Compound data structures

List (linked lists)

- Adding — $O(n)$
- Deleting — $O(n)$
- Searching — $O(n)$

Tree

- Adding — $O(n)$ if we have to find the parent node.
- Deleting — $O(n)$. There are three cases here:
    - Leaf node — just remove it
    - One child — swap and remove
    - Two children — remove it and replace it with the in-order successor
- Searching — $O(n)$

BST

- If a tree is balanced then all operations take $O(\log n)$, if its not then $O(n)$.
- Adding a node to BST:

```
def insert(self, key):
    if self.root is None:
        self.root = TreeNode(key)
    else:
        self._insert(self.root, key)

def _insert(self, root, key):
    if key < root.val:
        if root.left is None:
            root.left = TreeNode(key)
        else:
            self._insert(root.left, key)
    else:
        if root.right is None:
            root.right = TreeNode(key)
        else:
            self._insert(root.right, key)
```

- Deleting a node in BST:

```
def delete(self, key):
    self.root = self._delete(self.root, key)

def _delete(self, root, key):
    if root is None:
        return root

    if key < root.val:
        root.left = self._delete(root.left, key)
    elif key > root.val:
        root.right = self._delete(root.right, key)
    else:
        if root.left is None:
            return root.right
        elif root.right is None:
            return root.left

        temp = self._min_value_node(root.right)
        root.val = temp.val
        root.right = self._delete(root.right, temp.val)

    return root
```