Lecture 10

Sorting Trees

Lists

mergesort

work $O(n \log n)$

span $O(n)$

$\frac{W}{S} = \frac{n \log n}{n} = \log n$ Processors

Trees

mergesort

work $O(n \log n)$

span $<<< n$ $(\log n)^3$

Procs $\frac{n \log n}{(\log n)^3} \approx \frac{n}{(\log n)^2}$

time on p processors is

$$\gtrsim \max\left(\frac{W}{p}, S\right)$$

break even $\frac{W}{p} = S$

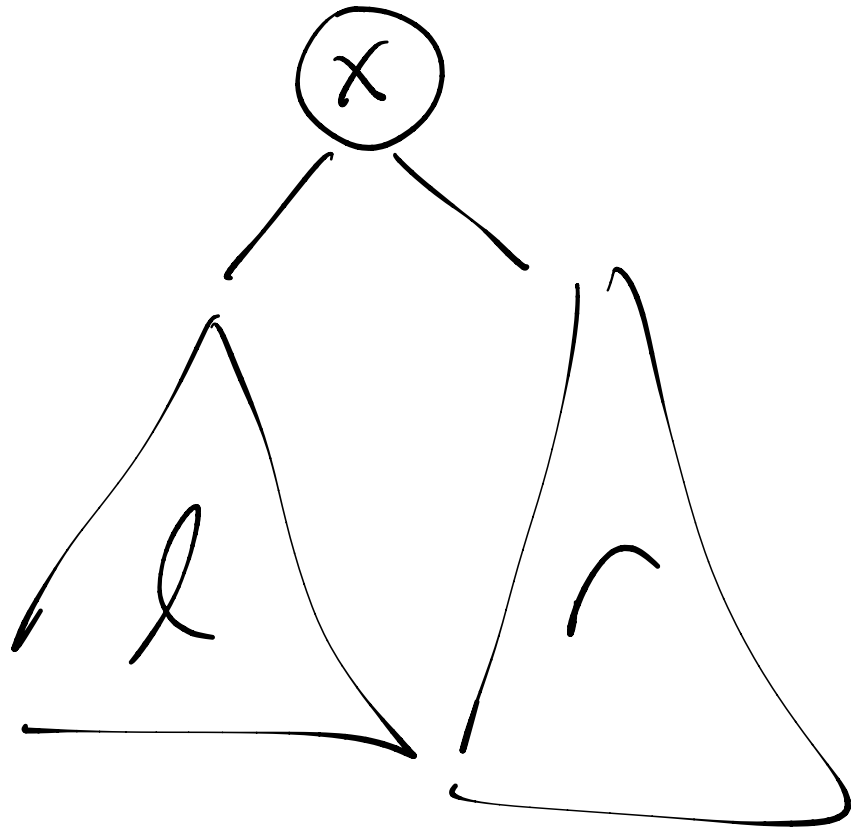
$$\boxed{p = \frac{W}{S}}$$

roughly how many procs can we use?

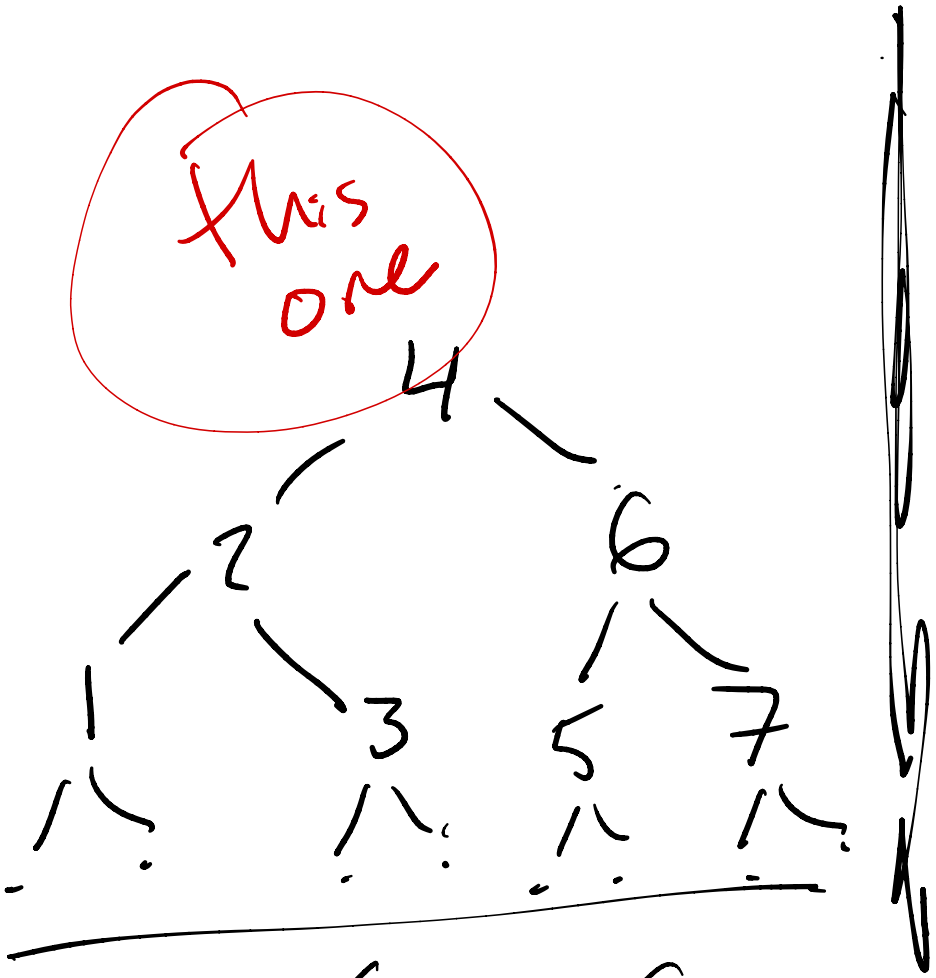
A tree is either

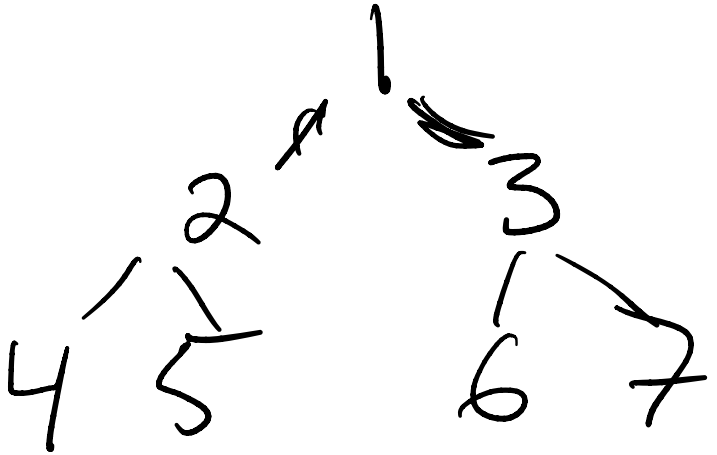
Empty, or

Node(l, x, r) where l : tree
x : int
r : tree



A tree is sorted when

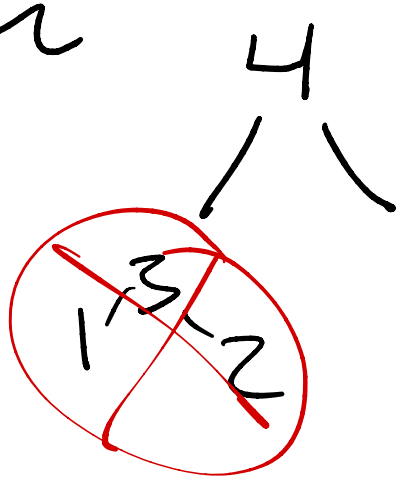




?

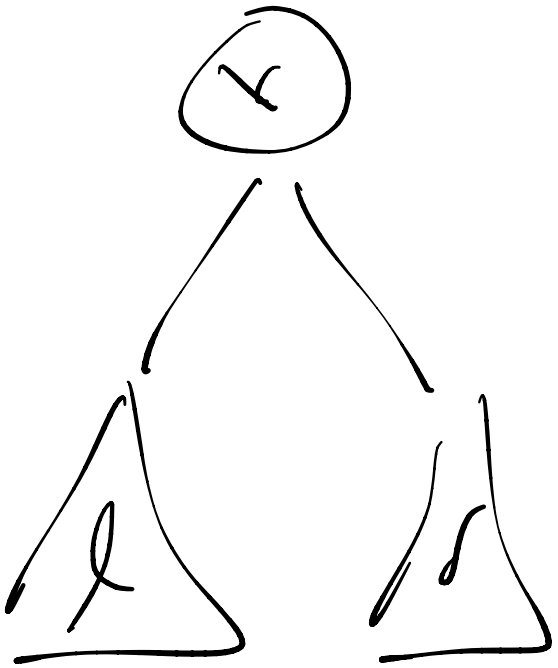
↳ [1, 2, 3, 4, 5, 6, 7]

A tree is sorted when either it's Empty



or it's

Node(l, x, r), and



everything in $l \leq x$

everything in $r \geq x$

l r are both sorted

```
fun ms(t:tree):tree =
    case t of
      Empty => Empty
    | Node(l, x, r) =>
rebalance(merge(merge(ms(l), ms(r)),
                Node(Empty, x, Empty)))
```

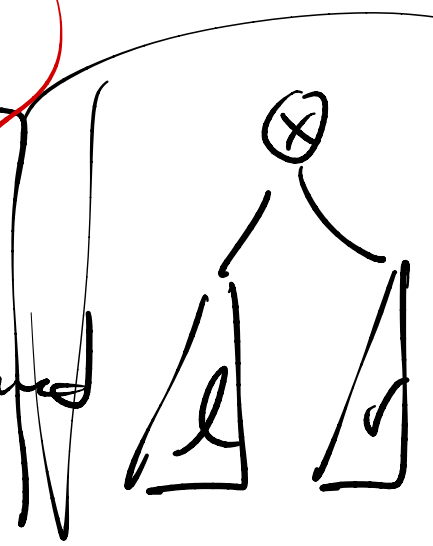
1. divide
2. recur
3. combine

$W_{ms}(n) = \text{"}W_{merge}\text{"} + 2W_{ms}\left(\frac{n}{2}\right)$

size of t

if balanced

$S_{ms}(n) = \text{"}S_{pan\ of\ merge}\text{"} + 1S_{ms}\left(\frac{n}{2}\right)$



```
fun ms(t:tree):tree =
  case t of
    Empty => Empty
  | Node(l, x, r) =>
      merge(merge(ms(l), ms(r)),
            Node(Empty, x, Empty))
```

(A) (B)

If a tree is balanced, then depth is $\log_2$ size

$$S_{ms}(n) = S_{merge}(\log_2 n, \log_2 n) \text{ is } (\log_2 n)^2 \quad \text{(A)}$$

$$+ S_{merge}(2\log_2 n, 1) \text{ is } (\log_2 n)$$

$$+ S_{ms}\left(\frac{n}{2}\right)$$

$$= (\log_2 n)^2 + S_{ms}\left(\frac{n}{2}\right)$$

$$S_{ms}(n) = (\log n)^2 + S_{ms}\left(\frac{n}{2}\right)$$

$$\leq \underbrace{(\log n)^2 + (\log n)^2 + (\log n)^2 + \cdots}_{(\log n)}$$

$$= (\log n)^3$$

$$W_{ms}(n) = \text{"}W_{merge}\text{"} + 2W_{ms}\left(\frac{n}{2}\right) + 1$$

size of t

If balanced

need this $O(n)$

$W_{ms}(n)$ to be $O(n \log n)$

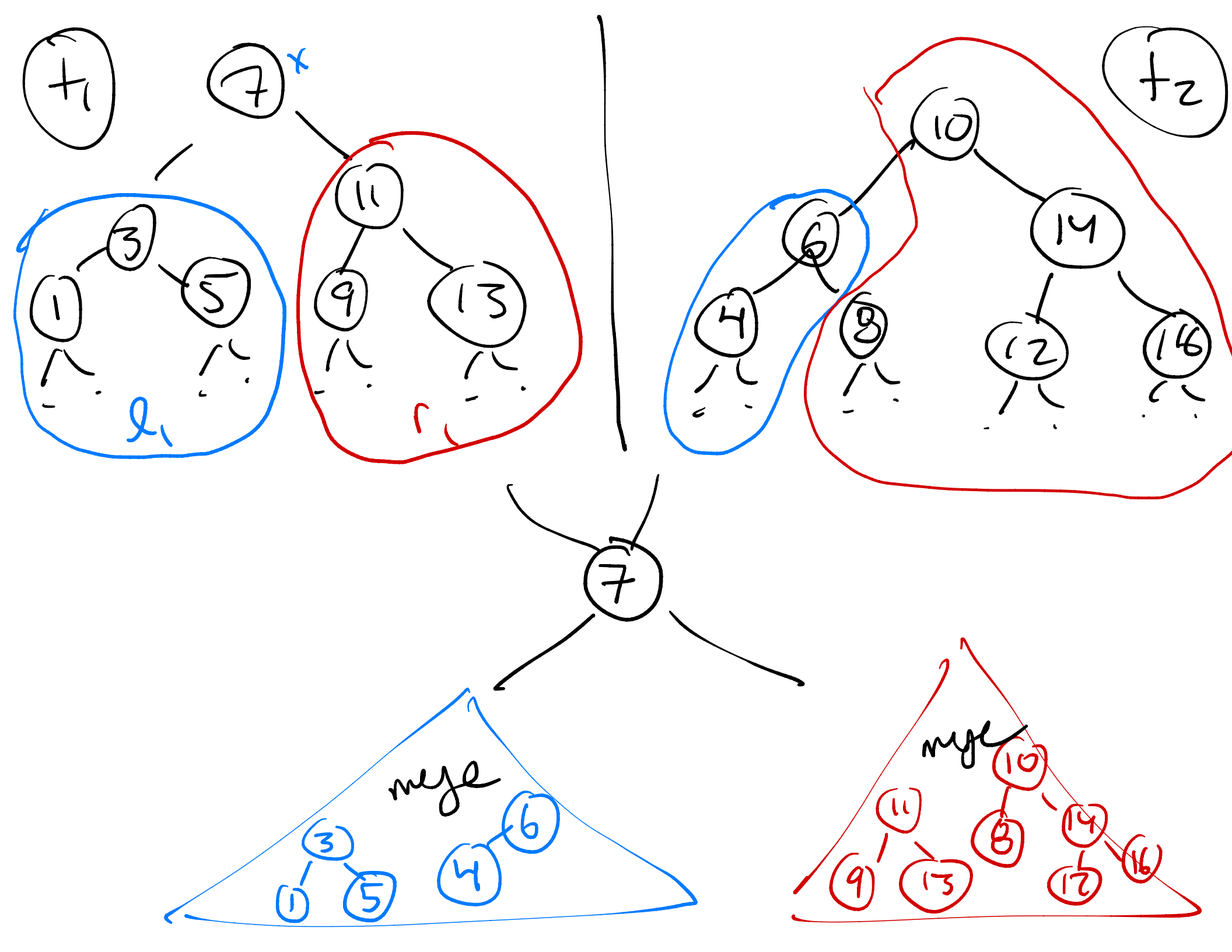
$$S_{ms}(n) = \text{"}\text{Span of merge}\text{"} + 1 S_{ms}\left(\frac{n}{2}\right)$$

want $S_{ms}(n)$ is $O((\log n)^3)$

need is span of merge is $O((\log n)^2)$

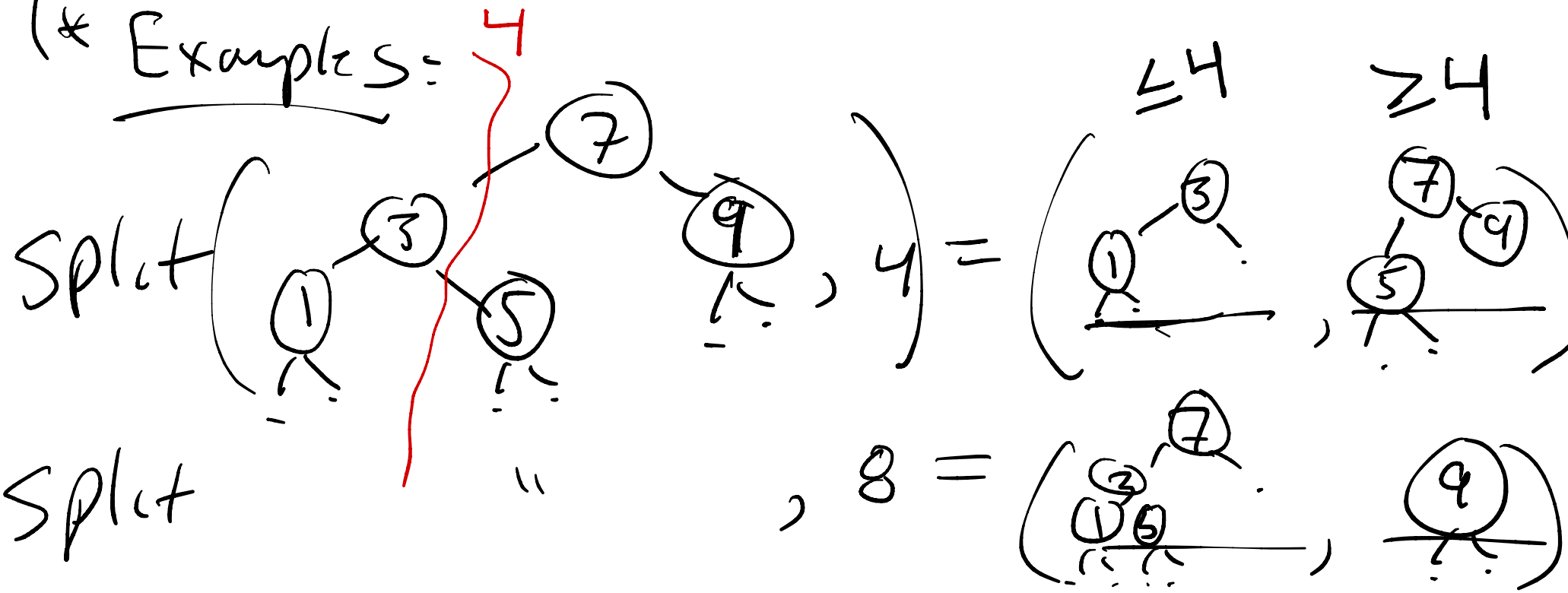
```
(* Purpose: given 2 sorted trees,
   compute a sorted tree w/ all of
     their numbers in it                *)

fun merge (t1:tree, t2:tree) :tree =
  case t1 of
     Empty => t2
   | Node(l1, x1, r1) =>
       let val (l2, r2) = splitAt(t2, x1)
       in
       Node(merge(l1, l2), x1, merge(r1, r2))
       end
```

goal: linear work
sublinear span


t1
7
x
3
1
5
l1
11
9
13
r1
t2
10
6
14
4
8
12
16
7
myε
3
1
5
6
4
myε
10
11
8
14
9
13
12
16

(* Given a sorted tree t and int x,
splitAt(t) = (l, r) where
$l \leq x$ and $r \geq x$
and every in t goes into exactly one of l or r *)

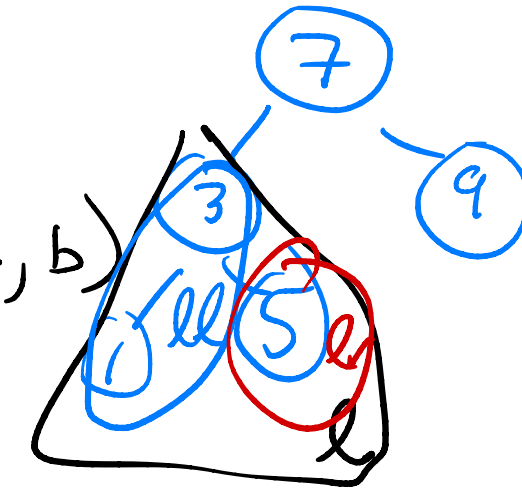
(* Examples:



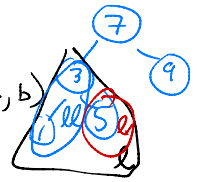
depth is at most depth of t

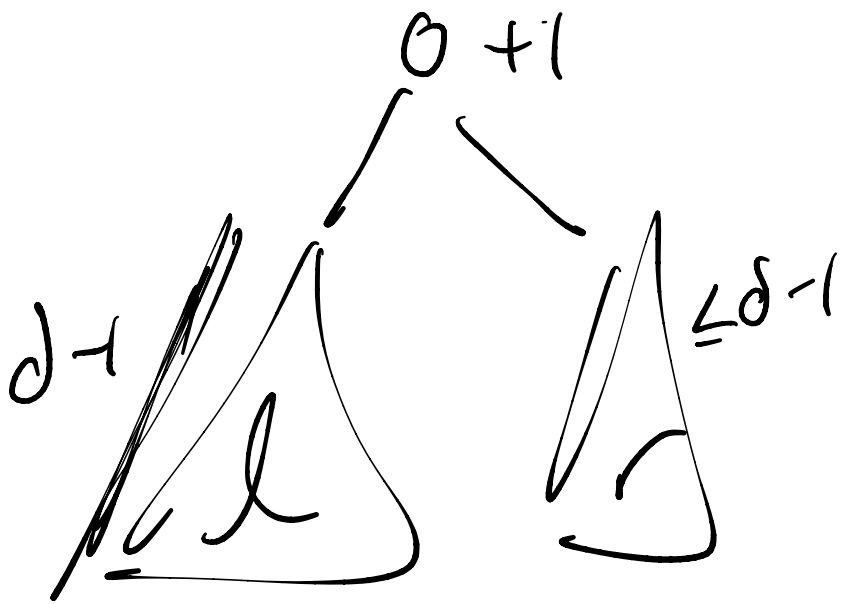
```
fun splitAt(t:tree, b:int): tree * tree =
  case t of
    Empty => (Empty, Empty)
  | Node(l,x,r) =>
      case b < x of
        true => let val (ll,lr) = splitAt(l,b)
                in (ll, Node(lr,x,r)) end
      | false => let val (rl,rr) = splitAt(r,b)
                 in (Node(l,x,rl), rr) end
```

≤b ≥b



```
fun splitAt(t:tree, b:int): tree * tree =
  case t of
    Empty => (Empty, Empty)
  | Node(l,x,r) =>
      case b < x of
        true => let val = (ll, lr) = splitAt(l, b)
                in (ll, Node(lr, x, r))
                end
      | false => let val (rl, rr) = splitAt(r, b) in
                 end (Node(l, x, rl), rr)
```





$$S_{splitAt}(d) = 1 + S_{splitAt}(d-1)$$

depth of t

$O(d)$

```
fun merge(t1:tree, t2:tree):tree =
  case t1 of
    Empty => t2
  | Node(l1, x1, r1) =>
      let val (l2, r2) = splitAt(t2, x1)
      in
      Node(merge(l1, l2), x1, merge(r1, r2))
      end
```

goal: linear work, sublinear span

← depth of output is ≤ depth t_1 + depth t_2

$$S_{merge}(d_1, d_2) \leq S_{splitAt}(d_2) + 1\, S_{merge}(d_1 - 1, d_2)$$

d_1: depth of t_1; d_2: depth t_2 ≤

upper bound; upper bound

$$\approx d_2 + S_{merge}(d_1 - 1, d_2)$$

is $O(d_1 \cdot d_2)$