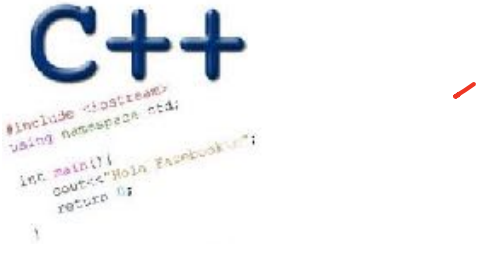
BINARY SEARCH TREES (CONTD)
C++ TEMPLATES

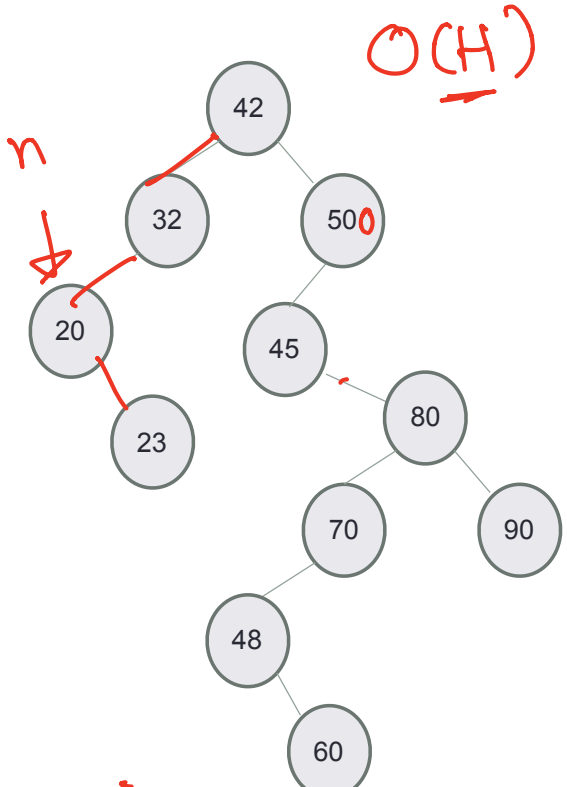
Problem Solving with Computers-II



How is PA02 going?

A. Done
B. On track to finish
C. Having difficulty debugging
D. Stuck
E. Haven't started

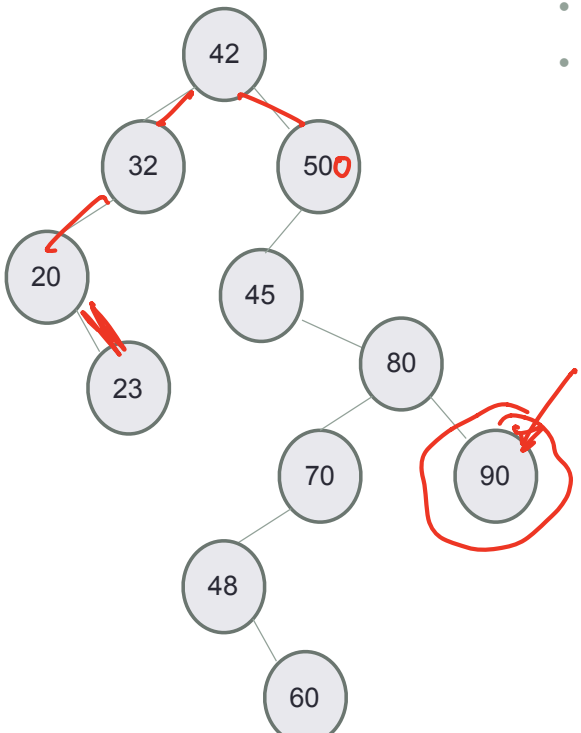
Successor: Next largest element



- What is the successor of 45? 48
- What is the successor of 48? 60
- What is the successor of 60? 70

```
if (n → right) {
  // return min element
     in n's right subtree
  return min(n→right)
} else {
  // traverse parent pointers
     until you find a node
     with key > n→value
}
```

Delete: Case 1: Node is a leaf node



- Set parent's appropriate child pointer to null
- Delete the node

① Set parent's left/right pointer to null

② Delete the node.

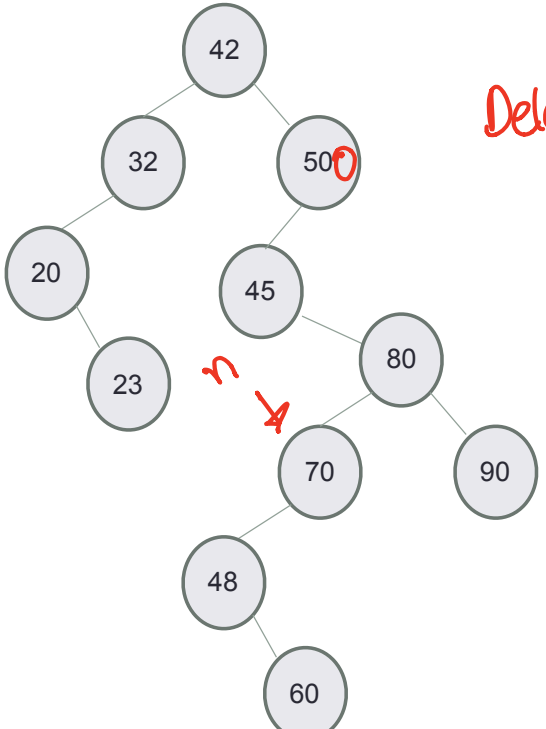
e.g.

Delete 90

```
n->parent->right = 0;
delete n;
```

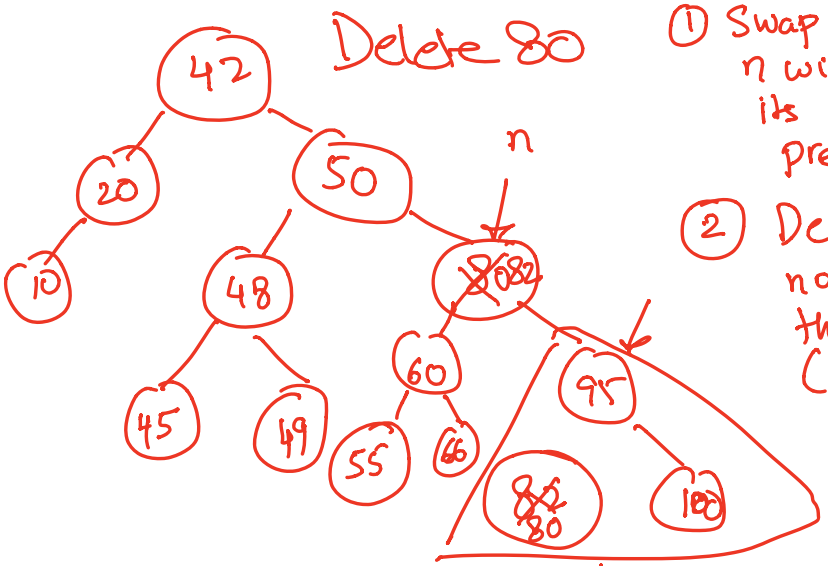
Delete: Case 2 Node has only one child

- Replace the node by its only child



Delete 70

```
if (n->left && n->right==0) {
    n->parent->left = n->left;
    n->left->parent = n->parent
    n->left = 0; // In case using a
                 recursive destructor
    delete n
}
```



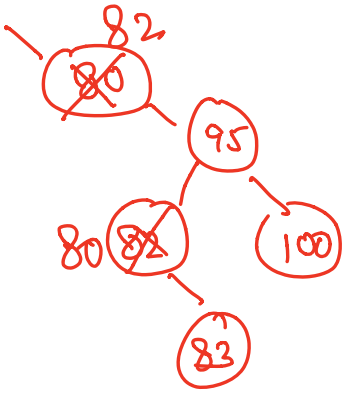
① Swap the value of n with that of its successor/predecessor — O(H) + O(1) = O(H)

② Delete the node (80) at the new location (prev location of successor) → O(1)

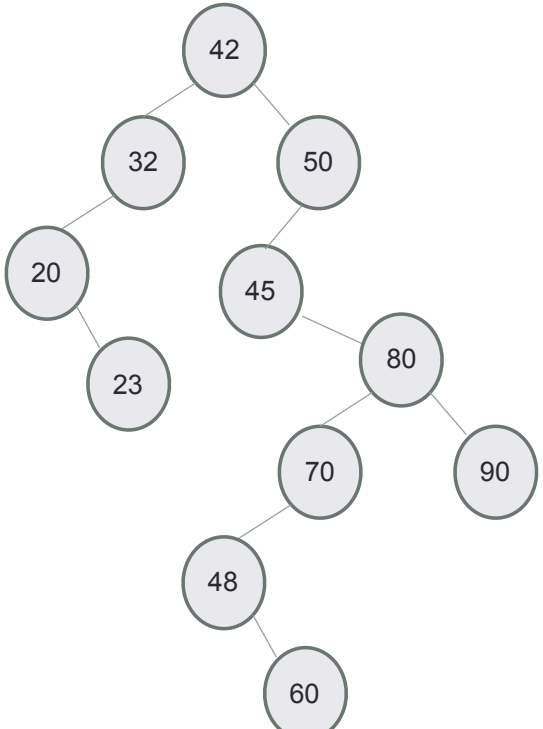
O(H)

Successor of 80 will have no more than 1 child because it is the leftmost node in 80's right subtree

Fallback to case 1 or case 2



Delete: Case 3 Node has two children

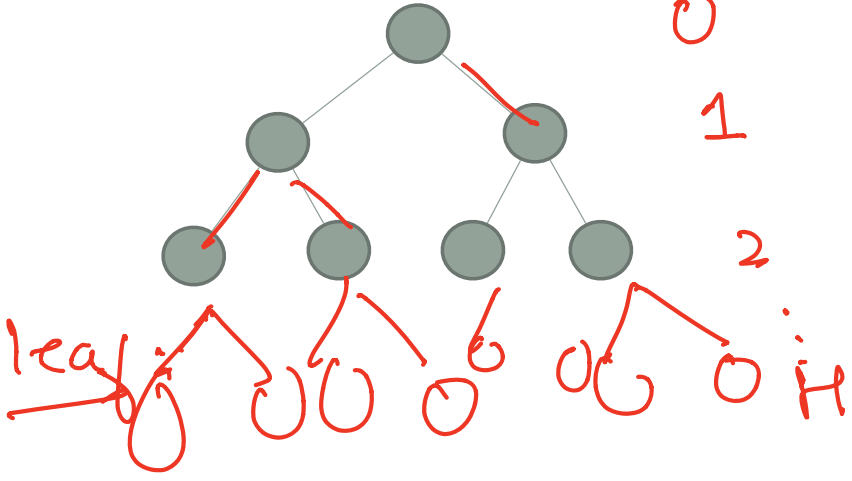


- Can we still replace the node with one of its children? Why or Why not?

Completely filled BSTs

Balanced tree

$H = O(\log_2 N)$



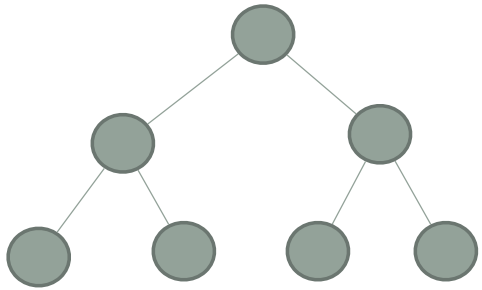
N : Total # of nodes

Relating H (height) and N (#nodes)
find is O(H), we want to find a f(N) = H

$$\sum_{i=0}^{H} 2^i = N$$

$$2^{H+1} - 1 = N$$

$$H = \log(N+1) + 1$$



1
2
⋮
2^L

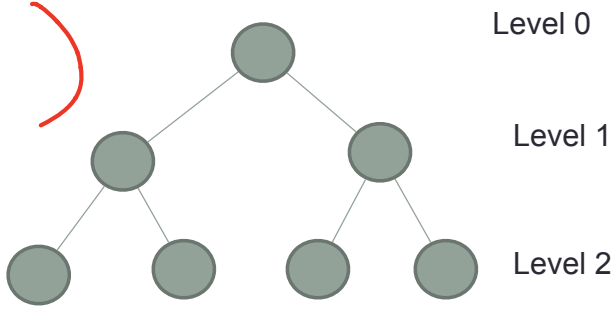
How many nodes are on level L in a completely filled binary search tree?

A.2
B.L
C.2**L
D.2^L

$$O(H) = O(\log_2 N)$$

Relating H (height) and N (#nodes)
find is O(H), we want to find a f(N) = H

$O(H) = O(\log_2 N)$

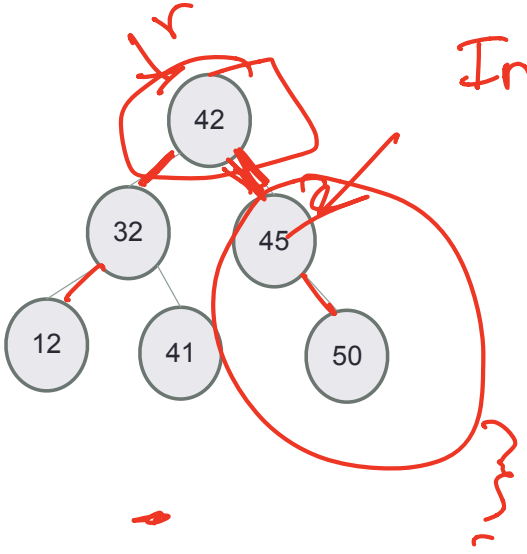


Finally, what is the height (exactly) of the tree in terms of N?

$H = \log(N+1) + 1$

And since we knew finding a node was O(H), we now know it is $O(\log_2 N)$

In order traversal: print elements in sorted order

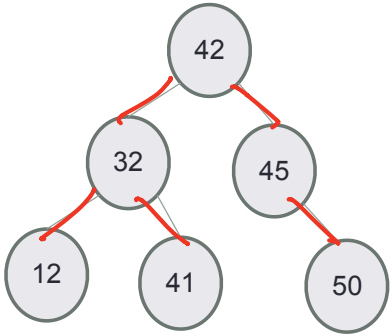



```
InOrder ( Node * r) {
    if (r) {
        InOrder (r→left);
        cout << r→value;
        InOrder (r→right);
    }
}
```

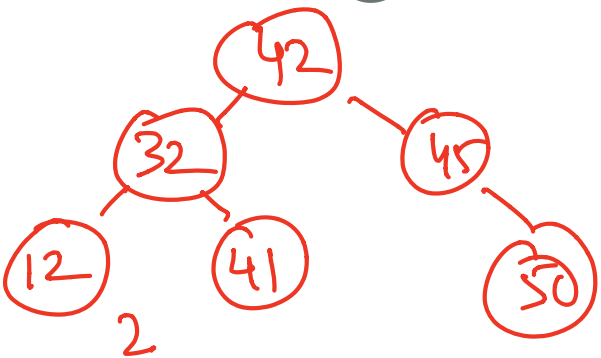
10 12 20 32 41 42 45 50

10 12 20 32 41 42 45 50

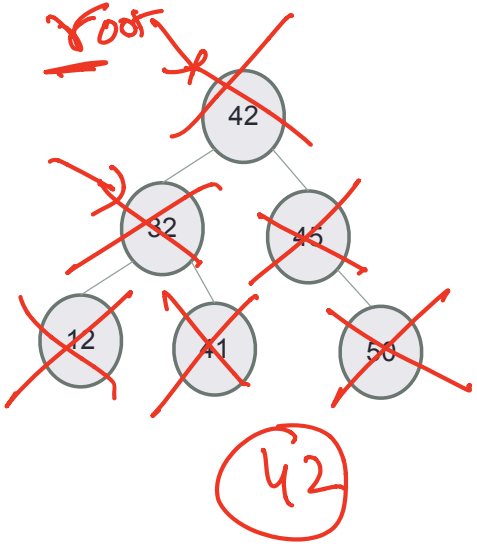
Pre-order traversal: nice way to linearize your tree!




```
PreOrder (Node * n) {
  if (n) {
    cout << n->value;
    PreOrder (n->left);
    PreOrder (n->right);
  }
}
```



Post-order traversal: use in recursive destructors!




```
PostOrder (Node * n) {
  if (n) {
    PostOrder(n->left);
    PostOrder(n->right);
    cout << n->value;
  }
}
```

delete root

```
~Node() {
  delete left;
  delete right;
}
```

Sorted arrays, linked-lists, Balanced Binary Search Trees

(unsorted)

Operations	Sorted Array	Balanced BST	Linked list
Min	O(1)	O(log N)	O(N)
Max	O(1)	"	O(N)
Successor	O(1)	"	O(N)
Predecessor	O(1)	"	O(N)
Search	O(log N)	"	O(N)
Insert	O(N)	"	O(1)
Delete	O(N)	"	O(N)
Print elements in order	O(N)	O(N)	O(N)