Augmented AVL Trees

Def: The <u>rank</u> of an integer wrt to a set S of integers is:

$$\text{rank}_S(n) = \left|\{ y \in S,\ y < x \}\right| + 1$$

Order-Statistic Tree:

keeps set of integers and supports Find / Insert / Delete, and also:

- Select(i): returns elements of S of rank i
- Rank(x): given pointer to some $x \in S$, returns $\text{rank}_S(x)$

Attempt 1: With each node u, let $u.key = x$ stores $\text{rank}_S(x)$

Problem: Insert() would be $\Omega(n)$

Ranks need to change for all larger elements.

Cannot get rank from only local information.

$\boxed{\text{Attempt 2:}}$ With each node u, store $u.size$ = # nodes in subtrees

Remark: sizes do not need to be similar.

We have: $u.size = u.l.size + u.r.size + 1$.

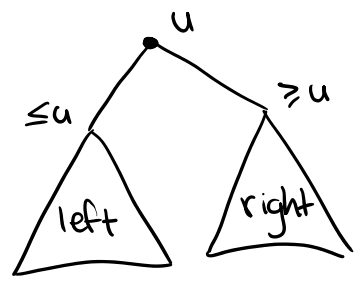
Hence, during update, we can recompute only from information regarding children.

Implementation:

Implementation:

- Select(u, i): returns element of rank i among nodes rooted at u.

```
Select(u, i):
    if i == u.left.size + 1 :
        return u.key
    if i ≤ u.left.size :
        return Select(u.left, i)
    return Select(u.right, i - u.left.size - 1)
```



- Rank: go up. accumulate if right child.

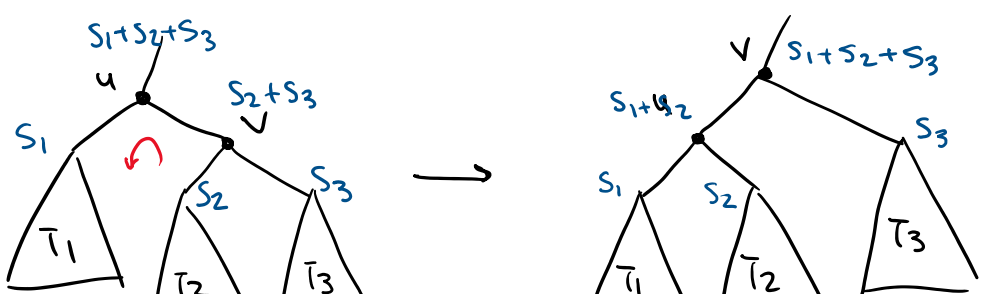
Complexity:

Both operations can be done in $O(\log n)$.

All recursive calls take at most "height" operations.

Maintaining size under Insert / Delete.

- Insert(x):
  a) Insert x, update size fields (traverse up) $O(\log n)$ more operations
  b) Balance when $BF = \pm 2$.



As there are a constant number of pointer changes. and recomputations are $O(1)$.

- Delete is basically symmetrical to Insert().

General Principle: (see CLRS)

If we augment an AVL tree with a new field $u.f$ at every node u, such that $u.f$ can be computed in $O(1)$ from information in $u.left$ and $u.right$.

Then, we can maintain the new field under Insert / Delete in $O(\log n)$.