LECTURE 15 : Treaps

"God does not play dice
with the universe"
Albert Einstein 1926

```
> data Treap a = Empty
>              | Node (Treap a) a Int (Treap a)
```
(Int: priority)

```
> delete :: Ord a => a -> Treap a -> Treap a
> delete x Empty = Empty
> delete x (Node a y q b)
>     | x == y = merge a b
>     | x < y  = Node (delete x a) y q b
>     | otherwise = Node a y q (delete x b)

> merge :: Treap a -> Treap a -> Treap a
> merge Empty r = r
> merge l Empty = l
> merge l@(Node a x p b) r@(Node c y q d)
>     | p < q = Node a x p (merge b r)
>     | otherwise = Node (merge l c) y q d
```

RANDOMIZED TREAPS.

Last time:

priority

```
> pinsert :: Ord a => a -> Int -> Treap a -> Treap a
```

seed

```
> data RTreap a = RTreap StdGen (Treap a)

> rinsert :: Ord a => a -> RTreap a -> RTreap a
> rinsert x (RTreap seed t)
>   = RTreap seed' (pinsert x p t)
>   where
>     (p, seed') = random seed
```

This uses `random :: StdGen -> (Int, StdGen)`

Treap a

```
> empty :: RTreap a
> empty = RTreap (mkStdGen 42) empty
```

```
mkStdGen :: Int -> StdGen
```

```
> fromList :: Ord a => [a] -> RTree a
> fromList xs = foldr insert empty xs

> toList :: RTree a -> [a]
> toList (RTree seed t) = toList t


>rqsort :: Ord a => [a] -> [a]
>rqsort xs = toList (fromList xs)
```

Randomized Binary Search Trees

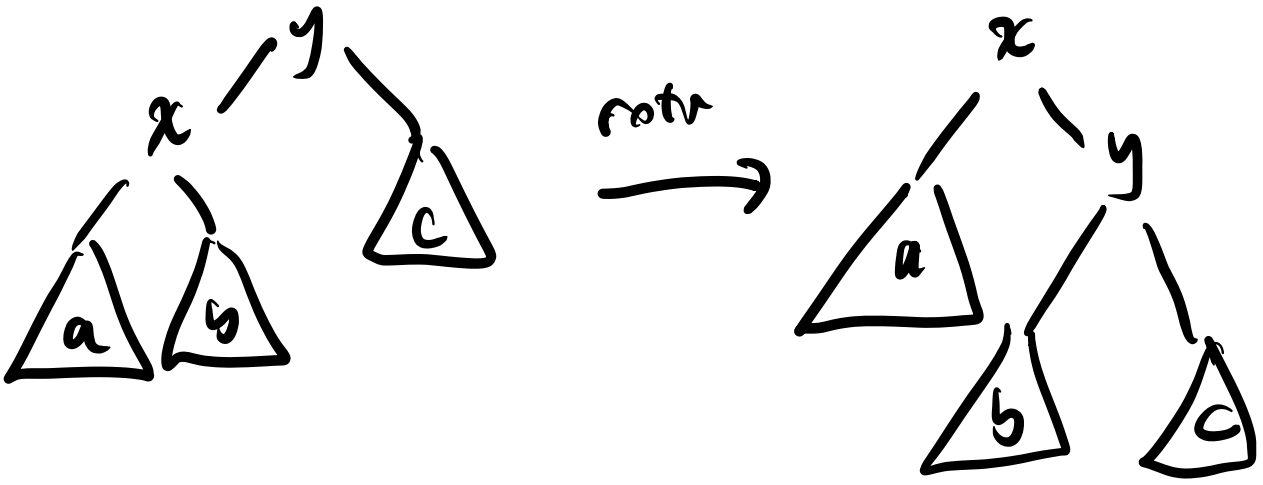
```
> data Tree a = Empty
>             | Node (Tree a) a (Tree a)

          Ord a =>
> insert :: a -> Tree a -> Tree a
>   ...

> insertRoot :: Ord a => a -> Tree a -> Tree a
> insertRoot x Empty = Node Empty x Empty
> insertRoot x t@(Node l y r)
>   | x == y = t
>
```

```
>   | x < y = rotr (insertRoot x l) y r
>   | x > y = rotl l y (insertRoot x r)
```

```
> rotr :: Tree a → a → Tree a → Tree a
> rotr (Node a x b) y c = Node a x (Node b y c)
```



```
> data RTree a = RTree StdGen Int (Tree a)
```

seed ↑ ↑ size

```
> empty :: RTree a
> empty = RTree (mkStdGen 42) 0 Empty
```

```
> insert :: a -> RTree a -> RTree a
> insert x (RTree seed n t)
>     | p == 0 = RTree seed' (n+1) (insertRoot x t)
>     | otherwise = RTree seed' (n+1) (insert x t)
>   where
>     (p, seed') = randomR (0, n) seed
```

---

randomR :: (Int, Int) -> StdGen -> (Int, StdGen)

m ... n

```
> toList :: Tree a -> [a]
> toList Empty          = []
> toList (Node l x r)   = toList l ++ [x] ++ toList r
```

(x:)

DList