RUNNING TIME ANALYSIS OF BINARY SEARCH TREES

Problem Solving with Computers-II

Big O → Was it weird??

$$f(n) = 10\,N^2 \log N + 5N + 2000$$

no. of primitive operations (steps)

We said that Big-O of f(n) is an "upper bound" on how fast f(n) grows... but then to find the Big O we proceeded to drop lower order terms and constants..... to get:

$$= O(N^2 \log N)$$

how is this an "upper bound" if we dropped the other two terms $(5N + 2000)$ and the constant coefficient 10

This is where you have to understand the rather nuanced definition of Big O. When we say $f(n) = O(g(n))$ we mean $f(n)$ is bounded by $c \cdot g(n)$, for $n > k$ where c & k are some constants.

If $f(n) = 10N^2 \log N + 5N + 2000$

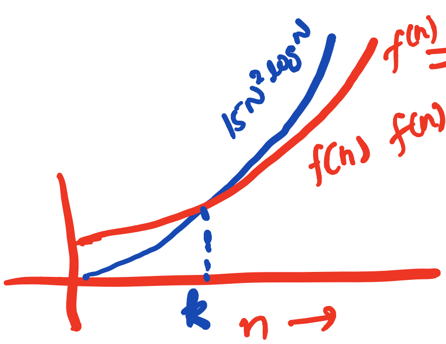
Can we show that $f(n) < c \cdot N^2 \log N$ for some constant c?

$$f(n) = 10N^2 \log N + 5N + 2000$$

$$f(n) < 10N^2 \log N + 5N^2 \log N + 2000$$

$$f(n) = 15N^2 \log N + 2000 \text{, for all } N$$

$$f(n) < 15N^2 \log N \text{, for large enough } N$$



So Big O says that f(n) grows no faster than a constant times g(n) for a large enough n!!

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

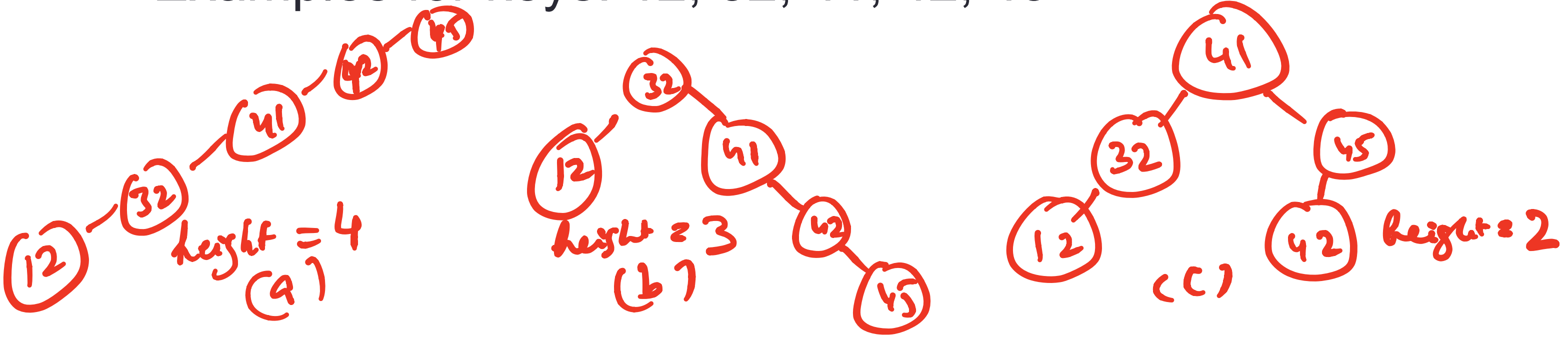
- What does f(n) = O(g(n)) really mean?

See first slide

$f(n) < c \cdot g(n)$ for a large enough n

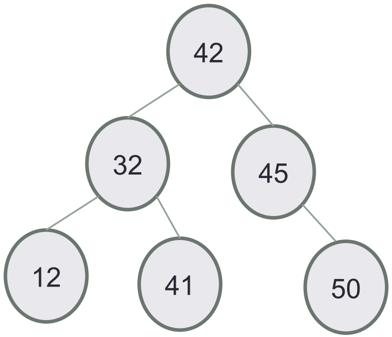
Height of the tree

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



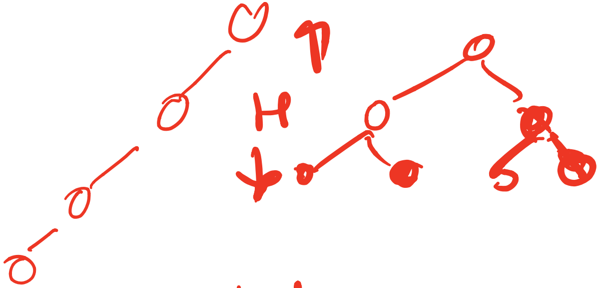
- Path – a sequence of nodes and edges connecting a node with a descendant.
- A path starts from a node and ends at another node or a leaf
- Height of node – The height of a node is the number of edges on the longest downward path between that node and a leaf.

Worst case Big-O of search



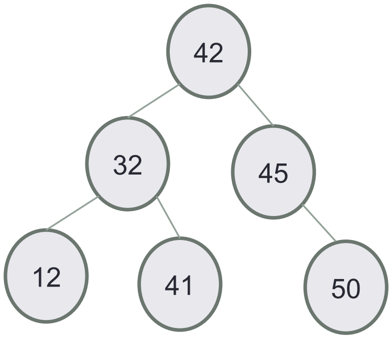
- Given a BST of height H and N nodes, what is the worst case complexity of searching for a key?

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



H = N-1

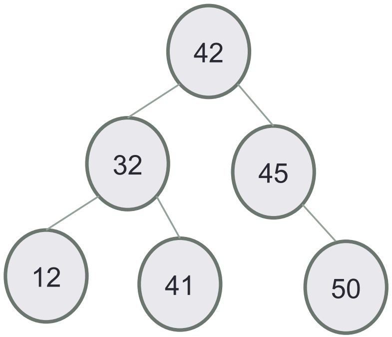
Worst case Big-O of insert



- 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?

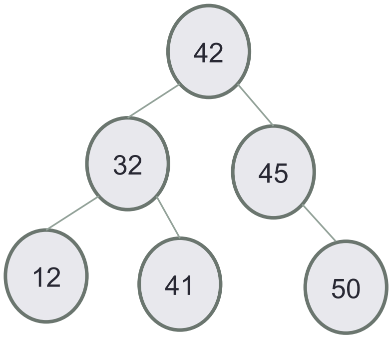
A. O(1)
B. O(log N)
C. O(H)
D. O(log H)

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

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?

pred or succ

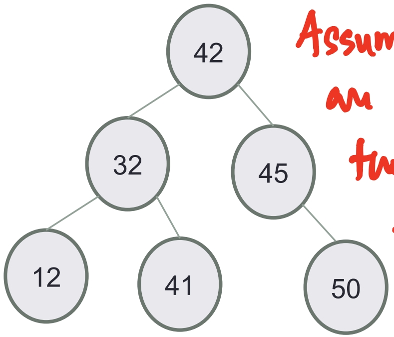
A. O(1)

B. O(log N)

C. O(H)

D. O(log H)

Worst case Big-O of delete



Assume we are not including the cost to find the key!

Case1: deleting a leaf node
O(1)

Case 2: one child
O(1)

Case 3: two children
- Find succer/pred O(H)
- swap O(1)
- delete → case1 or case2 O(1)

$\Big\}$ → O(H)

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

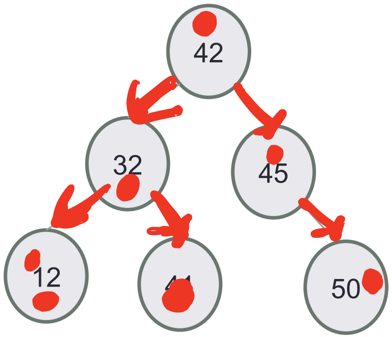
A. O(1)

B. O(log N)

C. O(H) → This is the running time even if we include the cost to find the key

D. O(log H)

Big O of traversals



In Order: $O(N)$

Pre Order: $O(N)$

Post Order: $O(N)$

```
Inorder ( n) {
    Inorder (n->left)
    cout<< n->data;
    Inorder (n->right)
}
```

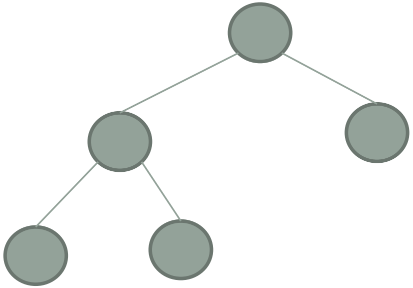
Inorder is called on each node only once!

N nodes $\rightarrow$ N calls

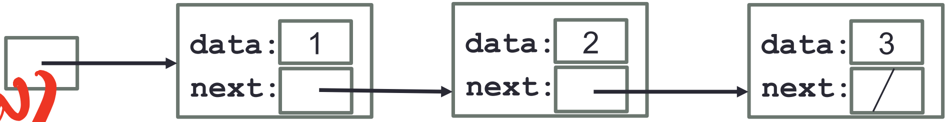
Worst case analysis

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

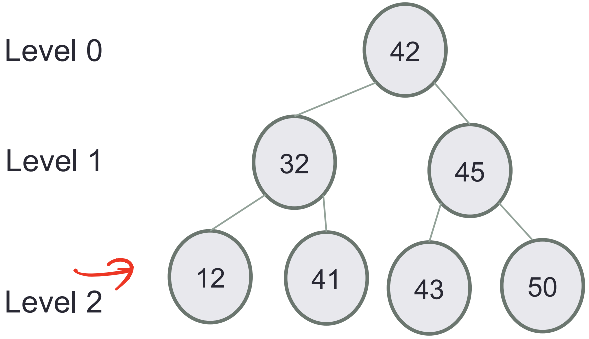
- A. Yes
- B. No



min – O(h)
next O(h)
insert O(h)
del
pred
succ
printing O(n)



Completely filled binary tree



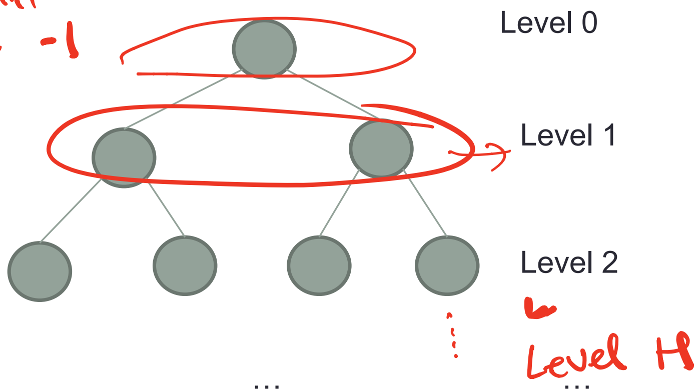
Nodes at each level have exactly two children, except the nodes at the last level

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

$2^0 + 2^1 + 2^2 \ldots\ldots 2^H = 2^{H+1} - 1$

$2^0 + 2^1 + 2^2 + \ldots\ldots 2^7 =$
$2^8 - 1$

Sum the nodes at each level to relate H with N



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

A. 2

B. L

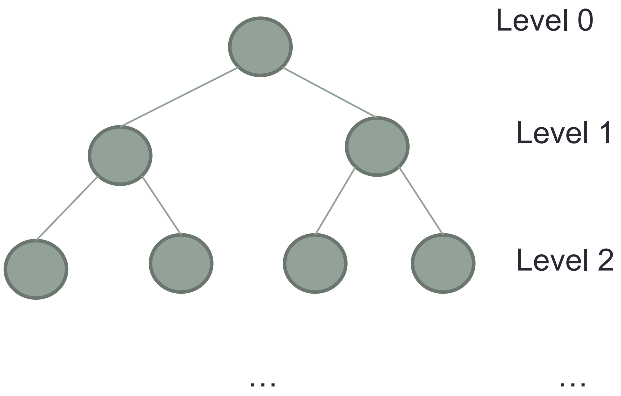
C. 2*L

D. 2^L

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

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

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?

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

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

Balanced! Double unsorted

Operation	Sorted Array	Binary Search Tree	Linked List
Min	O(1)	O(log N)	O(N)
Max	O(1)	"	"
Median	"	—	"
Successor	"	O(log N)	"
Predecessor	"	"	"
Search	O(log N)	"	"
Insert	O(N)	"	O(1)
Delete	O(N)	"	O(1)

(don't include time to find)

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
```