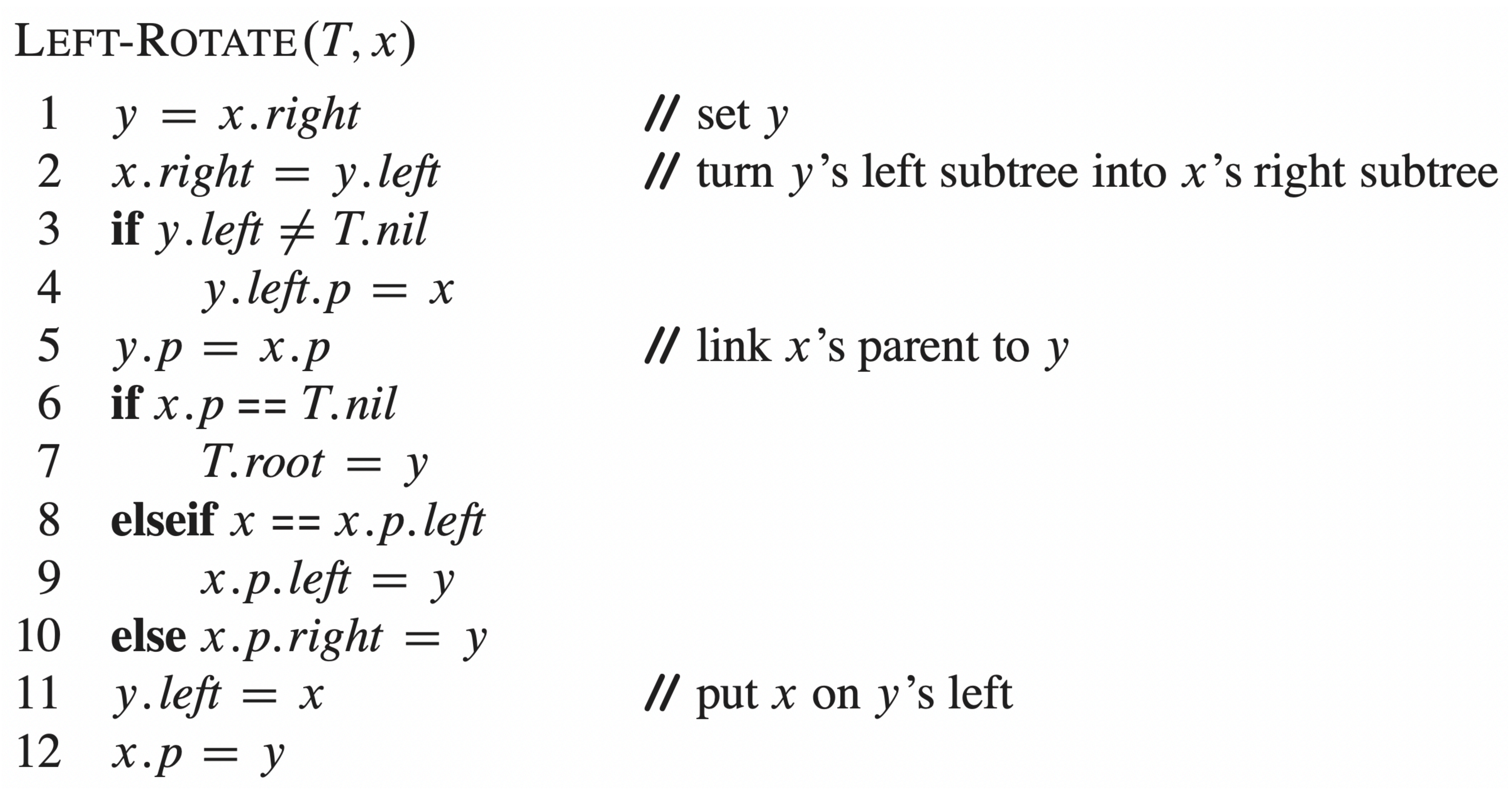
# Red-Black Trees

## Chapter 13 of Cormen's book

Giulia Bernardini
*giulia.bernardini@units.it*

Algorithmic Design
a.y. 2022/2023

# Rotations



```
LEFT-ROTATE$(T, x)$
    $y = x.right$                      // set $y$
    $x.right = y.left$                 // turn $y$'s left subtree into $x$'s right subtree
    if $y.left \neq T.nil$
        $y.left.p = x$
    $y.p = x.p$                        // link $x$'s parent to $y$
    if $x.p == T.nil$
        $T.root = y$
    elseif $x == x.p.left$
        $x.p.left = y$
    else $x.p.right = y$
    $y.left = x$                       // put $x$ on $y$'s left
    $x.p = y$
```

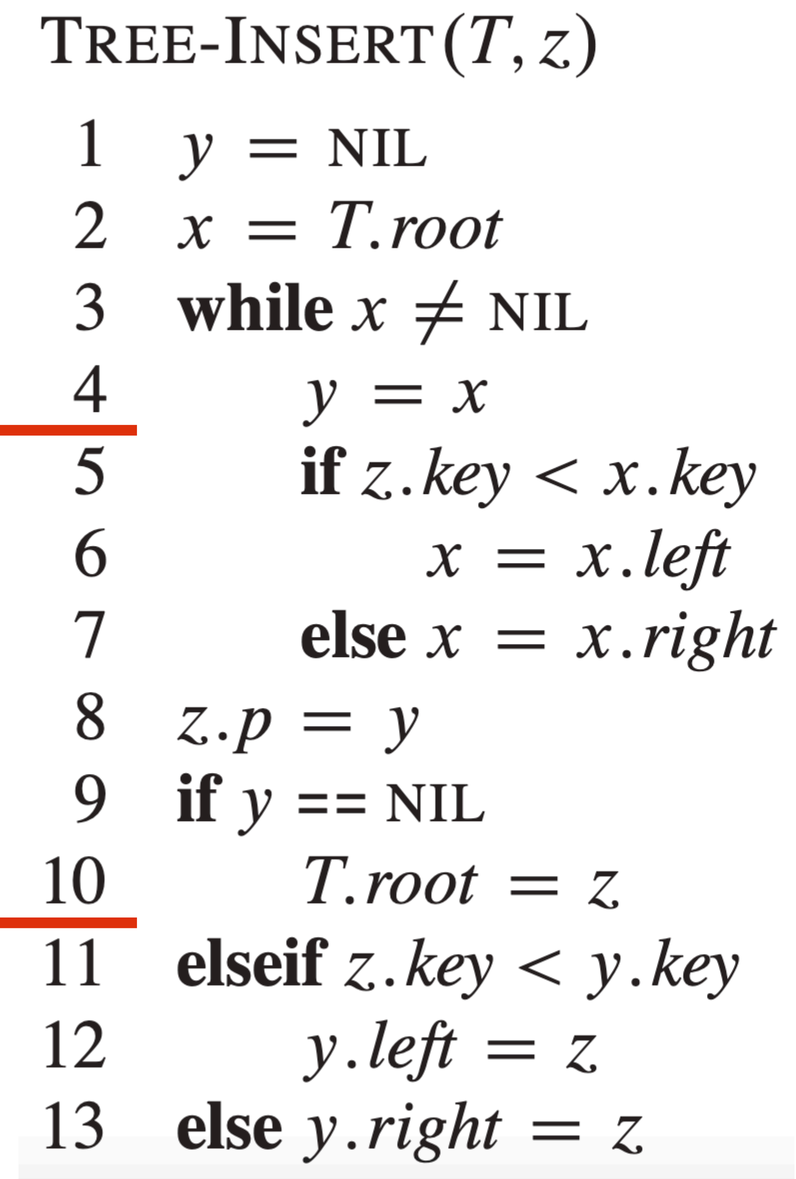
# Exercises

**Cormen Problem 12-1.** Equal keys pose a problem for the implementation of binary search trees.

**a.** What is the asymptotic performance of TREE-INSERT when used to insert *n* items with identical keys into an initially empty binary search tree?

# Exercises

**Cormen Problem 12-1.** We propose to improve TREE-INSERT by testing before line 5 to determine whether *z.key* = *x.key* and by testing before line 11 to determine whether *z.key* = *y.key*.



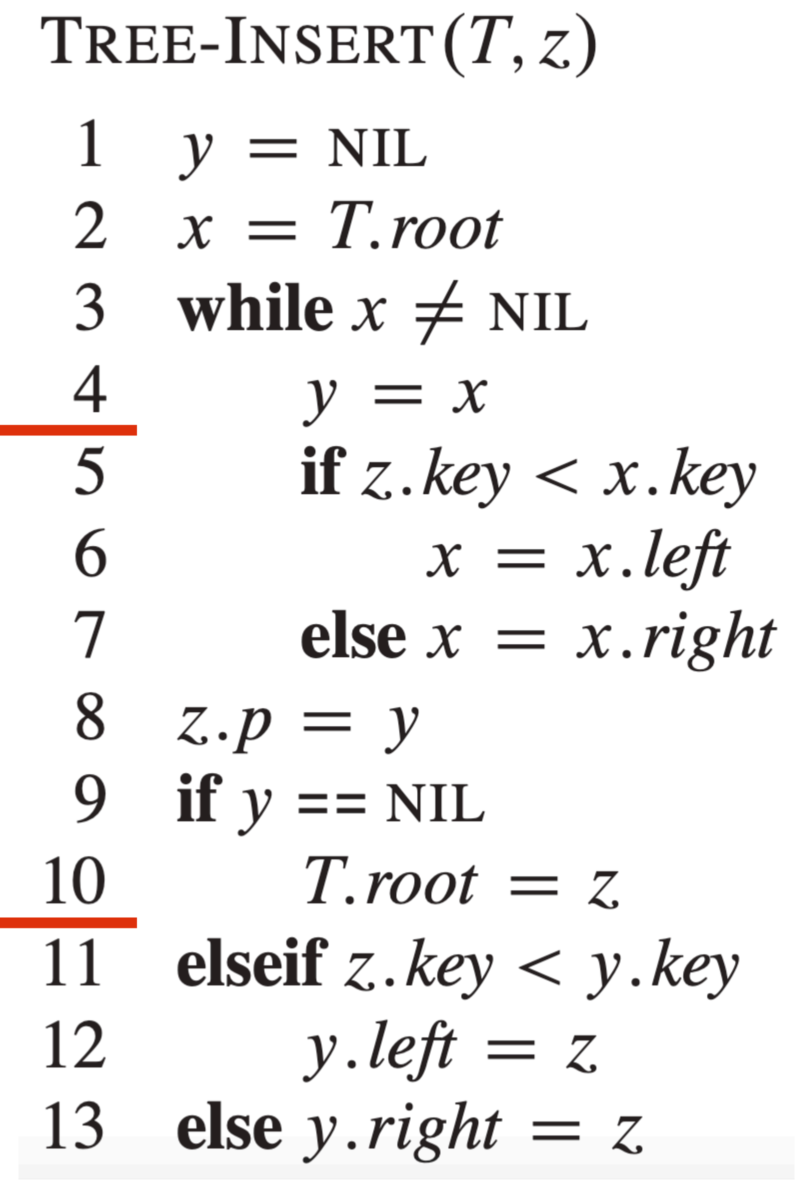
```
TREE-INSERT(T, z)
 1  $y = \text{NIL}$
 2  $x = T.root$
 3  while $x \neq \text{NIL}$
 4      $y = x$
 5      if $z.key < x.key$
 6          $x = x.left$
 7      else $x = x.right$
 8  $z.p = y$
 9  if $y == \text{NIL}$
10      $T.root = z$
11  elseif $z.key < y.key$
12      $y.left = z$
13  else $y.right = z$
```

If equality holds, we implement one of the following strategies. For each strategy, find the asymptotic performance of inserting *n* items with identical keys into an initially empty binary search tree. (The strategies are described for line 5, in which we compare the keys of *z* and *x*. Substitute *y* for *x* to arrive at the strategies for line 11.)

**b.** Keep a boolean flag *x.b* at node *x*, and set *x* to either *x.left* or *x.right* based on the value of *x.b*, which alternates between FALSE and TRUE each time we visit *x* while inserting a node with the same key as *x*.

# Exercises

**Cormen Problem 12-1.** We propose to improve TREE-INSERT by testing before line 5 to determine whether *z.key* = *x.key* and by testing before line 11 to determine whether *z.key* = *y.key*.



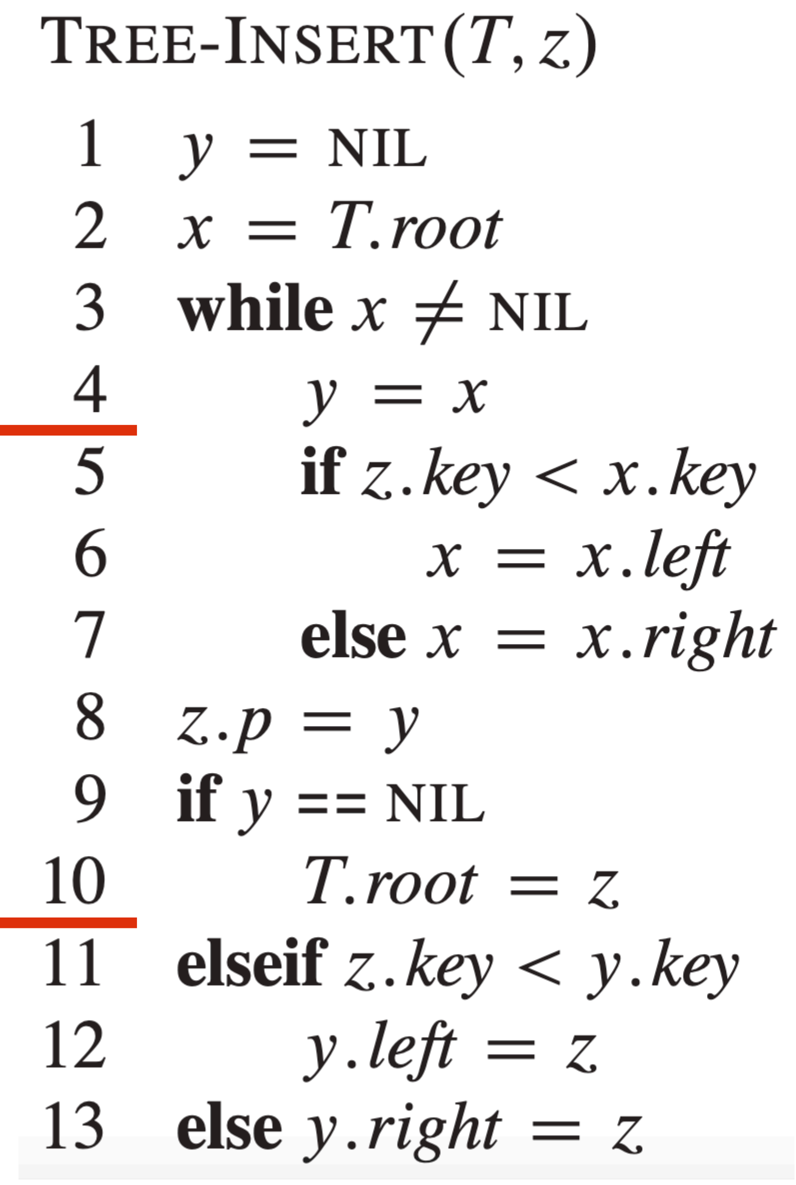
```
TREE-INSERT(T, z)
1  y = NIL
2  x = T.root
3  while x ≠ NIL
4      y = x
5      if z.key < x.key
6          x = x.left
7      else x = x.right
8  z.p = y
9  if y == NIL
10     T.root = z
11 elseif z.key < y.key
12     y.left = z
13 else y.right = z
```

If equality holds, we implement one of the following strategies. For each strategy, find the asymptotic performance of inserting *n* items with identical keys into an initially empty binary search tree. (The strategies are described for line 5, in which we compare the keys of *z* and *x*. Substitute *y* for *x* to arrive at the strategies for line 11.)

**c.** Keep a list of nodes with equal keys at *x*, and insert *z* into the list.

# Exercises

**Cormen Problem 12-1.** We propose to improve TREE-INSERT by testing before line 5 to determine whether *z.key* = *x.key* and by testing before line 11 to determine whether *z.key* = *y.key*.



```
TREE-INSERT(T, z)
1   y = NIL
2   x = T.root
3   while x ≠ NIL
4       y = x
5       if z.key < x.key
6           x = x.left
7       else x = x.right
8   z.p = y
9   if y == NIL
10      T.root = z
11  elseif z.key < y.key
12      y.left = z
13  else y.right = z
```

If equality holds, we implement one of the following strategies. For each strategy, find the asymptotic performance of inserting *n* items with identical keys into an initially empty binary search tree. (The strategies are described for line 5, in which we compare the keys of *z* and *x*. Substitute *y* for *x* to arrive at the strategies for line 11.)

**d.** Randomly set x to either *x.left* or *x.right*. (Give the worst-case performance and informally derive the expected running time.)

# Exercises

A preorder traversal of a tree is given by the following procedure:

- Visit (print) the root node
- Traverse the left sub-tree in pre-order
- Traverse the right sub-tree in pre-order

A postorder traversal of a tree is given by the following procedure:

- Traverse the left subtree by calling the postorder function recursively.
- Traverse the right subtree by calling the postorder function recursively.
- Visit (print) the current node.

**EX.** Given a BST in pre-order as {13,5,3,2,11,7,19,23}, draw this BST and determine if this BST is the same as one described in post-order as {2,3,5,7,11,23,19,13}.