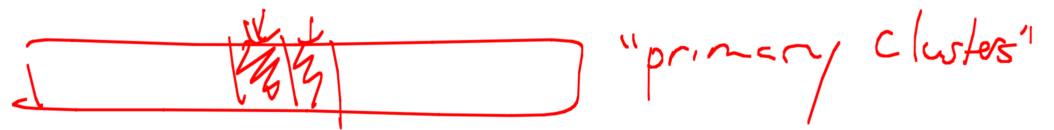
CS180 - Treaps

Note Title 5/4/2011

Announcements

- Program due tomorrow
- HW due next Monday
  - Review next Friday at 10:30 (check webpage / Thursday in case of a room change)
  - Final Monday at noon
  - No conflict (so far)



Final comments on Hashing (recap)

1. convert key to 32-bit int, k
2. Compute h(k), a value between 0 & N-1

   Goals:
   - fast - O(1)
   - minimize collisions
3. Deal with collisions. How?
   - auxiliary data structure (list, tree, etc.)
   - linear probing
   - quadratic probing

Issues with Quadratic Probing:

- Can still cause "secondary" clustering
- N really must be prime for this to work
- Even with N prime, starts to fail when array gets half full

(Runtimes are essentially the same)

Even worse: this can just fail: the array has an empty spot, but this function can't get to it

Secondary Hashing

- Try $A[h(k)]$
- If full, try $A[h(k) + f(j) \bmod N]$ for $j = 1, 2, 3, \ldots$

quadratic, we had $f(j) = j^2$

where
$f(j) = j \cdot h'(k)$ with h' a different hash function

another hash fcn

Load Factors

Separate chaining actually works as well as most others in practice, although it does use more space.

Most of these methods only work well if $\frac{n}{N} < .5$.

(Even chaining starts to fail if $\frac{n}{N} > .9$)

Rehashing

Because we need $\frac{n}{N} < .5$, most hash code checks if the array has become more than half full.

If so, it stops & recomputes everything for a larger N, usually at least twice as big.

(Still not too bad in an amortized sense – think vectors.)

In practice

Hashing is the fastest thing!
Given a good load factor, this uses a small amount of space and runs in $O(1)$ time.

This is a good example when the guaranteed run times are very different from what we see in practice.

Tree + heap

Treaps: a new binary tree data structure

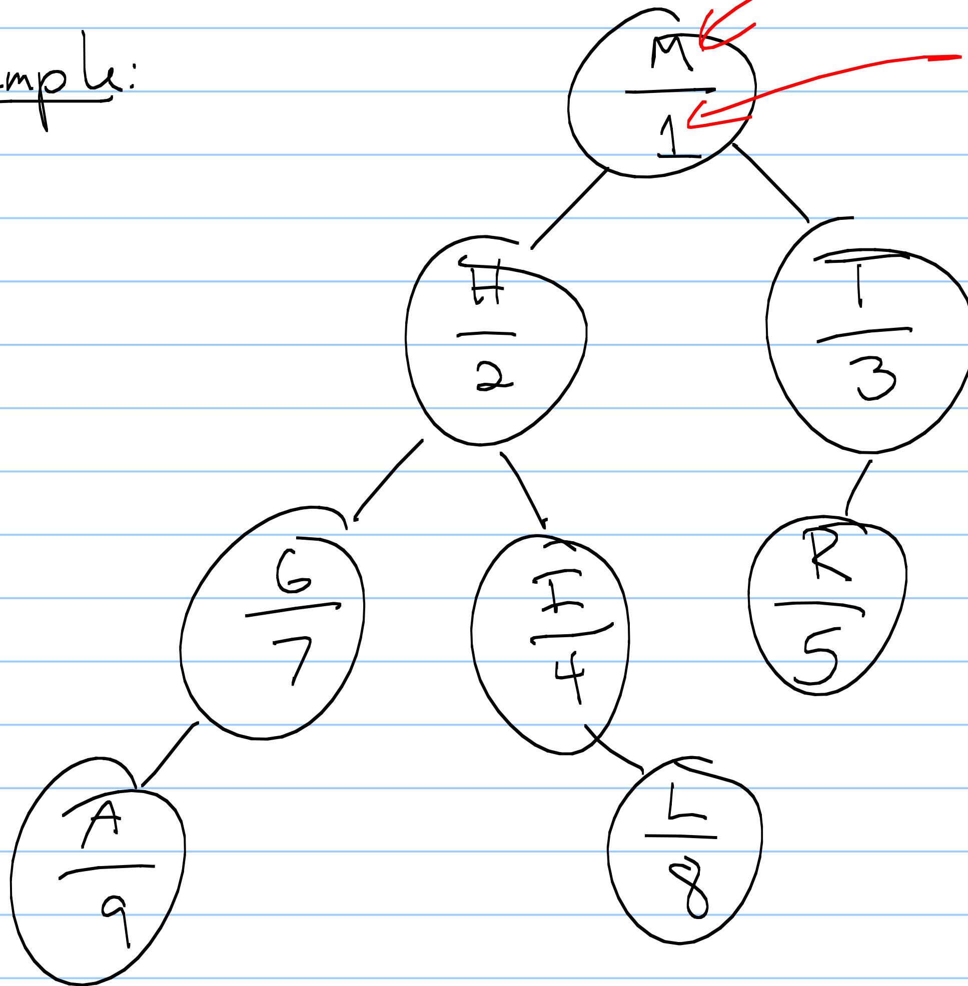
- Nodes will contain both values and priorities
- A treap is a BST over the values and a heap over the priorities.

← min heaps

Use letters + int

↑ values ↑ priorities

Example:



letters form a BST

numbers give a min heap

(letters are in alpha. order)

Insert

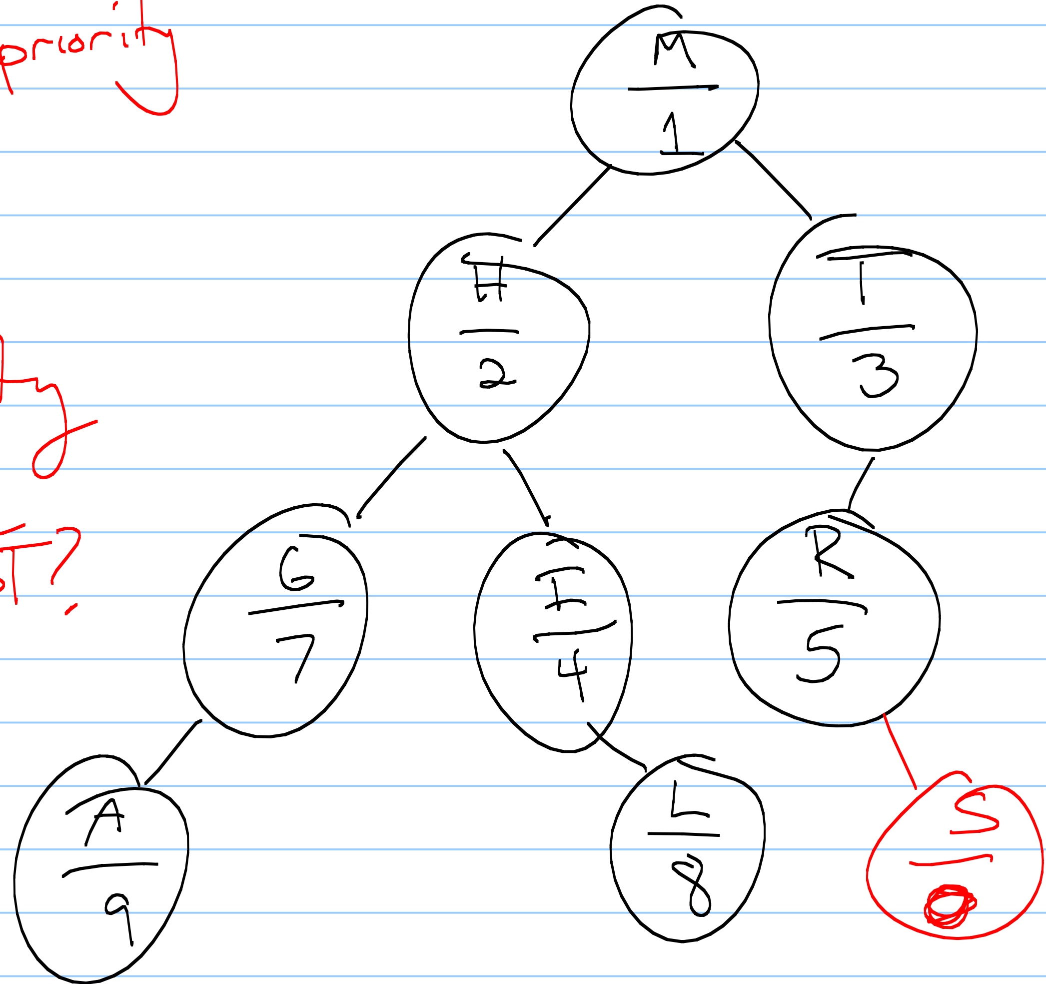
insert: (S, 0)

data ↙ ↘ priority

forget priority

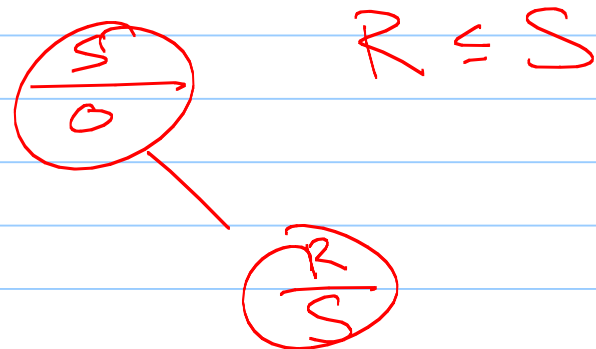
where should S go in BST?

Am I a min heap?



In heap, we "bubbled up".
Will that work here?

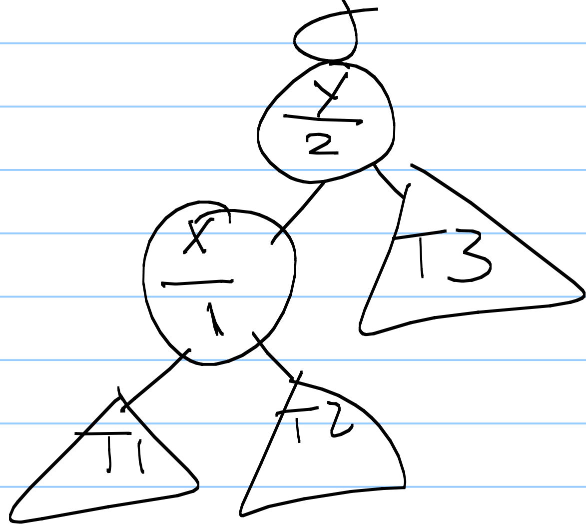
No. If I swap of R & S nodes, no longer a BST over values.



$R \leq S$

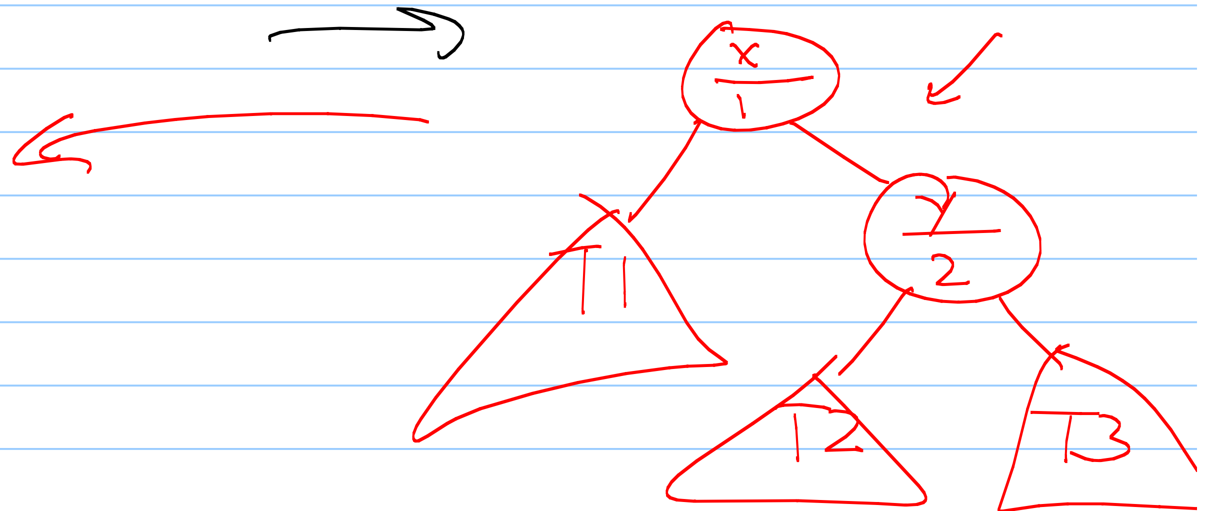
Rotations

x & y are in correct BST order, with $x \le y$, but priorities are wrong

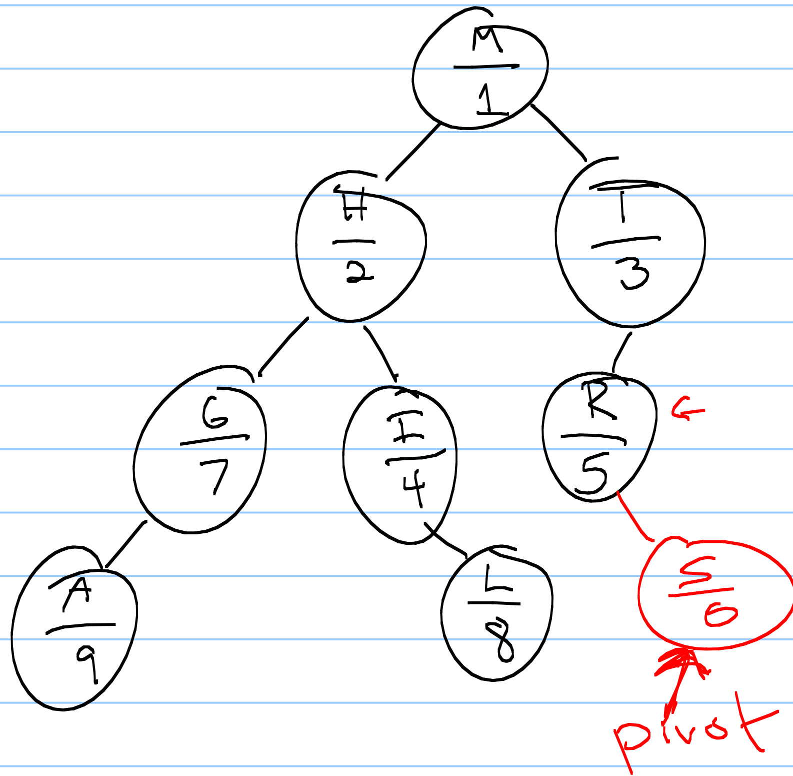


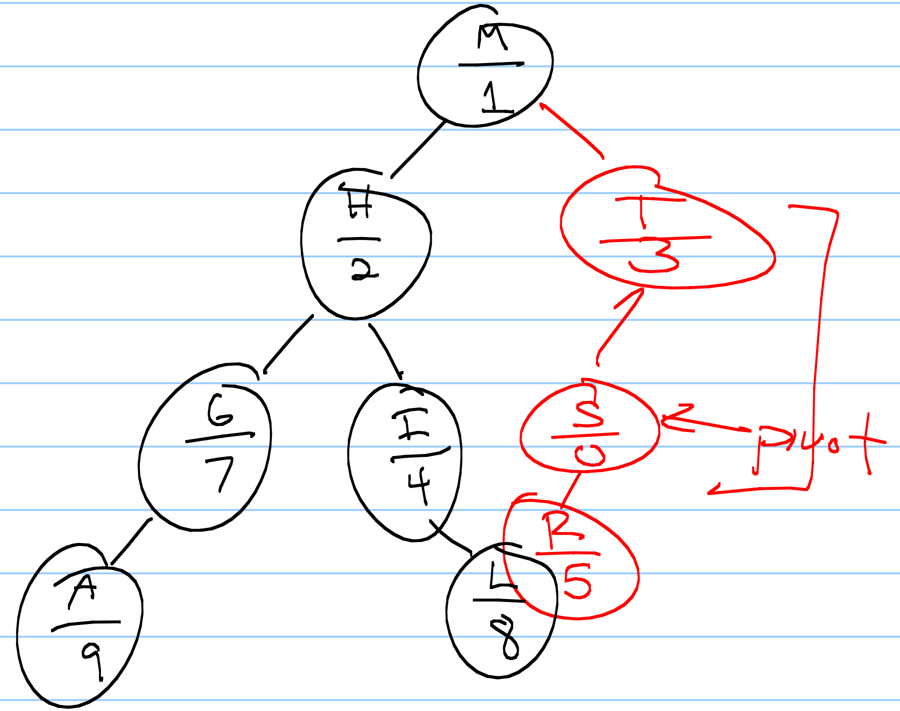
Ex:

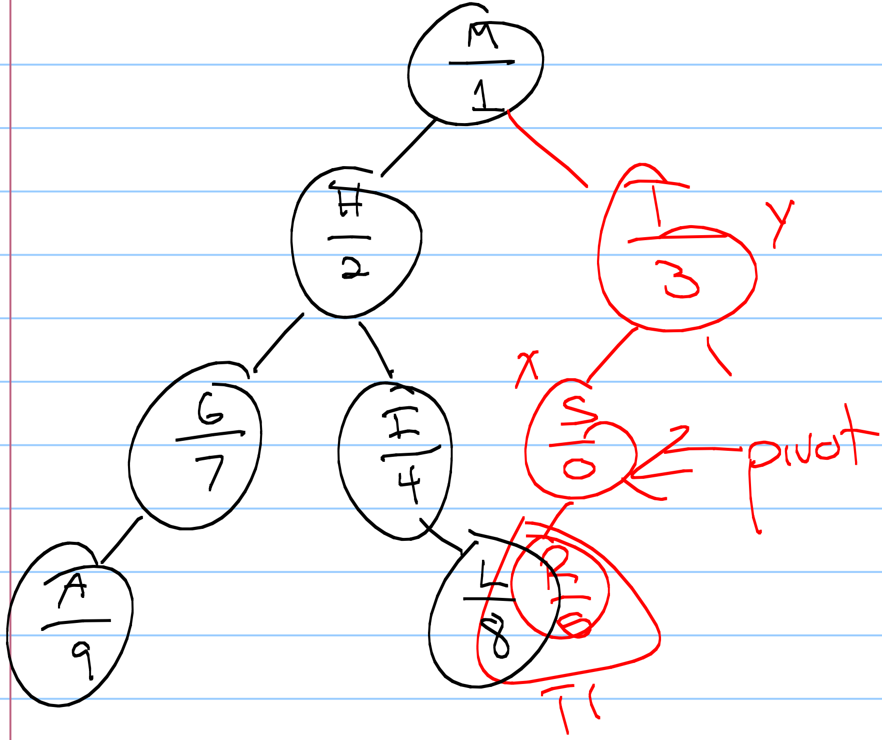
Nice: this is just our pivot.



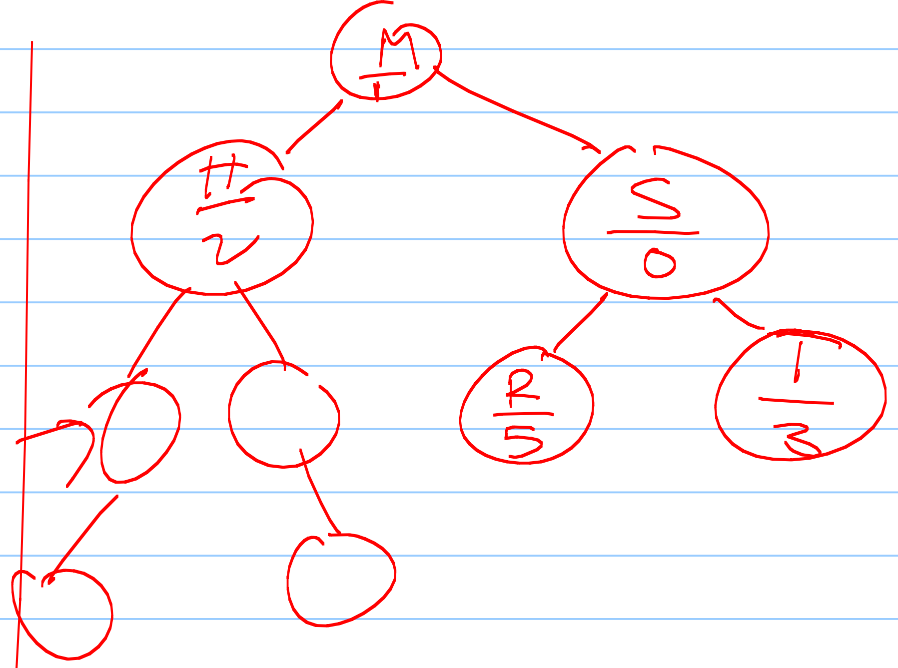
S_0 : insert (S, 0)





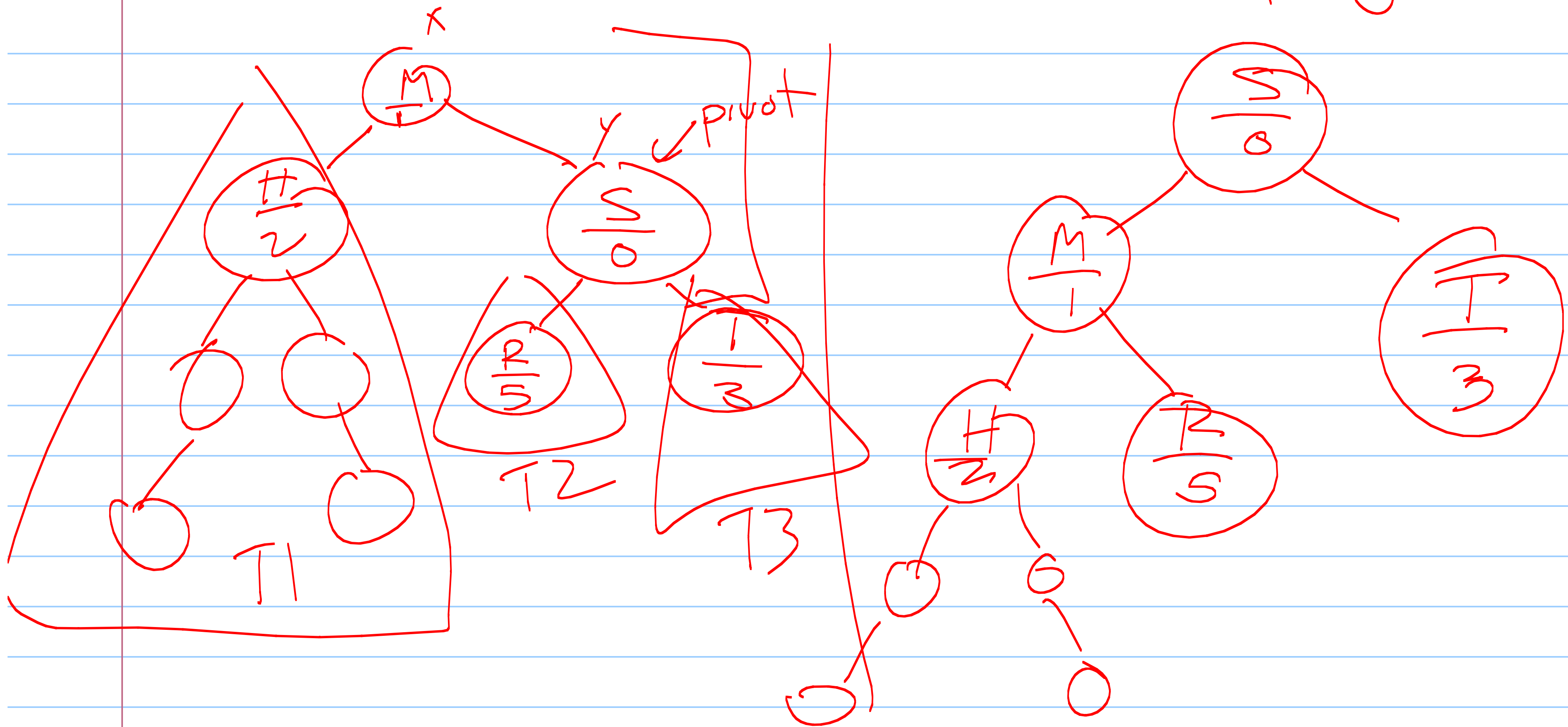


M
1
H
2
T
3
y
G
7
I
4
x
S
0
pivot
A
9
L
8
R
5



M
1
H
2
S
0
R
5
T
3

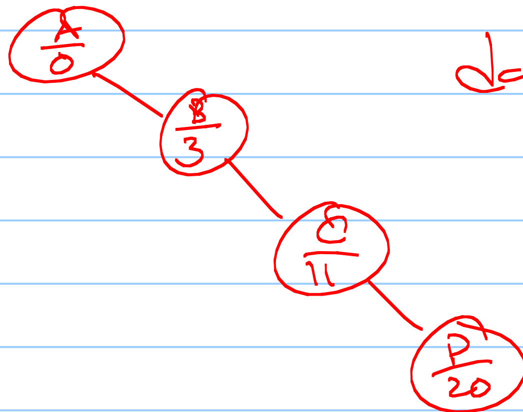
Treap again!



Downside: What can height be?

Can we force them to be balanced?

No: (A, 0) (B, 3) (C, 11) (D, 20)

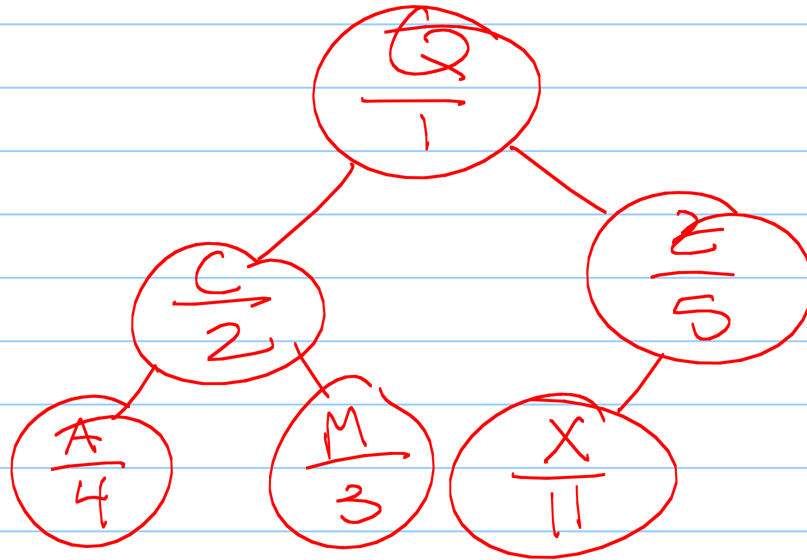


Each set of data has a **unique** treap.

Draw treap with (A, 4), (C, 2), (X, 11), (M, 3), (Q, 1), (Z, 5)

Who is root? root

(save tons of time)



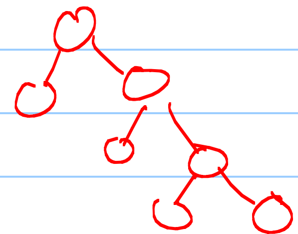
Randomized treaps:

Alternative to AVL trees.

Each element will get a random priority.

Expected height of the treap will be $O(\log n)$.

Worst case is still $O(n)$.



Code: How do we implement?

Inherit from Binary Search Tree.

- → priorities are in _aux.
- → insert (use pivot to fix)
- → delete

(look for extra lecture notes)