Data Structures - Binary Search Tree

- We saw a data structure already: sorted array.
  - Searching for a key is fast: $O(\lg n)$
  - Inserting an element takes $O(n)$ time!

- We want a data structure T where the following operations are fast.

1) Search(T, x)
2) Insert(T, x)
3) Delete(T, x)
4) Min(T)
5) Max(T)
6) Predec(T, x)
7) Succ(T, x)

New:

8) OrdUnion(T, T'): where $T < T'$
9) Split(T, x): Find T', T'' s.t. $T' < \{x\} < T''$ & $T = T' \cup T''$.
10) Rank$(T, x) := \#\{\text{elts.} < x\}$.

- Qn: Can we achieve each of them in $O(\lg n)$ time?

YES - Use height-balanced binary search tree.

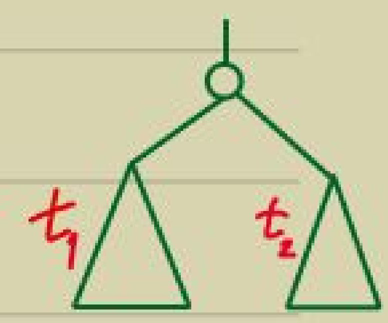
AVL Tree

- Named after Adelson-Velskii-Landis (1962).

- The properties/invariants of an AVL tree T:
  1) BST: values in left subtree < root < values in right subtree.
  2) Height-balanced: ht(left subtree) − ht(right subtree) is $-1, 0$ or 1.

▷ Height of T, storing n elements, is $\leq 2\lg n$.

Proof:



- Moving up ≤ 2 levels the #nodes covered becomes $t_1 + t_2 + 1$.
  $\Rightarrow$ Doubles wrt one of the subtrees.
- Since, the left (or right) subtree size can grow at most $\lg n$ times, the depth$(T) \leq 2\lg n$. $\square$

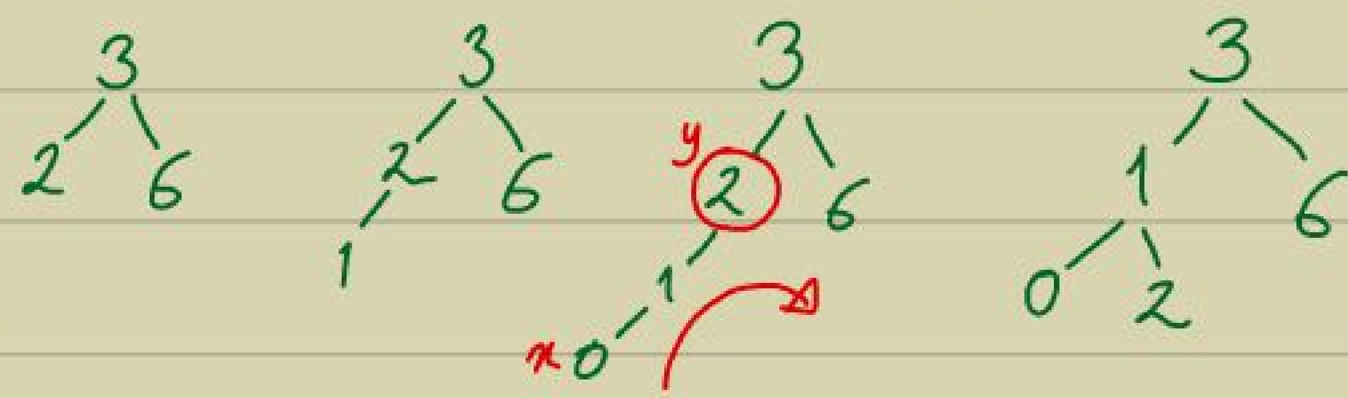
max(left(x))

Exercise: Search, Min, Max, Predec & Succ can be done in $O(\text{depth}(T))$-time.

▷ Moreover, in an AVL tree: Insert(T,x) can be done in $O(\lg n)$ time.

Proof idea:

- Add x in T following the BST rule.
- If one of the nodes y in the traversed path gets unbalanced, then rotate the tree rooted at y.

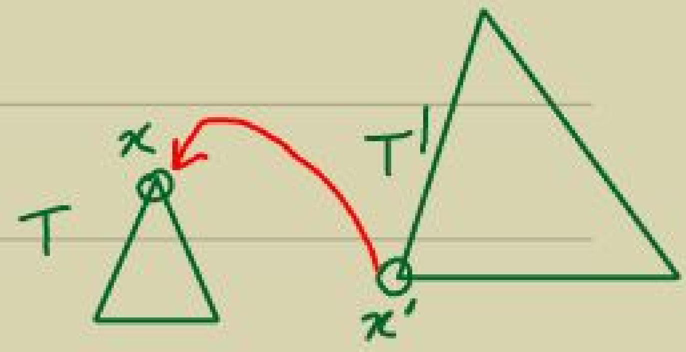


- Show that at most two rotations are needed.
- Doable in $O(\text{depth}(T))$-time. $\square$

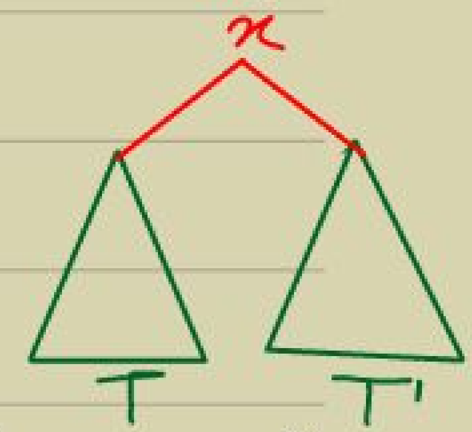
▷ Delete(T,x) takes $O(\lg n)$ time.

- Again, the idea is rotation. Here $O(\text{depth}(T))$ many rotations may be required.

OrdUnion(T, T')

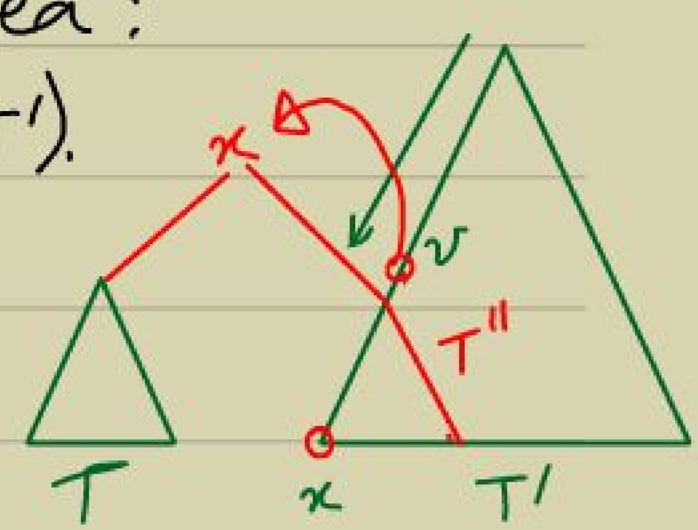


- Simply placing x under its predecessor x' gives an unbalanced tree.



- Simpler example: Suppose T & T' have equal heights.
  Then we can make left/right subtrees.

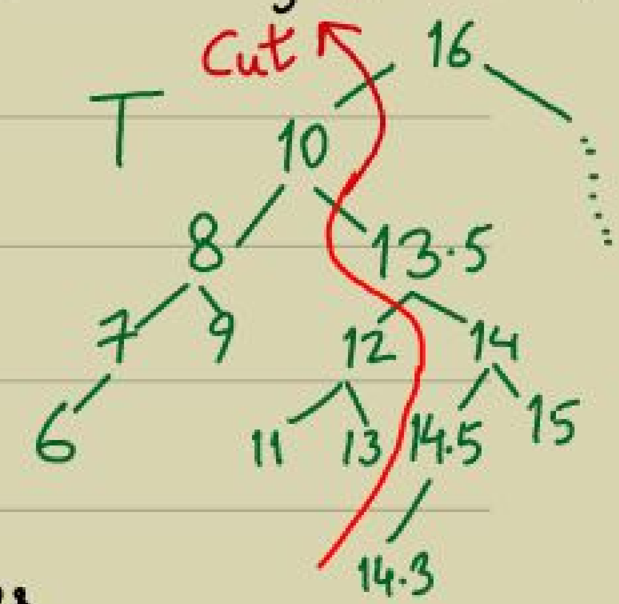
- How to extend this idea?



- Assume $depth(T) \leq depth(T')$.
- Find $x := least(T')$.
- Find $v \in T'$ such that $depth(left(v)) = depth(T)$.
- Define $T'' := left(v)$.

- Delete x from T''.
- Create a new node x with $left(x) := T$ & $right(x) := T''$.
- Reset $left(v) := x$.

<u>Exercise</u>: OrdUnion(T, T') takes $O(\lg n)$ time.



<u>Split(T, x)</u>

- Example: • For $x = 13.4$, start cutting T into subtrees, bottom-up.
  - In the eg, we get two subtrees $< x$.
  - Join them using OrdUnion.
  - Similarly, join the subtrees $> x$.
  - This gives us the split $T = T' < x < T''$.

<u>Exercise</u>: Split(T, x) can be done in $O(\lg n)$ time.

Rank(T, x)

- Idea 0: • Repeatedly find predecessors of x & stop when we reach min(T).
  • If Rank(T, x) =: k then this takes $O(k \lg n)$ time.
  (Exercise: $O(k + \lg n)$ time.)

- Better idea:
  • Follow the search-path from the root to x.
  • Which elements contribute to Rank(T, x)?

Ans: The left subtrees of the path & some vertices on the path.

• So, store size(v) := # elts in the subtree rooted at v, in every node v in T.

- Implementing this requires a change in the data structure.

  Augmented AVL Tree:

  Each node has an extra field size(v).

- Accordingly, one has to reimplement Insert(T, x) & Delete(T, x).

  Exercise: Do it in $O(\lg n)$ time.

- Finally, Rank(T, x) in the augmented tree:
  - Modify Search(T, x) algorithm using a new variable rank $\leftarrow 0$ as:
    - When taking a right link of v update rank $\leftarrow$ rank + size(left(v)) + 1.

$\triangleright$ Rank(T, x) can be computed in $O(\lg n)$ time.