
1: B Trees

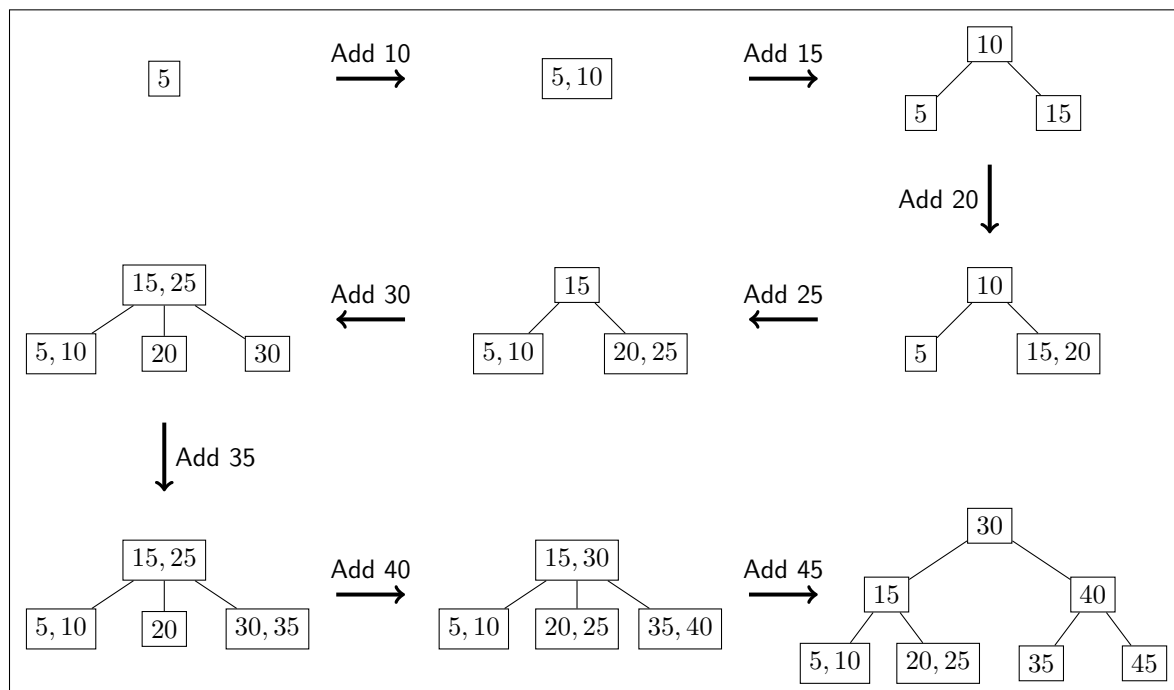
1. Draw the 2,3-tree that results from inserting the following keys in sequential order into an initially empty tree: 5, 10, 15, 20, 25, 30, 35, 40, 45.
2. Show the 2,3-tree that results from deleting 10, then 20, then 30 from the above tree.
3. Design a simple algorithm that, given any B-tree, can find the minimum key stored in the tree.
4. Design an algorithm that, given a B-tree of minimum order 3 and a key k , returns the predecessor of k . The predecessor is the key that is the next smallest key to k when looking at all the keys in the B-tree. If there exists no predecessor for k (ie. k is the smallest element in the tree), the algorithm should return null.

2: Solutions

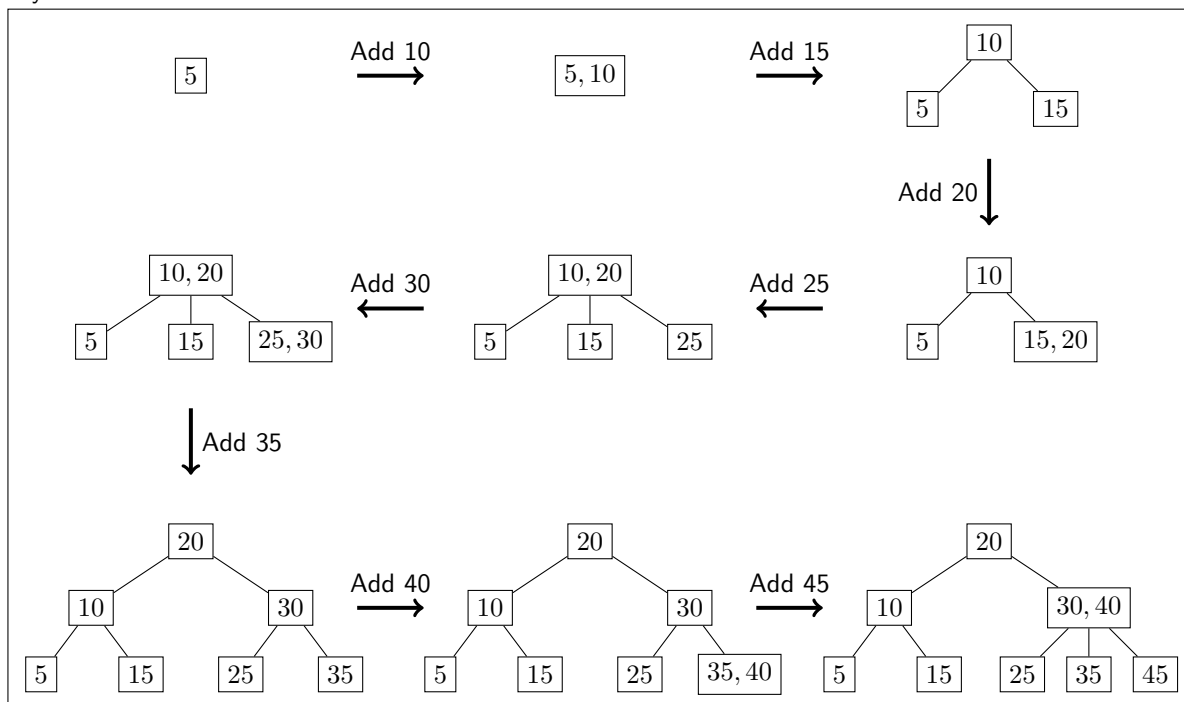
B-Trees

1. This question has multiple solutions depending on whether or not you use rotations in the implementation of B-trees. For the purposes of this class, both solutions are equally correct. However, traditional implementations of the B-tree do not use rotations.

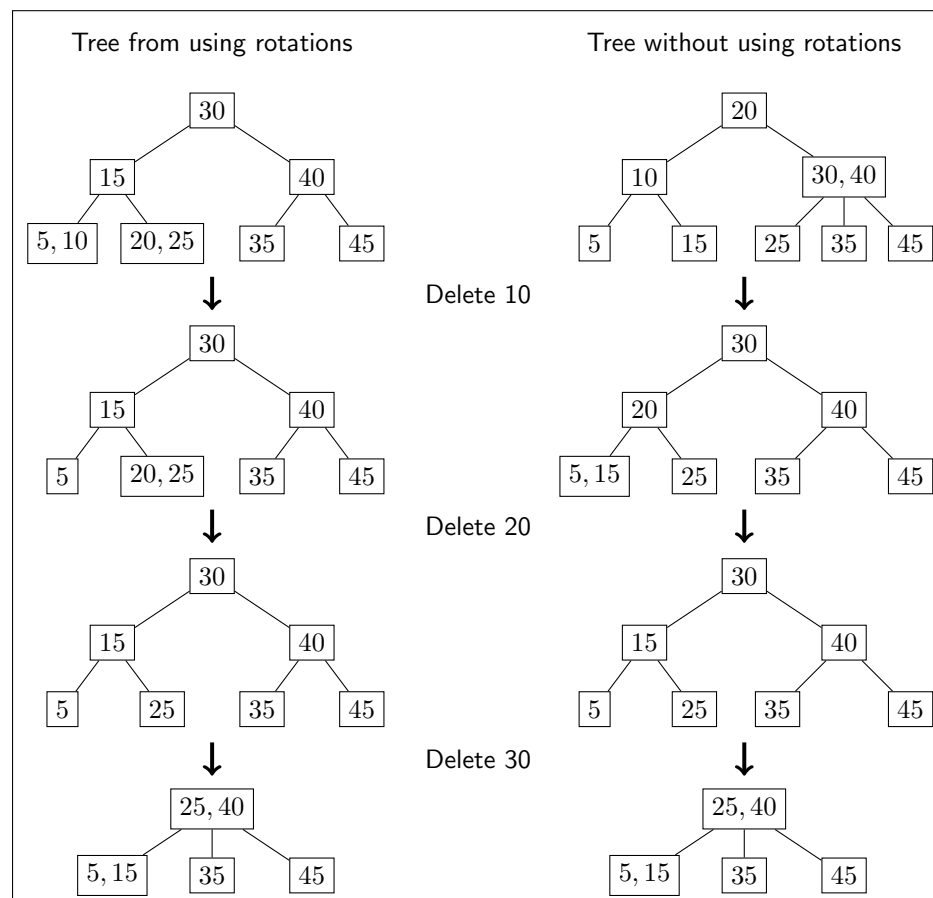
If you choose to use rotations:



If you choose to not use rotations:



2. Both solutions are correct, depending on the ending tree from part (1).



3. To find the minimum key stored in a B-tree, we simply need to continually perform left-traversals until we find a NULL, and return the smallest key in the node where NULL was found. The properties of a B-tree ensure that the leftmost node of a subtree will always contain the smallest key value in that particular subtree.
4. To find the predecessor of a key stored in a B-tree, we need to consider either if k is a leaf node or an internal node.
 - If the key is in an internal node and the key is NOT the smallest key in the node, then we simply need to return the largest key to the left of the key in the same node.
 - If the key is in an internal node, and if the key is the smallest key in the node, AND the node possesses a left child, then the predecessor would be the largest key in the left subtree of the key.
 - If the key is in a leaf node, and the key is NOT the smallest key in the node, then we need to return the largest key to the left of the key in the same node.
 - If the key is in a leaf node and the key is the smallest key in the node, then we need to move up one level until a node with a smaller key is found; the largest key (rightmost key) in the subtree of this node will be the predecessor.