

BINARY SEARCH TREES - PART 2

Problem Solving with Computers-II

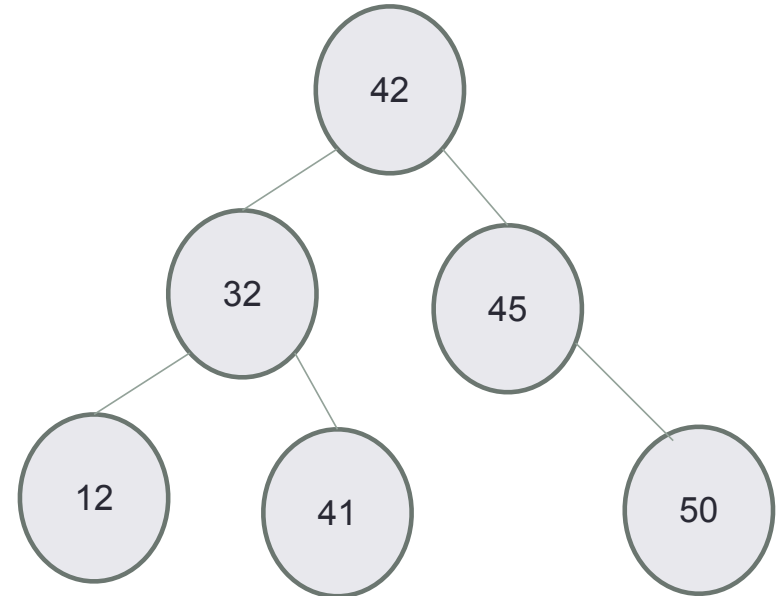
The image shows the C++ logo in blue, followed by a snippet of C++ code in a monospaced font. The code is:

```
#include <iostream>
using namespace std;
int main(){
    cout<<"Hola Facebook!n";
    return 0;
}
```

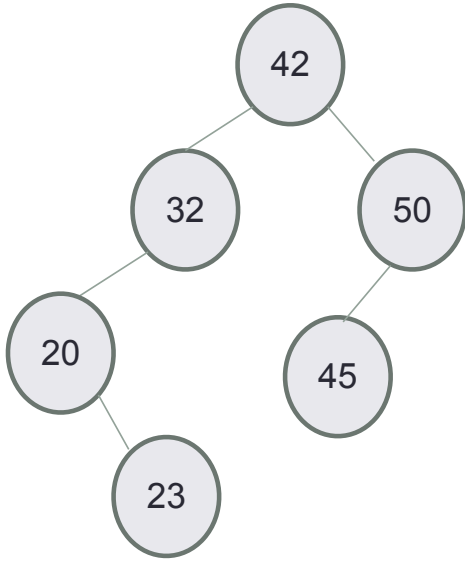
Define the BST ADT

68, 30, 63, 71, 77

Operations
Search
Insert
Min
Max
Successor
Predecessor
Delete
Print elements In order Preorder, Post order



Predecessor: Next smallest element



- What is the predecessor of 32?
- What is the predecessor of 45?

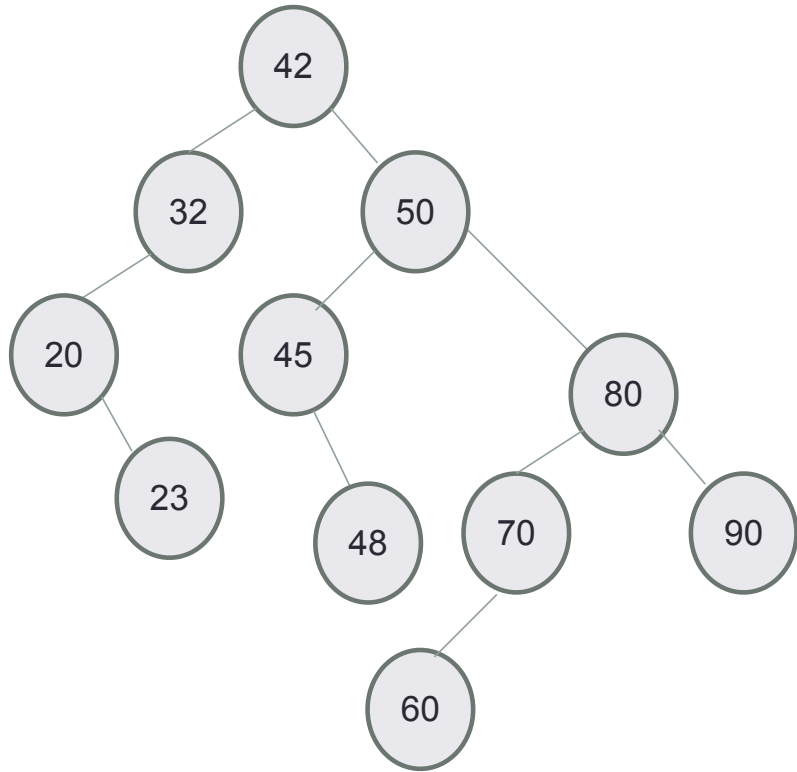
```
int bst::predecessor(BSTNode* n, int value) const{
    if(!n) return std::numeric_limits<int>::min();
    if(n->left){
        //Case 1
        return max key in the left subtree
    }else{
        //Case 2
        traverse parent pointers until a smaller key is found
    }
}
```

Fill in the blank for case 1 using min/max helper functions

- A. n->left;
- B. min(n)
- C. max(n)
- D. min(n->left)
- E. max(n->left)



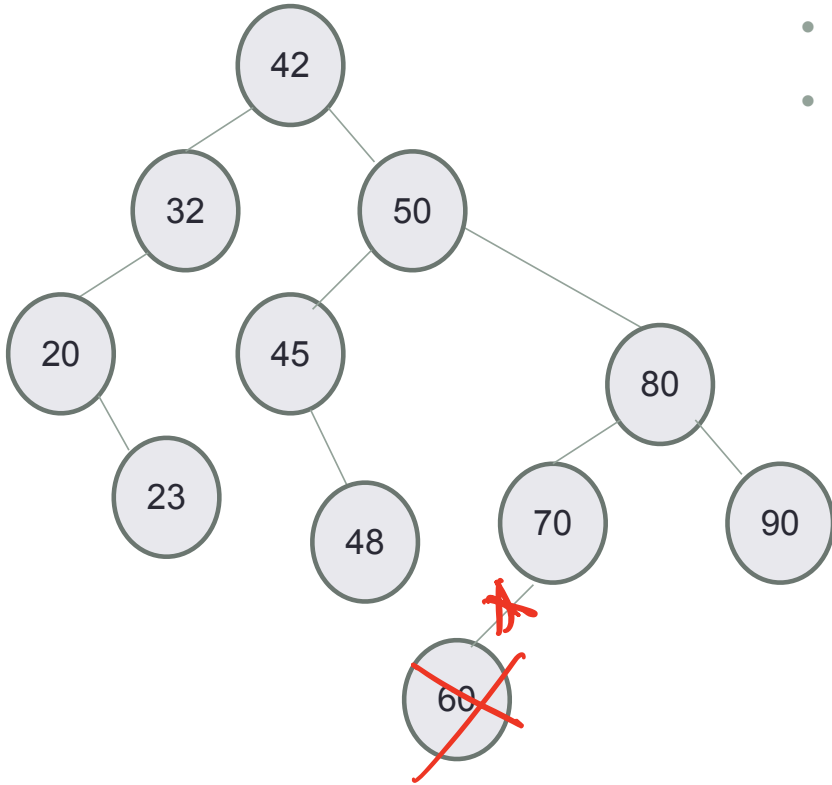
Successor: Next largest element



- What is the successor of 45?
- What is the successor of 50?
- What is the successor of 60?

Delete: Case 1 - Node is a leaf node

- Set parent's (left/right) child pointer to null
- Delete the node

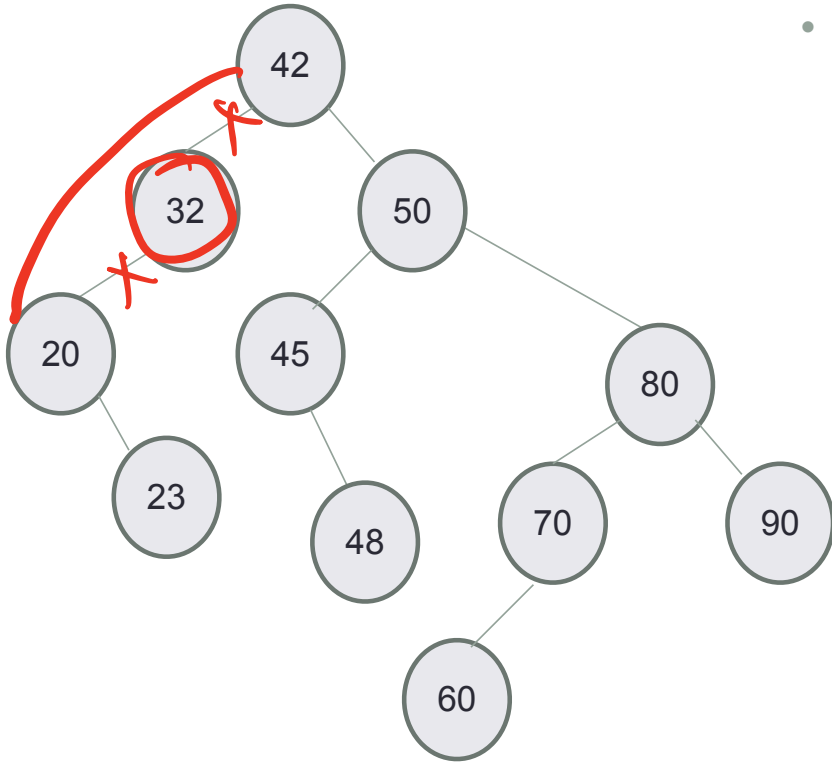


Delete 60

Delete: Case 2 - Node has only one child

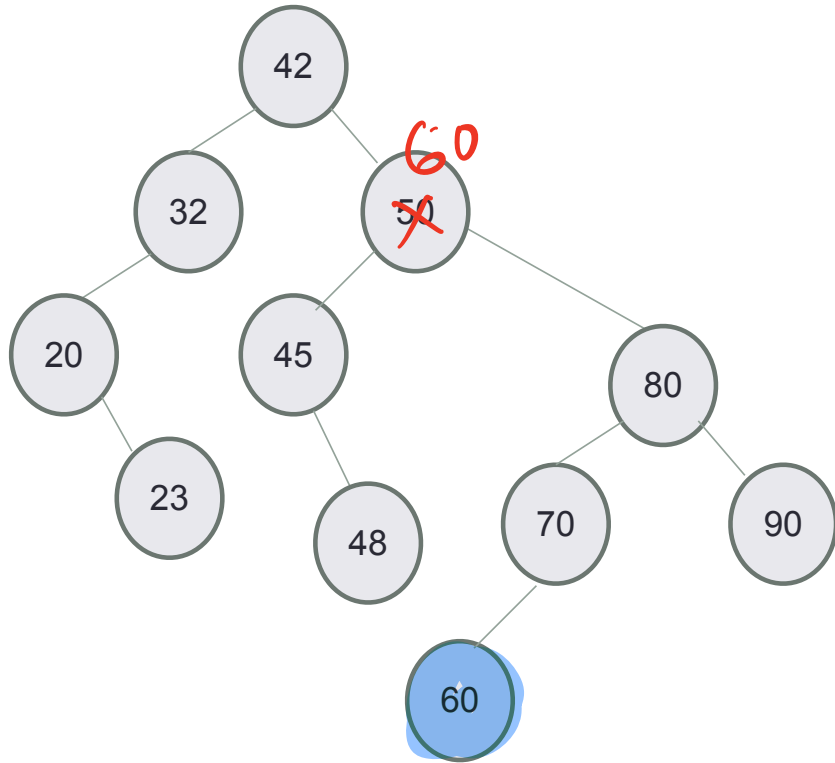
- Replace the node by its only child

Delete 32



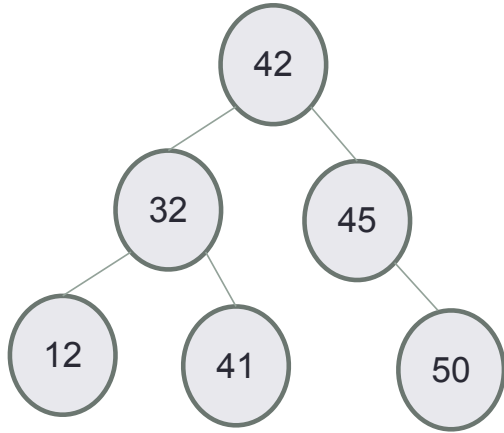
Delete: Case 3 - Node has two children

- Can we still replace the node by one of its children? Why or Why not?



Delete 50
Replace 50's key by the
successor's key
Delete the successor node
(in blue)
(Defaults to one of the
two previous cases)

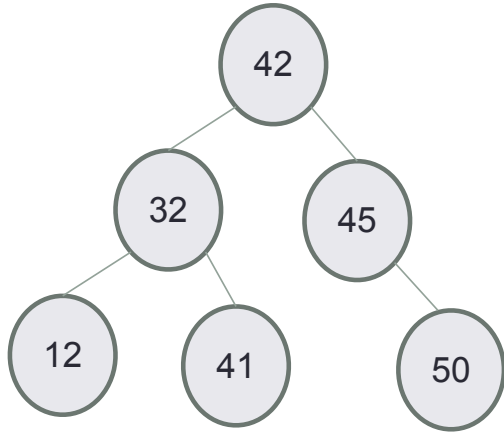
In order traversal: print elements in sorted order



Algorithm Inorder(tree)

1. Traverse the left subtree, i.e., call Inorder(left-subtree)
2. Visit the root.
3. Traverse the right subtree, i.e., call Inorder(right-subtree)

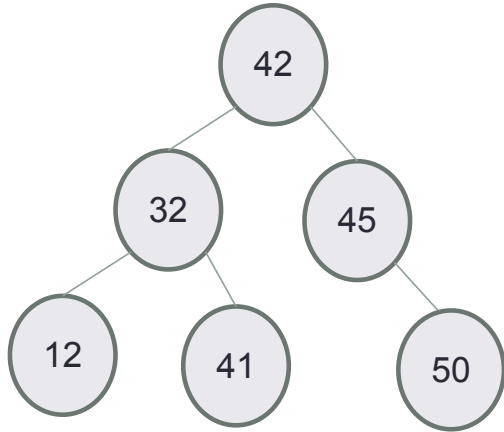
Pre-order traversal: nice way to linearize your tree!



Algorithm Preorder(tree)

1. Visit the root.
2. Traverse the left subtree, i.e., call Preorder(left-subtree)
3. Traverse the right subtree, i.e., call Preorder(right-subtree)

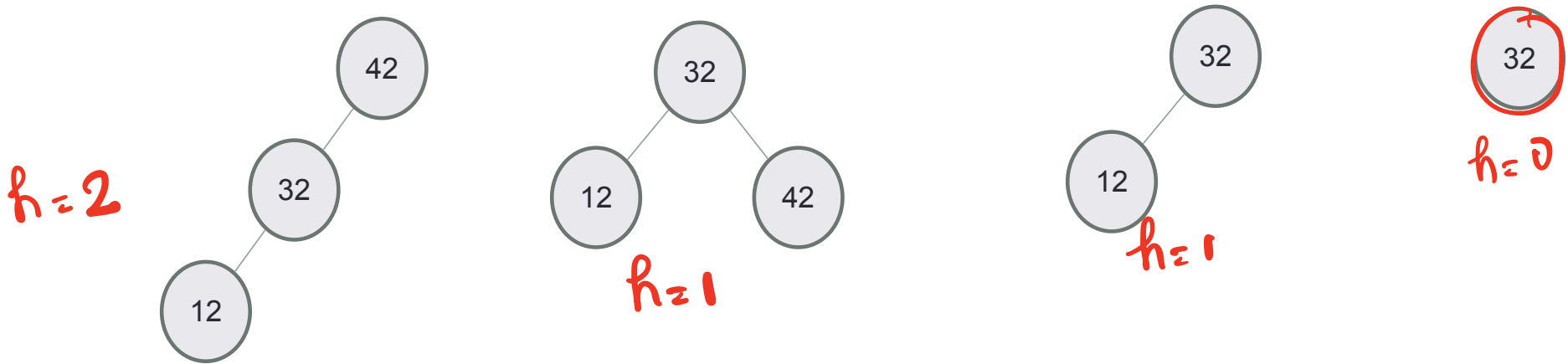
Post-order traversal: use to recursively clear the tree!



Algorithm Postorder(tree)

1. Traverse the left subtree, i.e., call Postorder(left-subtree)
2. Traverse the right subtree, i.e., call Postorder(right-subtree)
3. Visit the root.

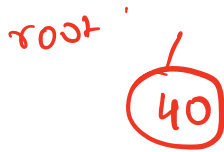
Write a member function for the BST ADT to compute its height



- How confident are you about your solution and overall approach?
 - A. Not at all
 - B. Somewhat confident
 - C. Very confident

root 

expected return value of getHeight
 $h = -1$

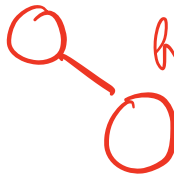


$h = 0$

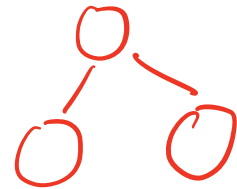
getHeight(r->left)
returns 0



getHeight(r->right)
returns -1



$h = 1$

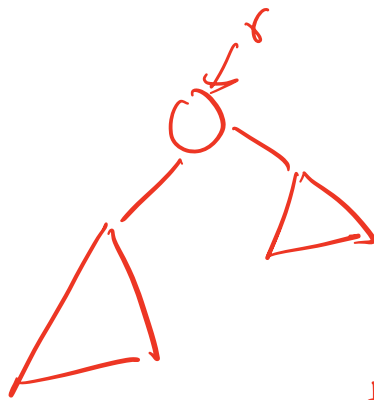


$h = 1$

(A)

(B)

(C)



Recursive case
getHeight(r)

$$= 1 + \max(\text{getHeight}(r \rightarrow \text{left}), \text{getHeight}(r \rightarrow \text{right}))$$

Figure out how to compute the height for node r given the height of its left & right subtrees -

Practice problem

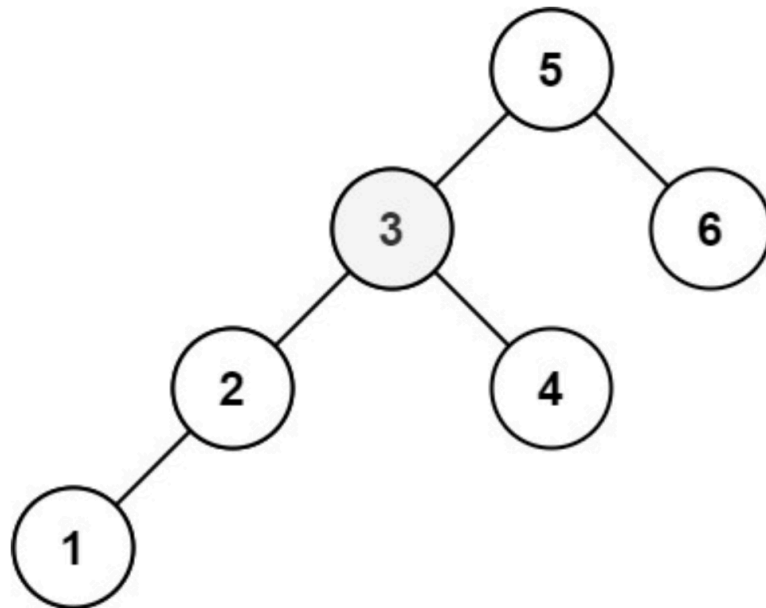
- <https://leetcode.com/problems/kth-smallest-element-in-a-bst/description/>

Input: tree on the right, $k = 3$

Output: 3


Constraints:

- The number of nodes in the tree is n .
- $1 \leq k \leq n \leq 10^4$
- $0 \leq \text{Node.val} \leq 10^4$
-

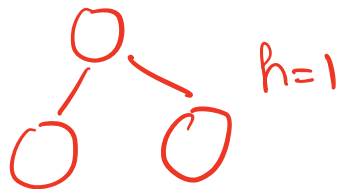
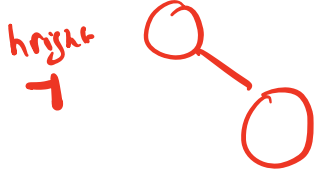
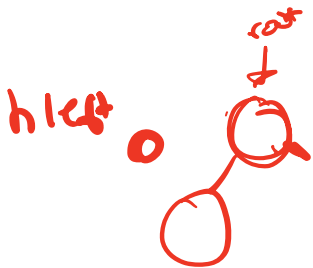


root 

$h = -1$

root 

$h = 0$



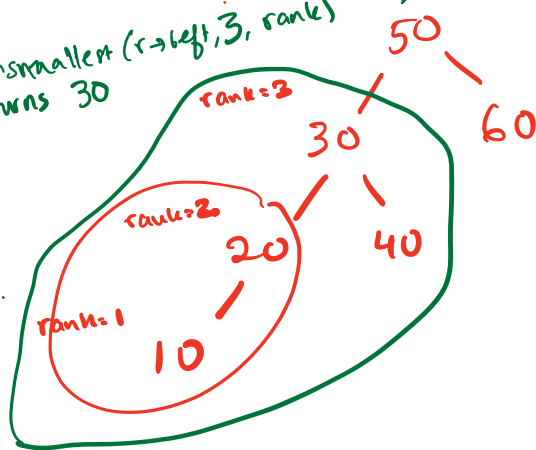
(1)

(2)

(3)

k^{th} smallest value in a BST
Input: bst-root and k value, where $1 \leq k \leq n$
 n : no. of nodes in BST

k^{th} smallest ($r \rightarrow$ left, 3, rank)
returns 30



If rank of a node exceeds k
e.g. in the case of the root (50)
the function should return the
result of the recursive call on
the left subtree of 50

If rank of node is equal to k
return the key of the node
If rank of node is less than k
return the result of recursive call
on the right subtree

Input: root of bst, $k = 3$

Expected output: 30

One approach:
(Visits all n nodes)

Use a modified inorder traversal to populate a
vector, return the element at index $k-1$

Second approach
(Terminates early)

Compute the rank of each node in order
if $\text{rank}(x) > k$, return the result of k^{th} smallest on the left subtree
if $\text{rank}(x) == k$, return $x \rightarrow$ value
if $\text{rank}(x) < k$, return k^{th} smallest on the right subtree