Random Trees

A **Cartesian tree** is a binary tree, introduced by Vuilleman (1980) for geometric range searching.

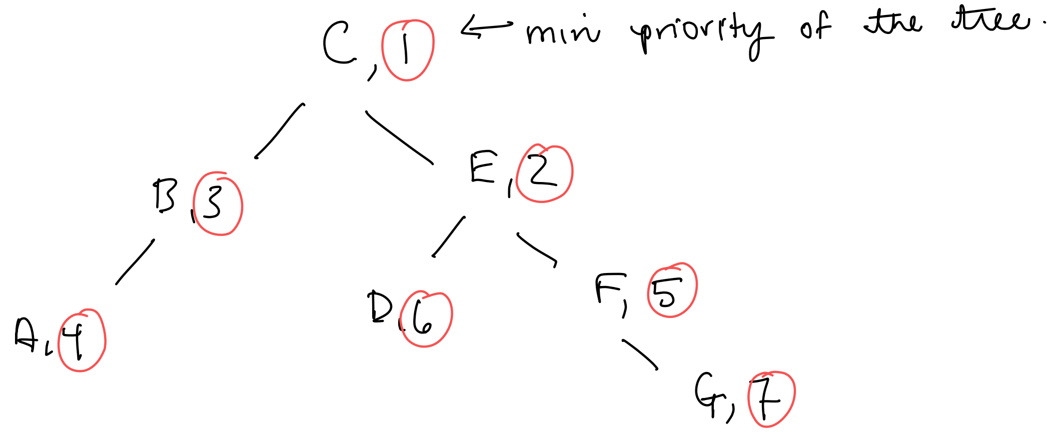
Ex.

Suppose given: A list of Keys and associated priorities:

Keys:	A	B	C	D	E	F	G.
Priorities:	4	3	1	6	2	5	7

Priorities $\Longleftrightarrow$ also called "time stamps".

Build a BST on the keys by inserting them in the order given by their priorities.



Note that the tree has min heap priority based on the *Time stamps/priorities*

The *inorder* traversal of the tree results in the sorted keys. This is equivalent to BSTs.

DEFINITION: Cartesian tree

Given a set $(x_1, y_1), \ldots (x_n, y_n)$

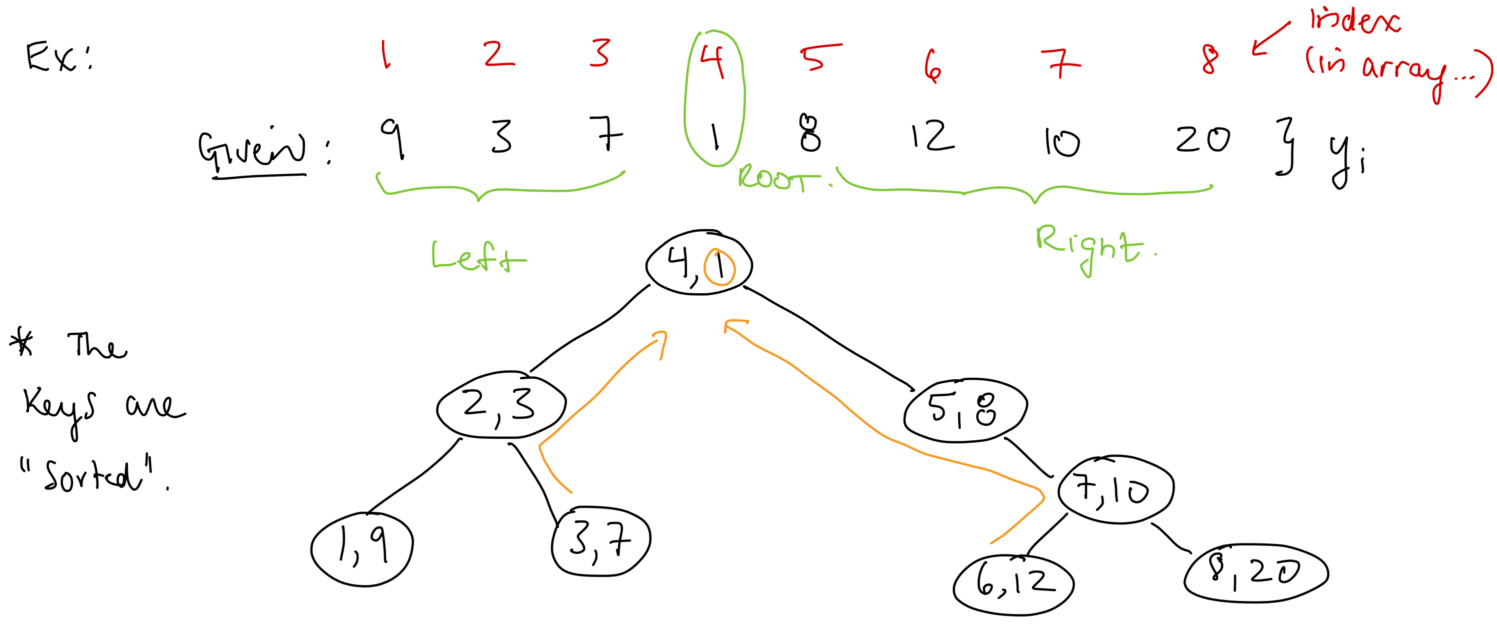
x_1: Key; y_1: priority (time stamp).

A cartesian tree is a B.S.T. with respect to the Keys and a min. heap with respect to the priorities.

Fact: Given data $(x_1, y_1,) \ldots (x_n, y_n)$, the cartesian tree is unique.

* The choice of root is unique, the L/R subtrees are determined by the root.

A Cartesian tree can be built on a sequence of items $y_1, \ldots, y_n$ by using the $y's$ as the priorities and the order of the items as the keys:

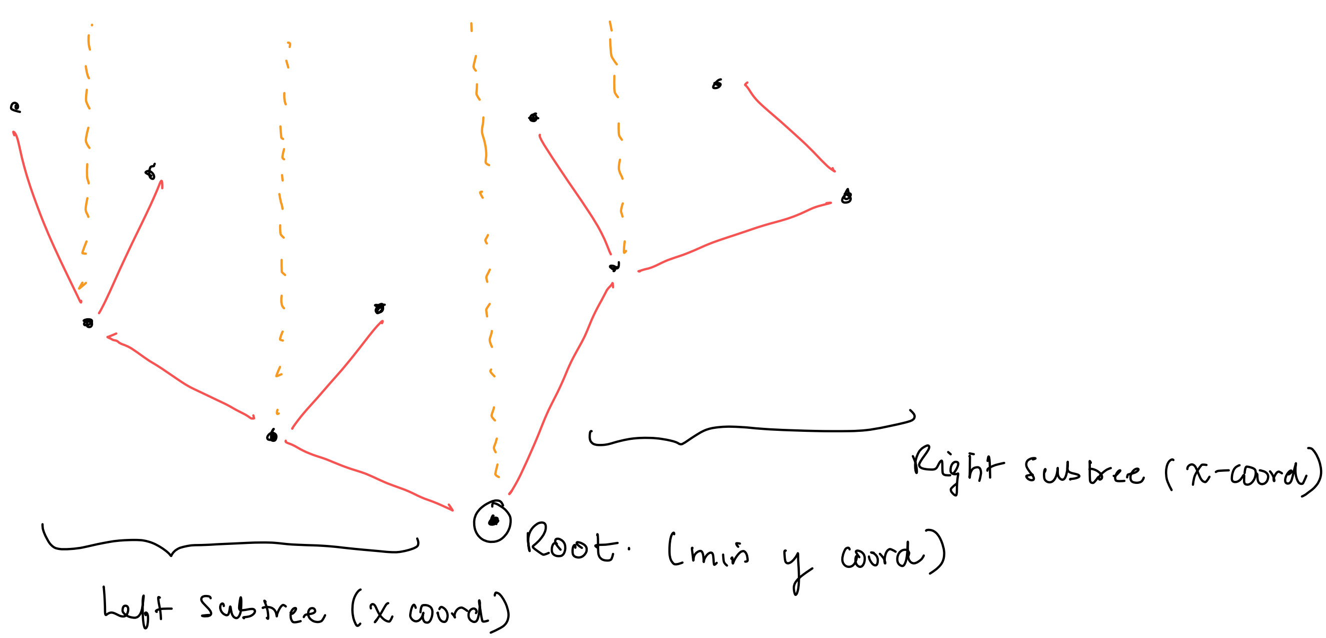


Application: Range Searching: Given 2 elemens in the above sequence/array, the minimum value in the sbuarray is the **lowest common ancestor** in the tree.

eg. 9 3 7 1 8 12 10 20

min. element. * see above.

Application: Cartesean trees can be used to store 2-dim data:
BST keys: x-coord
Priorities: y-coord



Suppose we have keys $x_1, \ldots, x_n$ for a **Binary search tree**. To build a Cartesian tree, we would need priorities. We could use the priorities $1, \ldots, n$.

$x_1 \; \ldots\ldots \; x_n$ Keys
$1 \; \ldots\ldots \; n$ priorities.

The result ⇒ Keys get inserted into BST in order $x_1 \to x_n$. Could be unbalanced!!

In a **Random binary search tree** the keys are inserted in **Random** order. This could be achieved by **permuting** the data and the inserting them one by one into the Cartesean tree with priorities $1, \ldots, n$.

Keys : (after permutation) $x_{\sigma_1} \; \ldots\ldots \; x_{\sigma_n}$
priorities : $1 \; \ldots\ldots \; n$

⇒ The result ⇒ Keys are inserted into the tree in random order.

The above construction of a Cartesean tree is equivalent to a Random binary search tree. So it's depth is logarithmic with high probability!.

Problem: How could we maintain this randomized property as items are added/removed from the tree? This happens if we receive only 1 item at a time...

TREAPS

First introduced by Aragon, Seidel (1989). It is a Cartesean tree where each node received a **Randomly chosen priority**

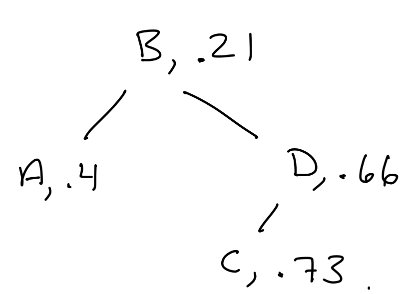
- Given Keys $x_1 \; \ldots \; x_n$
- Assign priority T_i to each x_i.

$\downarrow$

Random value (ex. uniform # in $[0,1]$).

ex

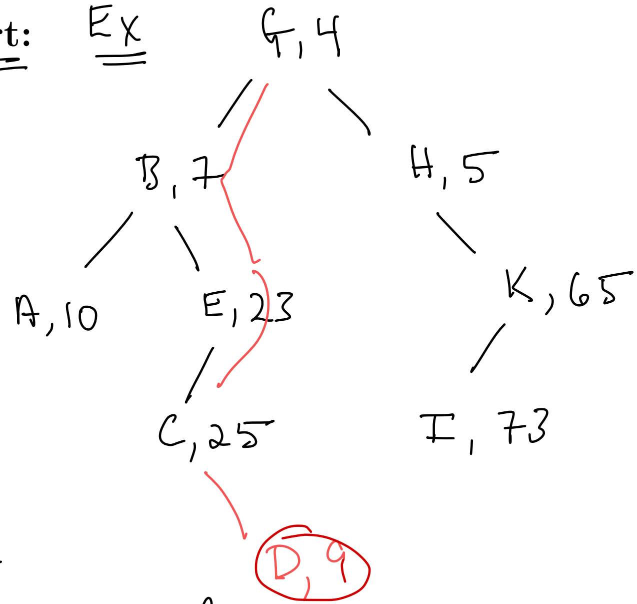
Keys	A	B	C	D
assign priorities	.4	.21	.73	.66



Treaps are easy to maintain as new items are added/removed ...

Operations: Goal is to "maintain" the random BST relationship...

Insert: Ex

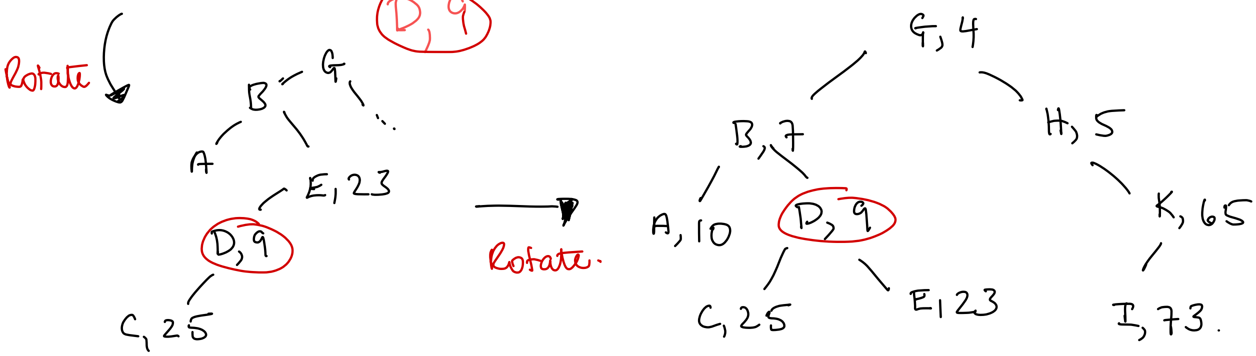


Insert 'D':

→ assign priority (random) 9.

→ B.S. to insert D into tree.

→ perform rotations to fix the heap.

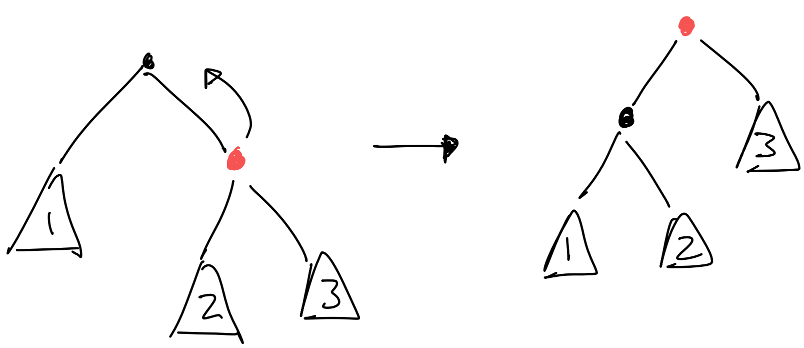


The insert operation:

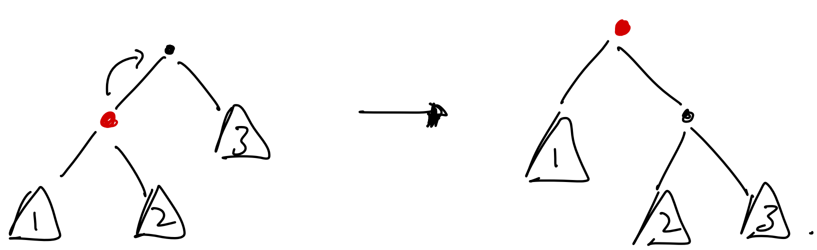
- assigns the new node a random priority
- searches for the position according to the BST
- performs a sequence of roations to maintain the heap property

The rotate operation:

Left:



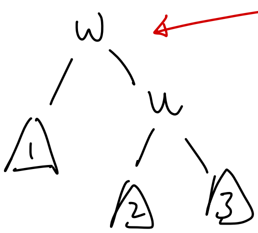
Right:



Pseudocode for Moving node u up: (Rotate up).

while $u \neq root$ && $time[parent[u]] > time[u]$

Before:



$w = parent[u]$

if $right[w] == u$ then:

$right[w] = left[u]$.

$left[u] = w$

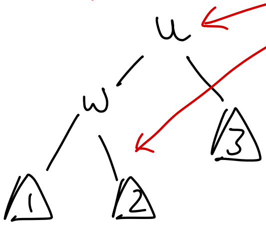
$parent[u] = parent[w]$

$parent[right[w]] = w$

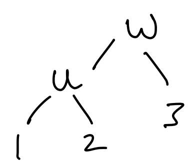
$parent[w] = u$

Left rotation.

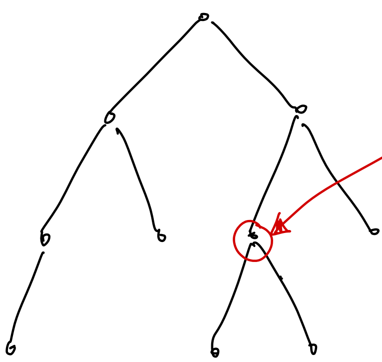
after:



else (right rotation ....) * see notes.



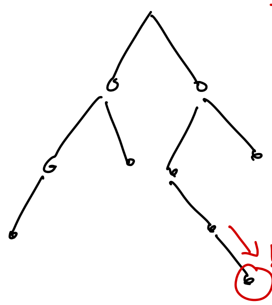
Delete a node:



. To delete this node, make his priority ∞.

. Perform rotations (down) to maintain the heap.

Node will go to leaf!



Now chop off leaf!!

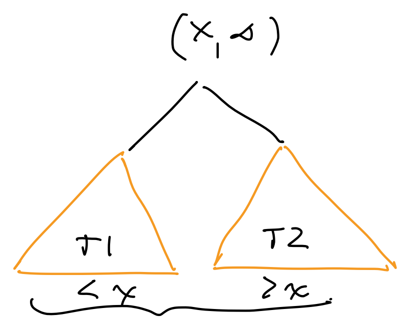
Split a treap: Given a value x, we wish to split the treap into 2 treaps, T1, T2 where:

$$Keys(T1) < x < Keys(T2).$$

How?

original treap

$\longrightarrow$ $\underbrace{\text{insert}(x, \infty)}_{\text{will become root.}}$ $\longrightarrow$ $\longrightarrow$



These are your 2 treaps.

Join 2 Treaps:

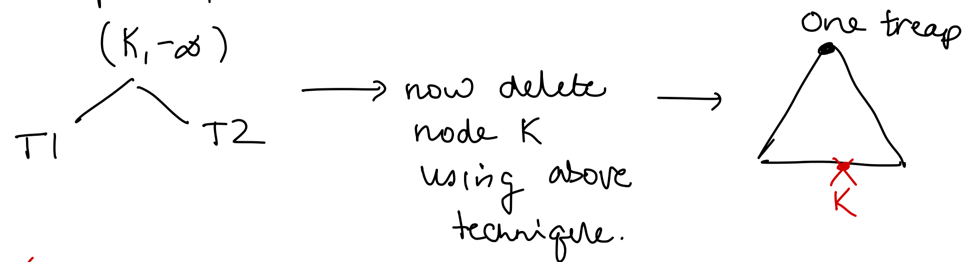
- Assume we are given 2 treaps, T1, T2 such that

  Keys(T1) < Keys(T2).

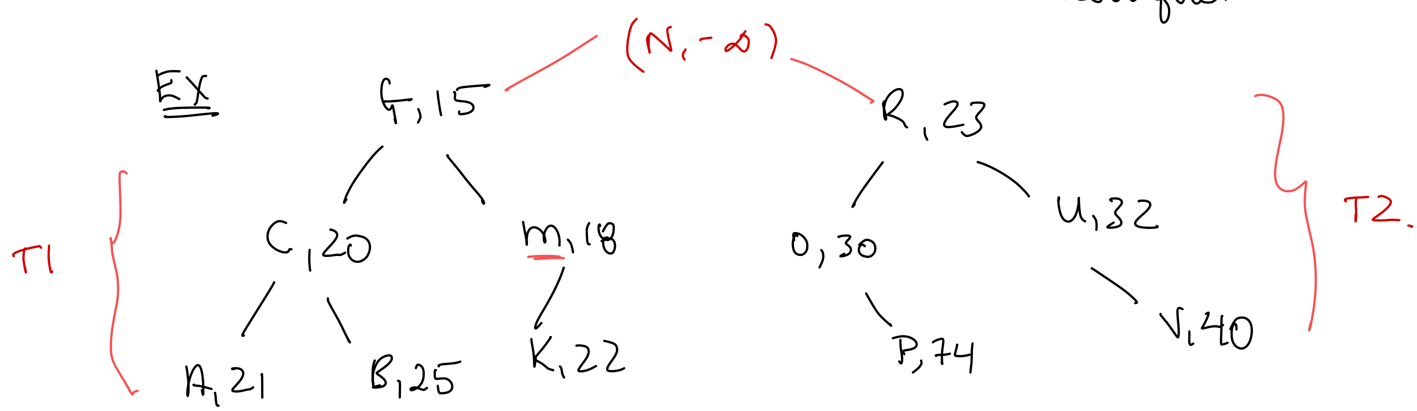
- Want to join the 2 treaps into one treap.
- How?
  - Create a node with key 'K' such that

    Keys(T1) < K < Keys(T2).

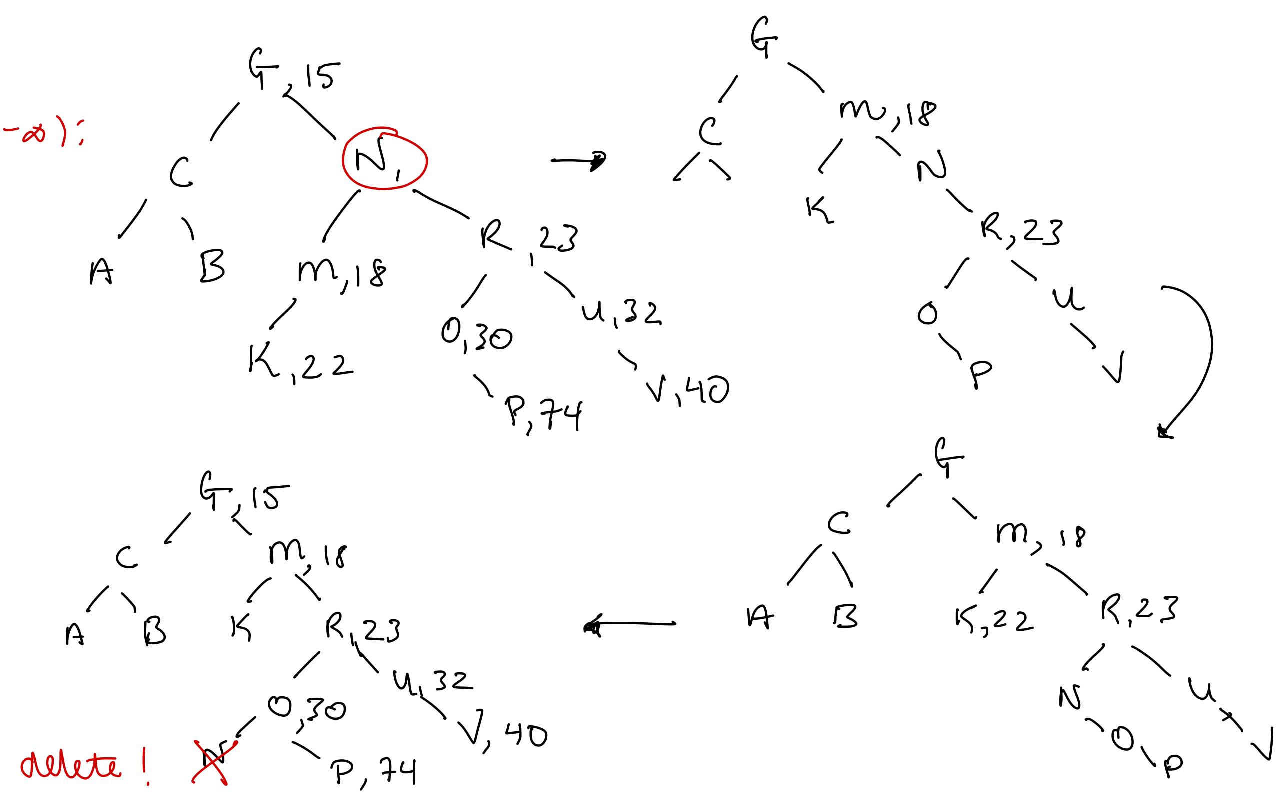
  - Join the 2 treaps by first placing K at the root, with priority $-\infty$:



EX



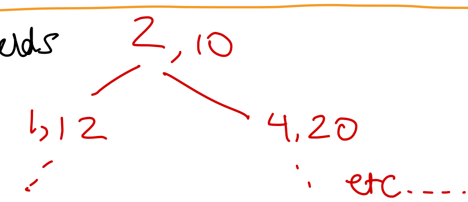
Now delete (N, $-\infty$):



Recall the Cartesian tree, given a sequence of priorities:

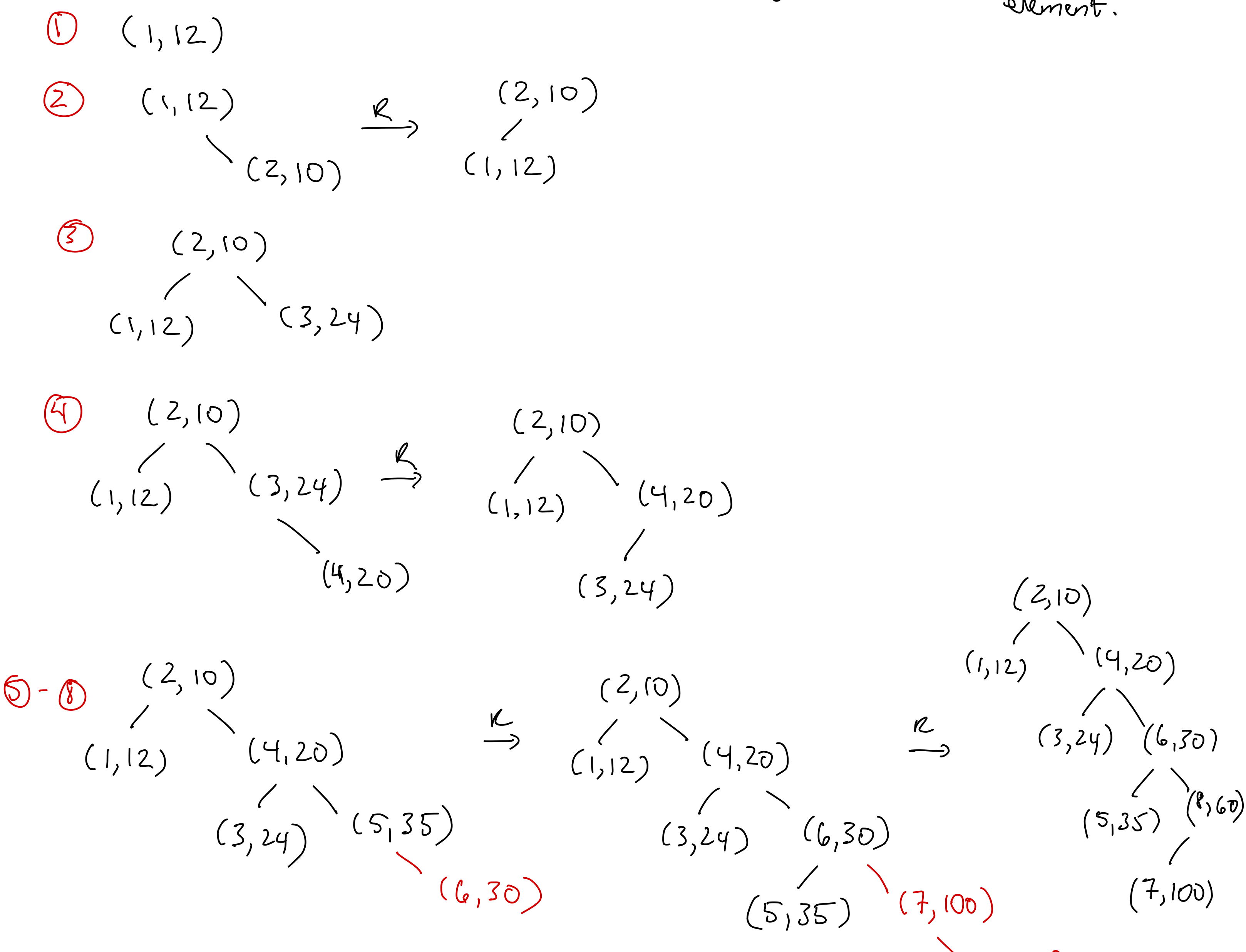
	Left		Right						
Keys:	1	2	3	4	5	6	7	8	9
Priorities:	12	10	24	20	35	30	100	60	18

Building this starting at the root, yields



The tree can also be constructed **sequentally** in linear time, using the above insert algorithm:

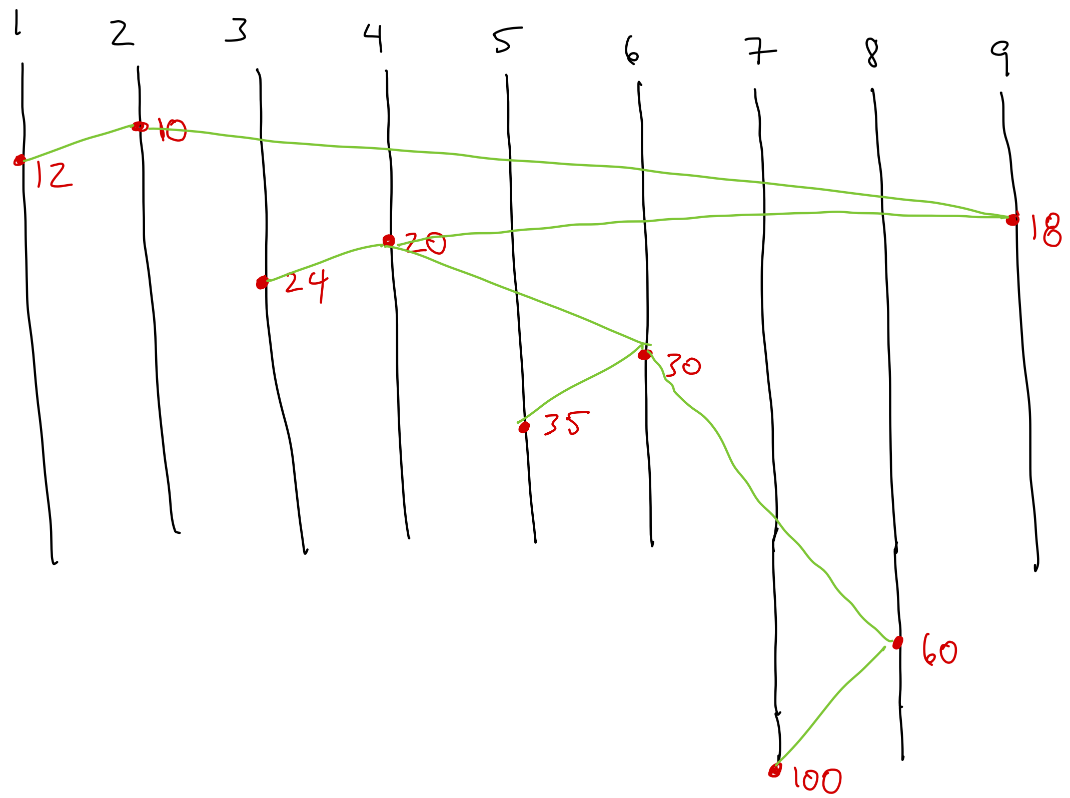
- Insert the items one by ones maintaining a cartesian tree at each step...
- The new Key will be inserted as the right child of the last element.



ETC...

A Cartesean tree on the sequence above can be pictured on an abacus...

The final tree can be drawn on vertical lines (Keys) where the 'height' represents the time stamp (priority...)



Towards the analysis of treap depth:

Recall that for a discrete Random Variable,

$$E(X) = \sum_n x_n P(X = x_n)$$

Assume that the data of our treap is $(1, T_1), \ldots, (n, T_n)$. In other words, our keys are $1 \ldots n$ for simplicity.

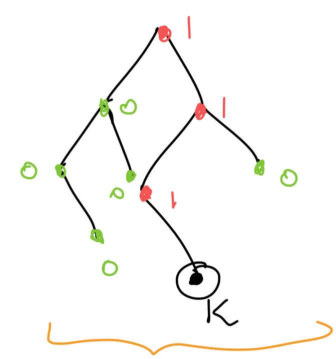
Keys

time stamps. (random).

Let the depth of node k be D_k:

$$D_K = \sum_{j \neq K} X_{jK}$$

$$X_{jK} = \begin{cases} 1 & \text{if } j \text{ is ancestor of } K \\ 0 & \text{otherwise.} \end{cases}$$



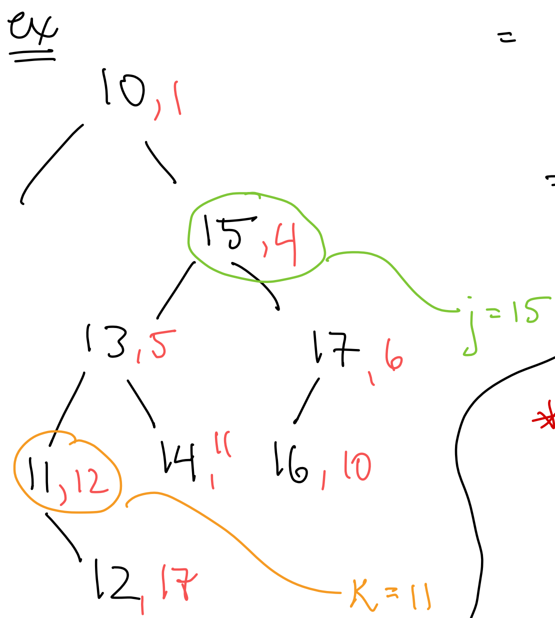
depth of $K = 3$

$= \sum X_{jK}$

$= 1+1+1+0+0+0+0+0$

To find the **Expected** depth we sum up the chance that each node in the tree is an ancestor of k:

$$E(D_K) = \sum_{j \neq K} P(j \text{ is ancestor of } K)$$



$$= \sum_{j<K} P(j \text{ is ancestor of } K) + \sum_{j>K} P(j \text{ is ancestor of } K)$$

$$= \sum_{j<K} P(T_j \text{ is smallest of } T_j \dots T_K) + \sum_{j>K} P(T_j \text{ is smallest of } T_K \dots T_j)$$

* Note 15 is ancestor of 11 because!

$$\begin{array}{ccccc} T_{11}, & T_{12}, & T_{13}, & T_{14}, & T_{15} \\ 12 & 17 & 5 & 11 & 4 \end{array} \quad \text{min!!}$$

$$= \sum_{j<K} \frac{1}{K-j+1} + \sum_{j>K} \frac{1}{j-K+1}$$

$$= \left(\tfrac{1}{2} + \dots + \tfrac{1}{K}\right) + \left(\tfrac{1}{2} + \dots + \tfrac{1}{(n-K+1)}\right)$$

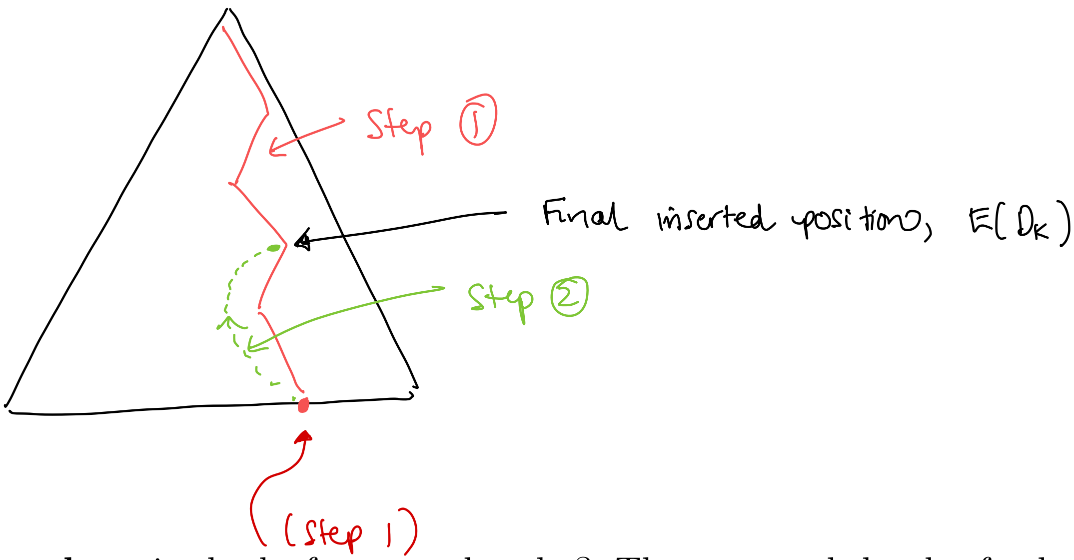
$$= H_K - 1 + H_{n-K+1} - 1$$

$$= \boxed{H_K + H_{n-K+1} - 2} \quad E(D_K).*$$

$$\cong 2\ln n \cong 1.39 \log_2 n.$$

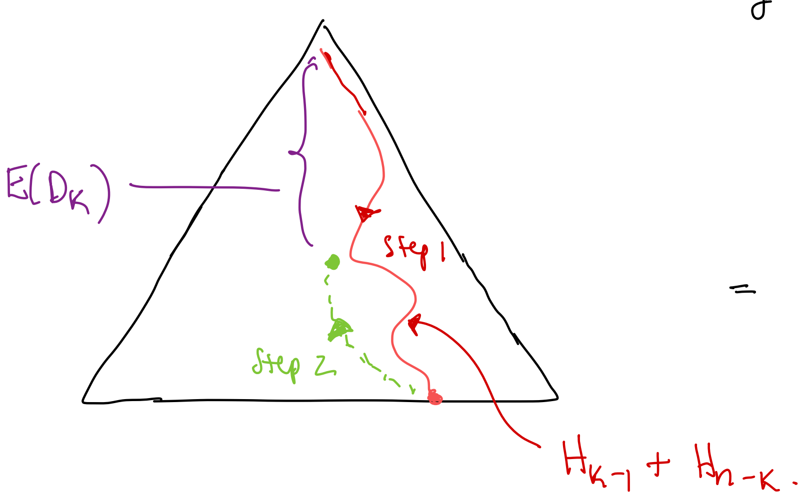
The **insert** operation:

Recall it first searches through the BST to insert the key as a leaf, and the rotates up to maintain the heap property of the priorities.



How **deep** is the leaf expected to be? The expected depth of a leaf is computed in the same way, however we assume it's priority is ∞ since it is at the bottom of the tree.

$$E(\text{depth of leaf}) = \sum_{j<K} \underbrace{\frac{1}{K-j}}_{P(T_j \text{ is the min of } T_j \cdots T_{K-1})} + \sum_{j>K} \underbrace{\frac{1}{j-K}}_{P(T_j \text{ is the min of } T_{K+1} \cdots T_j)}$$



$$= H_{K-1} + H_{n-K}$$

After the node is rotated up to its finial position, its depth is that of a typical random node:

$$E(D_K) = H_K + H_{n-K+1} - 2.$$

Thus the number of rotations is the difference between these 2 paths:

$$\text{Green} = (\overset{\text{red}}{H_{K-1} + H_{n-K}}) - (\overset{\text{purple}}{H_K + H_{n-K+1} - 2}) \le 2$$

The insert time is bounded by the expected depth of the leaf.

$\sim 2n \ln n.$

Similary, delete in a treap is $O(\log n)$.

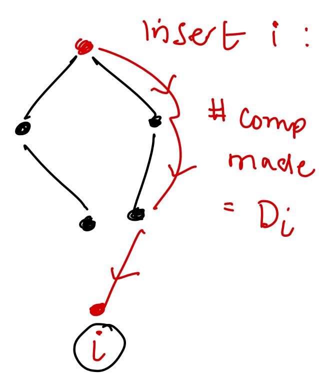
Note that the expected height of the treap is $O(\log n)$, (not shown).

QuickSort:

Note that the algorithm for quicksort is essentially the same as building a RBST.

- Pick pivot $\longrightarrow$ like picking root of RBST.
- Partition into L/R in quicksort $\rightarrow$ Partition into L/R subtrees.

The number of comparisons is thus the same as those made when building a BST:



$$\# \text{ comp} = \sum_{i=1}^{n} D_i$$

$$E(\#\text{comp}) = \sum_{i=1}^{n} E(D_i)$$

$$= \sum_{i=1}^{n} H_i + H_{n-i+1} - 2$$

ex. if $n=3$...

$(H_1 + H_3) + (H_2 + H_2) + (H_3 + H_1)$

$$= \left(2 \sum_{i=1}^{n} H_i\right) - 2n$$

$$= 2 \sum_{1}^{n} \underbrace{\sum_{j=1}^{i} \frac{1}{j}}$$ $- 2n$

$1 + (1 + 1/2) + (1 + 1/2 + 1/3) + \cdots$

Collect terms...

$$= 2 \sum_{1}^{n} \frac{1}{j} (n - j + 1) - 2n$$

$$= \left(2n H_n - 2\sum(1) + 2\sum 1/j\right) - 2n$$

$$= 2n H_n - 2n + 2 H_n - 2n$$

$$= 2n H_n + 2H_n - 4n$$

$$\cong 2n \ln n \simeq 1.38 n \log_2 n$$