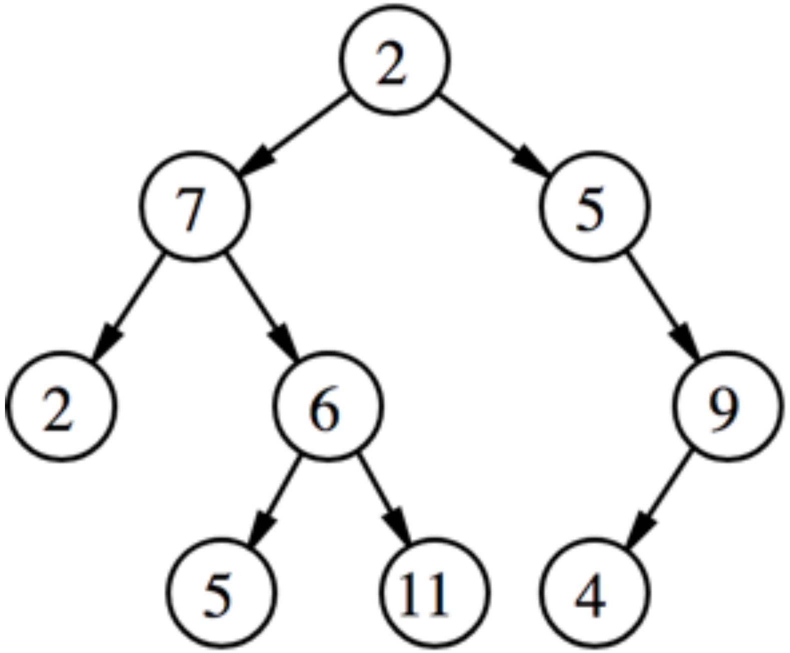
BINARY SEARCH TREES

Problem Solving with Computers-II

Sorted arrays, Balanced Binary Search Trees

Operations	**Sorted Array**	**Balanced BST**
Min		
Max		
Successor		
Predecessor		
Search		
Insert		
Delete		
Print elements in order		

Trees



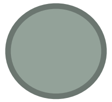
A tree has following general properties:

- One node is distinguished as a **root**;
- Every node (exclude a root) is connected by a directed edge *from* exactly one other node;

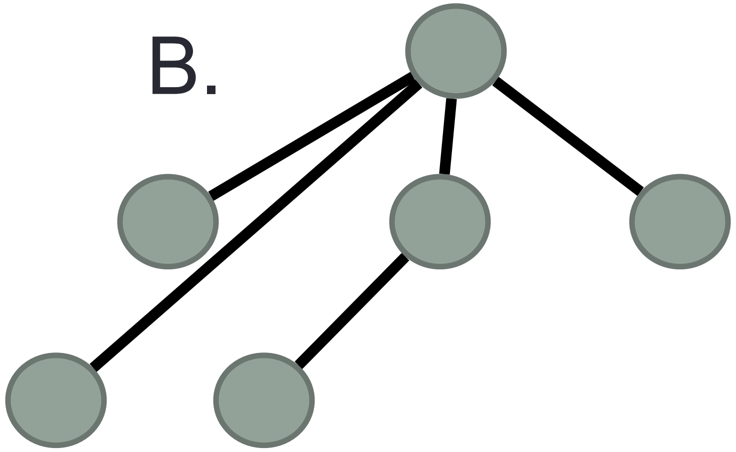
  A direction is: *parent -> children*

Which of the following is/are a tree?

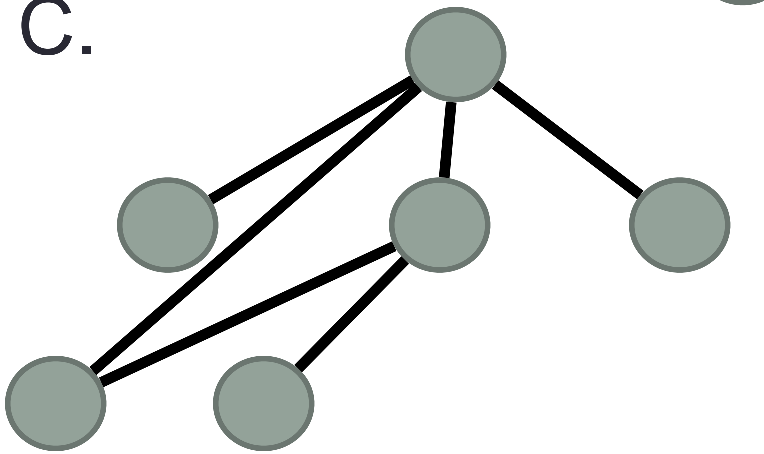
A.



B.



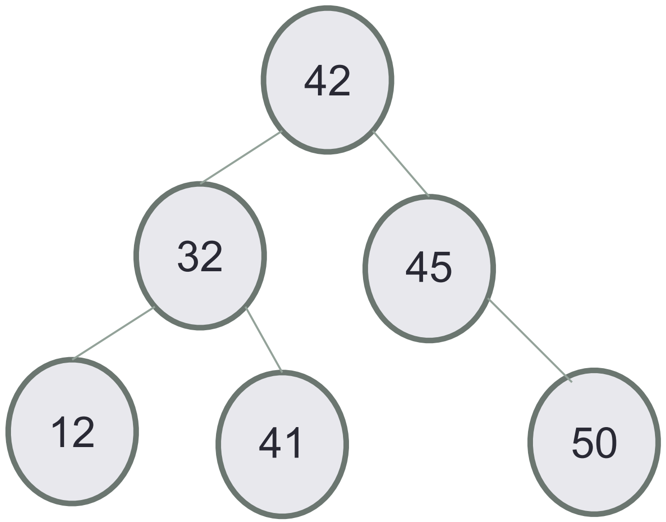
C.



D. A & B

E. All of A-C

Binary Search Tree – What is it?



- Each node:
  - stores a key (k)
  - has a pointer to left child, right child and parent (optional)
  - Satisfies the Search Tree Property

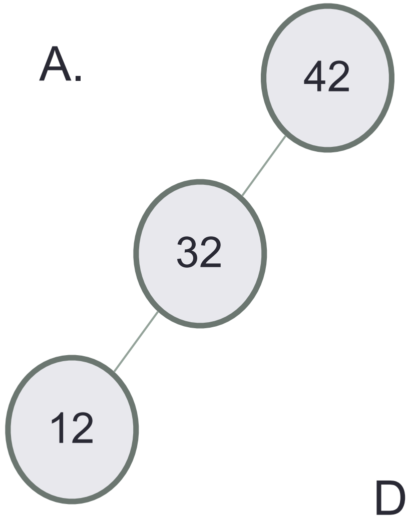
Do the keys have to be integers?

A node in a BST

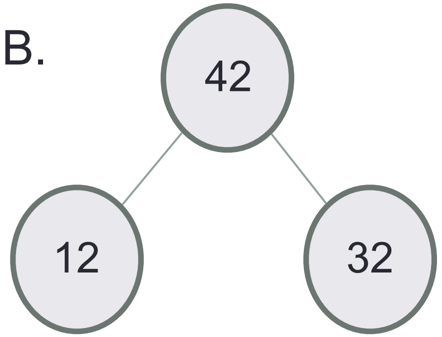
```
class BSTNode {

public:
   BSTNode* left;
   BSTNode* right;
   BSTNode* parent;
   int const data;

   BSTNode( const int & d ) : data(d) {
      left = right = parent = 0;
   }
};
```

Which of the following is/are a binary search tree?

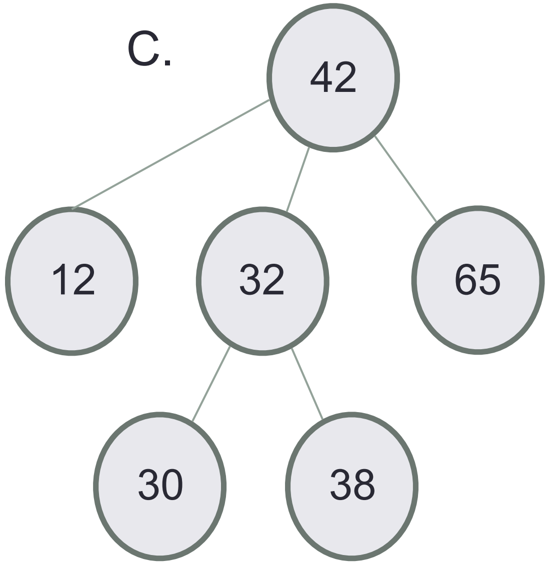
A.



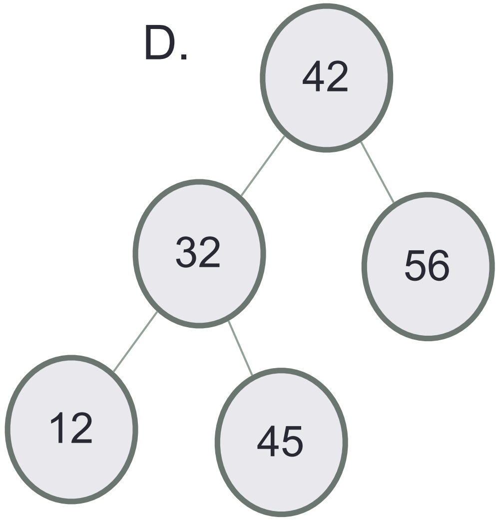
B.



C.

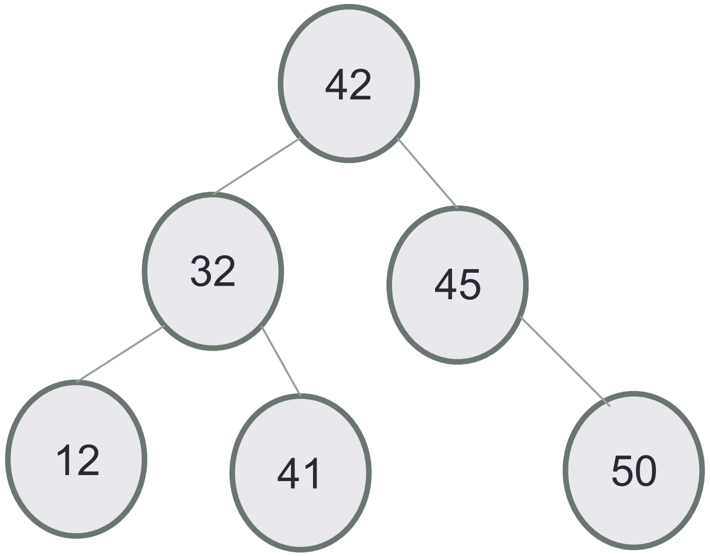


D.



E. More than one of these

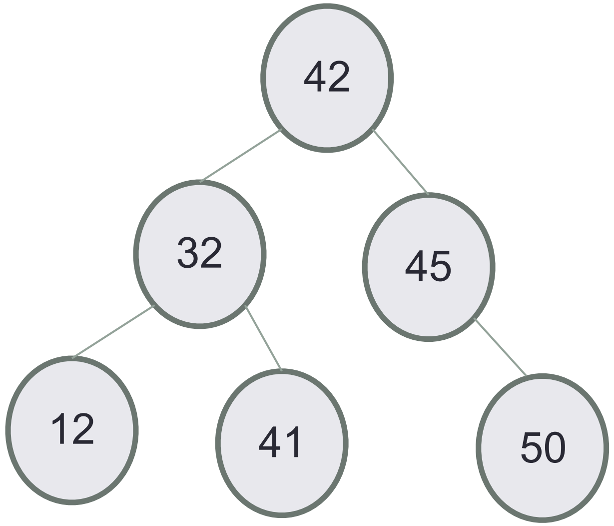
BSTs allow efficient search!



- Start at the root; trace down a path
  by comparing **k** with the key of the current node x:
  - If the keys are equal: we have found the key
  - If **k** < key[x] search in the left subtree of x
  - If **k** > key[x] search in the right subtree of x

Search for 41, then search for 53

Search



How many edges need to be traversed to search for 40?

A. One
B. Two
C. Three
D. Four

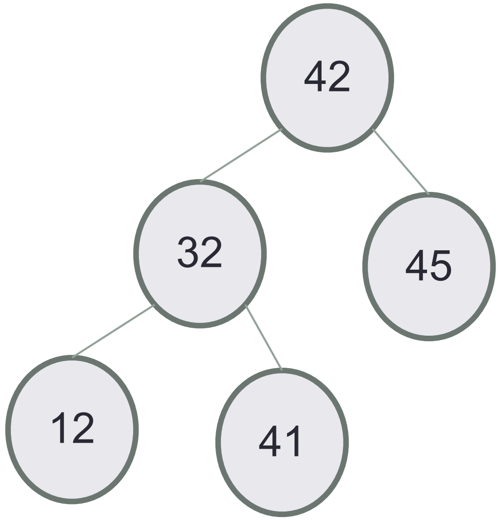
How fast is the BST search algorithm?

Many different BSTs are possible for the same set of keys

Examples for keys: 12, 32, 41, 42, 45

How fast is BST search algorithm?

The complexity of search depends on the HEIGHT of the BST!



Height of a node: the height of a node is the number of edges on the longest path from the node to a leaf

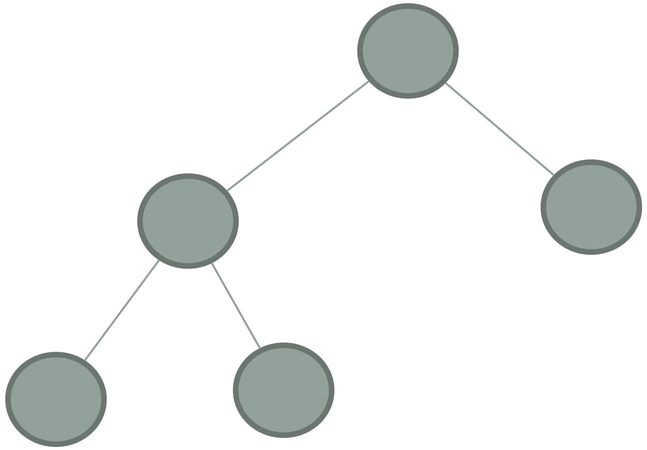
Height of a tree: the height of the root of the tree

Height of this tree is 2.

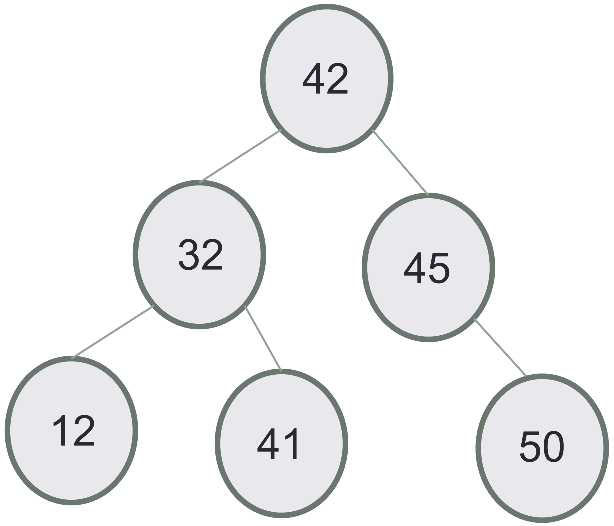
Worst case analysis

Are binary search trees *really* faster than linked lists for finding elements?

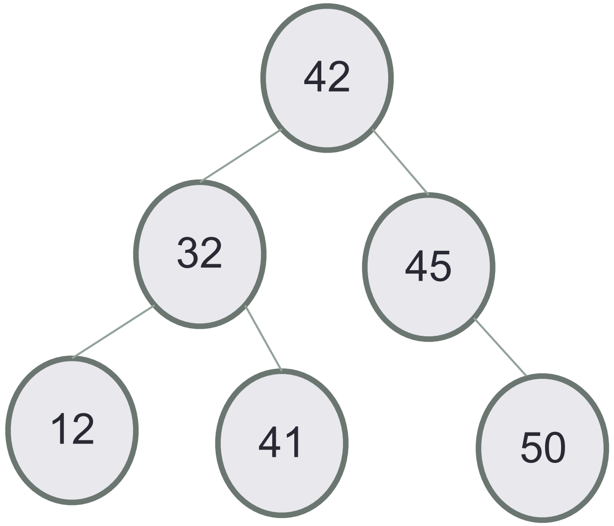
- A. Yes
- B. No



Balanced BSTs



Insert



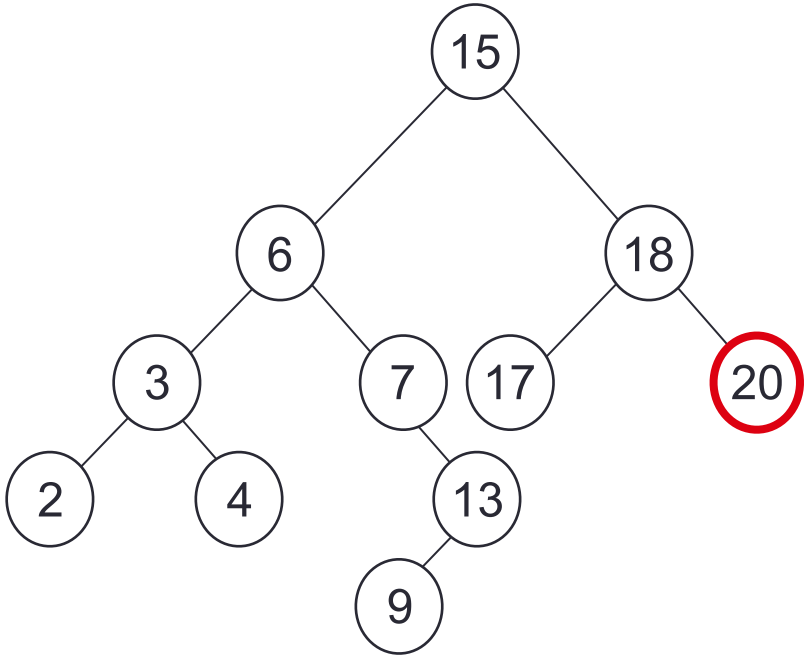
- Insert 40
- Search for the key
- Insert at the spot you expected to find it

Max

Goal: find the maximum value in a BST

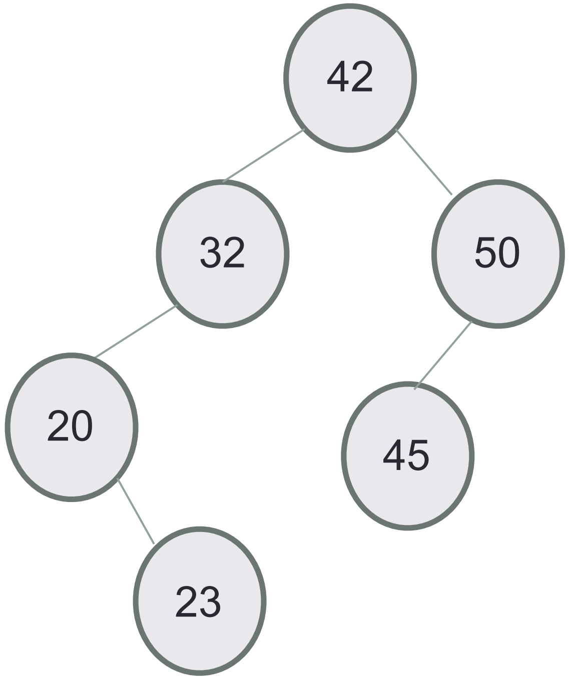
Following right child pointers from the root, until a leaf node is encountered

Alg: `int BST::max()`



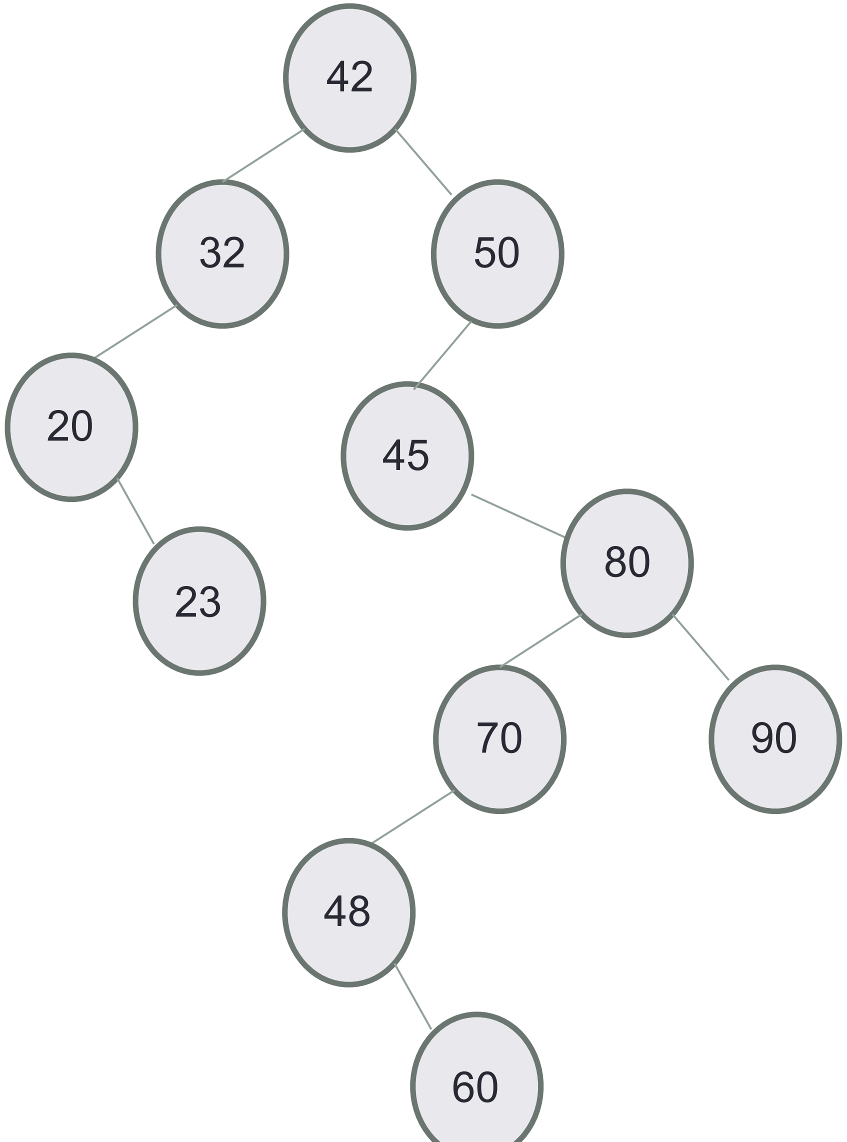
Maximum = 20

Predecessor: Next smallest element



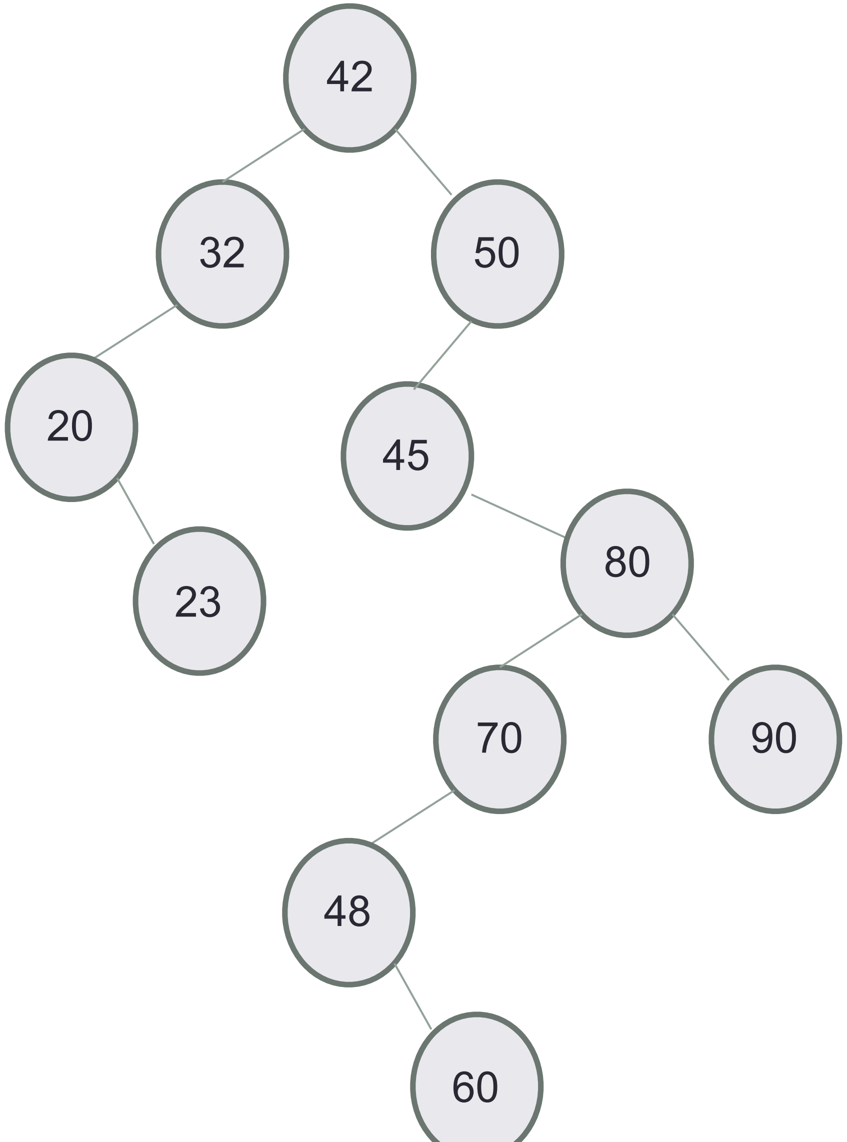
- What is the predecessor of 32?
- What is the predecessor of 45?

Successor: Next largest element



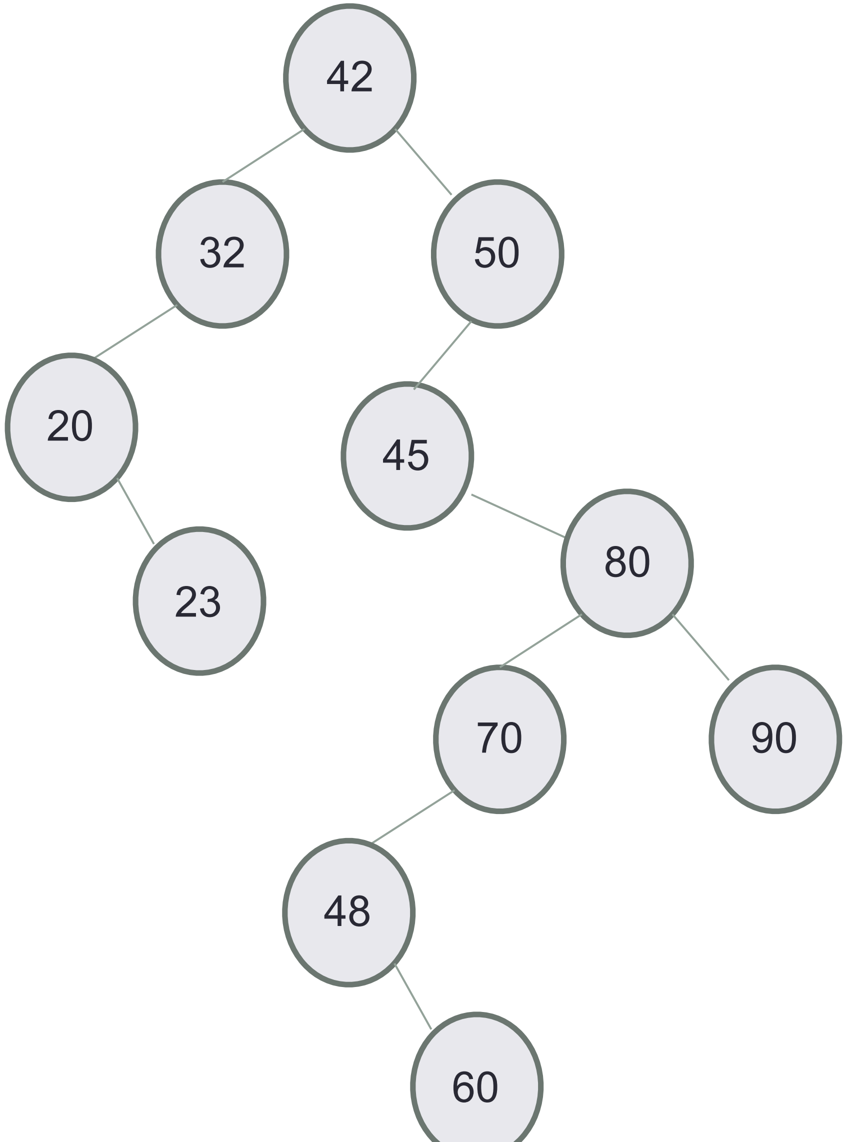
- What is the successor of 45?
- What is the successor of 48?
- What is the successor of 60?

Delete: Case 1: Node is a leaf node



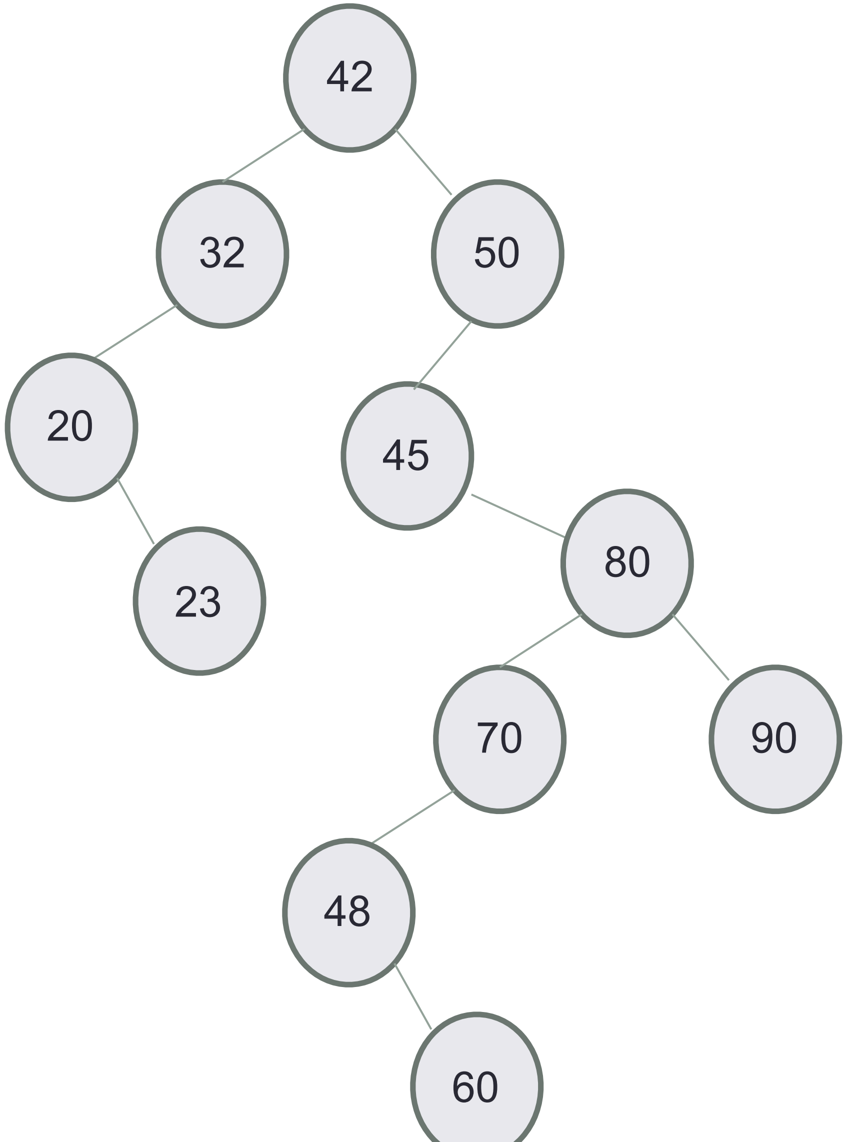
- Set parent's appropriate child pointer to null
- Delete the node

Delete: Case 2 Node has only one child



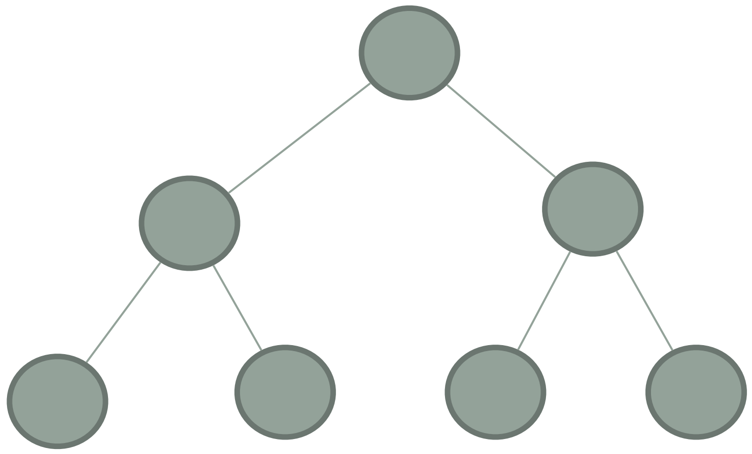
- Replace the node by its only child

Delete: Case 3 Node has two children

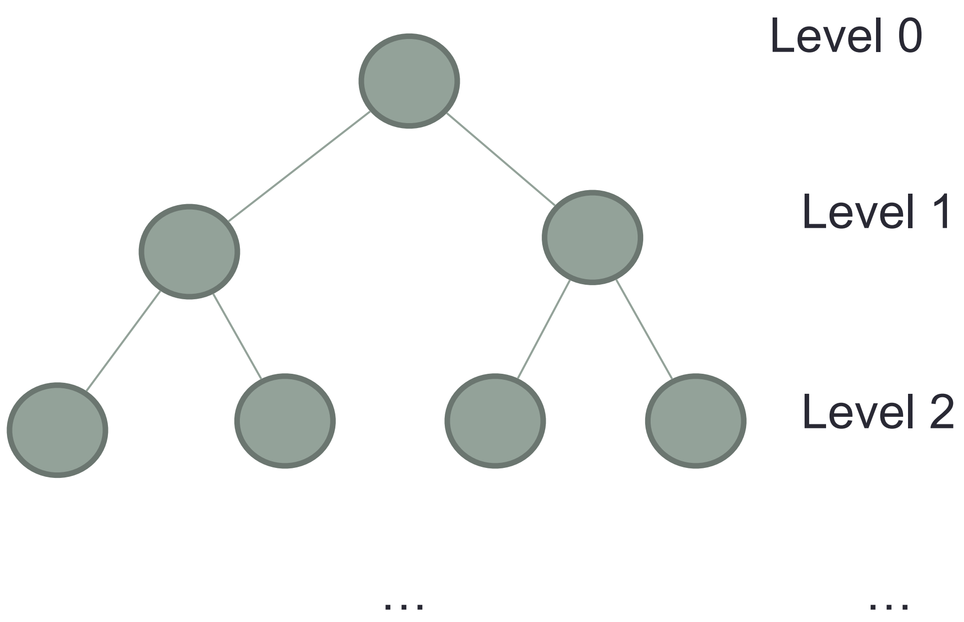


- Can we still replace the node by one of its children? Why or Why not?

Completely filled BSTs



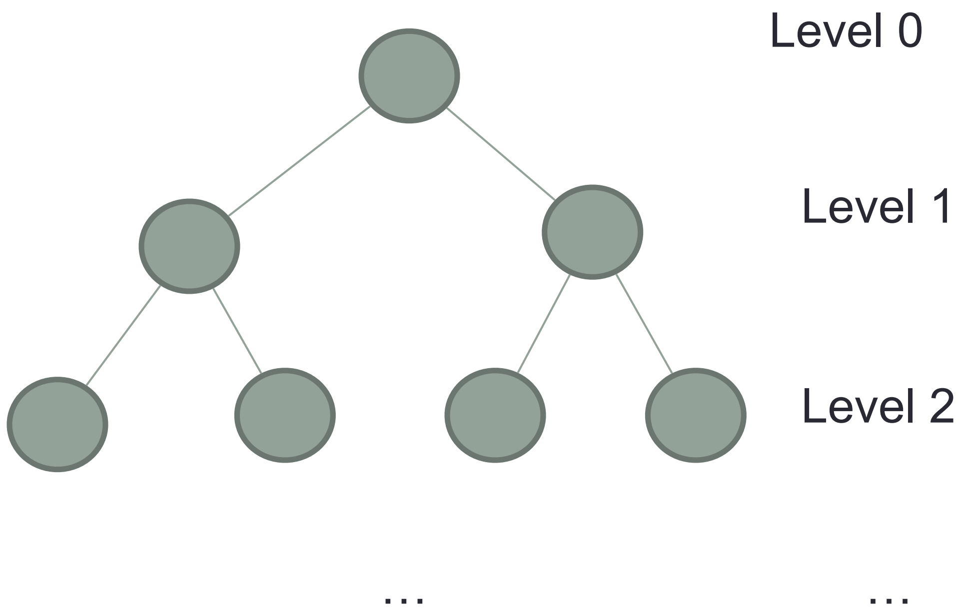
Relating H (height) and N (#nodes)
find is O(H), we want to find a f(N) = H



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

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

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



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

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

Sorted arrays, linked-lists, Balanced Binary Search Trees

Operations	Sorted Array	Balanced BST	Linked list
Min			
Max			
Successor			
Predecessor			
Search			
Insert			
Delete			
Print elements in order			