Lec 8

Merge sorting tree

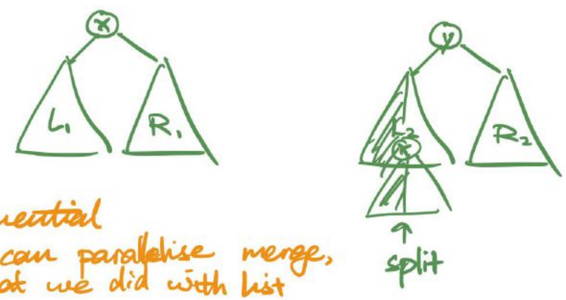
```
(* msort : tree -> tree
   REQ  true
   ENS  sorted tree with same elems
*)
fun msort (Empty : tree) : tree = Empty
  | msort (Node (L, x, r)) =  insert ( x, merge ( msort L , msort R))
```

We don't have to do split when doing it on tree

```
(* merge : tree * tree -> tree
   REQ  both input sorted
   ENS  merge them and is sorted
*)
fun   merge ( Empty , T2 ) = T2
  |   merge ( Node (L1 , x , R2) , T2 ) =
        let
          val (L2,R2) = SplitAt (x, T2)
        in
          Node (Merge (L1, L2), x, Merge (r1 , r2))
        end
```

though this is sequential

then we can parallelise merge, unlike what we did with list



```
(* SplitAt : int * tree -> tree * tree
   REQ   tree sorted
   ENS   => T1, T2  with  ∀n∈T1, n ≤ x,  ∀n∈T2, n ≥ x,  same set of nodes.
*)
fun  splitAt (_, Empty) = (Empty, Empty)
   | SplitAt (x, Node (L, y, R)) =
       (case Int.compare (x, y) of
           LESS => let
                     val (L1, L2) = SplitAt (x, L)
                   in
                     (L1, Node (L2, y, R))
                   end
         | _    => let
                     val (R1, R2) = SplitAt (x, R)
                   in
                     (Node (L, y, R1), R2)
                   end
       )
```

SplitAt Analysis — in terms of depth d

$S_{splitAt}(0) = c_0$

$$S_{splitAt}(d) = \begin{cases} c_1 + S_{splitAt}(d_1) & \text{if LESS} \\ c_2 + S_{splitAt}(d_2) & \text{else} \end{cases}$$

$[\, d = \max(d_1, d_2) + 1 \Rightarrow d_1 \le d-1,\ d_2 \le d-1$

$\le c_3 + S_{splitAt}(d-1)$

$\in O(d)$

Merge analysis — d_1, d_2 are depth of the two inputs

$$S_{merge}(0, d_2) = c_0$$

$$S_{merge}(d_1, d_2) = c_1 + S_{splitAt}(d_2) + \max(S_{merge}(d_{1l}, d_{2l}), S_{merge}(d_{1r}, d_{2r}))$$

$$\left[\begin{array}{l} d_{1l}, d_{1r} \le d_1 - 1 \\ d_{2l}, d_{2r} \le d_2 \end{array}\right.$$

Lemma: splitAt doesn't return deeper trees than input

$$\le c_1 + c_2 d_2 + S_{merge}(d_1 - 1, d_2)$$

$$= c_1 + c_2 d_2 + c_1 + c_2 d_2 + S_{merge}(d_1 - 2, d_2)$$

$$\ldots = d_1 c_1 + d_1 (c_2 d_2) + c_0$$

$$\in O(d_1 d_2)$$

"so our code doesn't work, end of lecture" (?)

Msort analysis — d is depth of input tree

uh oh ... we don't know these

$$S_{msort}(0) = c_0$$

$$S_{msort}(n) = c_1 + \max(S_{msort}(d_1), S_{msort}(d_2)) + S_{merge}(d_1', d_2') + S_{insert}(d_3)$$

$$\left[\begin{array}{l} d_1, d_2 \le d - 1 \\ \text{uh oh} \end{array}\right.$$

Umm let's do this and retry —

```
fun msort(Empty : tree) : tree = Empty
  | msort(Node(L, x, r)) = rebalance(insert(x, merge(msort L, msort R)))
```

$$S_{msort}(0) = c_0$$

$$S_{msort}(n) = c_1 + \max(S_{msort}(d_1), S_{msort}(d_2)) + S_{merge}(d_1', d_2') + S_{insert}(d_3) + S_{rebalance}(d_4)$$

$$\left[\begin{array}{l} d_1, d_2 \le d - 1 \\ d_1', d_2', d_3, d_4 \in O(d) \end{array}\right.$$

$$\le c_1 + S_{msort}(d_1 - 1) + c_2 d^2 + c_3 d + \underline{c_4 d^2}$$

will show

$$\le c_5 d^2 + S_{msort}(d, -1)$$

$$\le c_6 d^3$$

$$\in O(d^3)$$

If input balanced i.e. $n = 2^d$ then $O(\log^3 n)$

there's a better analysis that gets it down to $O(\log^2 n)$