1. List: implementation – array & linked list
2. Linear list – each element has only one predecessor and one successor
   1. Implementation: linked-list
      1. Differences between linked list and array
      2. Basic types of linked list: singly linked list, doubly linked list, circularly linked list, doubly linked circular list
      3. Basic operations: create a list, insertion, deletion, search, traversal, destroy a list
      4. Basic coding and big O analysis for different operations on sorted/unsorted linked list; on linked list with head pointer only or head pointer & tail pointer.
3. Non-linear list – each element can have multiple successors
   1. Tree – each element can have multiple successors but only one predecessor
      1. Concepts:
         1. Degree, in-degree, out-degree
         2. Root node, leaf node, internal node
         3. Parent, children, sibling, ancestor, Descendent
         4. Level, height
         5. Subtree – note that a single node is also a subtree.
         6. Balance factor (only exists for binary trees): it can be positive, negative or zero.
         7. Balanced tree; complete tree
      2. General Tree Operations: (how to conduct operations in preorder & postorder)

Note that, these operations are all done in a recursive way.

* + - 1. Traverse a tree: e.g. Print out all the nodes. What orders? Both preorder & postorder
      2. Count tree node: what order? Doesn’t matter
      3. Count tree height
      4. Destroy a tree: what order? Only postorder.
      5. Note that, the general tree is not sorted, so we have to traverse the whole tree to search a specific node or the min/max node. It’s very similar to traversal operation. Therefore, we did not explicitly mention search and Max/min.
    1. Binary Search Tree Operations: (Basic coding & big O analysis & illustration of different operations)
       1. Traverse a tree: similar to general tree case. Preorder & postorder & inorder. can only be done recursively.
       2. Count tree height: similar to general tree case.
       3. Search a tree: can be done both recursively and iteratively
       4. Max/Min: can be done both recursively and iteratively
       5. Insert a node: search + add; All inserts take place at a leaf or at a leaflike node
    2. AVL Tree Operations: (bigO analysis & illustration of different operations)
       1. Insertion (BST insertion + rebalancing)
       2. Deletion (BST deletion + rebalancing)
       3. Search (The same as BST search)

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **SET** | **Unsorted Array** | **Sorted Array** | **Hash Table** | **Unsorted Linked List** | **Sorted Linked List** | **BST** | **AVL** |
| Search | O(n) | O(log n) | O(n)  (expected O(1)) | O(n) | O(n) | O(h)  (Log(n) <= h <= n) | O(logn) |
| Add | O(n) + O(1) = O(n) | O(log n)+O(n) = O(n) | O(n)  (expected O(1)) | O(n) + O(1) = O(n) | O(n) + O(1) = O(n) | O(h)+ O(1) = O(h) | O(logn) |
| Remove | O(n) + O(1) = O(n) | O(log n)+O(n) = O(n) | O(n)  (expected O(1)) | O(n) + O(1) = O(n) | O(n) + O(1) = O(n) | O(h)+ O(1) = O(h) | O(logn) |
| Min/Max | O(n) | O(1) | O(**m**) | O(n) | O(1)(assuming fast access to tail) | O(h) (Log(n) <= h <= n) | O(logn) |