RUNNING TIME ANALYSIS OF BINARY SEARCH TREES

Problem Solving with Computers-II

How is PA02 going?

A. Done!
B. On track to finish
C. On track to finish but my code is a mess
D. Stuck and struggling
E. Haven’t started

Midterm – Monday 2/25

- Cumulative but the focus will be on
  - BST
  - running time analysis
  - use of the C++ STL

Review Big O

- What does $f(n) = O(g(n))$ really mean?

Binary Search Trees

- WHAT are the operations supported?
- HOW do we implement them?
- WHAT are the (worst case) running times of each operation?

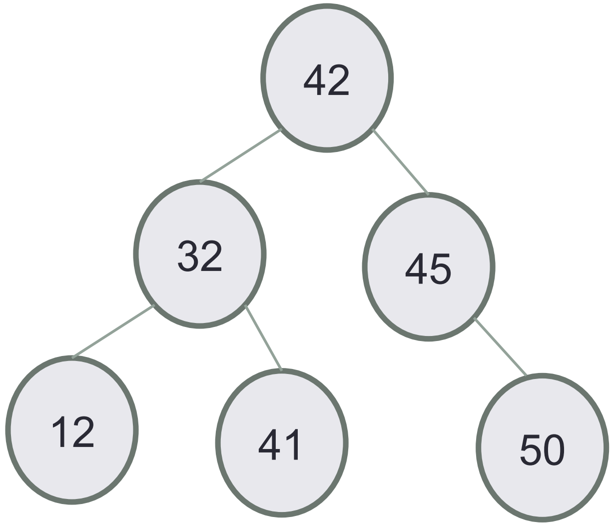
Visualize BST operations: https://visualgo.net/bn/bst

Height of the tree

Many different BSTs are possible for the same set of keys
Examples for keys: 12, 32, 41, 42, 45

- Path – a sequence of nodes and edges connecting a node with a descendant.
- A path starts from a node and ends at another node or a leaf
- Height of node – The height of a node is the number of edges on the longest downward path between that node and a leaf.

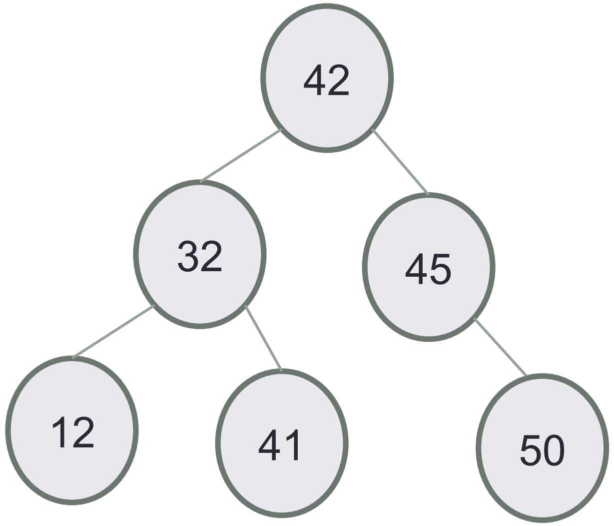
Worst case Big-O of search



- Given a BST of height H and N nodes, what is the worst case complexity of searching for a key?

A. O(1)
B. O(log N)
C. O(H)
D. O(log H)

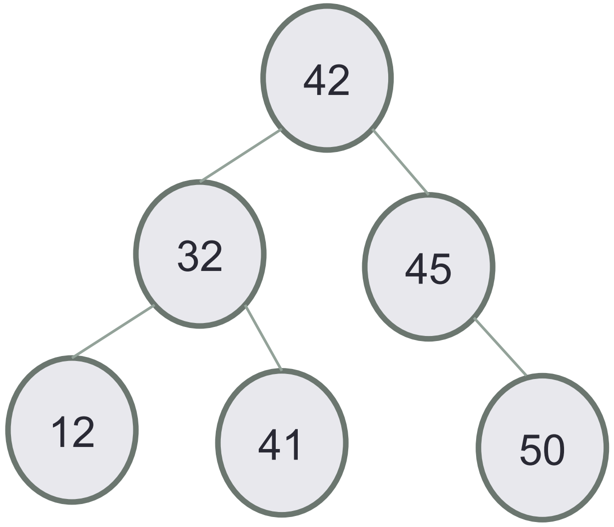
Worst case Big-O of insert



- Given a BST of height H and N nodes, what is the worst case complexity of inserting a key?

A. O(1)
B. O(log N)
C. O(H)
D. O(log H)

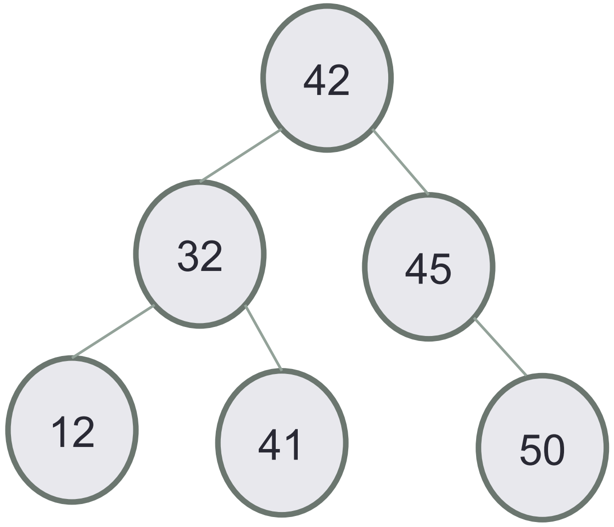
Worst case Big-O of min/max



- Given a BST of height H and N nodes, what is the worst case complexity of finding the minimum or maximum key?

A. O(1)
B. O(log N)
C. O(H)
D. O(log H)

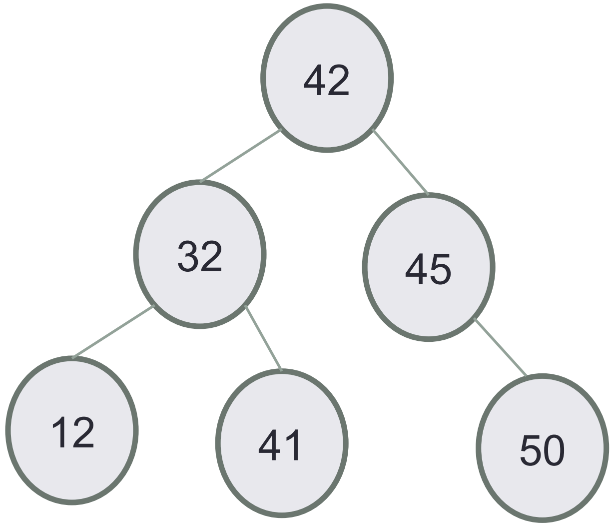
Worst case Big-O of predecessor/successor



- Given a BST of height H and N nodes, what is the worst case complexity of finding the minimum or maximum key?

A. O(1)
B. O(log N)
C. O(H)
D. O(log H)

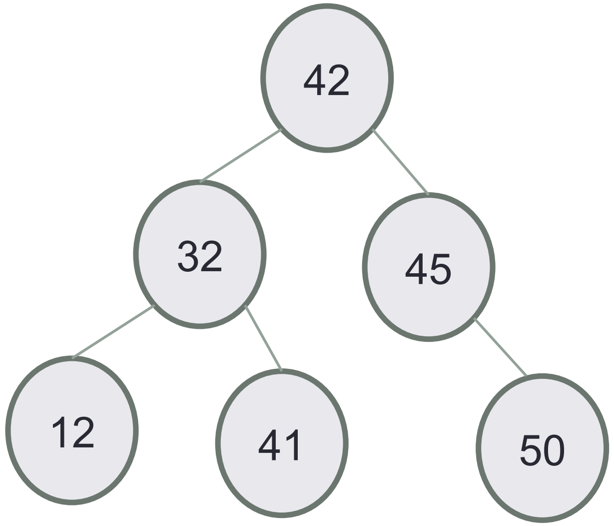
Worst case Big-O of delete



- Given a BST of height H and N nodes, what is the worst case complexity of deleting the key (assume no duplicates)?

A. O(1)
B. O(log N)
C. O(H)
D. O(log H)

Big O of traversals



In Order:

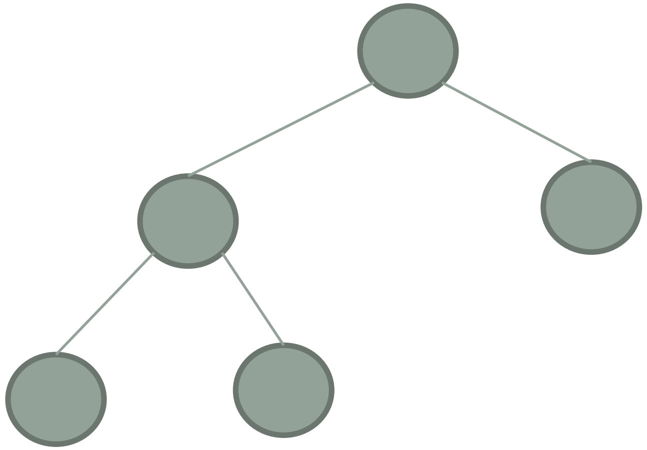
Pre Order:

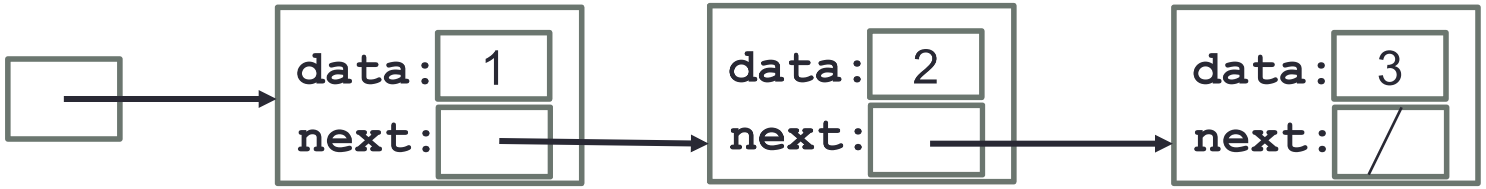
Post Order:

Worst case analysis

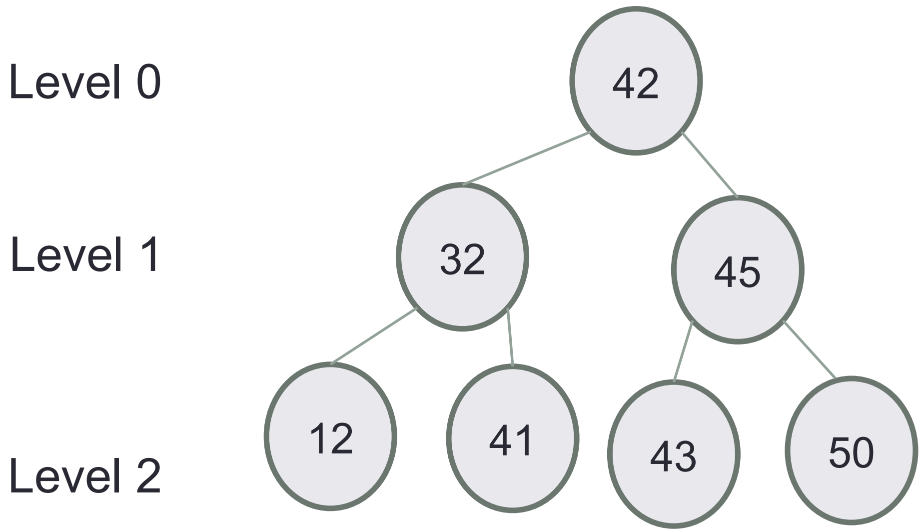
Are binary search trees *really* faster than linked lists for finding elements?

- A. Yes
- B. No



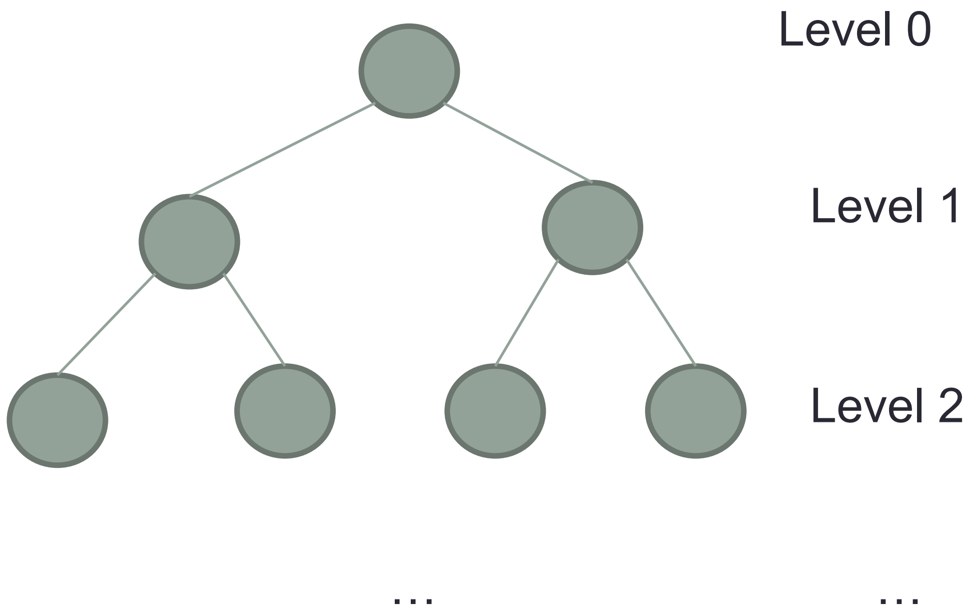


Completely filled binary tree



Nodes at each level have exactly two children, except the nodes at the last level

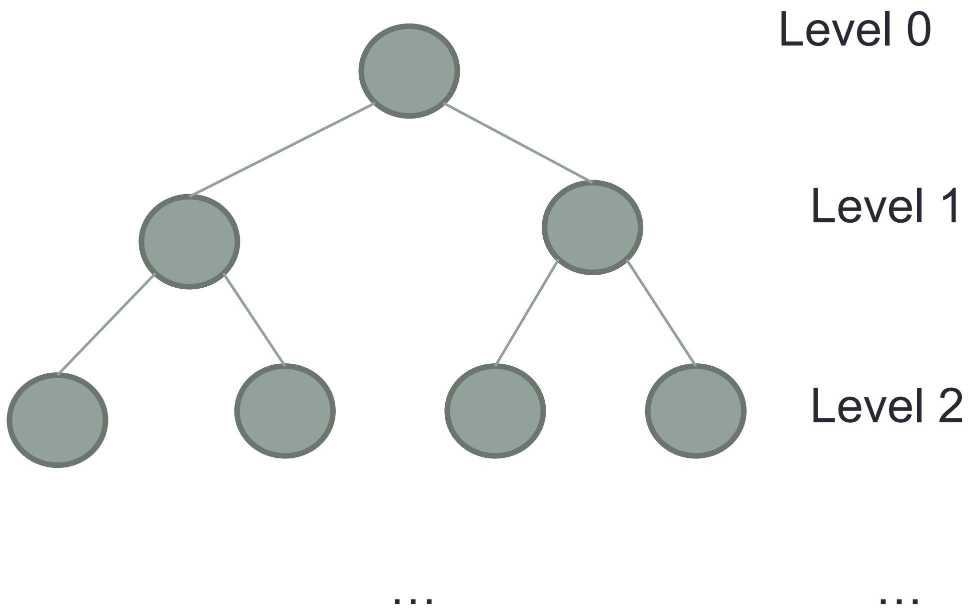
Relating H (height) and N (#nodes)
find is O(H), we want to find a f(N) = H



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?

Balanced trees

- Balanced trees by definition have a height of O(log N)
- A completely filled tree is one example of a balanced tree
- Other Balanced BSTs include AVL trees, red black trees and so on
- Visualize operations on an AVL tree: https://visualgo.net/bn/bst

Summary of operations

Operation	Sorted Array	Binary Search Tree	Linked List
Min			
Max			
Median			
Successor			
Predecessor			
Search			
Insert			
Delete			

CHANGING GEARS: C++STL

- The C++ Standard Template Library is a very handy set of three built-in components:
  - Containers: Data structures
  - Iterators: Standard way to search containers
  - Algorithms: These are what we ultimately use to solve problems

C++ STL container classes

```
array
vector
forward_list
list
stack
queue
priority_queue
set
multiset (non unique keys)
deque
unordered_set
map
unordered_map
multimap
bitset
```