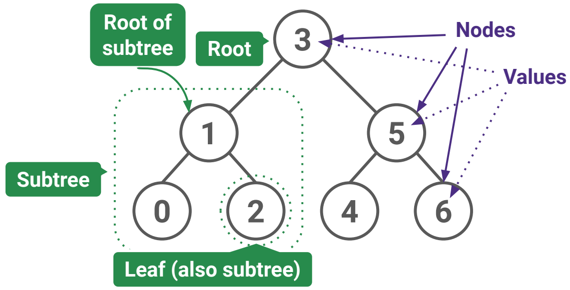


Recursive Description: **root** and **subtree**.
Each subtree is itself a valid tree.
A tree with zero subtrees is a **leaf**.

Relative Description: **node** and **value**.
Each node has a value. A **parent** node is joined with an **edge** to each **child** node.

Tree Abstraction

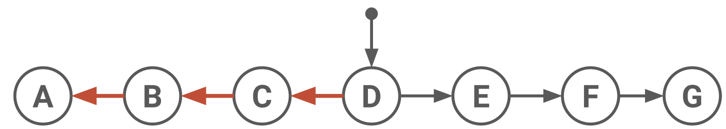
We oftentimes abuse the terminology a bit by saying things like, “each parent is the sum of its children”.

?: What does the “root node” refer to? What does the “root value” refer to?

Optimization: Move Entry Point, Flip Links

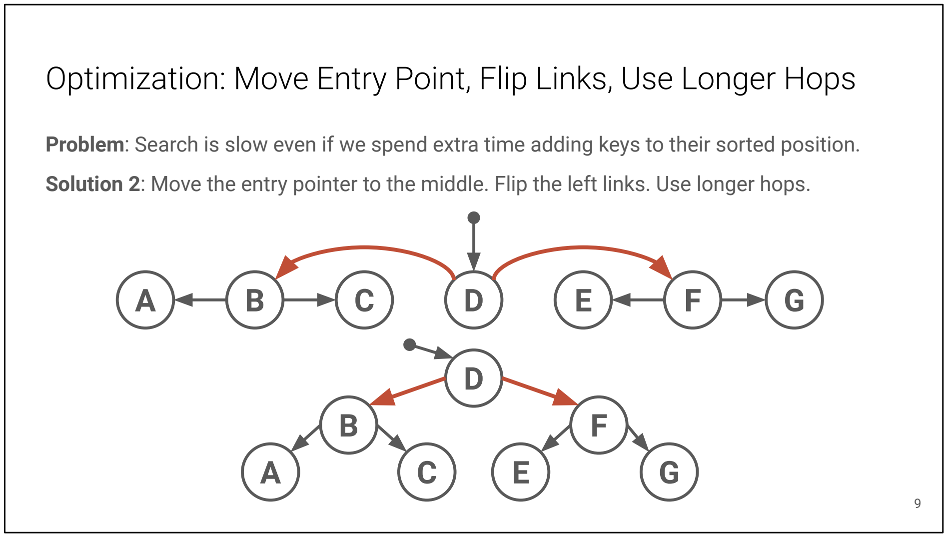
Problem: Search is slow even if we spend extra time adding keys to their sorted position.

Solution 2: Move the entry pointer to the middle. Flip the left links.



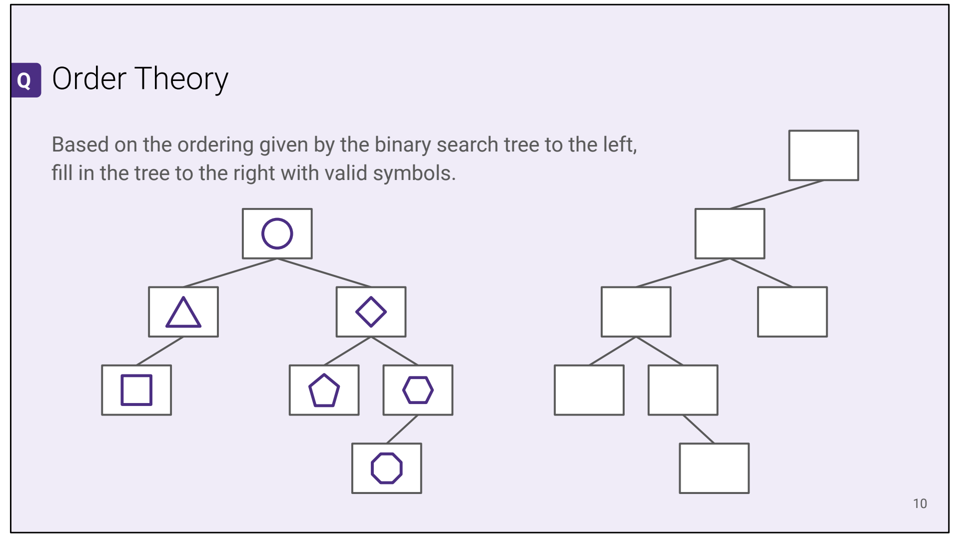
?: How does this change affect the worst-case search time?

?: How can we improve this optimization?



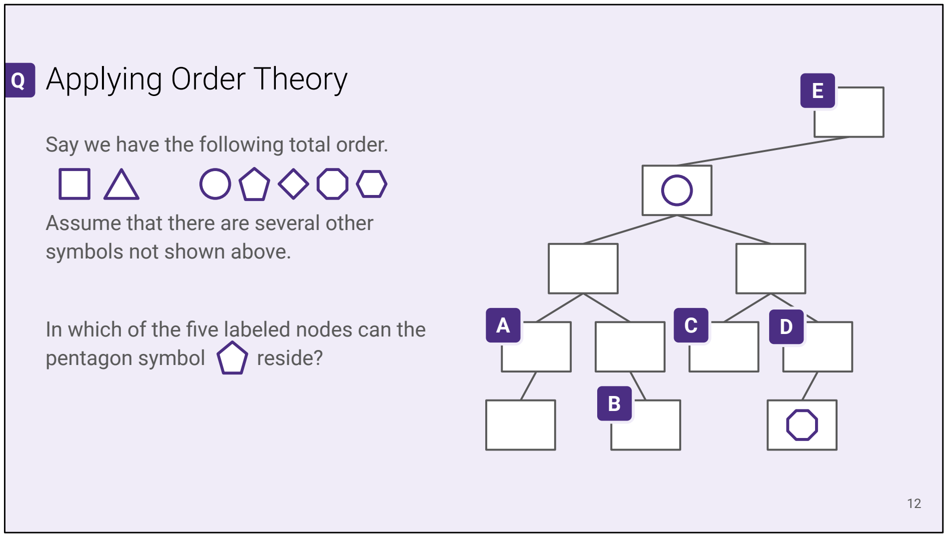
We saw this pattern of recursive subdivision in merge sort, and it's here again!

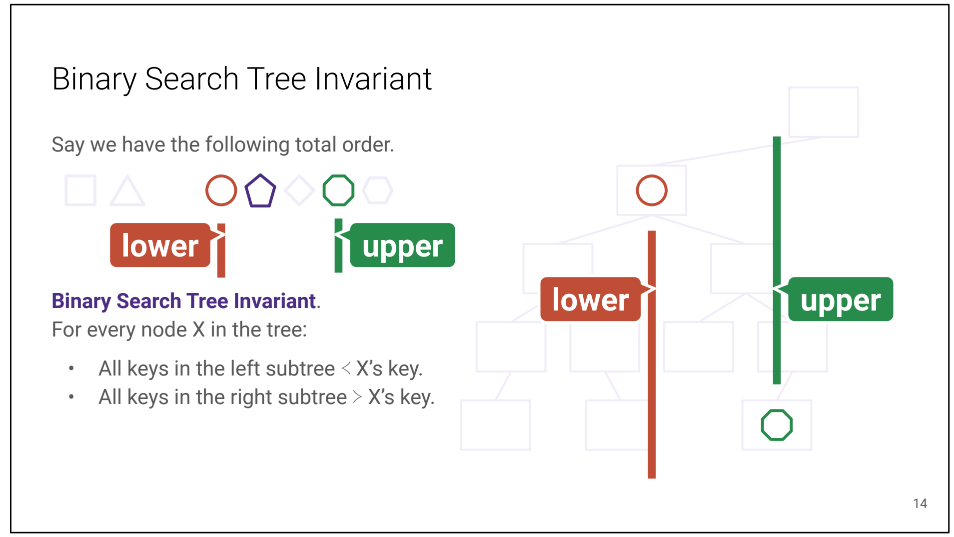
?: What is the worst-case search time?



?: Binary search trees are related to OrderedLinkedSets. What do we know about the relationship between the square symbol and the triangle symbol?

Q1: Based on the ordering given by the binary search tree to the left, fill in the tree to the right with valid symbols.

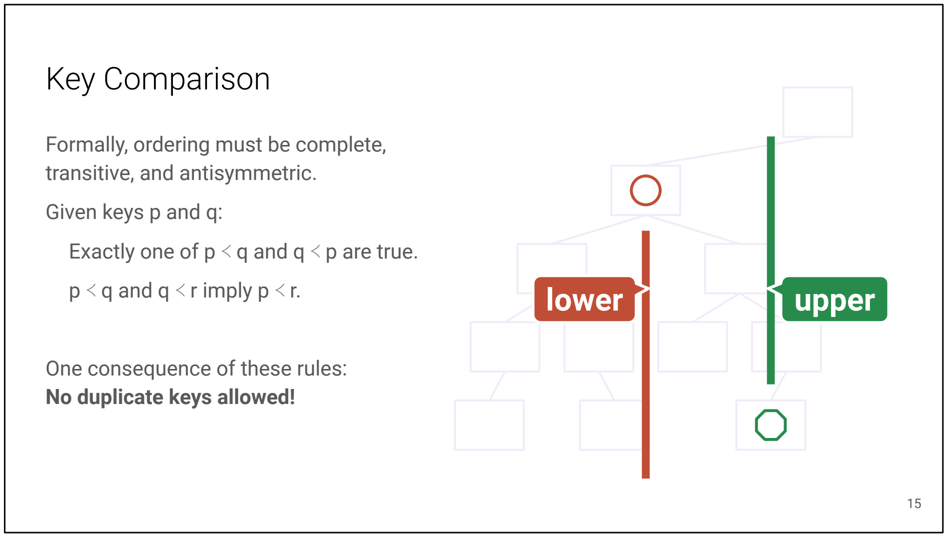




Q1: In which of the five labeled nodes can the pentagon symbol reside?

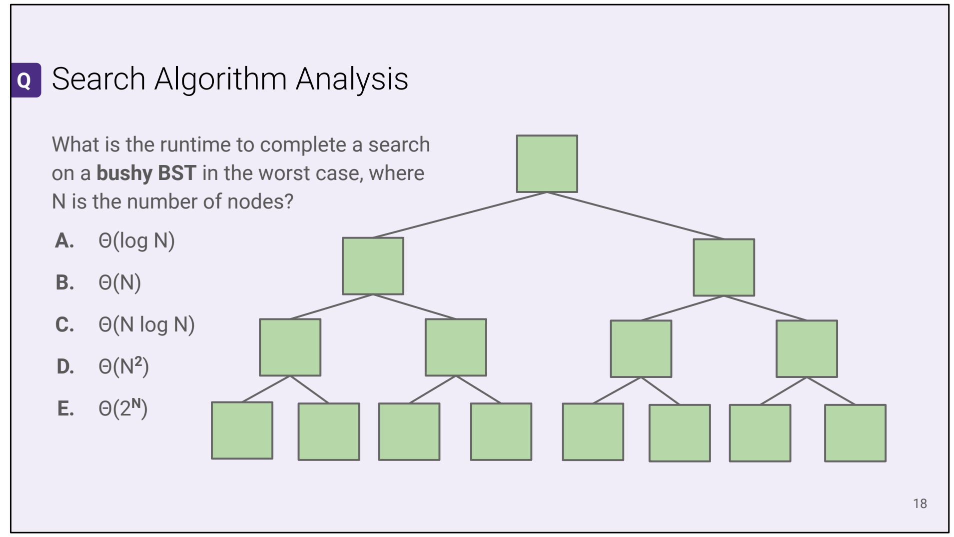
?: If we search a left subtree, how does that change the **lower** limit on the keys? The **upper** limit?

?: If we search a right subtree, how does that change the **lower** limit on the keys? The **upper** limit?



?: What is the purpose of this formal definition of key comparison? How do we apply these rules to numbers vs. arbitrary objects?

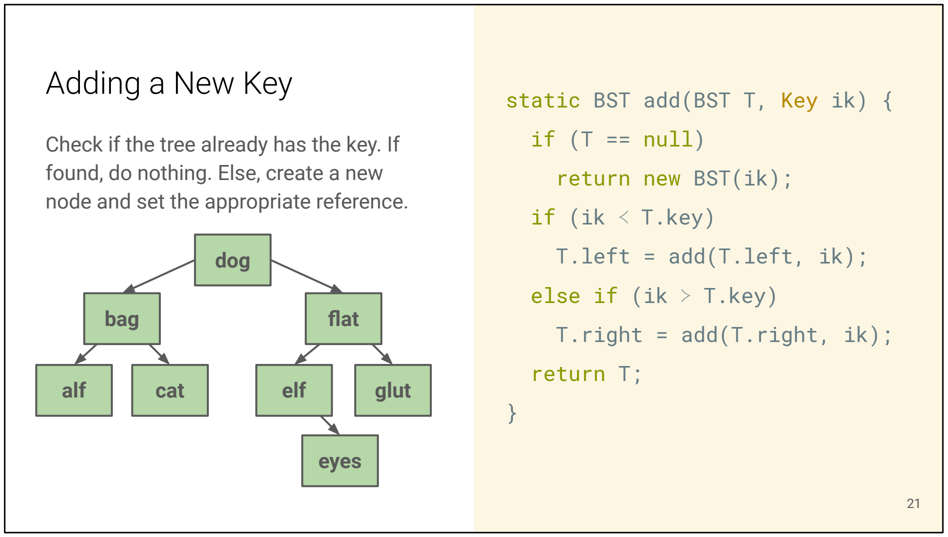
?: How might we allow duplicate keys in our binary search tree in spite of these rules? What are the potential problems that arise?



We don't yet have a formal definition for the concept of bushiness. Use the example as a visual aid.

Q1: What is the runtime to complete a search on a **bushy BST** in the worst case, where N is the number of nodes?

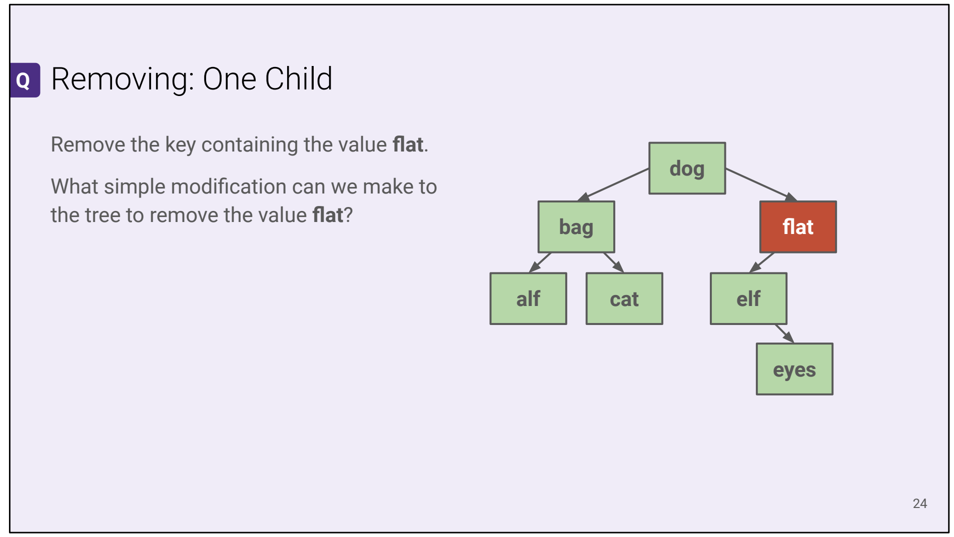
?: What is the best case runtime?



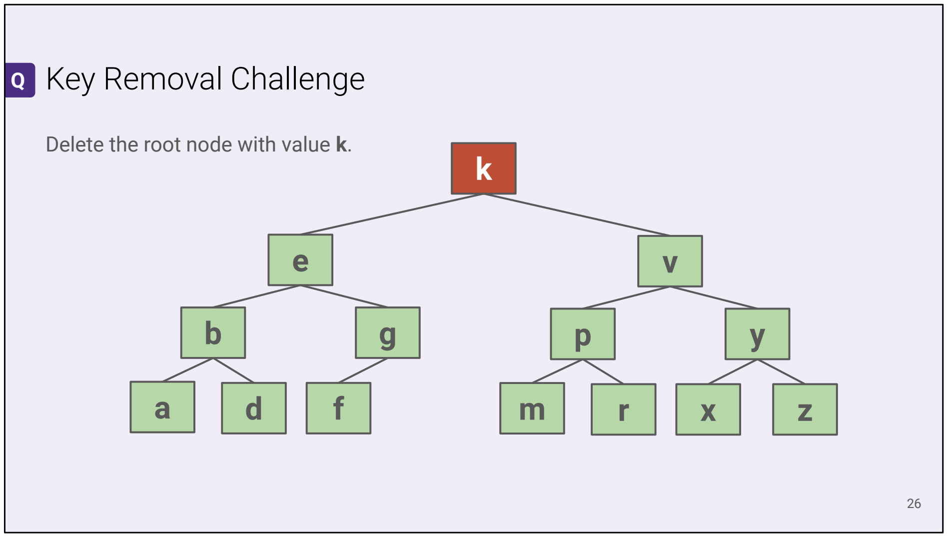
You might sometimes see code that exhibits “arm’s-length recursion.” Consider these two unnecessary base cases.

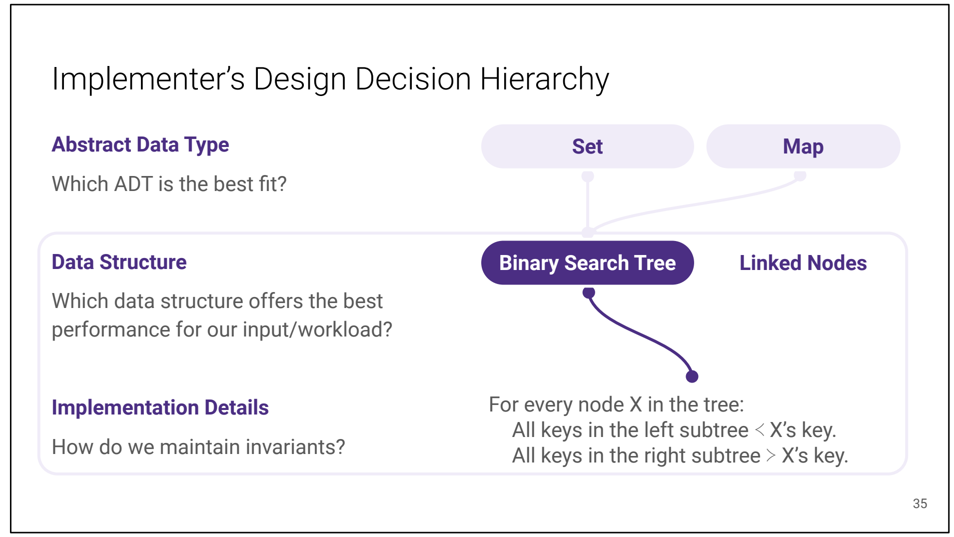
```
if (T.left == null)
  T.left = new BST(ik);
else if (T.right == null)
  T.right = new BST(ik);
```

?: How does the code given in the slide handle the arm’s-length recursion scenario?



Q1: What simple modification can we make to the tree to remove the value **flat**?





Q1: Delete the root node with value **k**.

As the ADT implementer, we always had to keep in mind our invariants when thinking through the problem.

?: How does the Binary Search Tree Invariant affect the implementation of contains, add, and remove?