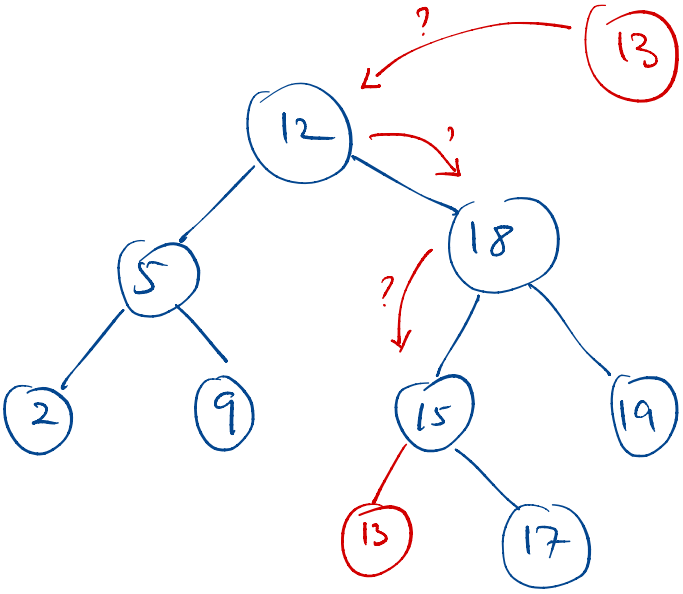
Binary Search Tree - III

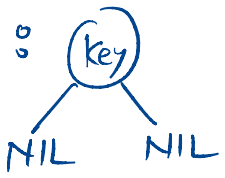
Insertion: Causes the dynamic set to be modified.

→ Insertion must ensure that the BST property continues to hold

Example:



Input: a BST node z whose key has been set and $z.left = NIL$, $z.right = NIL$



Tree - Insert (T, z)

```
x = T.root        // Node being compared with z
y = NIL           // y will be the parent of z
while x ≠ NIL
    y = x
    if z.key < x.key
        x = x.left
    else
        x = x.right
z.p = y           // found the location to insert z with parent y
if y == NIL       // Tree was empty
    T.root = z
else if z.key < y.key
    y.left = z
else
    y.right = z
```

O(h)

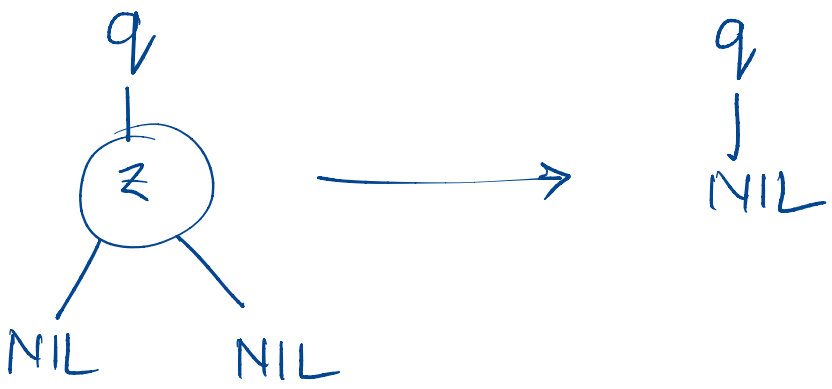
Complexity

Deletion: Delete a node z from the BST T

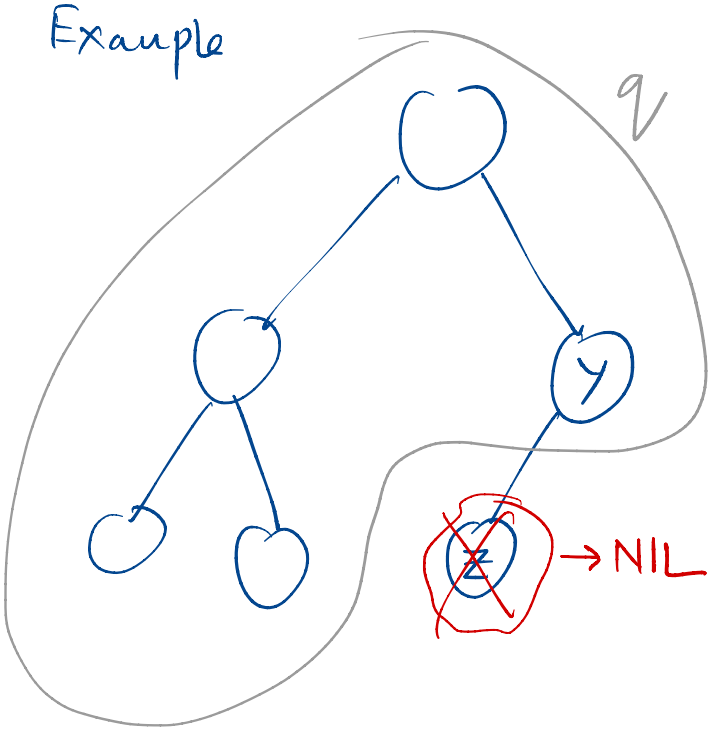
3 basic Cases:

(1) If z has no children then simply remove it by modifying its parent to replace z with NIL as child

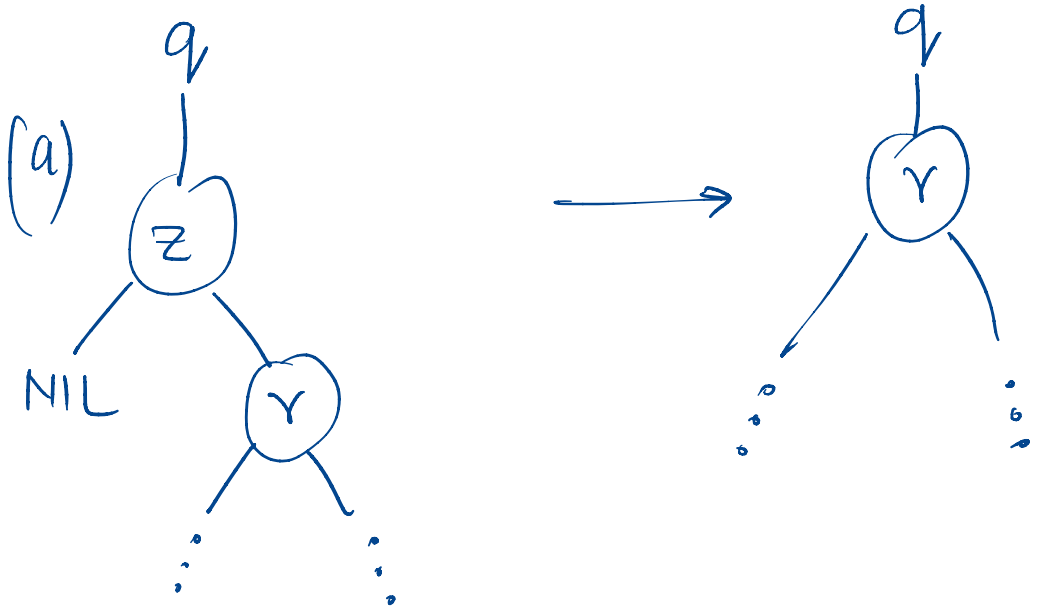
(q represents rest of the tree)

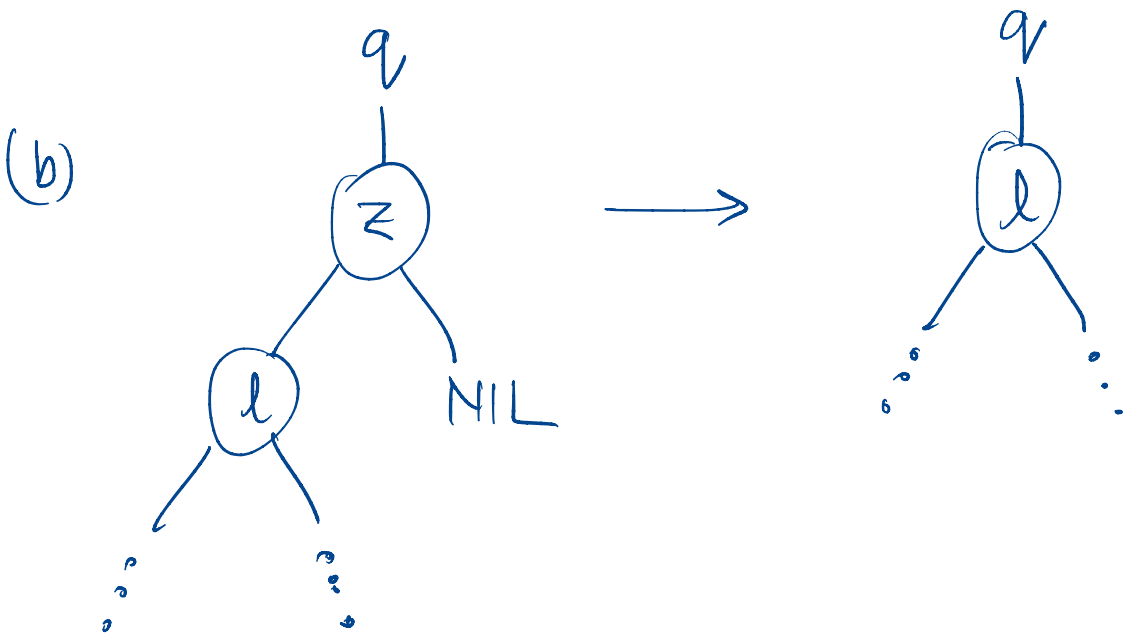


Example



(11) If z has just 1 child, then elevate that child to take z's position in the tree by modifying z's parent to replace z by z's child

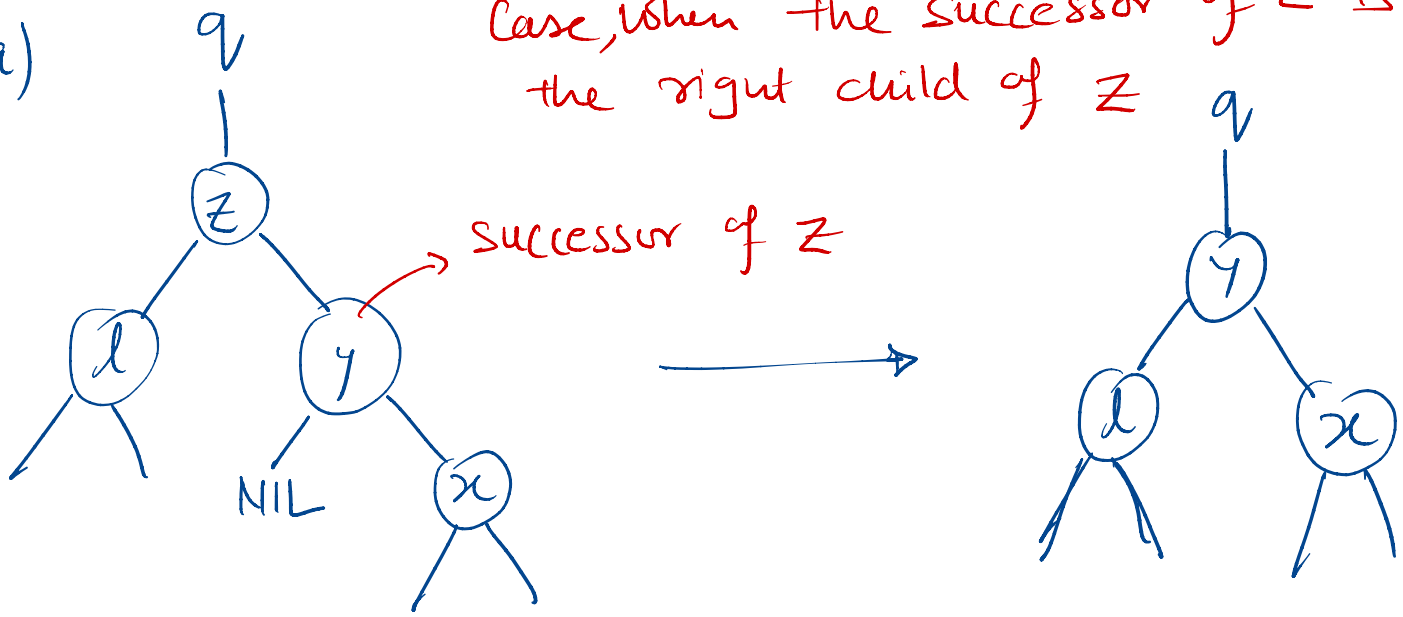




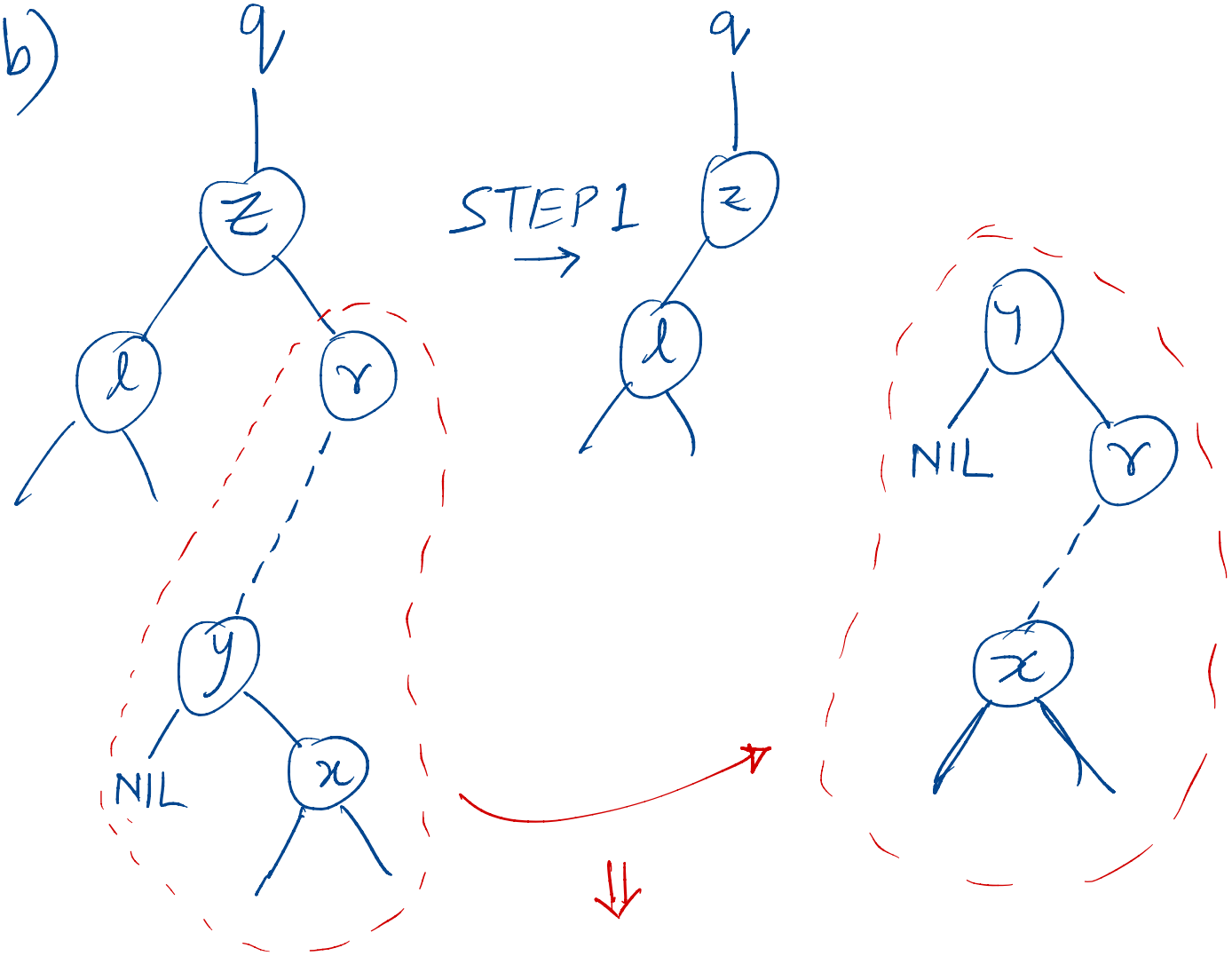
(III) If z has 2 children, find z's successor y — which must belong to z's right subtree — and move y to take z's position in the tree

The rest of the z's original right subtree becomes y's new right subtree and z's left subtree becomes y's new left subtree.

a)

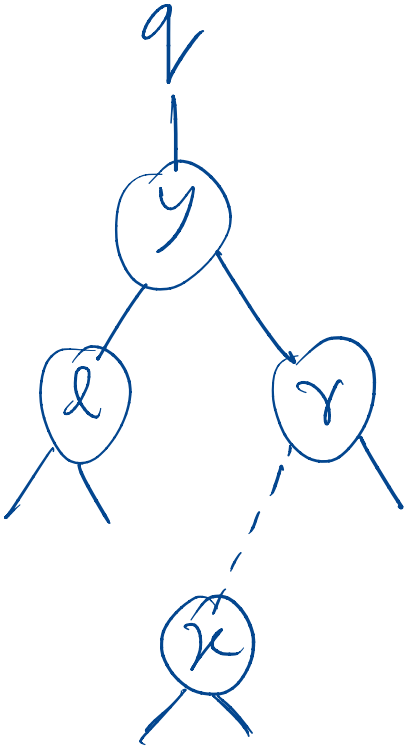


b)



STEP 2

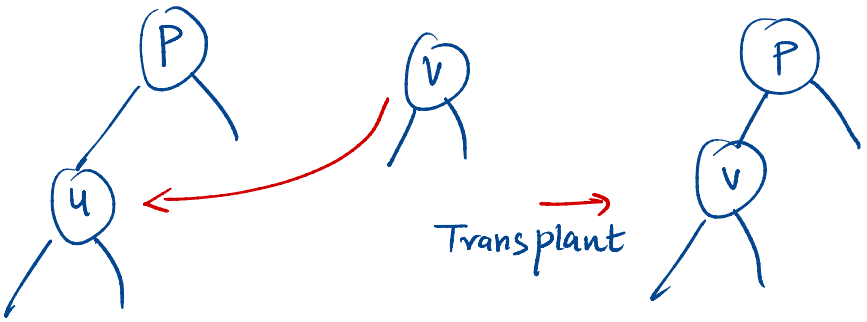
- remove y from the tree
  (y has only 1 child x, so replace y with x in the tree)
- Place y as the root with r being the right child of y
  (this transformation keeps y the smallest element)



Transplant Operation

↳ replaces one subtree as a child of its parent with another subtree

↳ Define transplant that replaces subtree rooted at node u with a subtree rooted at node v. Set parent pointers appropriately.



Transplant (T, u, v)

```
if u.p == NIL
    T.root = v
else if u == u.p.left
    u.p.left = v
else
    u.p.right = v
if v ≠ NIL
    v.p = u.p   // set the parent
```

Define node deletion using the transplant operation

Tree-Delete (T, z)

handle cases:
- No child
- only 1 child

```
if z.left == NIL                          → replace z by its right child
    Transplant(T, z, z.right)
else if z.right == NIL                    → replace z by its left child.
    Transplant(T, z, z.left)
else
    y = Tree-Minimum(z.right)             → successor of z
    if y ≠ z.right                        ← left has to be NIL
        Transplant(T, y, y.right)
        y.right = z.right
        y.right.p = y
    Transplant(T, z, y)                   → replace z by its successor
    y.left = z.left
    y.left.p = y
```

handle case of both children

make y the right child of z

O(h) complexity