7. Binary search tree

1. Search trees

A search tree is a data structure used for searching and sorting. It is a tree-like structure where each node contains a value and a pointer to its child nodes. Search trees are used for fast lookup and insertion of data and are commonly used to implement associative arrays or sets.

2. Binary search tree

A Binary Search Tree (BST) is a particular type of search tree where each node contains a key and a value, and **all nodes in the left subtree have keys less than the node's key, while all nodes in the right subtree have keys greater than the node's key**. This makes operations such as searching, insertion, and deletion in a BST have a time complexity of O(log n), where n is the number of nodes in the tree. BSTs are a very common data structure and are widely used in various algorithms and applications in computer science.

The time complexity of operations such as searching, insertion, and deletion in a Binary Search Tree (BST) is O(log n), where n is the number of nodes in the tree.

In a BST, all nodes in the left subtree have keys less than the node's key, while all nodes in the right subtree have keys greater than the node's key. Therefore, when **searching** for a node, the target value can be compared with the current node's key to decide whether to continue searching the left or right subtree, effectively reducing the search range by half each time. As a result, the time complexity of searching is O(log n).

For **insertion** and **deletion** operations, the node's position to be inserted or deleted needs to be found based on the definition of a BST, and the node is inserted or deleted accordingly. Again, since the search range is halved at each step, the time complexity is also O(log n).

In the worst case, a BST may degenerate into a linked list, resulting in a time complexity of O(n) for all operations. Therefore, unique data structures such as **balanced binary search trees** can be used to ensure performance and prevent worst-case scenarios.

Binary Search Tree Visualization: https://www.cs.usfca.edu/~galles/visualization/BST.html

3. Balanced binary search tree

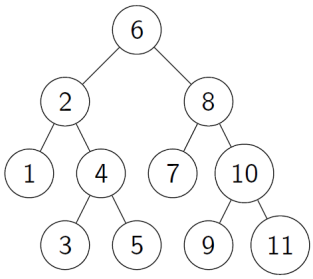
A balanced binary search tree (BBST) is structured to ensure that its operations do not degrade to O(n) time complexity. When a binary search tree becomes unbalanced, for example, when one branch is much deeper than the other, the time complexity of lookup operations will degrade to O(n), which is not a desired performance outcome.

Common BBSTs include AVL trees, red-black trees, and others. These trees automatically balance themselves during insertion and deletion operations to maintain their balance, ensuring that the time complexity of any process remains at the O(log n) level. Thus, BBSTs are highly efficient data structures.

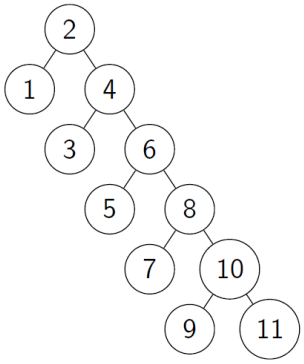
4. Exercises

1. **Decide whether a binary tree is a binary search tree or not.**

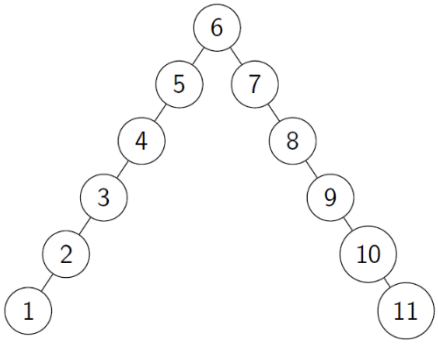
a.



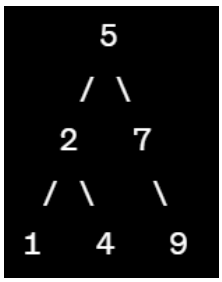
b.



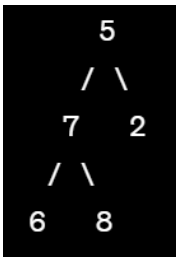
c.



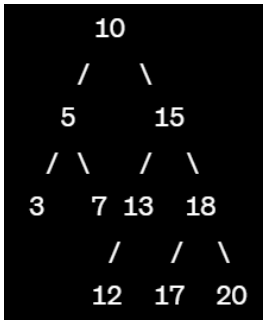
d.



e.



f.



2. Write down the in-order traversal on these binary search trees in exercise 1.
(inorder traversal: left subtree -> root nodes -> right subtree)

3. Build up a binary search tree (use Insert):
a. Insert(5), Insert(7), Insert(11), Insert(3), Insert(1), Insert(6), Insert(20), Insert(4), Insert(2), Insert(13).
b. Insert(8), Insert(4), Insert(5), Insert(2), Insert(11), Insert(1), Insert(16), Insert(3), Insert(13), Insert(15).

4. Find an element in a binary search tree (use Search):
a. In 3.a, Search(4), Search(20).
b. In 3.b, Search(3), Search(15).

5. Remove elements from a binary search tree (use Delete):
a. In 3.a, Delete(4), Delete(7), Delete(5).
b. In 3.b, Delete(2), Delete(3), Delete(16).