Complexity:

Based on the information provided in the search results, the time complexities of various operations on binary search trees (BSTs) are as follows:

1. Searching in a BST:
   - Average case: O(log n)
   - Worst case: O(n)

   The worst-case scenario occurs when the BST is skewed (e.g., a linked list), and the search time becomes linear in the number of nodes (O(n)). However, for a balanced BST, the search time is logarithmic in the number of nodes (O(log n)).

2. Insertion in a BST:
   - Average case: O(log n)
   - Worst case: O(n)

   Similar to searching, the insertion time is logarithmic in the average case for a balanced BST, but can degrade to linear time (O(n)) in the worst case for a skewed BST.

3. Deletion in a BST:
   - Average case: O(log n)
   - Worst case: O(n)

   Deleting a node in a BST involves searching for the node, finding its successor, and then performing the deletion. The time complexity follows the same pattern as searching and insertion.

4. Balancing an Unbalanced BST:
   - Time complexity: O(n)

   The search results indicate that the time complexity to balance an unbalanced BST is linear in the number of nodes (O(n)). This is typically achieved by first performing an inorder traversal to obtain a sorted array of the nodes, and then constructing a balanced BST from the sorted array.

In summary, the key points are:

- For a balanced BST, the time complexities of search, insertion, and deletion are O(log n) on average.
- For an unbalanced BST (e.g., a skewed tree), the time complexities can degrade to O(n) in the worst case.
- Balancing an unbalanced BST can be done in O(n) time.

The search results also mention that there are self-balancing binary search tree variants, such as AVL trees and Red-Black trees, which can guarantee logarithmic time complexities for all operations, even in the worst case.