BINARY SEARCH TREES (CONTD)

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

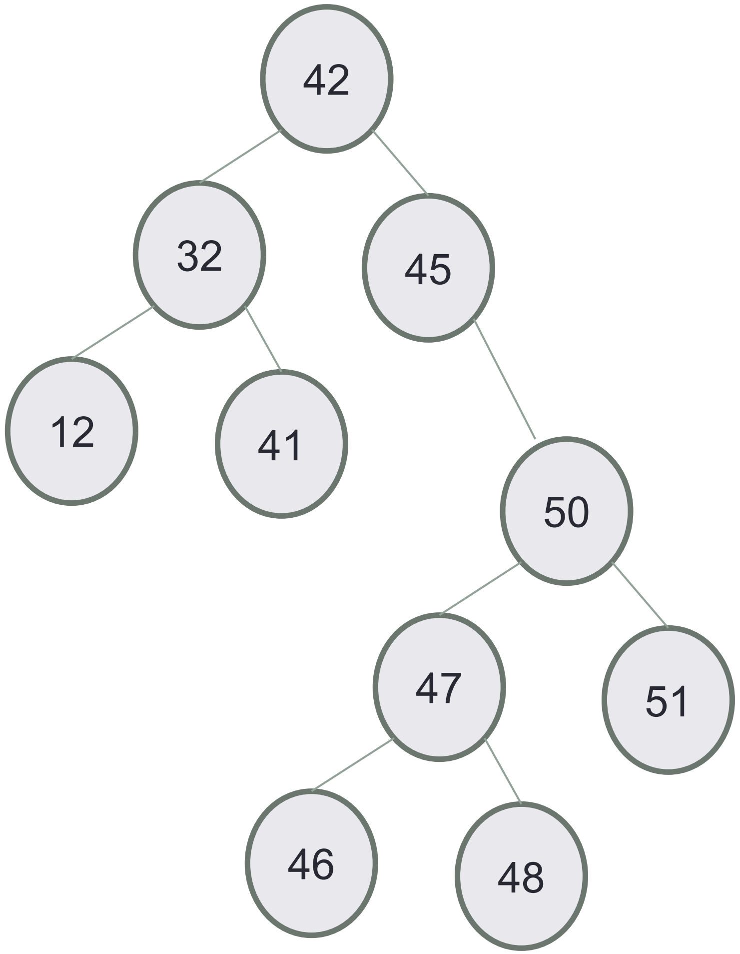
Midterm – Monday 2/26

- We will have a review session (by our TA Jack) on Friday 1-5p in Phelps 3526
- Which of the following review session times work best for you?

A. 1 - 3p
B. 3p - 5p
C. Neither

Please post the topics that you need help with on Piazza.

Binary Search Tree – Review



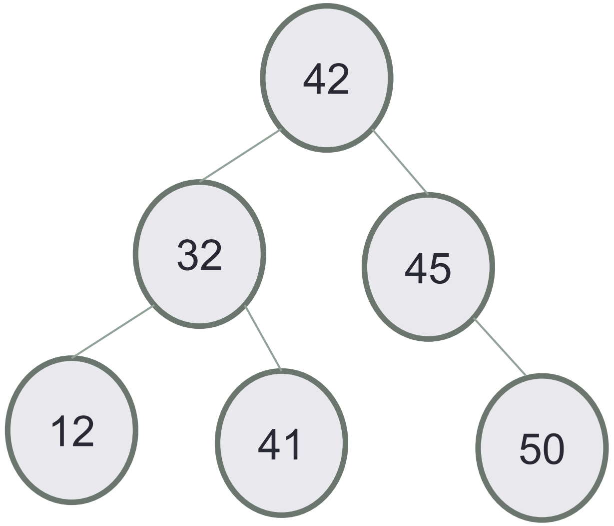
- Rooted binary tree
- Each node:
  - stores a key
  - has a pointer to left child, right child and parent (optional)
  - Satisfies the Search Tree Property

Height of the tree

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

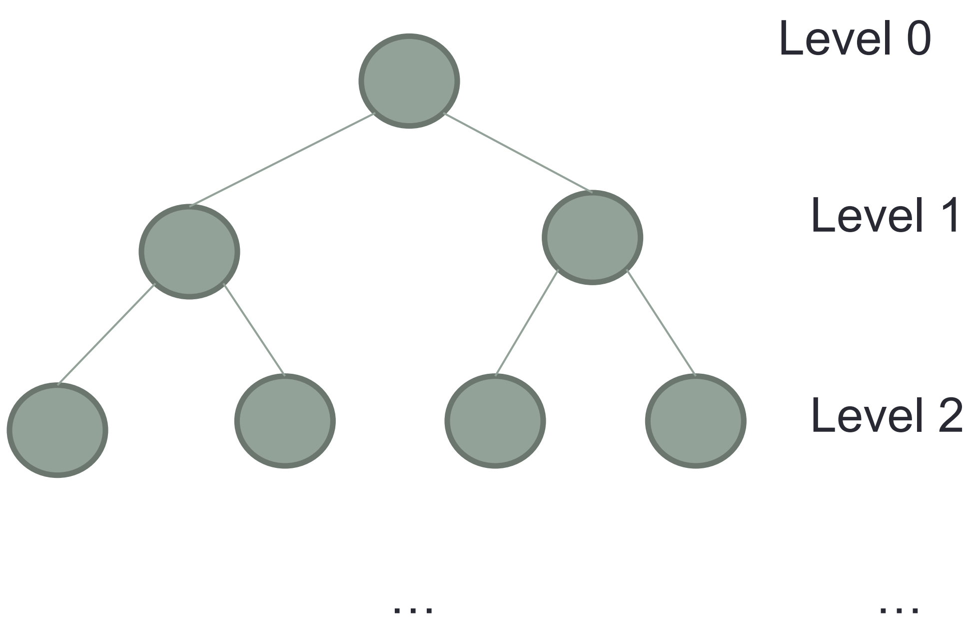
- *What is the height in each case?*
- *How do we define the height of the tree?*

Balanced trees



- What is a balanced tree?

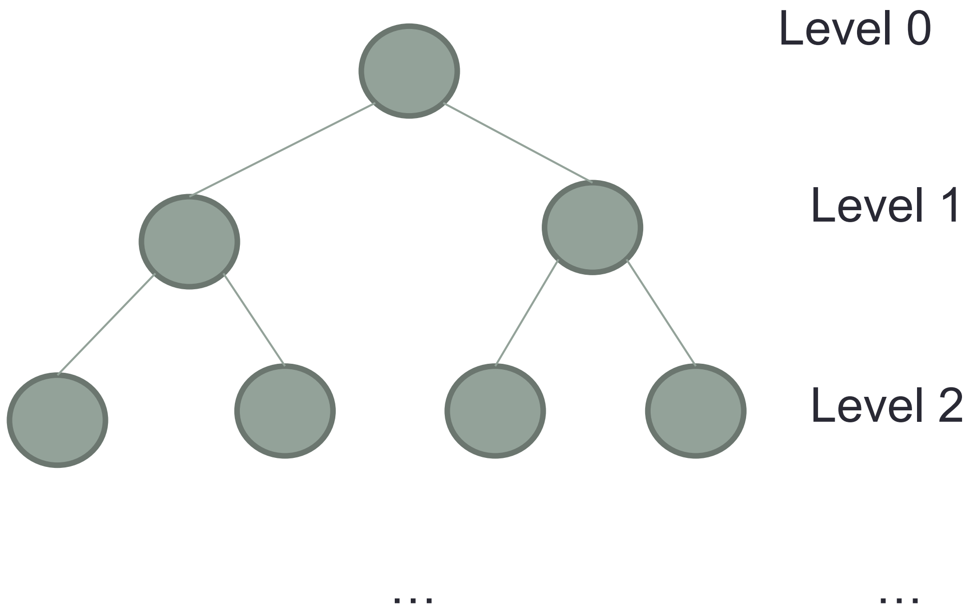
Relating H (height) and N (#nodes)
find is O(H), we want to find a f(N) = H



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

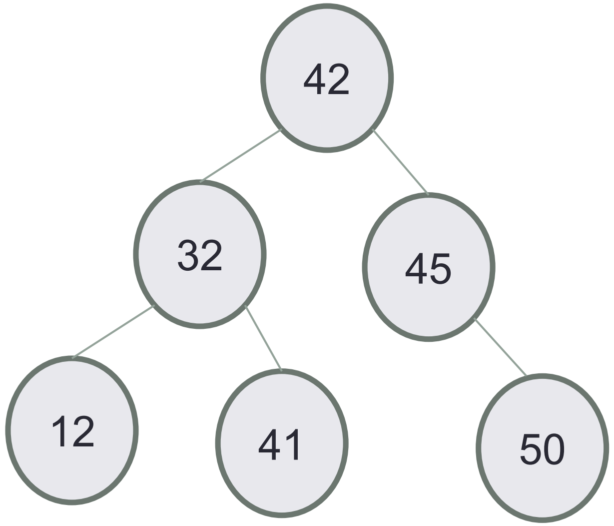


Finally, what is the height (exactly) of the tree in terms of N?

Binary Search Trees

- WHAT are the operations supported?
- HOW do we implement them?
- WHY do we get certain run times?

Search



- Search for 41, then 40

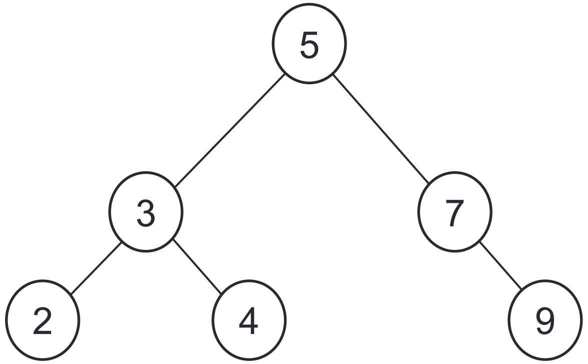
How many pointer traversals are needed to search for 40?

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

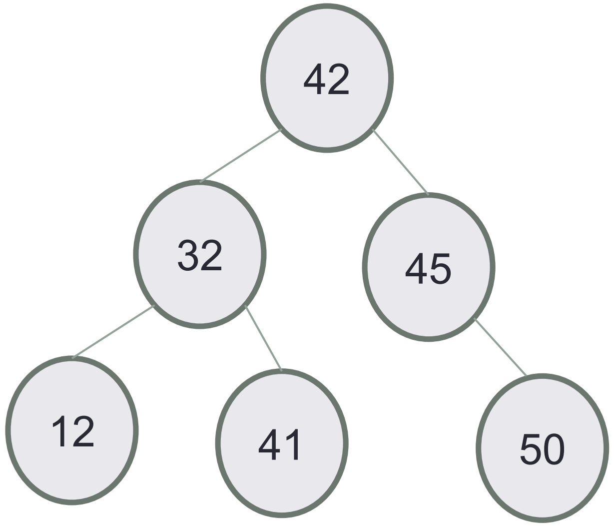
Searching for a Key

Alg: **TREE-SEARCH(x, k)**

```
if x = NIL or k = key [x]
    then return x
if k < key [x]
    then return TREE-SEARCH(left [x], k )
    else return TREE-SEARCH(right [x], k )
```



Insert

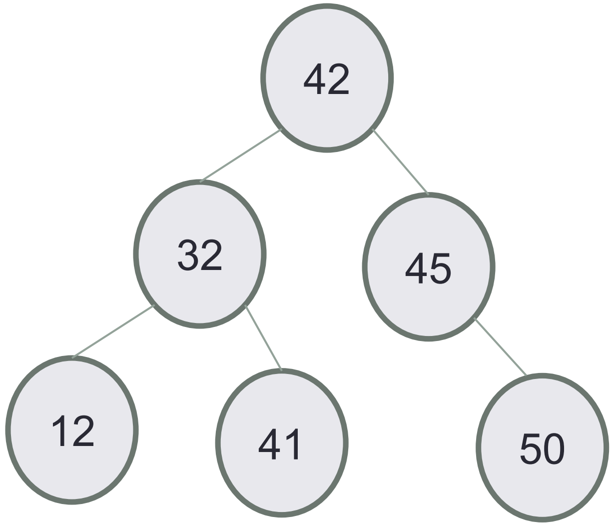


- Insert 40
- Search for the key
- Insert at the spot you expected to find it

What is the worst case running time of search or insert?

A. O(1)
B. O(logN)
C. O(height)
D. O(N)

Min: Search for negative infinity



Can you come up with a recursive implementation for min?

A. Yes
B. No, it has to be iterative

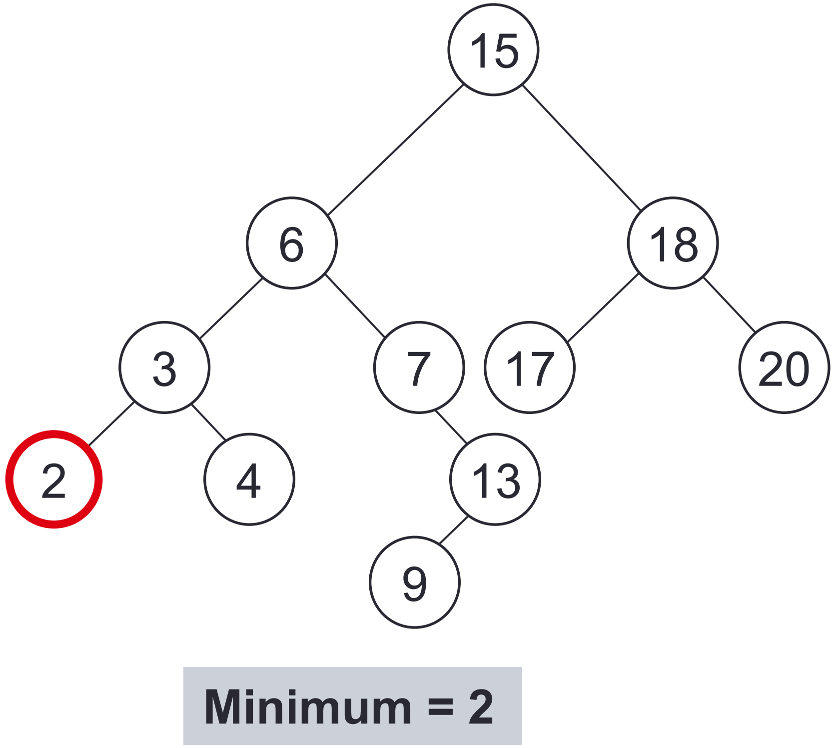
Finding the Minimum in a Binary Search Tree

Goal: find the minimum value in a BST

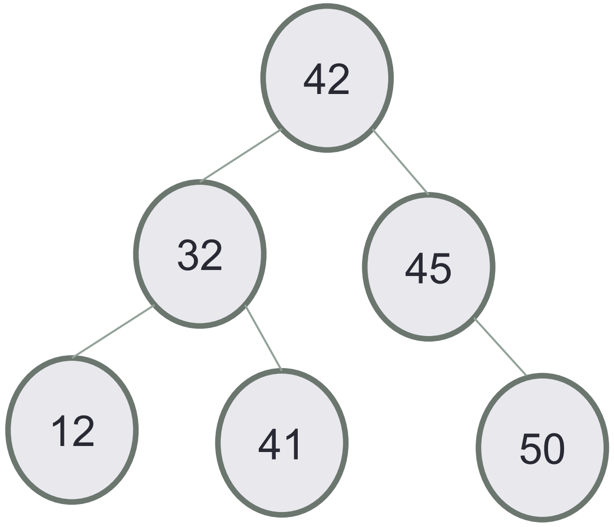
Following left child pointers from the root, until a NIL is encountered

Alg: TREE-MINIMUM(x)

1. **while** left [x] $\neq$ NIL
2. **do** x $\leftarrow$ left [x]
3. **return** x



Max: Search for positive infinity



Traversing a Binary Search Tree

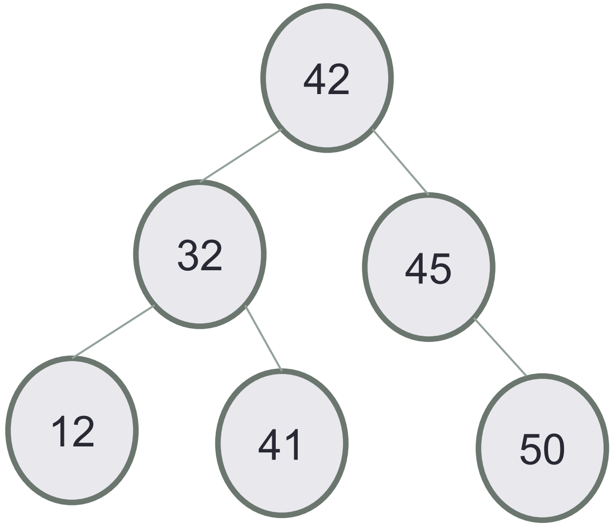
- **Inorder** tree walk:
  - Root is printed between the values of its left and right subtrees:

    left, root, right → keys are printed in sorted order

- **Preorder** tree walk: root printed first: **root, left, right**

- **Postorder** tree walk: root printed last: **left, right, root**

In order traversal: print elements in sorted order



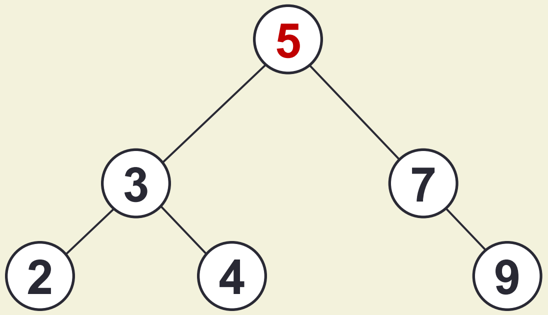
What is the running time?

Traversing a Binary Search Tree

Alg: **INORDER-TREE-WALK(x)**

```
if x ≠ NIL
  then INORDER-TREE-WALK ( left [x] )
       print key [x]
       INORDER-TREE-WALK ( right [x] )
```

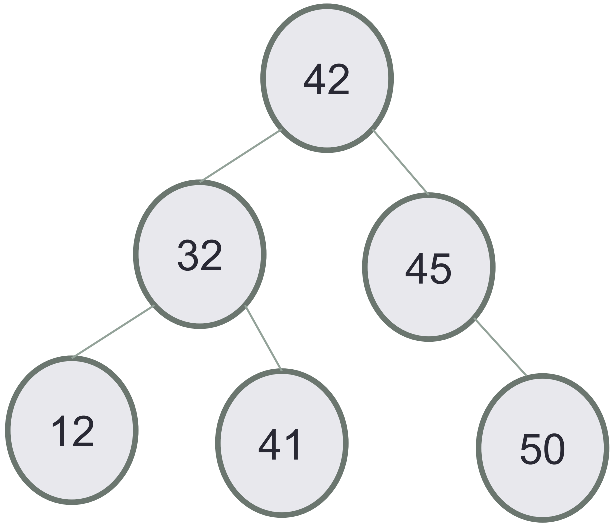
E.g.:



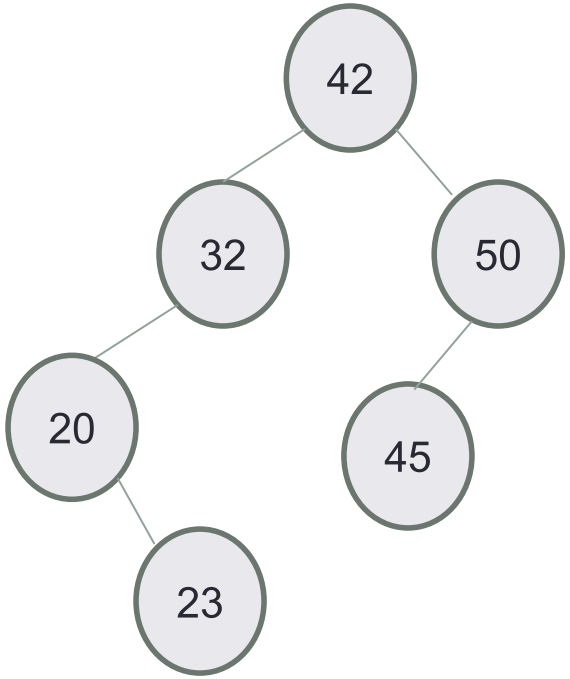
Output: 2 3 4 **5** 7 9

- Running time:
  - $\Theta(N)$, where N is the size of the tree rooted at x

Pre-order traversal: What is it good for?



Predecessor: Next smallest element

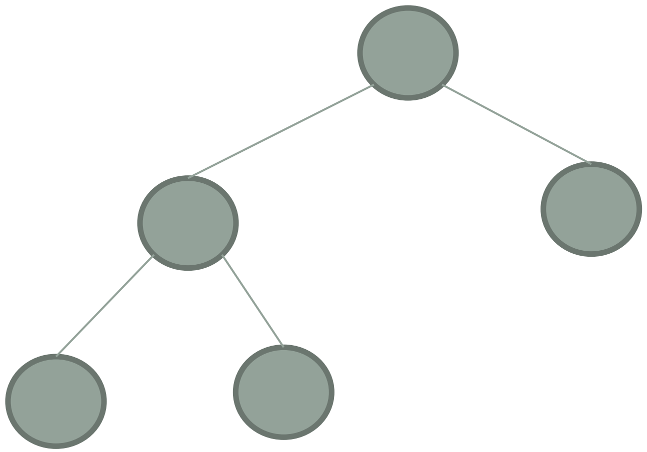


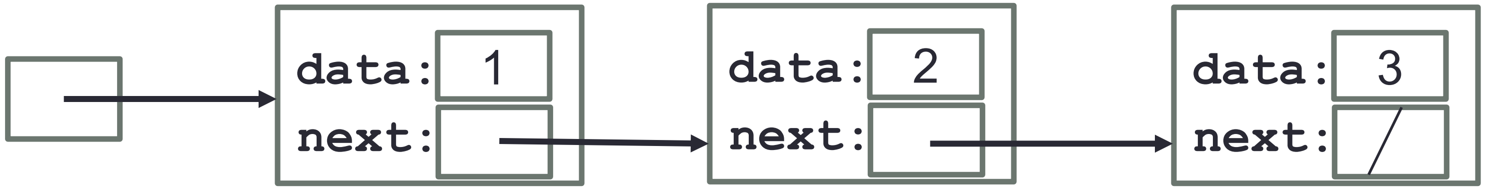
- What is the predecessor of 32?
- What is the predecessor of 45?

Worst case analysis

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

- A. Yes
- B. No





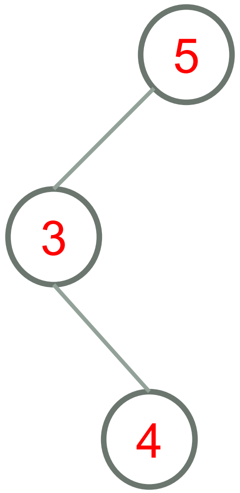
Summary of operations

Average case analysis of a "successful" find

Given a BST having N nodes x_1, .. $x_{N,}$ such that key(x_i) = k_i

How many compares to locate a key in the BST?

1. Worst case:
2. Best case:
3. Average case:



Here is the result! Proof is a bit involved but if you are interested in the proof, come to office hours

$D_{avg}(N)$	Average #comparisons to find a single item in any BST with N nodes

$$D_{avg}(N) \approx 1.386 \ \log_2 N$$

Conclusion: The average time to find an element in a BST with no restrictions on shape is $\Theta(\log N)$.