Advanced Data Structures and Algorithms

Associate Professor Dr. Raed Ibraheem Hamed

Computer Science Department
College of Science and Technology
University of Human Development,

2015 – 2016

Random Numbers

I flipped a coin 30 times:

HTHTHHHTTTHHHTHTTTTTTHHHHHHTTH

One may almost suggest such a sequence is not random;

what is the probability of getting 6 tails and then 6 heads?

- Unfortunately, this is exactly what we got with the first attempt to generate such a sequence, and is as **random a sequence** as we can generate.

Randomized Data Structures

One application of randomized algorithms in the area of data structures, specifically, **Treaps** and **lists**.

Randomized Data Structures

Treaps

Skip lists

Treaps

- First introduced in 1989 by Aragon and Seidel
- Randomized Binary Search Tree
- A combination of a binary search tree and a heap (a "tree - heap")
- Each node contains
  - Data / Key (comparable)
  - Priority (random integer number)
  - Left, right child reference
- Nodes are in BST order by data/key and in heap order by priority random integer number.
- Every subtree of a treap is a treap

Treaps Definition

A treap is a binary search tree in which every node has both a **search key** and a **priority**, where the inorder sequence of search keys is sorted and each node's priority is smaller than the priorities of its children.

Tree + Heap = Treaps

- The search tree has the structure that would result if elements were inserted in the order of their priorities.

Definition: A treap is a binary tree. Each node contains ***one element x*** with key(x) ∈ U and prio(x) ∈ R. The following properties hold.

Search tree property

For each element x:

- elements y in the left subtree of x satisfy: key(y) < key(x)
- elements y in the right subtree of x satisfy : key(y) > key(x)

Heap property

For all elements x, y:

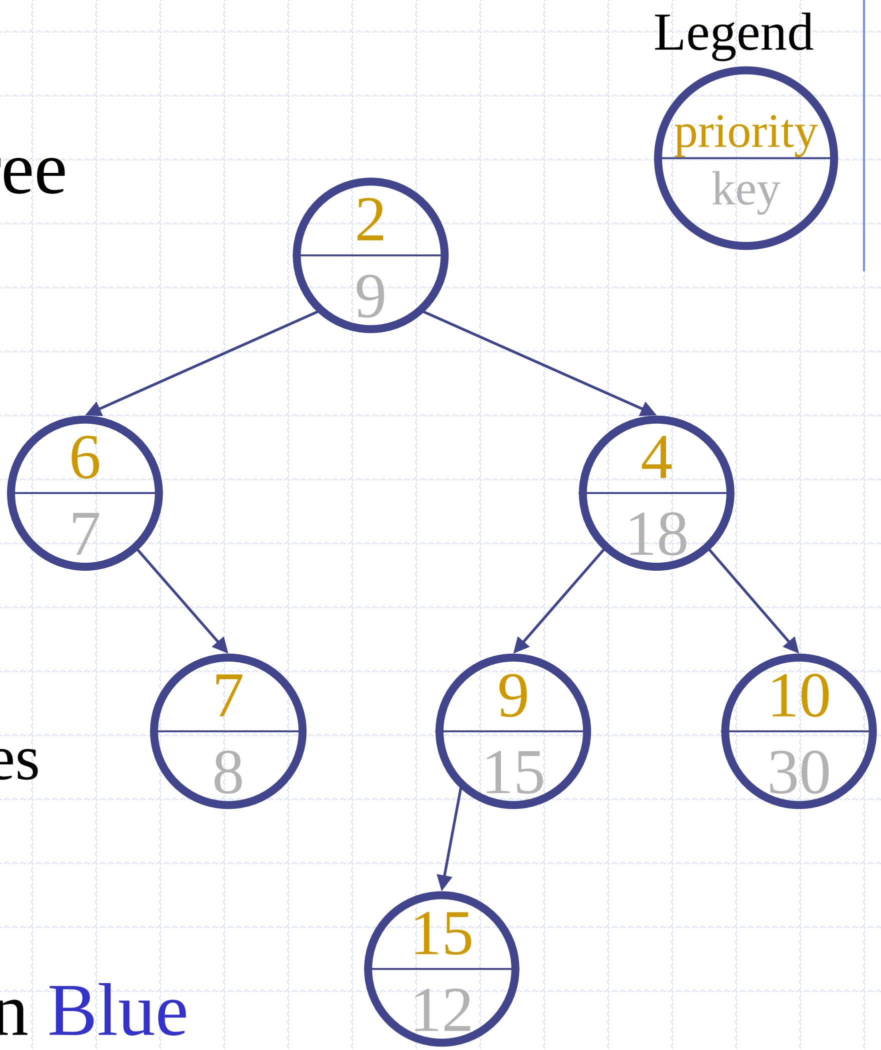
- If y is a child of x, then prio(y) > prio(x).
- All priorities are pairwise distinct.

Treap : Dictionary Data Structure

- Treap is a binary search tree
  - binary tree property
  - search tree property

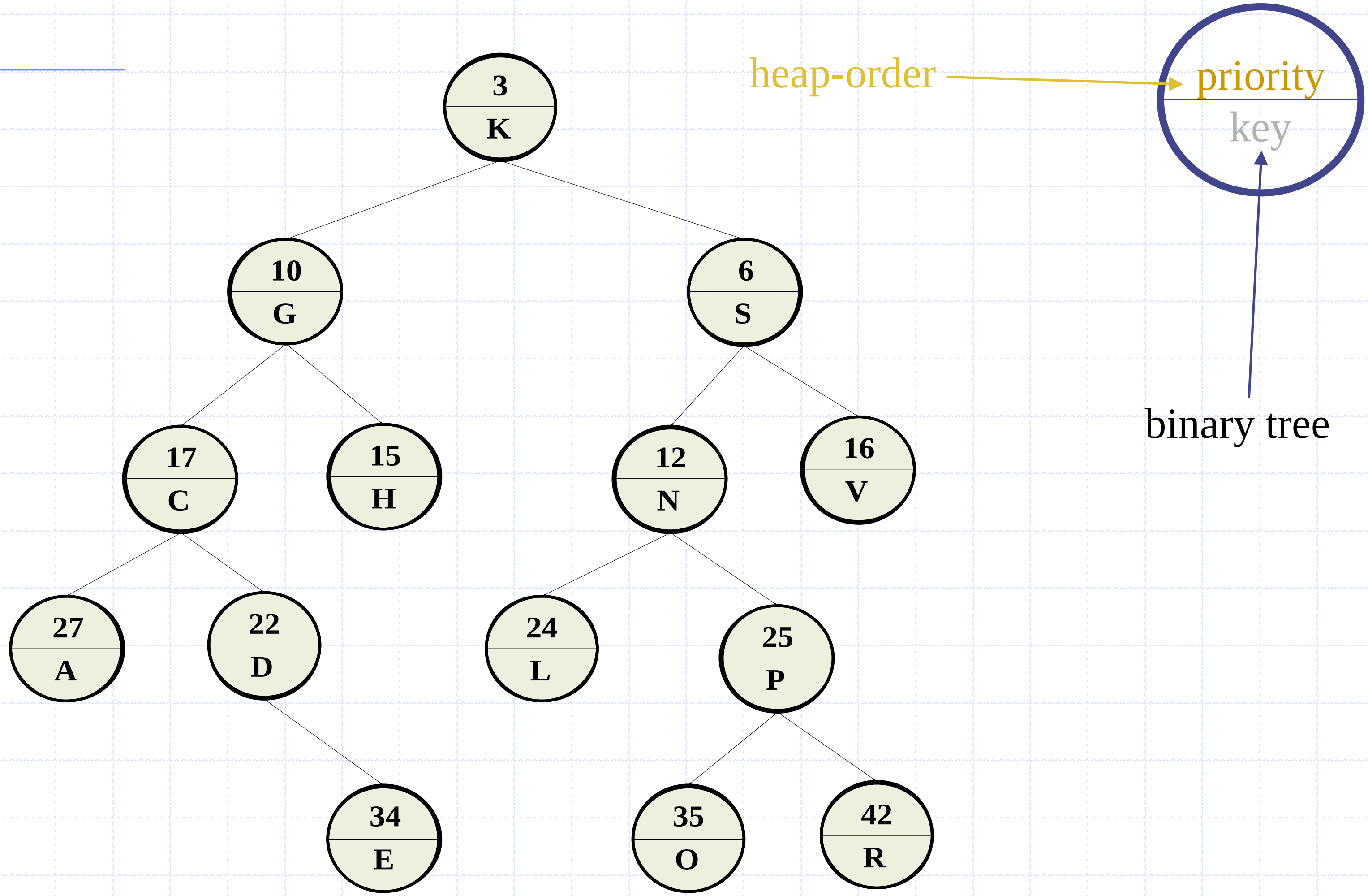
Treap is also a heap

- heap-order property
- randomly assigned priorities



Heap in Yellow; Search Tree in Blue

A Treap Example



A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

Treaps

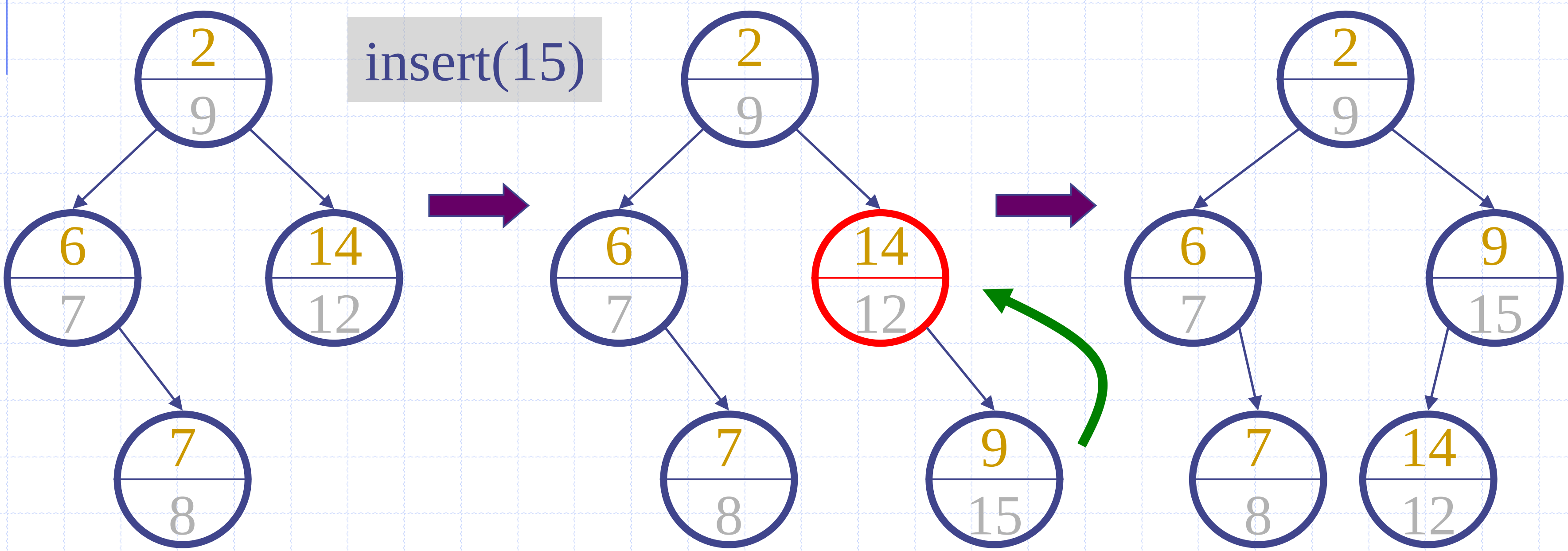
There is only one possible treap for a given **set of Keys** and priorities Proof:

- by heap property: **the key k**, with the **highest priority** must be the **root** of the treap
- by BST property: all keys < k must be in the left subtree of the root and all keys > k must be in the right subtree
- Inductively, the subtrees of the root must be constructed in the same manner

i.e. by heap property: the root of the LST, k1, must have the highest priority of any key in the LS **and** by BST: all keys in LST of k1 are < k1 and all keys in RST of k1 are > k1

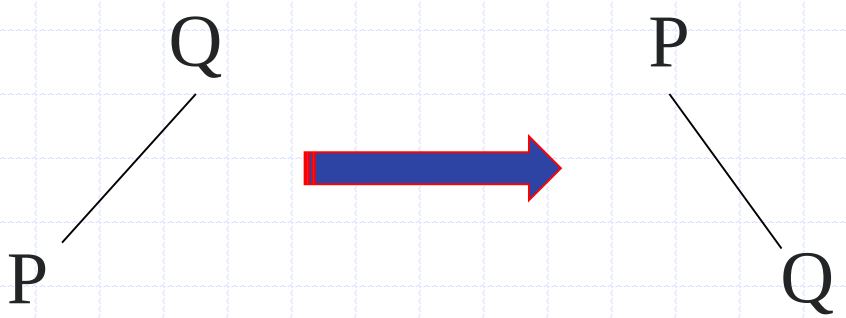
Treap Insert

- Choose a random priority
- Insert as in normal BST
- Rotate up until heap order is restored

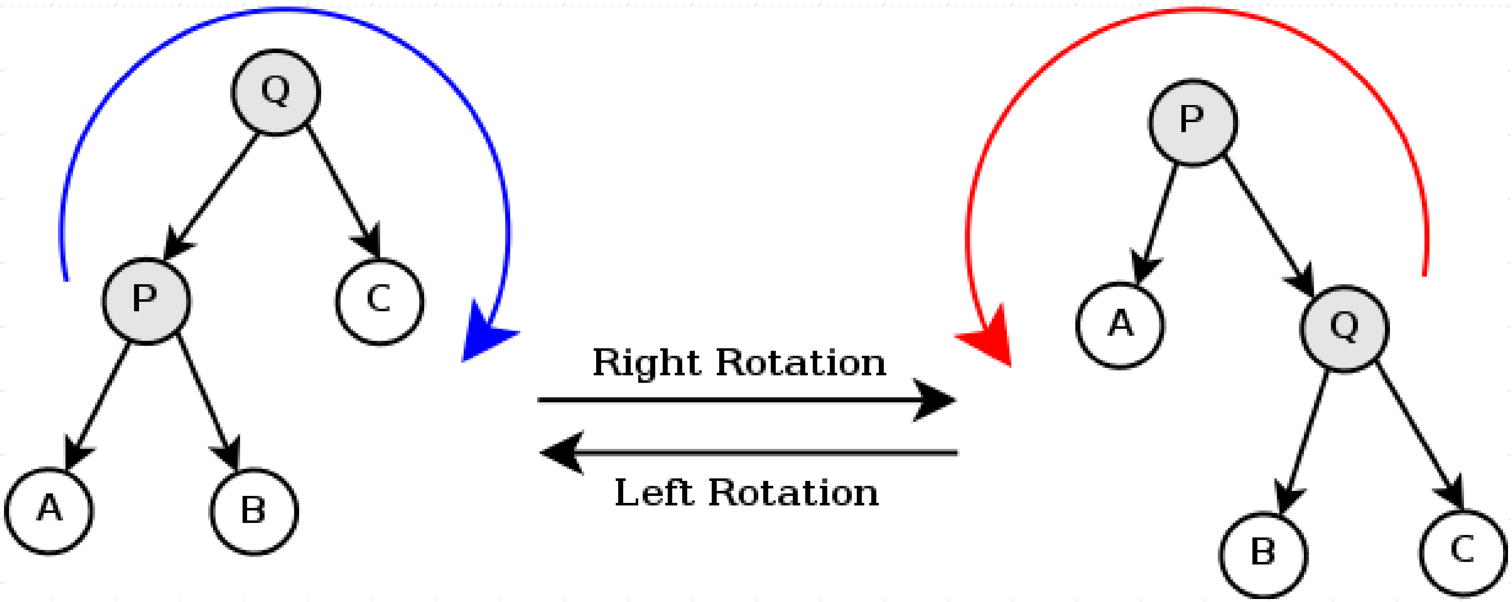


Binary Tree Rotation (Left or Right)

Let P be Q's left child. Set P to be the new root." Basically that's the description of the rotation to the **right** or clockwise:



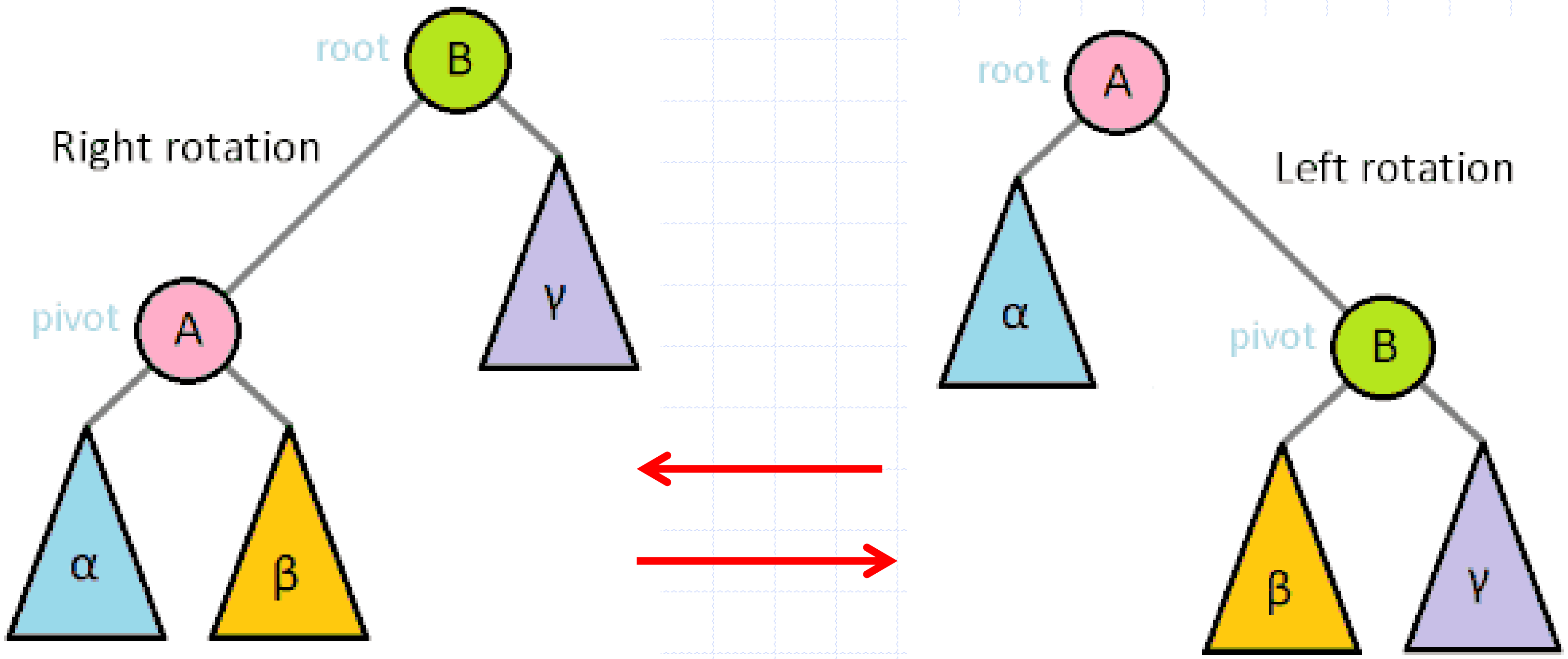
Binary Tree Rotation (Left or Right)



Binary Tree Rotation (Left or Right)

The right rotation operation as shown in the image to the left is performed with *Q* as the root and hence is a right rotation on, or rooted at, *Q*. This operation results in a rotation of the tree in the clockwise direction. The inverse operation is the left rotation, which results in a movement in a counter-clockwise direction (the left rotation shown above is rooted at *P*). The key to understanding how a rotation functions is to understand its constraints.

Binary Tree Rotation (Left or Right)



Binary Tree Rotation (Left or Right)

T1, T2 and T3 are subtrees of the tree rooted with y (on left side) or x (on right side)

```
         y                                x
        / \      Right Rotation          /  \
       x   T3    – – – – – – – – >      T1   y
      / \       < - - - - - - -              / \
    T1  T2     Left Rotation                T2  T3
```

Keys in both of the above trees follow the following order

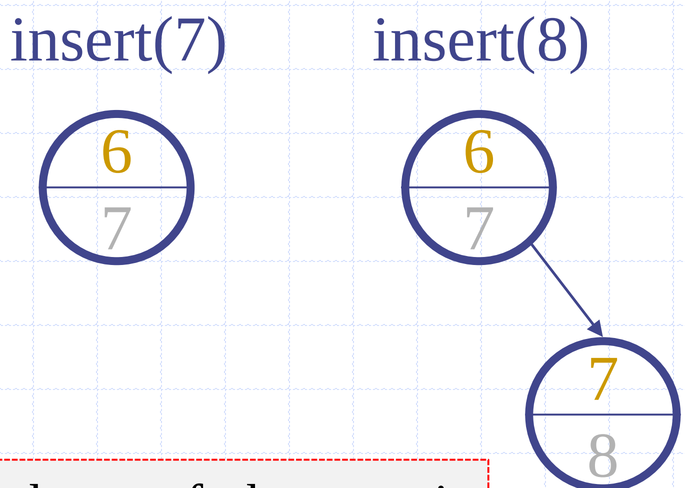
keys(T1) < key(x) < keys(T2) < key(y) < keys(T3)

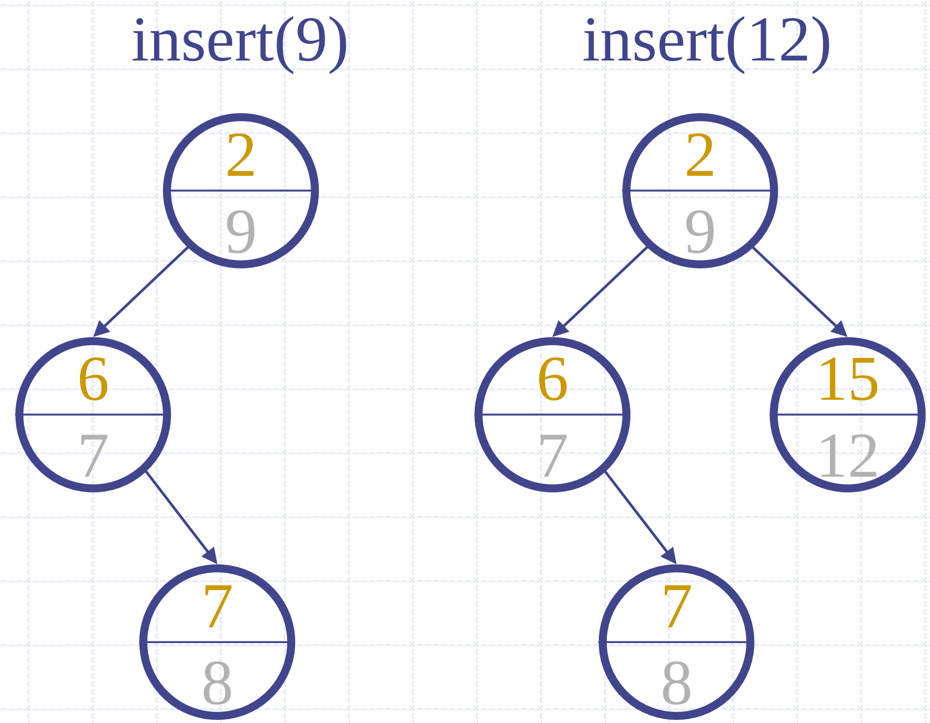
So BST property is not violated anywhere.

Tree + Heap… Why Bother?

- Insert data in sorted order into a treap …

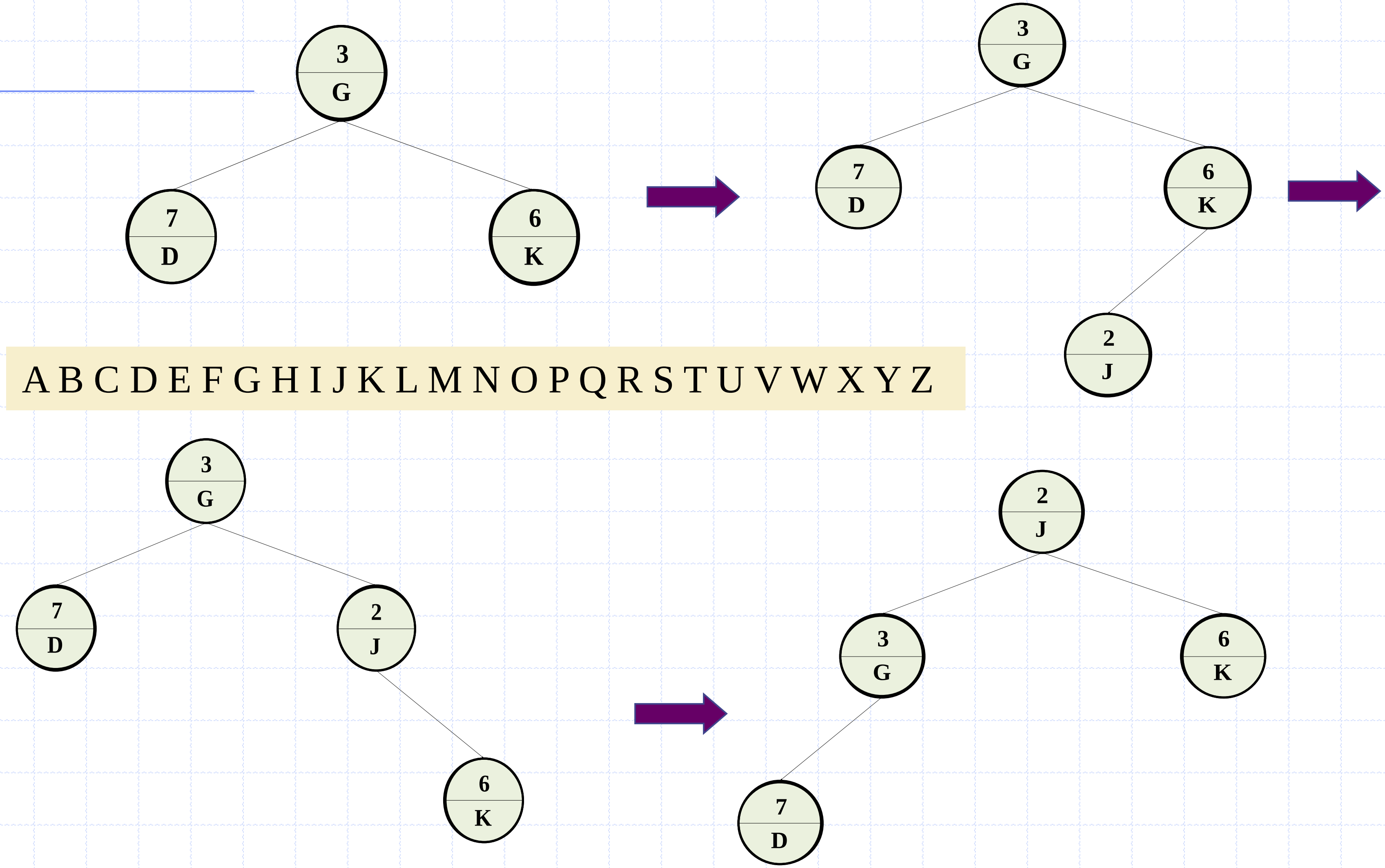
  What shape tree comes out?





The shape of the tree is **fully** specified by what the keys are and what their random priorities are!

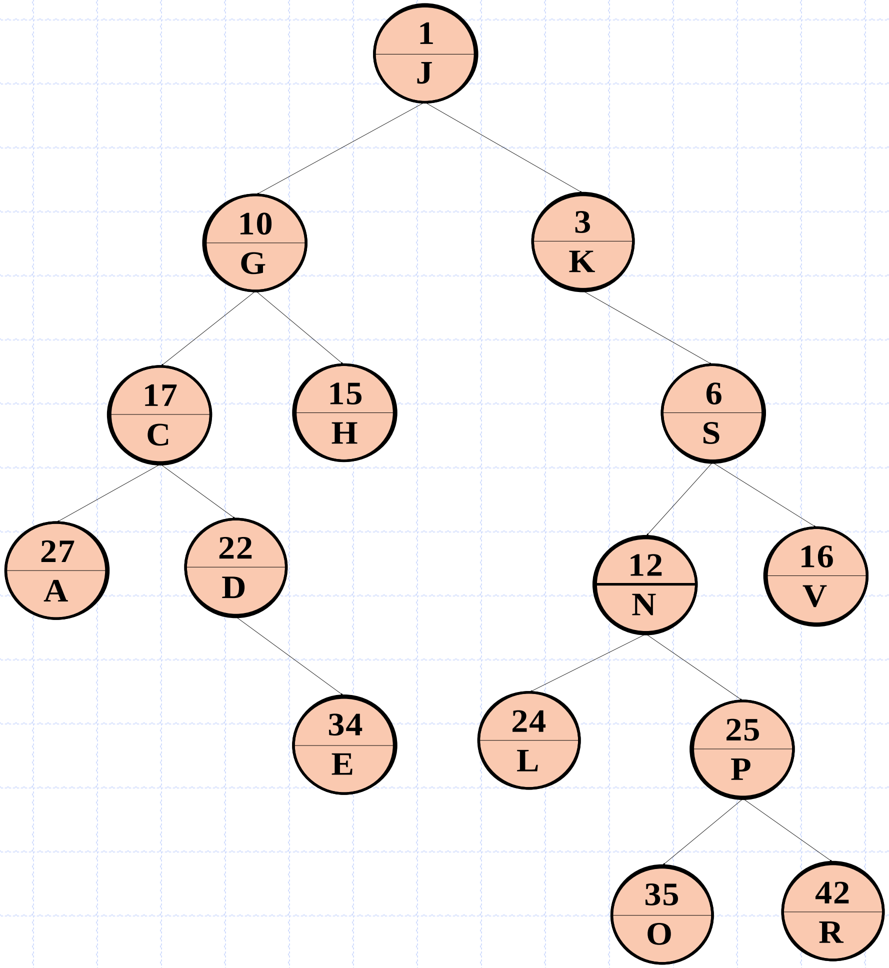
Insert J with priority 2



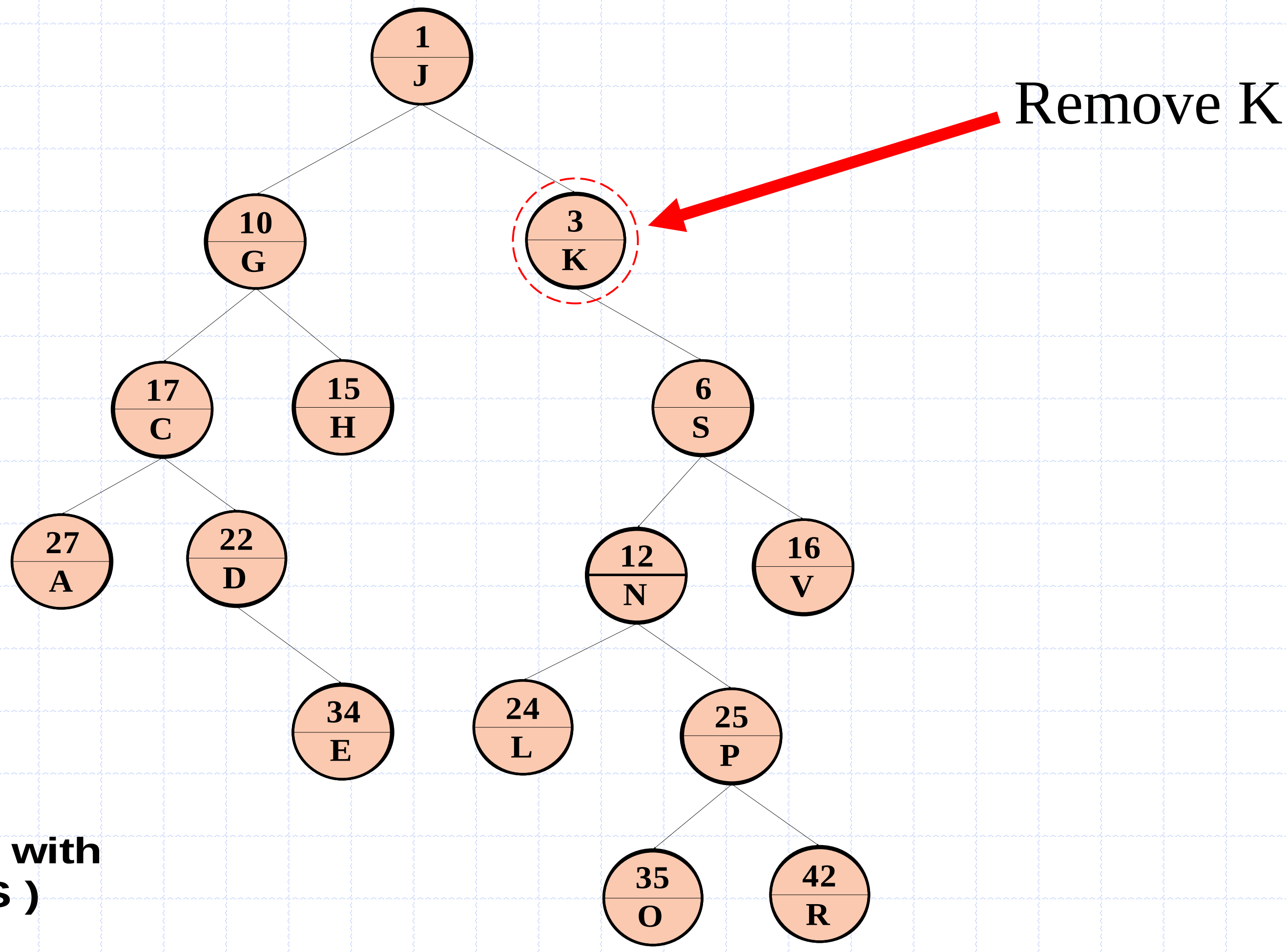
Treap Delete Strategy

- Find the key
- Increase its value to ∞
  - When X is found, rotate with child that has the smaller priority
  - If X is a leaf, just delete it
  - If X is not a leaf, recursively remove X from its new subtree
- Snip it off

Treap Delete Strategy

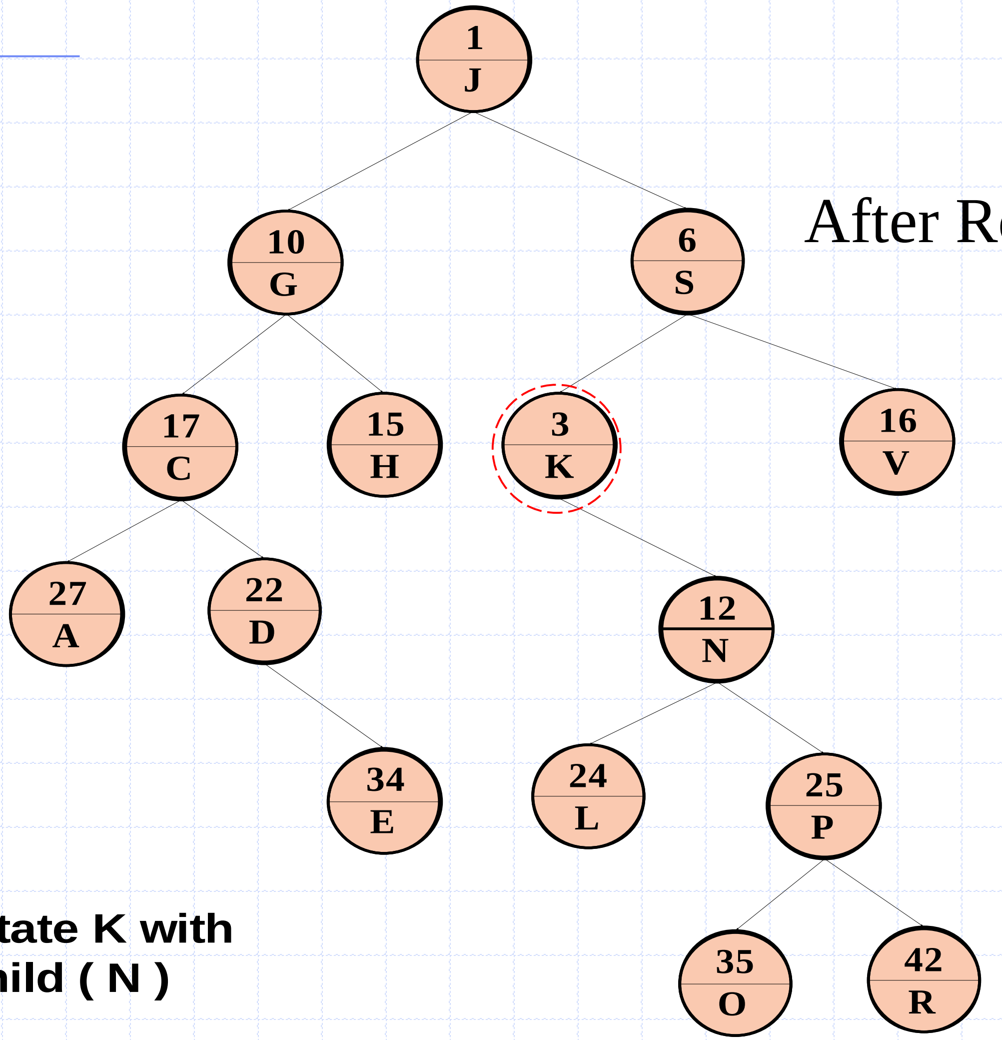


Treap Delete Strategy



Step 1 - Rotate K with Right Child (S)

Treap Delete Strategy

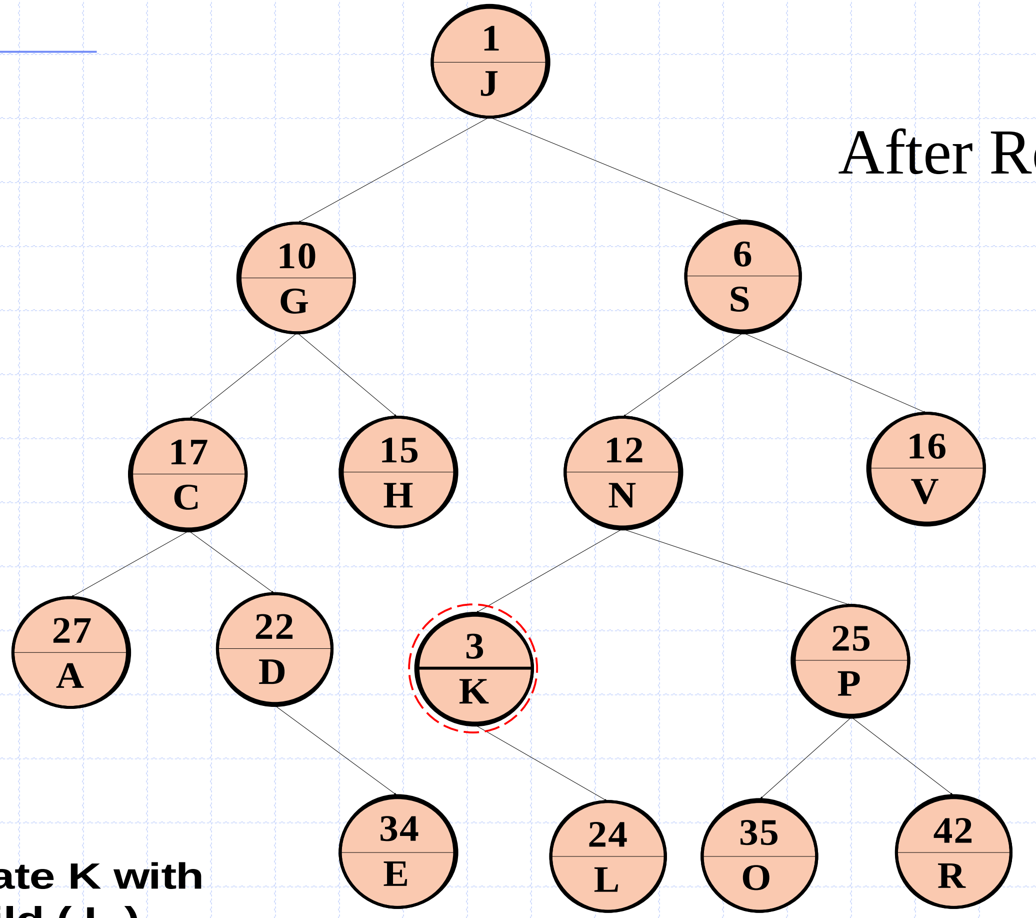


After Rotating K with S

Step 2 - Rotate K with Right Child (N)

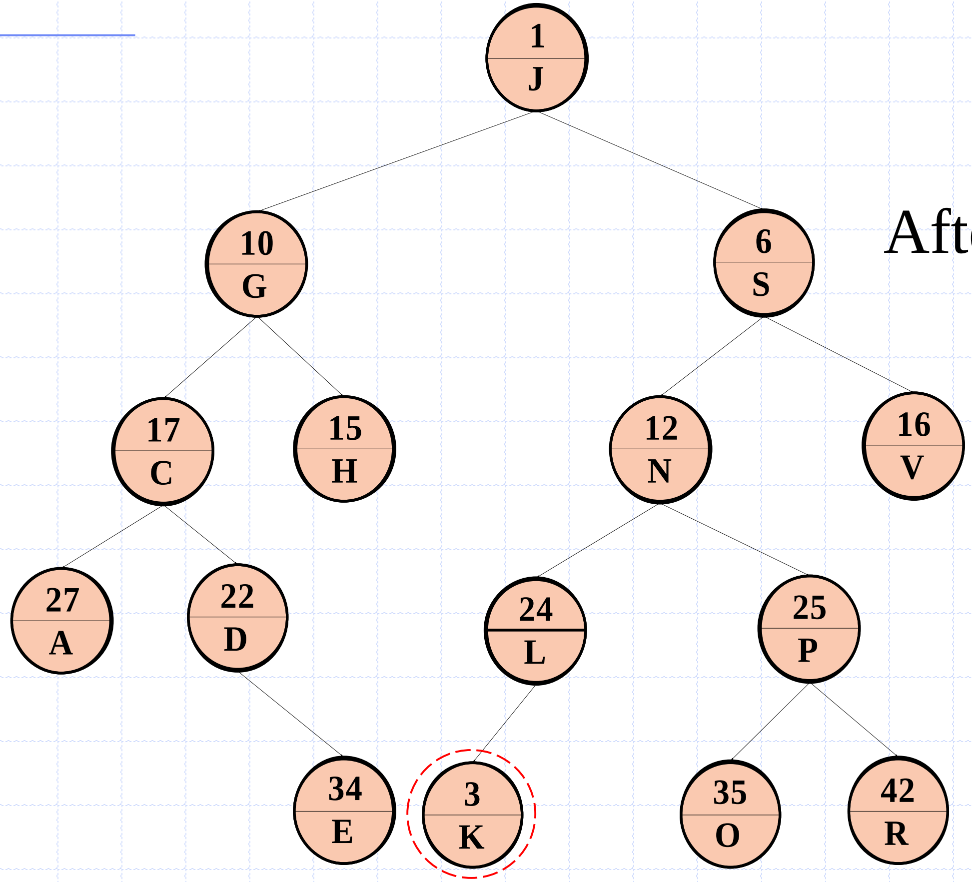
Treap Delete Strategy

After Rotating K with N



Step 3 - Rotate K with Right Child (L)

Treap Delete Strategy

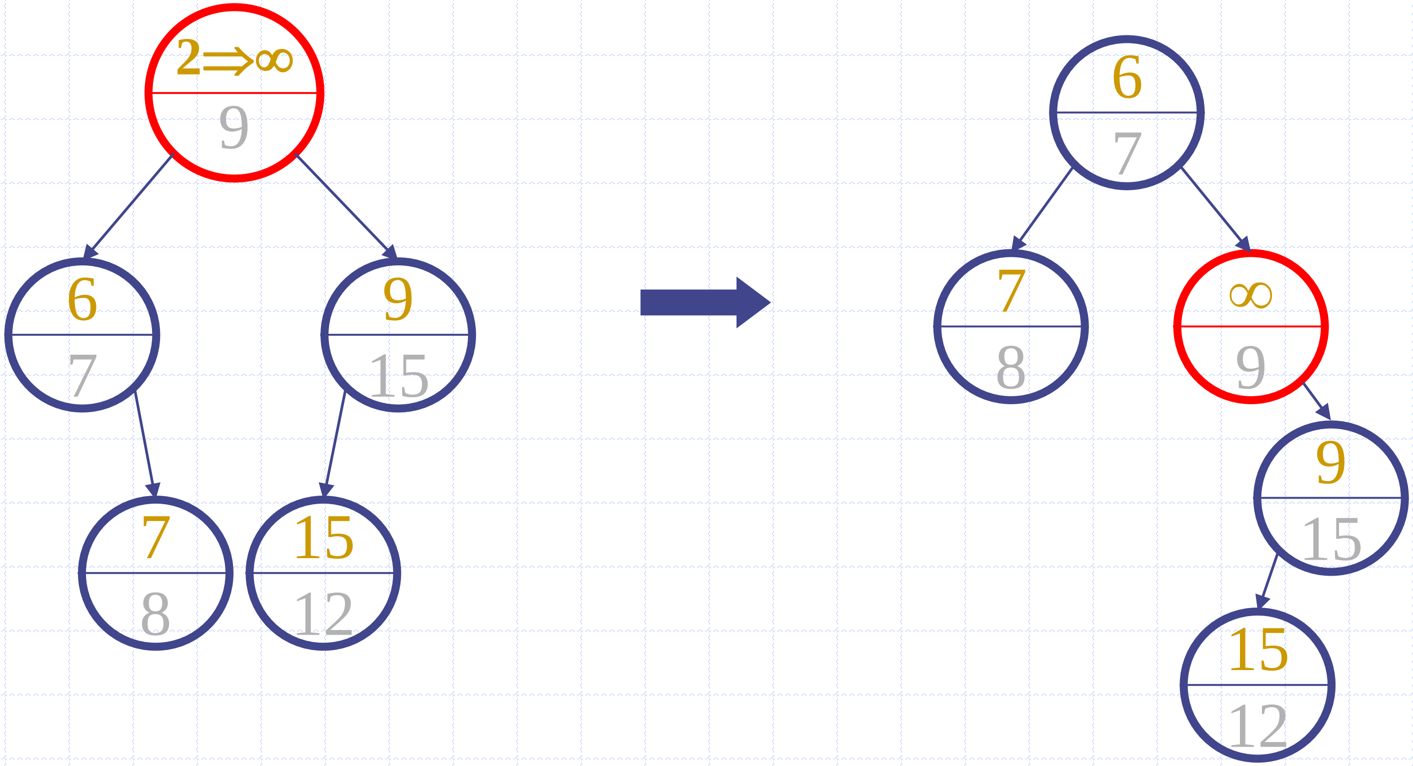


After Rotating K with L

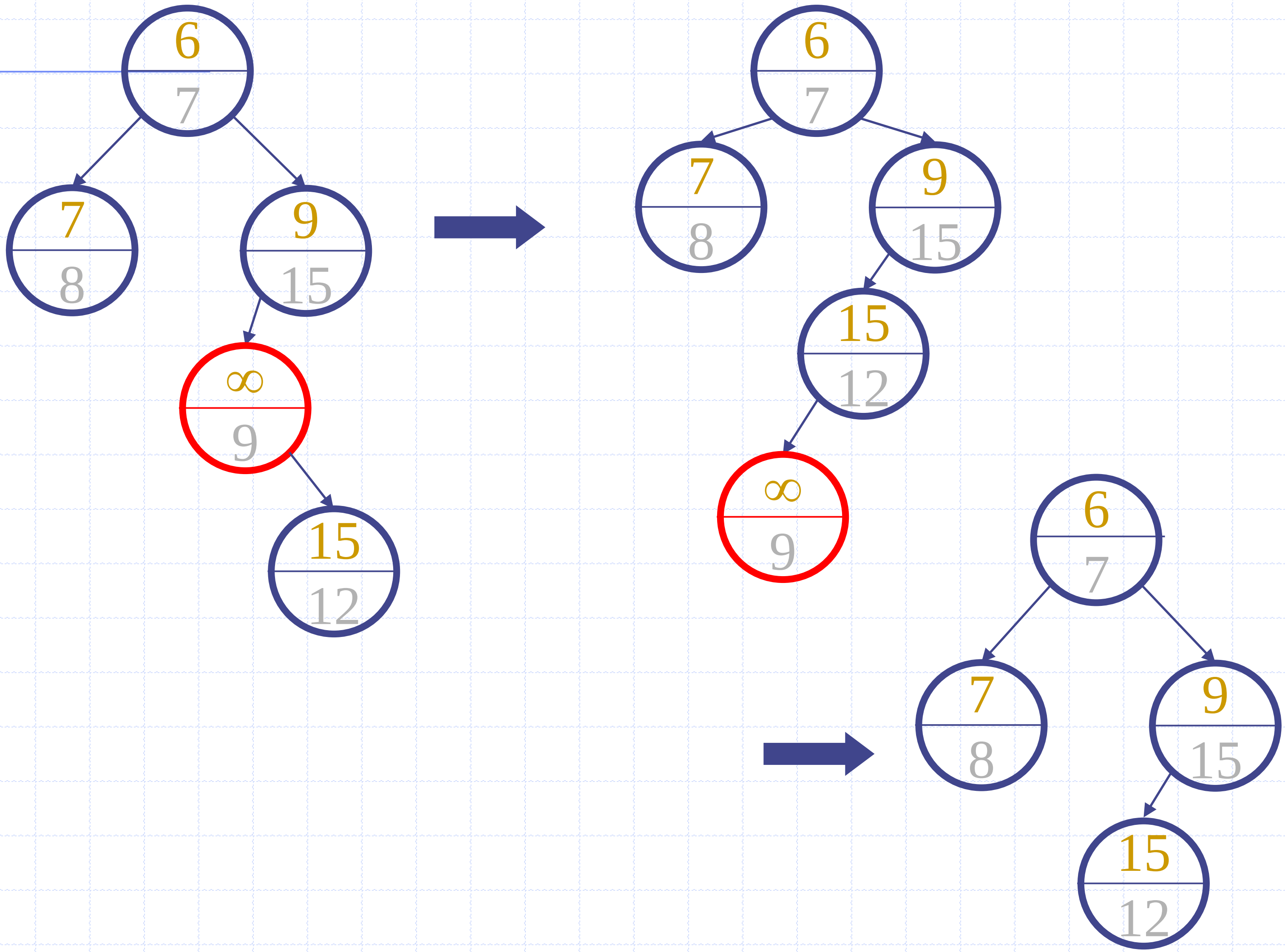
Step 4 - K is a leaf delete K

Treap Delete Strategy

Delete (9)



Treap Delete Strategy



Thank you

??