A Quick Review of Data Structure

- Data storing and manipulation.
- Sorting and searching, given n numbers stored in an array, binary search can be applied when they are sorted.
- We have $O(n \log n)$ "preprocessing" time, we can search fast in $O(\log)$ time.
- What if "dynamic" insertion and deletin are needed.

Membership problem

- $S = \{a_1, a_2, \ldots, a_n\}$
- operations are
  - membership of a, $a \in S$?
  - delete a, if $a \in S$, remove a from S.
  - insert a, if $a \notin S$, $S = \{a_1, a_2, \ldots, a_n\} \cup \{a\}$.
- Abstract data type, implementation details are ignore.

Implementation I: Unsorted Array

- Query: $O(n)$.
- Deletion: Search then delete, thus $O(n)$ and $O(1)$.
- Insertion: Search and insert, thus $O(n)$ and $O(1)$.

Implementation II: Sorted Array

- Preprocessing, i.e., sorting helps
- Query: $O(\log n)$.
- Deletion: Search then delete, thus $O(\log n) + O(n)$.
- Insertion: Search and insert, thus $O(\log n) + O(n)$.
- Fast searching but maintaining the data structure costs a lot. It doesn't improve the worst case time bound.

Implementation III: Linked List

- The big cost was for maintaining data structure, we replace the sorted array by sorted linked list. Deletion or insertion, we don't have to move data around.
- Query: $O(n)$, since binary search doesn't work.
- Deletion: $O(n) + O(1)$.
- Insertion: $O(n) + O(1)$.
- Linked structure avoids moving data around, but binary search won't work since we don't know where is the middle.
- Add link pointing to the middle. draw a figure

Implementation IV: Tree structure

- Query: $O(\log n)$ since tree height is $O(\log n)$.
- Deletion: $O(\log n) + O(\log n)$,

  still remember the deletion algorithm?
- Insertion: $O(\log n) + O(1)$.
- A problem, a sequence of insertions and deletions could cause the tree unbalance, thus we cannot have the $O(\log n)$ bound to the height of the tree.
- Need method to re-balance the tree if it is out of balance.

Balance Tree

- Height Balance Tree.
  - AVL-Tree.
  - Red-Black Tree.
  - 2-3 Tree or 2-3-4 Tree.
- Weight Balance Tree.

2-3-4 Tree
and
Concatenable Queue

- Contenable Queue
  - Store an order list $S = \{a_1, a_2, \ldots, a_n\}$.
  - Operations are:
    - membership, insertion, and deletion.
    - delete min or max.
    - concatenate two lists S_1 and S_2.
    - Split S into S_1 and S_2.
- In what cases that we need a concatenable queue?

2-3-4 Tree

- There are internal nodes and external nodes (leaves).
- Internal nodes can be a 2-node (a node has two children), 3-node, and 4-node.
- Internal nodes store branching information and external nodes store data.
- External nodes are at the same depth.
- draw figures