## delete(x,T): Case 1

**if** T has no children
 **if** x == T.item
  **return empty tree**
 **else**
  NOT FOUND

e.g. Delete 4 in Figure 1.

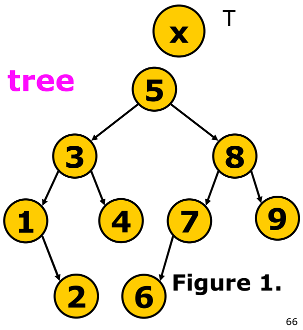



**Figure 1.**



This pseudo-code returns the new tree after item x is deleted from tree T.

## delete(x,T): Case 2 (A)

**if** T has only 1 child (left)
 **if** x == T.item
  **return** T.left
 **else**
  T.left = delete(x,T.left)
 **return** T

e.g. delete 7 in Figure 1.

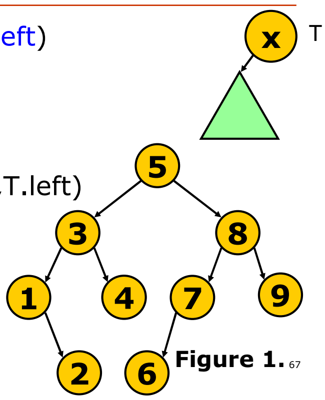



**Figure 1.**



## delete(x,T): Case 2 (B)

**if** T has only 1 child (right)
 **if** x == T.item
  **return** T.right
 **else**
  T.right = delete(x, T.right)
 **return** T

e.g. delete 1 in Figure 1.

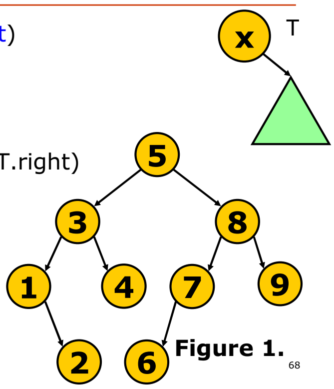
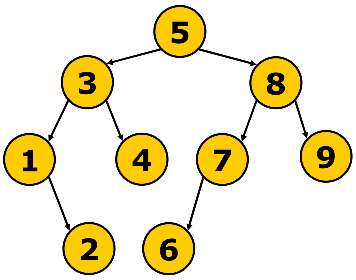



**Figure 1.**

## delete(x,T): Case 3

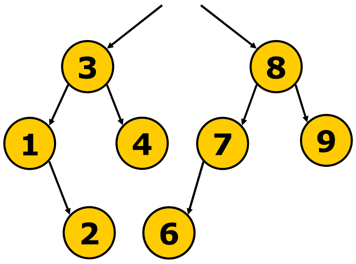
Node to be deleted has 2 children
e.g. delete 5






## delete(x,T): Case 3

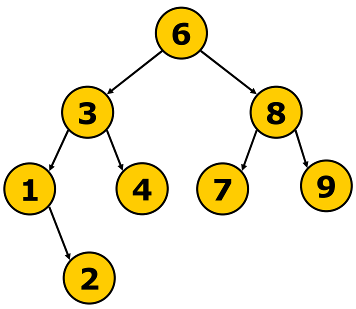
e.g. delete 5






## delete(x,T): Case 3

5 deleted!






After deleting 5 from the tree.

The node containing 6 is called the inorder successor of the node to be deleted.

We can also replace the node to be deleted by its inorder predecessor. Node with 4

## delete(x,T): Case 3

```
if T has two children
   if x == T.item
     T.item = findMin(T.right) // replace T.item by
                 // the min. item of the right subtree

     T.right = delete(T.item, T.right)
      // delete x (i.e. T.item) from the right substree
   else if x < T.item
            T.left = delete(x, T.left)
          else
            T.right = delete(x, T.right)
return T
```



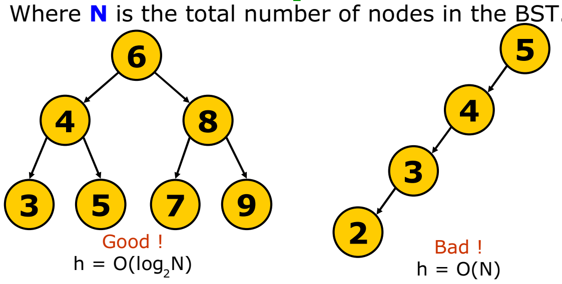
## Running Time of BST

- findMin O(h) where h is the height of the BST
- search O(h)
- insert O(h)
- delete O(h)



## Running time of BST (cont.)

- But h is not always $O(\log_2 N)$!
  Where **N** is the total number of nodes in the BST.




When you insert nodes in increasing or decreasing order, you get a **skewed** tree.



When you insert nodes in increasing order, you get a skewed tree. Therefore h is actually in O(N).