CS261 Data Structures

Binary Search Trees

Concepts

Goals

- Introduce the Binary Search Tree (BST)
- Conceptual implementation of Bag interface with the BST
- Performance of BST Bag operations

Binary Search Tree

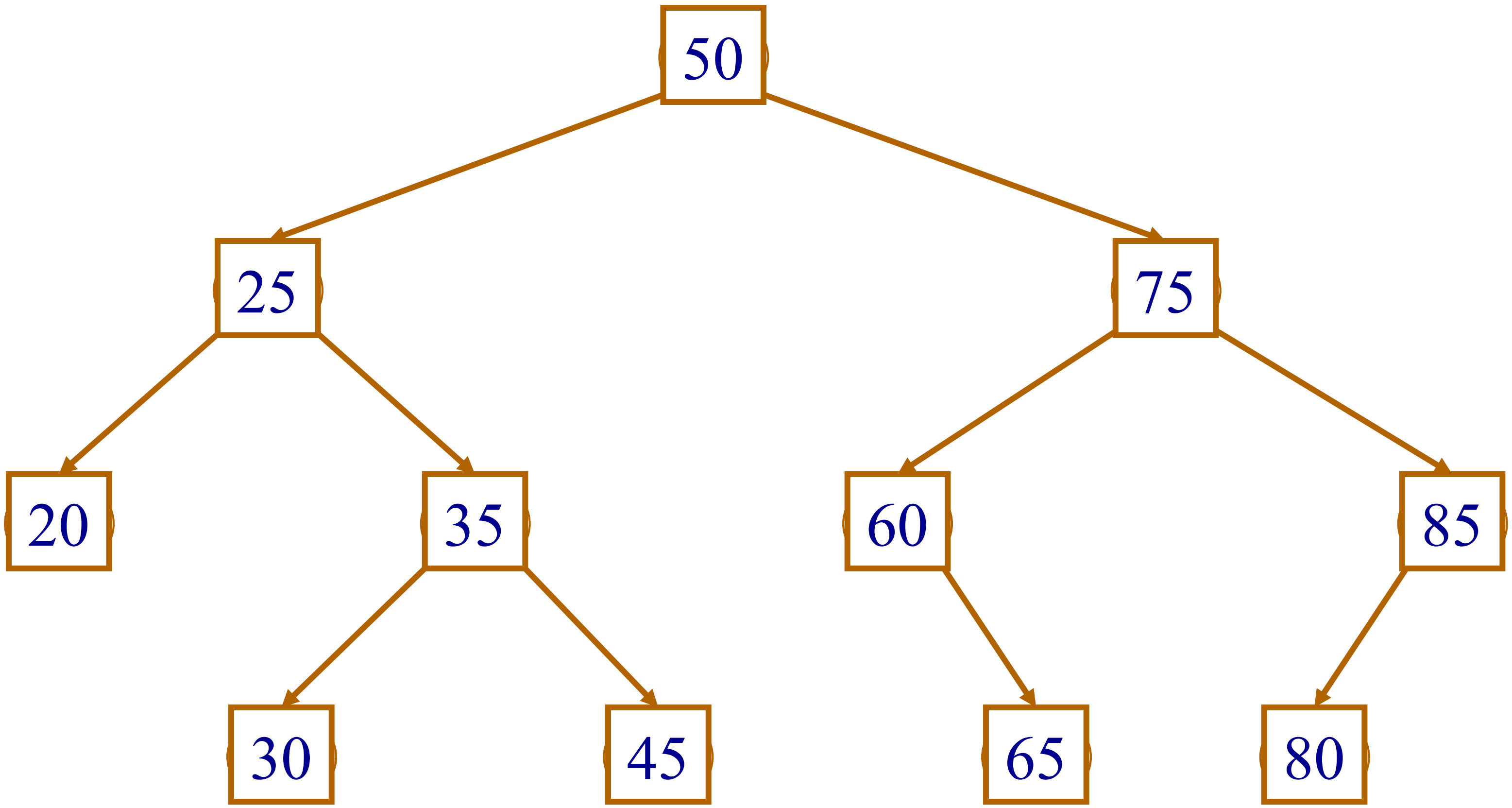
- Binary search trees are binary trees where every node's value is:
  - *Greater than* all its descendents in the *left subtree*
  - *Less than or equal to* all its descendents in the *right subtree*

Difference between binary tree and binary search tree is: search tree is ordered

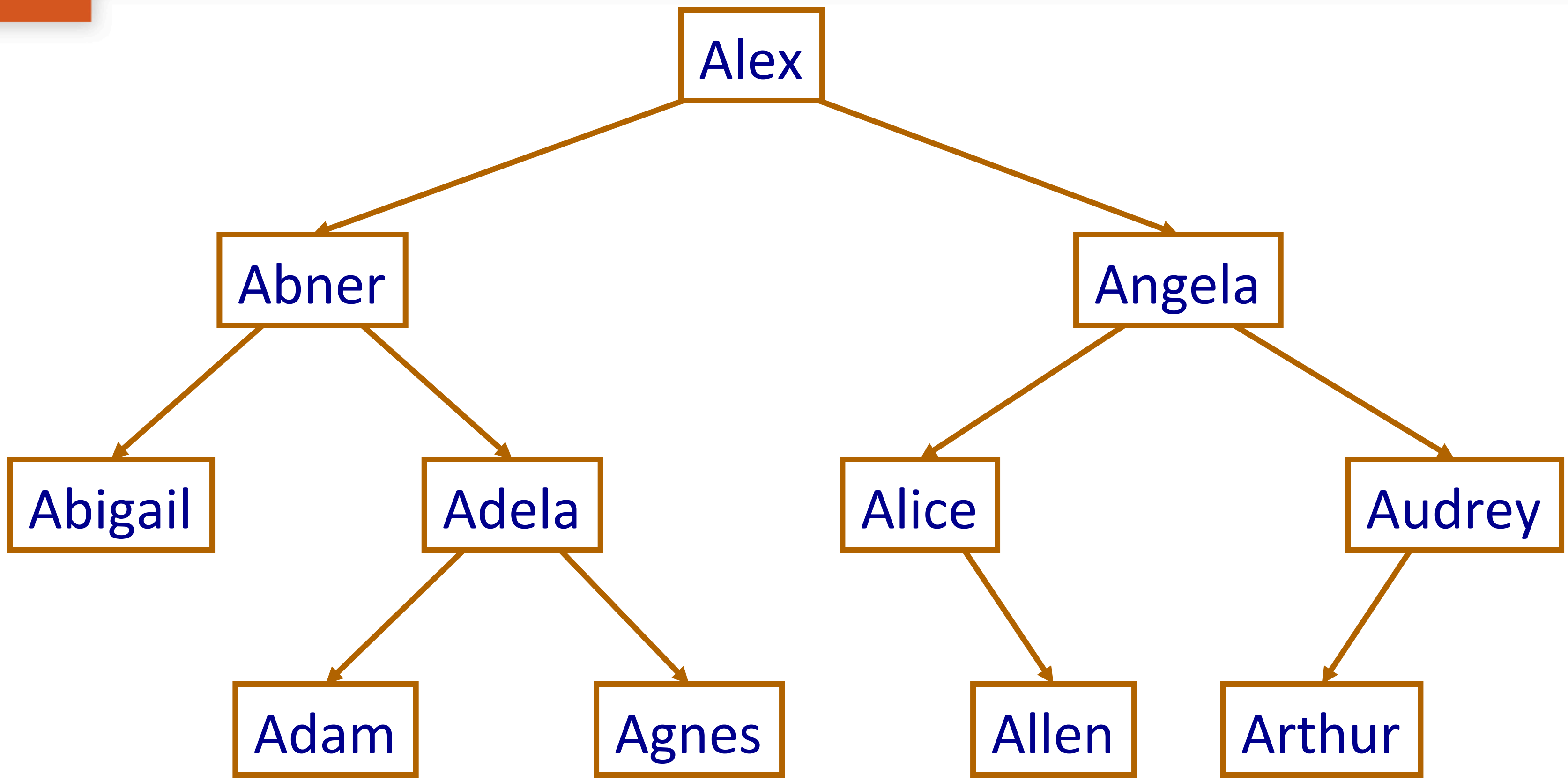
- If tree is reasonably full (well balanced), searching for an element is O(log *n*). Why?

cuz it's a binary search

Intuition



Binary Search Tree: Example

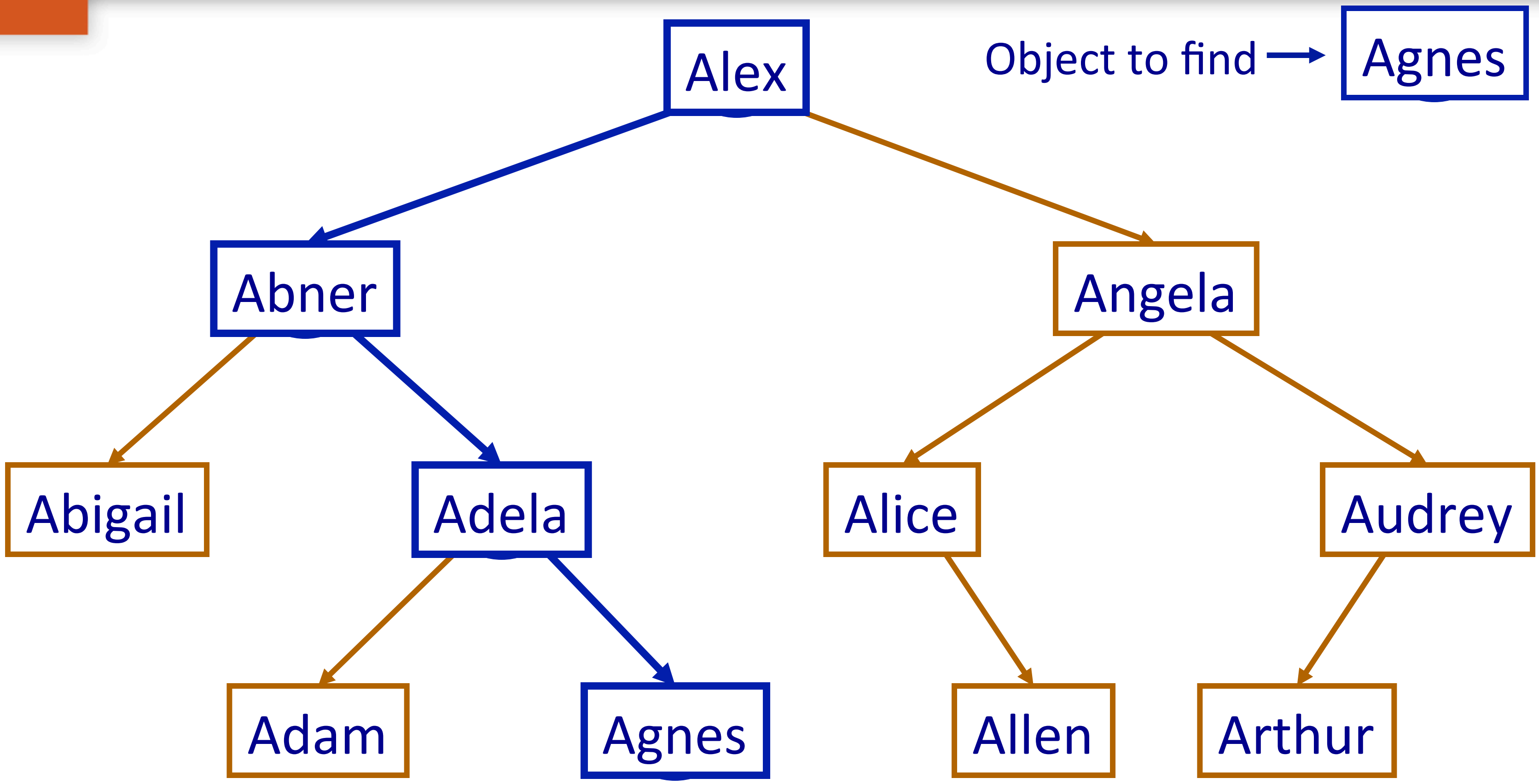


ordered alphabetically

BST Bag: Contains

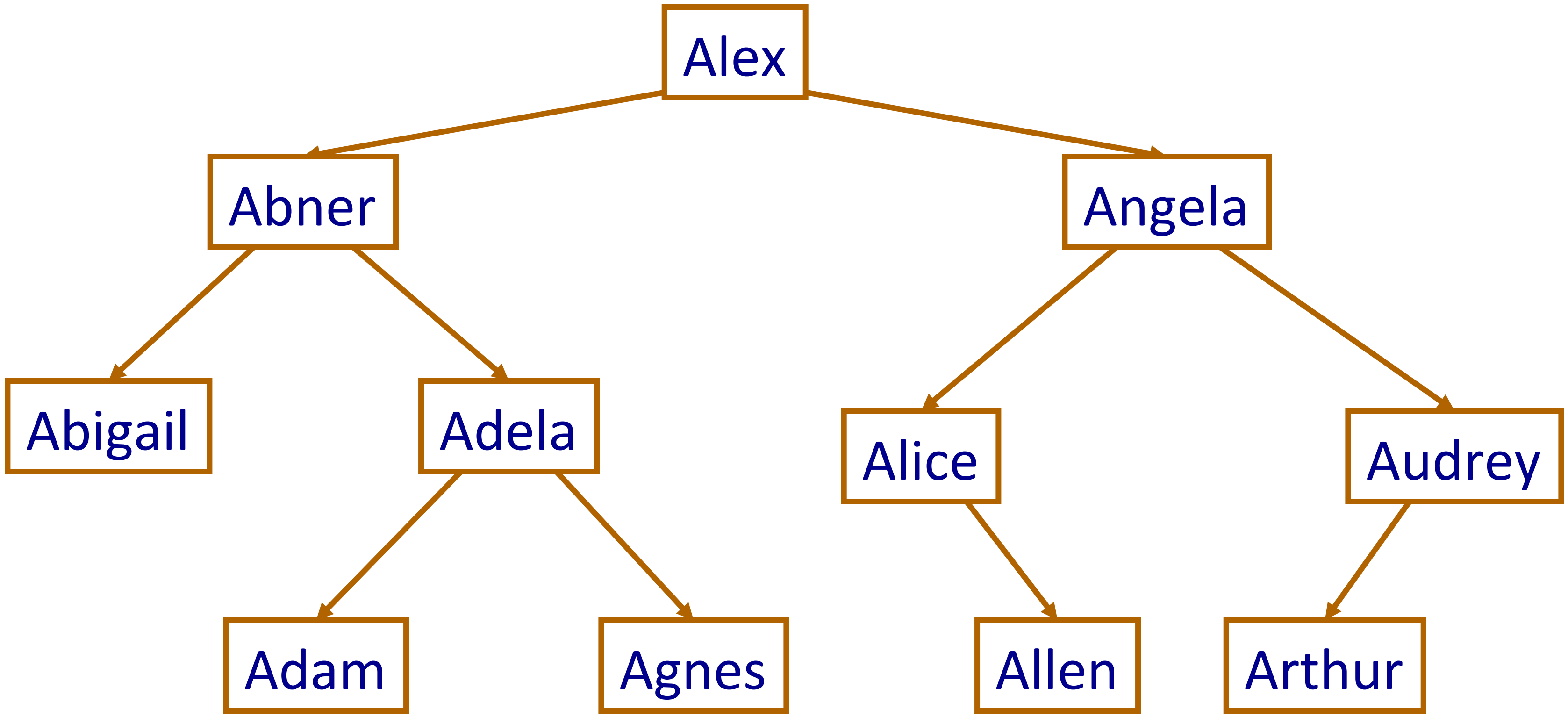
- Start at root
- At each node, compare value to node value:
  - Return true if match
  - If value is less than node value, go to left child (and repeat)
  - If value is greater than node value, go to right child (and repeat)
  - If node is null, return false
- Dividing in half each step as you traverse path from root to leaf (assuming reasonably full!!!)

BST Bag: Contains/Find Example

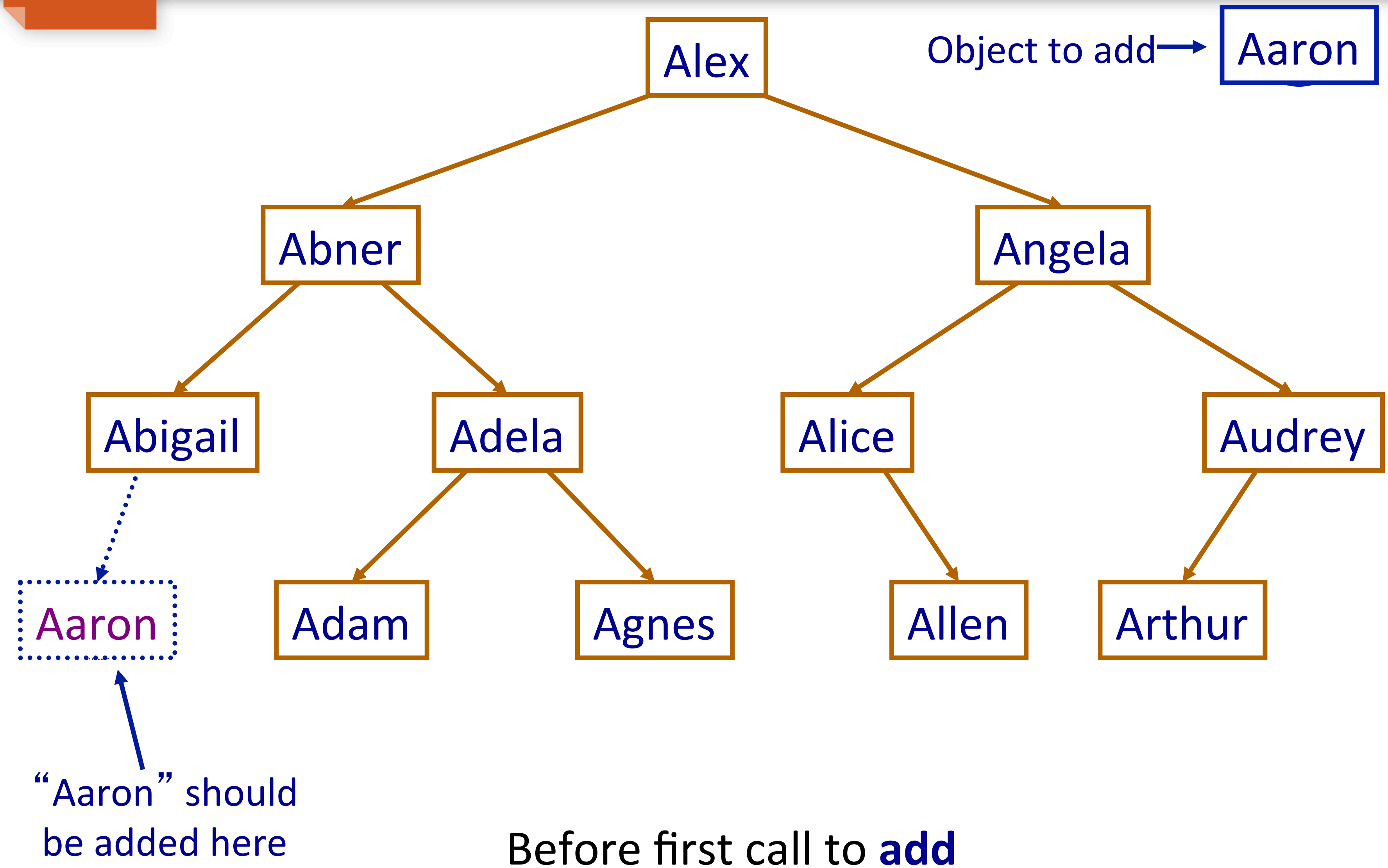


BST Bag: Add

- Do the same type of traversal from root to leaf
- When you find a null value, create a new node

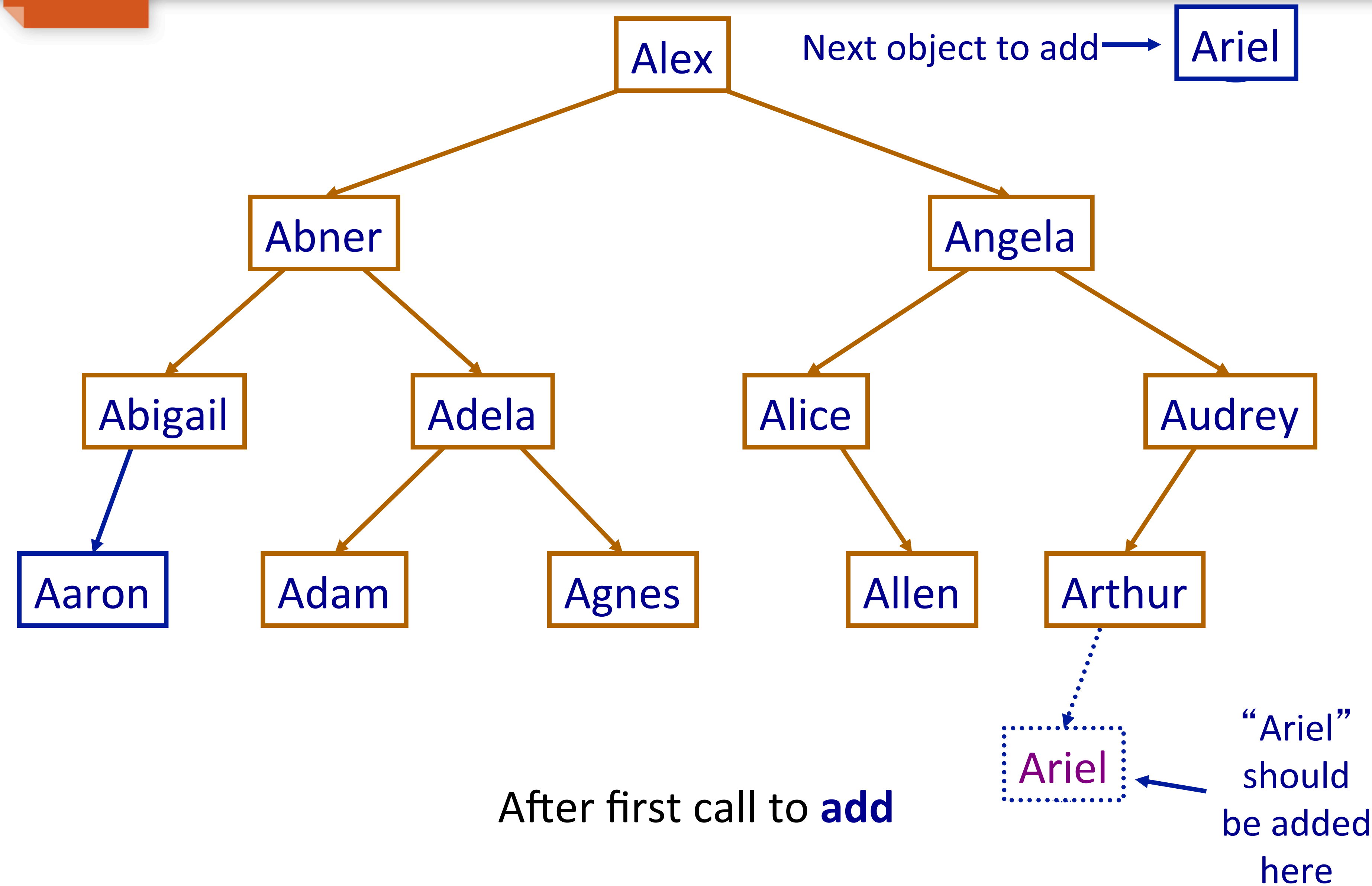


BST Bag: Add Example



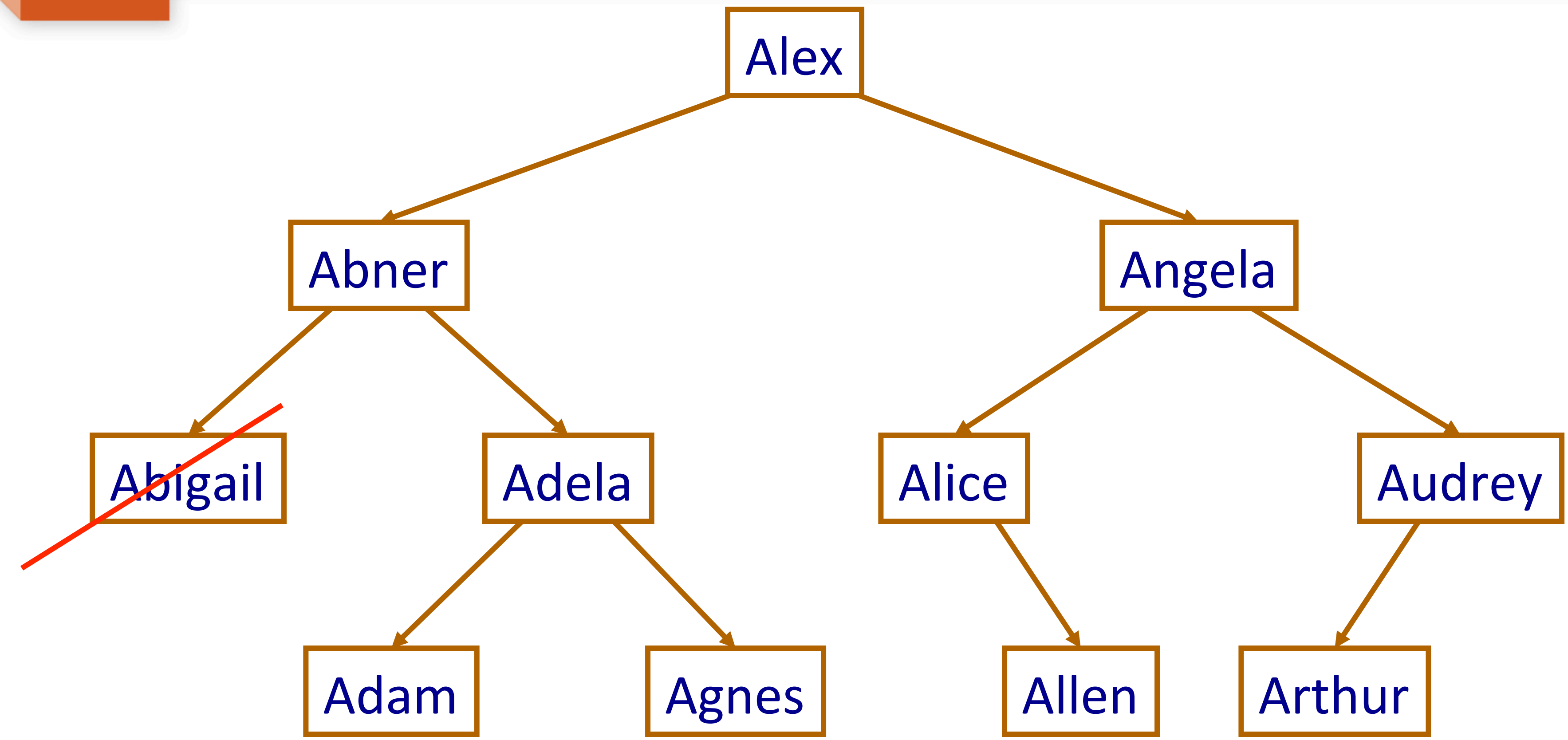
Before first call to **add**

BST Bag: Add Example



After first call to **add**

BST Bag: Remove

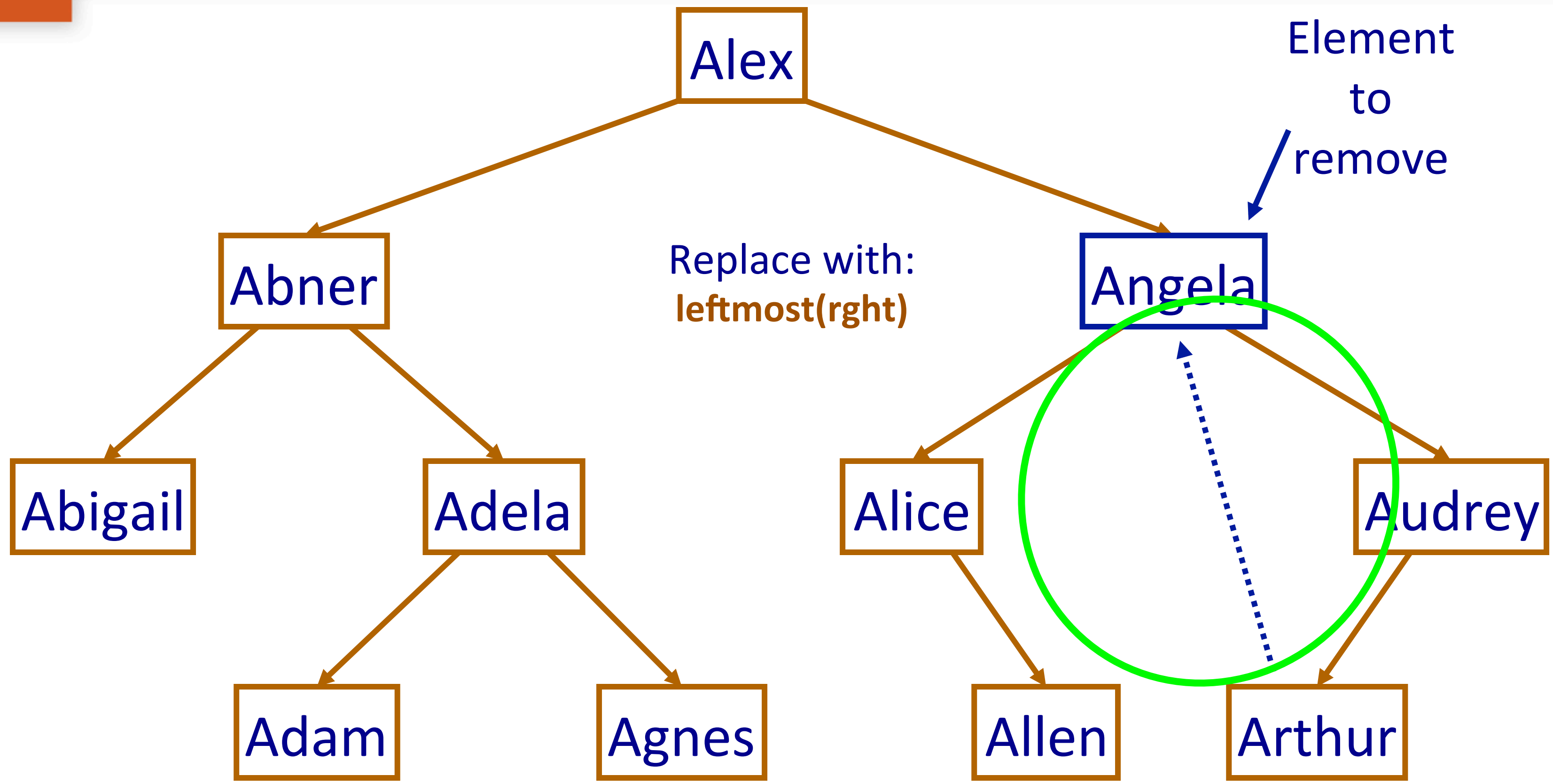


How would you remove Abigail? Audrey? Angela?

Who fills the hole?

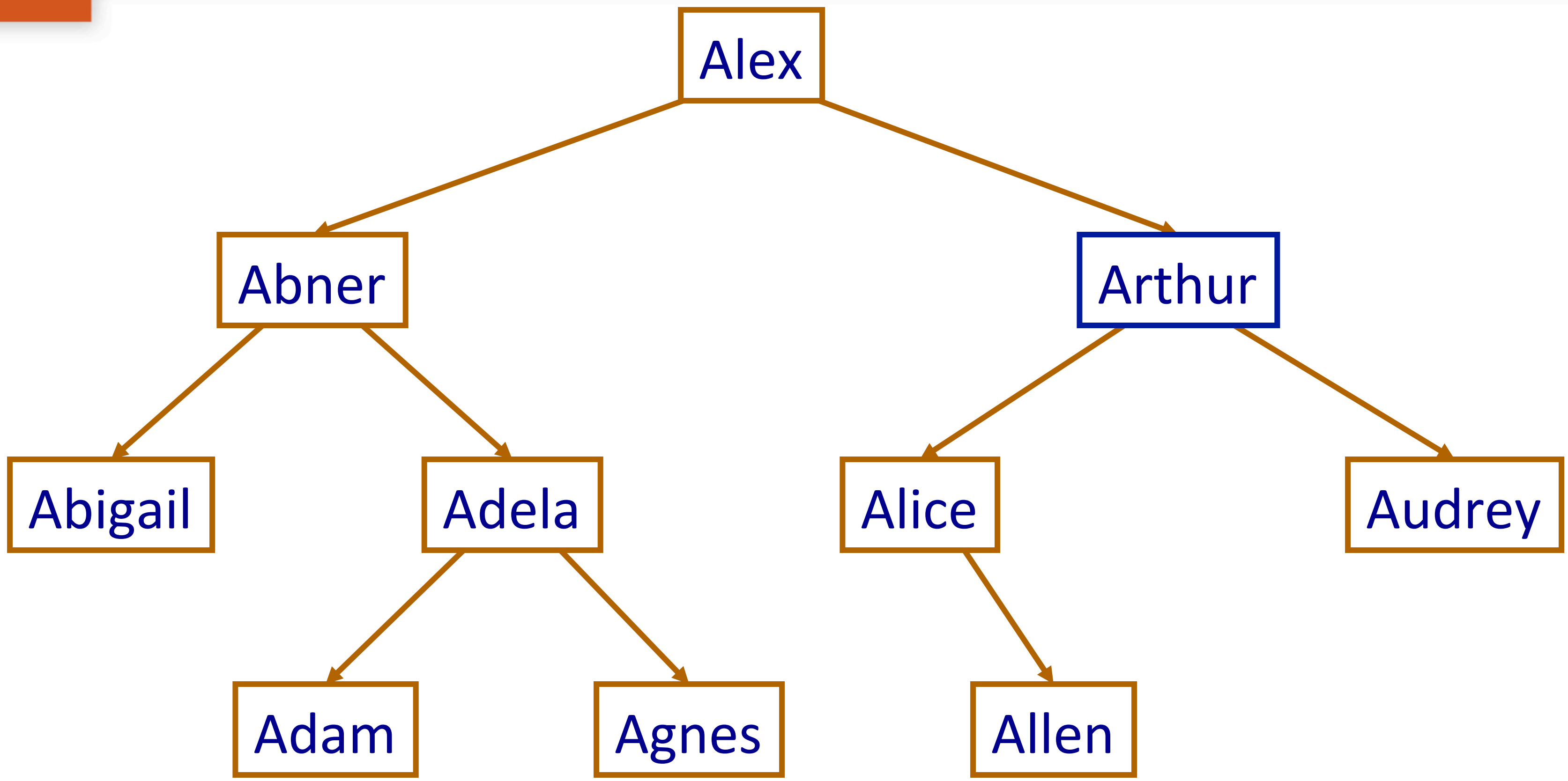
- Answer: the leftmost child of the right subtree (smallest element in right subtree)
- Try this on a few values
- Alternatively: The rightmost child of the left subtree

BST Bag: Remove Example



Before call to **remove**

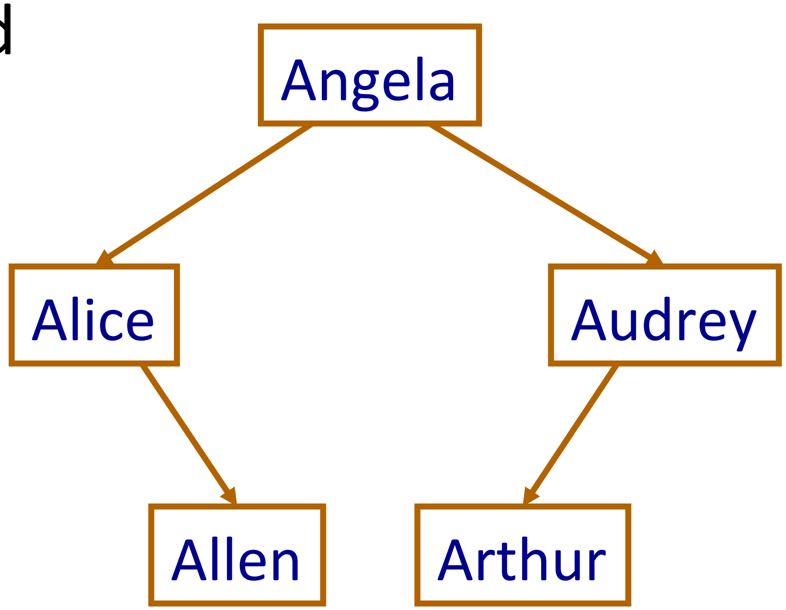
BST Bag: Remove Example



After call to **remove**

Special Case

- What if you don't have a right child?
- Try removing "Audrey"
  - Simply return left child



Complexity Analysis (contains)

- If reasonably full, you're dividing in half at each step: O(log n)

- Alternatively, we are running down a path from root to leaf

  – We can prove by induction that in a complete tree (which is reasonably full), the path from root to leaf is bounded by floor(log n), so O(log n)

Binary Search Tree: Useful Collection?

- We've shown all Bag operations to be proportional to the length of a path, rather than the number of elements in the tree
- We've also said that in a reasonably full tree, this path is bounded by : floor(($\log_2 n$))
- This Bag is faster than our previous implementations!

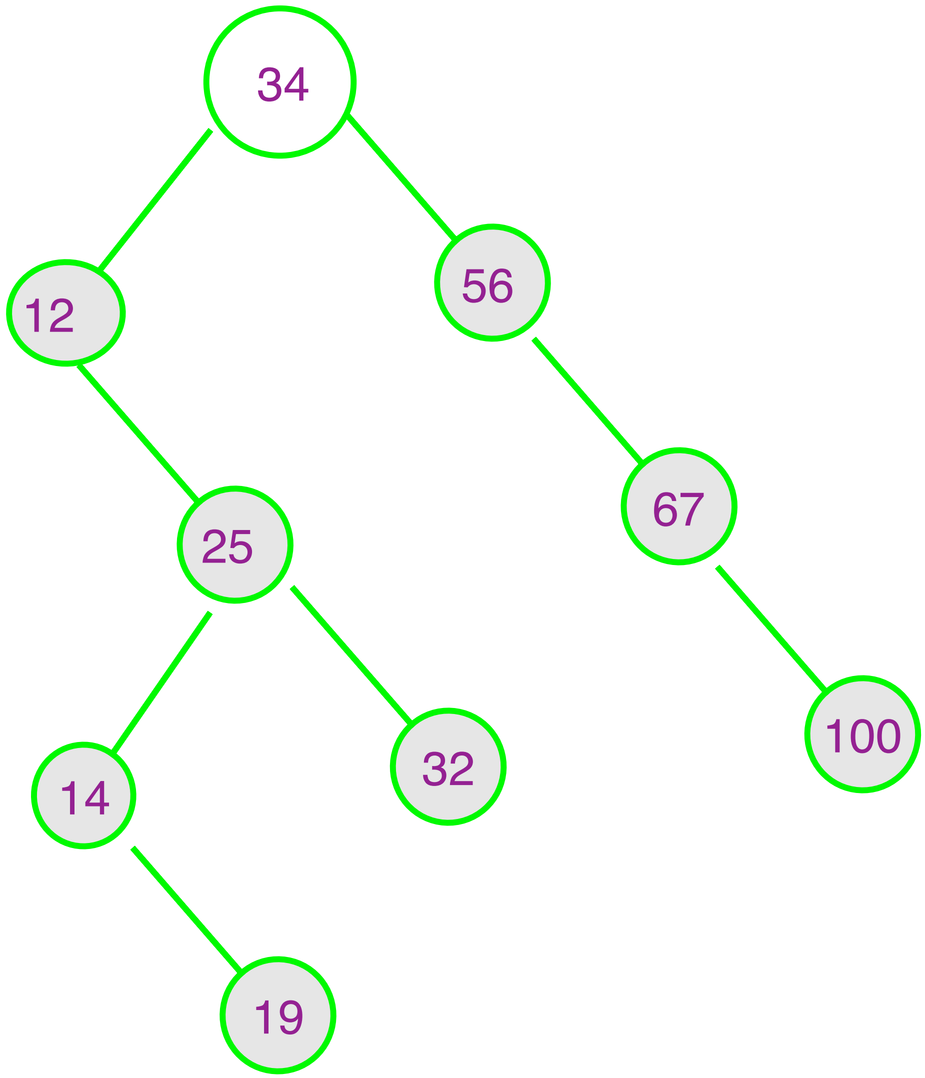
Comparison

- Average Case Execution Times

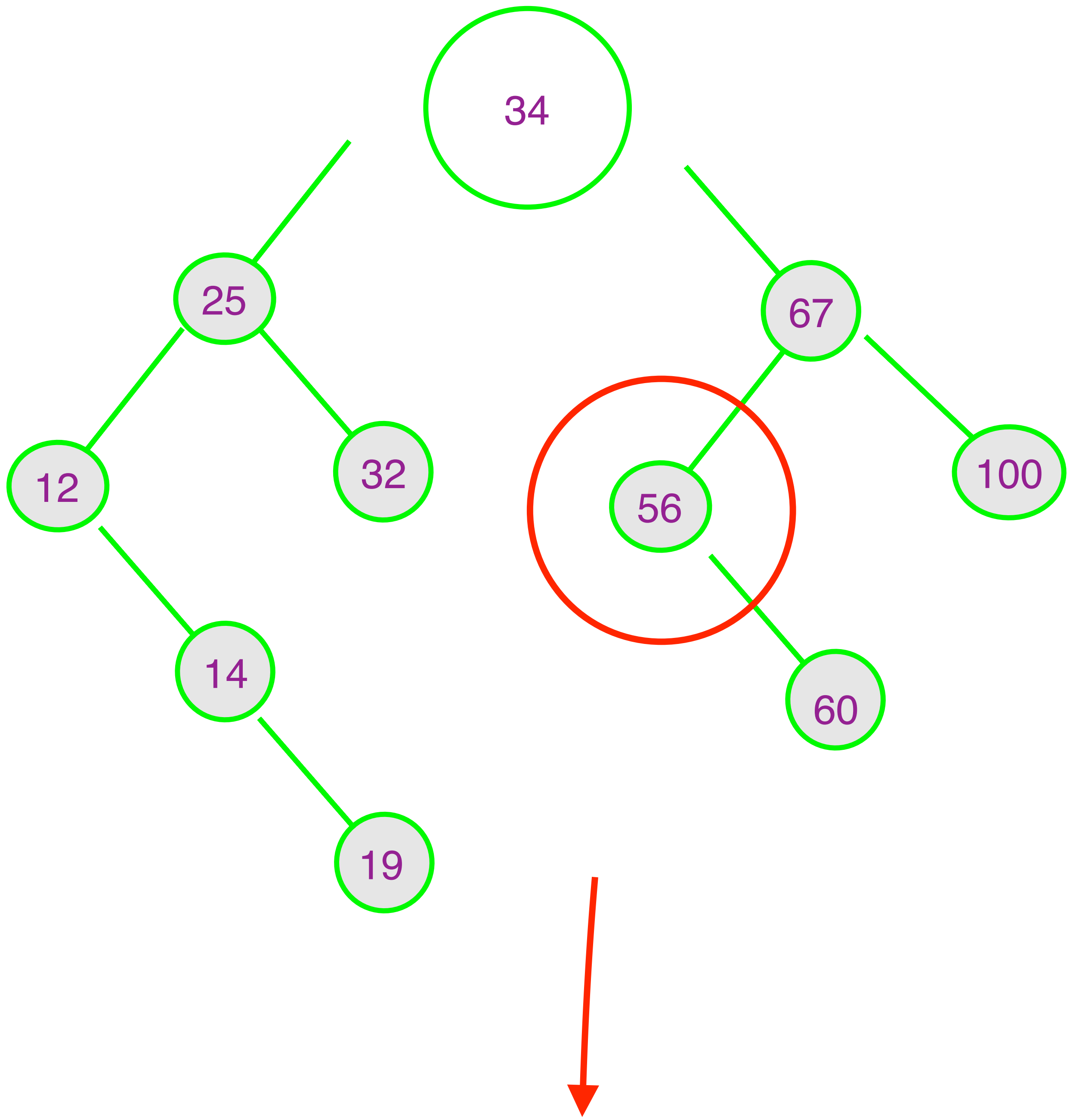
Operation	DynArrBag	LLBag	Ordered ArrBag	BST Bag
Add	O(1)	O(1)	O(n)	O(logn)
Contains	O(n)	O(n)	O(logn)	O(logn)
Remove	O(n)	O(n)	O(n)	O(logn)

Bag Add Ex:

- insert numbers into BST in order given (don't worry about balance)
- 34, 56, 67, 12, 25, 14, 100, 19, 32



Bag Add Ex:
- insert numbers into BST in order given (don't worry about balance)
- 34, 67, 56, 25, 12, 14, 100, 19, 32, 60
- THEN, remove 34



On RIGHT subtree, go as far LEFT as possible