Lecture 9

1) Span of mergesort

2) Trees

```
fun mergesort (l : intlist) : intlist =
  case l of
      [] => []
    | [x] => [x]
    | _ => let val (l1, l2) = split(l)
           in merge(mergesort l1,
                    mergesort l2)
           end
```

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

"$\approx$" $n + 2W_{ms}\left(\frac{n}{2}\right)$

$$S_{ms}(n) = S_{split}(n) + \max\left(S_{ms}\left(\frac{n}{2}\right), S_{ms}\left(\frac{n}{2}\right)\right) + S_{ms}(n)$$

```
fun split(l) = case l of
    []  => ([], [])
  | [x] => ([x], [])
  | x::y::xs => let val (l1, l2) = split(xs)
                in (x::l1, y::l2) end
```

$x :: y :: xs$ — n

xs — $n-2$

$$W_{split}(n) = 1 + W_{split}(n-2)$$

Closed form $\approx \frac{n}{2}$

$O(n)$

$$S_{split}(n) = 1 + S_{split}(n-2)$$

$O(n)$

not parallelizable algorithm

fun merge(l_1, l_2) = - -

s

$x :: xs, y :: ys \Rightarrow$
case $x \leq y$ of
true $\Rightarrow x ::$ merge($xs, y :: ys$)

$s-1$

| false $\Rightarrow y ::$ merge($x :: xs, ys$)

$s-1$

$W_{merge}(s) = 1 + W_{merge}(s-1)$

s

$|l_1| + |l_2|$ $O(s)$

$S_{merge}(s) = 1 + S_{merge}(s-1)$

not parallel

$O(s)$

$T(n) = 1 + T(n-1)$ $O(n)$

$T(n) = 1 + T(n-2)$ $O(n)$

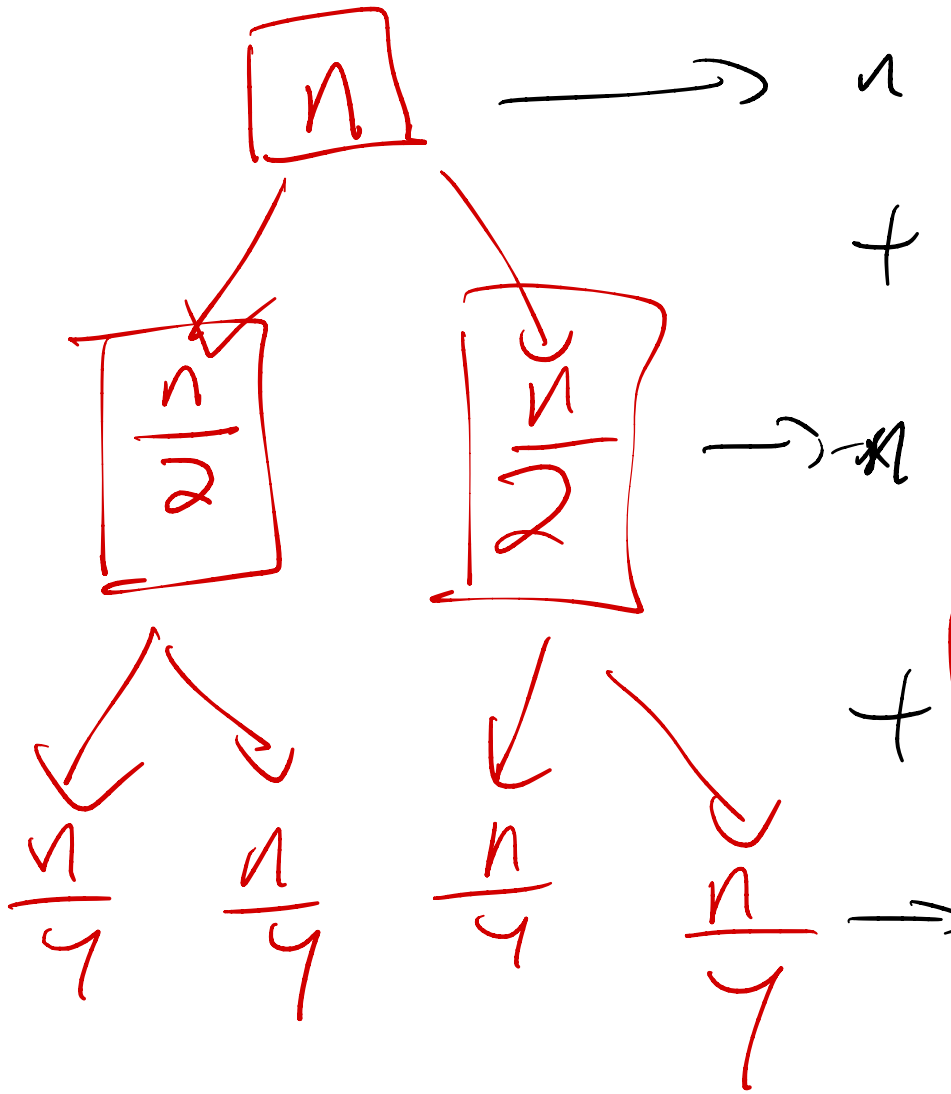
$T(n) = n + T(n-1)$ $O(n^2)$

$T(n) = n + 2T(\frac{n}{2})$ $O(n \log_2 n)$

$T(n) = n + T(\frac{n}{2})$ $O(n)$

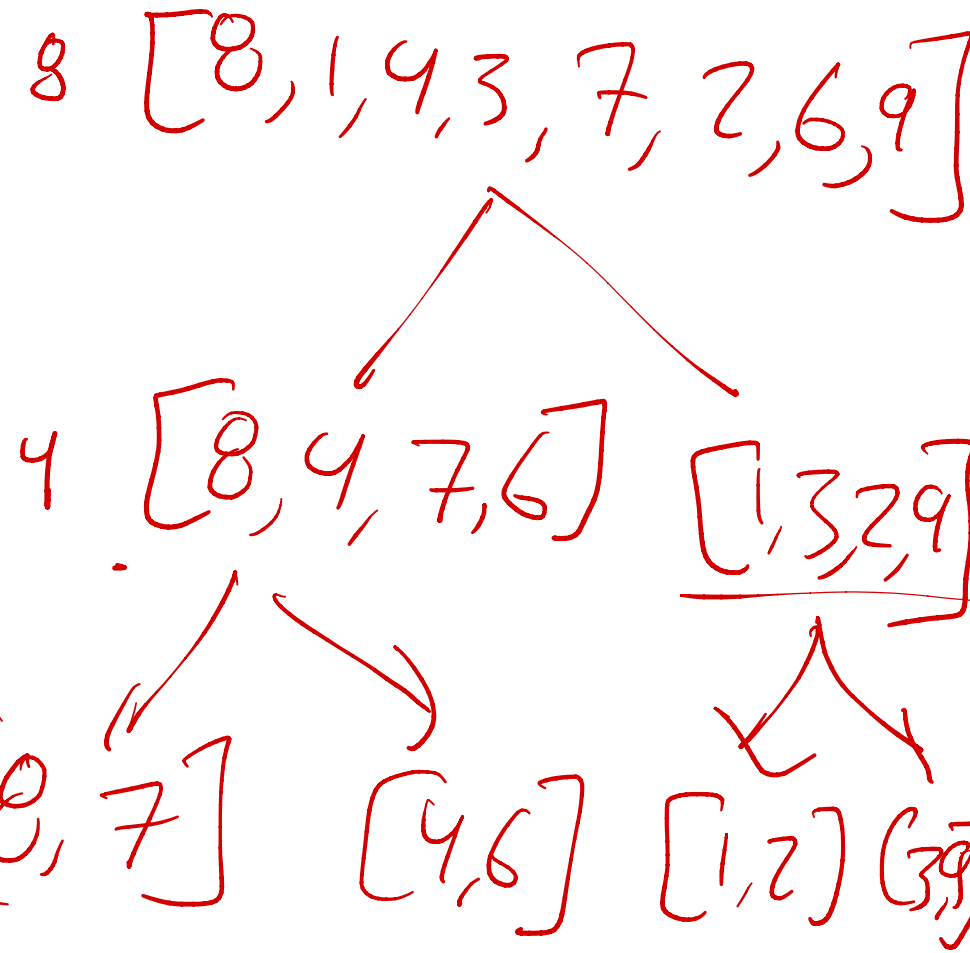
resources

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



$O(n \log_2 n)$

$\theta\frac{n}{\theta} = n$



n at each level

X

levels

$= \log_2 n$

Span: running time for "enough" processors

Idea: if 2 things can be done in parallel, take the maximum of their running time (span), not the sum ← sequential

max(x,y) means bigger of x or y

max(7,3) = 7

max(3,7) = 7

$S_{ms}(n) = \underline{S_{split}(n)}$ is $O(n)$
$+ \max(S_{ms}(\frac{n}{2}), S_{ms}(\frac{n}{2}))$
$+ S_{ms}(n)$ is $O(n)$

is $O(n)$

$= n + \max(S_{ms}(\frac{n}{2}), S_{ms}(\frac{n}{2}))$

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

$= n + \frac{n}{2} + \frac{n}{4} + \frac{n}{8} + \frac{n}{16} + \frac{n}{32} + \cdots$

$= n(1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \frac{1}{16} + \frac{1}{32} + \cdots)$

$\leq 2n$

n 1 billion

Goal: span is $(\log n)^k$

$n \log n$ 30 billion work for MS

n 1 billion span

time on p processors is $\approx \max\left(\frac{W}{P}, S\right)$

$\approx \max\left(\frac{30 \text{ billion}}{P}, 1 \text{ billion}\right)$

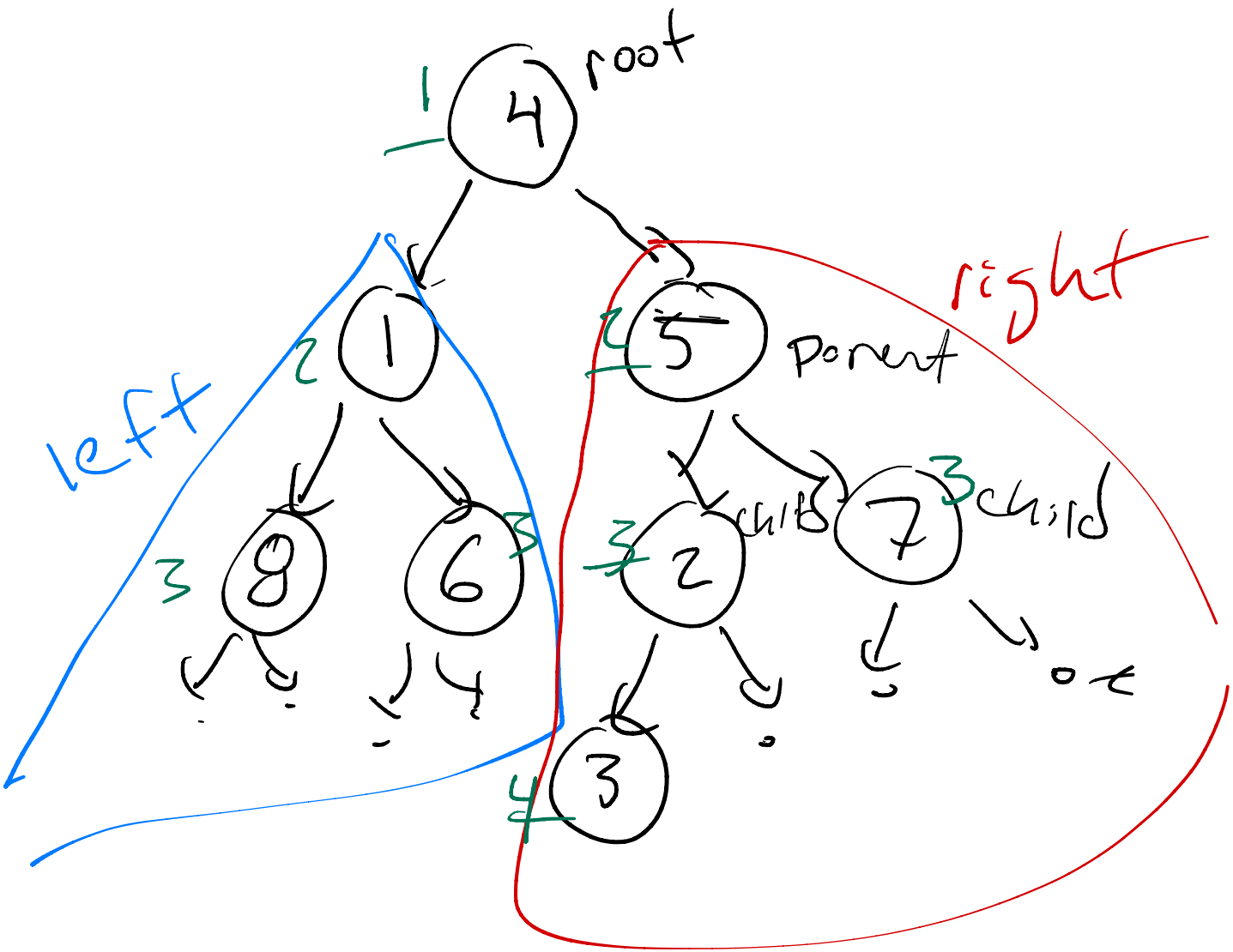
can only use roughly 30 procs

Goal: get the span of helpers down while keeping the same work

Lists ⟶ Trees

Trees

[8, 1, 6, 4, 3, 2, 5, 7]



Size

# of numbers

Depth

length of biggest path from the root to a leaf

A tree is either

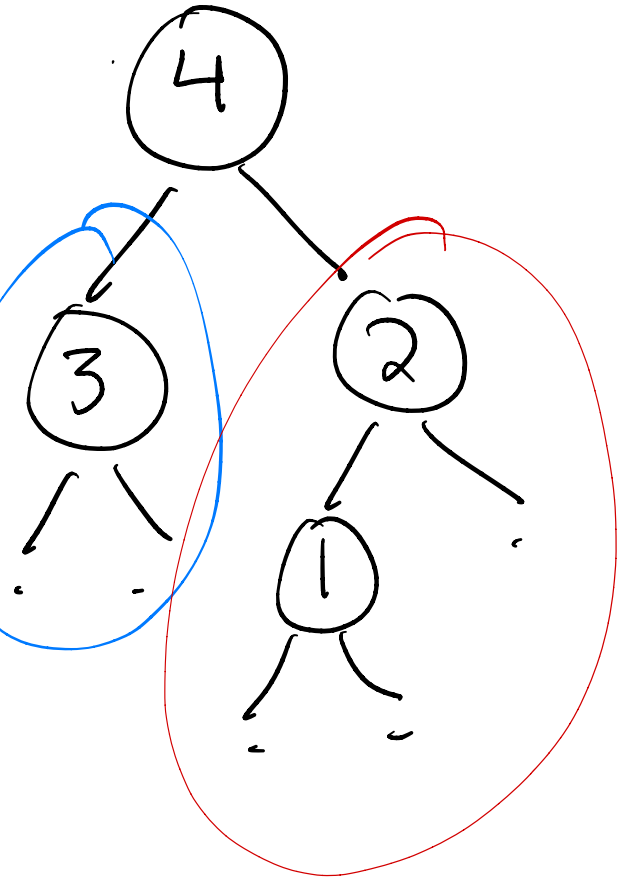
- Empty, or
- Node (l, x, r) where

  x: int

  l: tree

  r: tree

→ and that's it!



Node(

Node (Empty
3,
Empty)

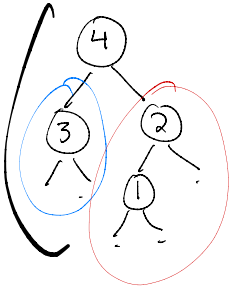
)

4,

Node(Node(Empty, 1, Empty)
2,
Empty)

)

(* Purpose: count how many #s are in the tree

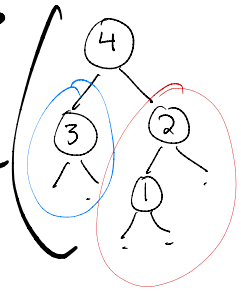
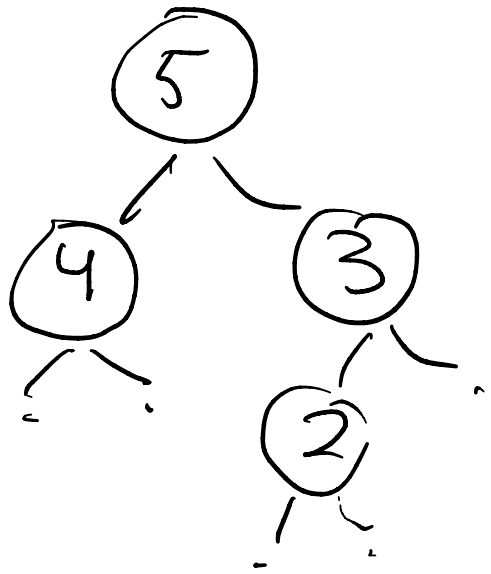
(* size(

) = 4 *)

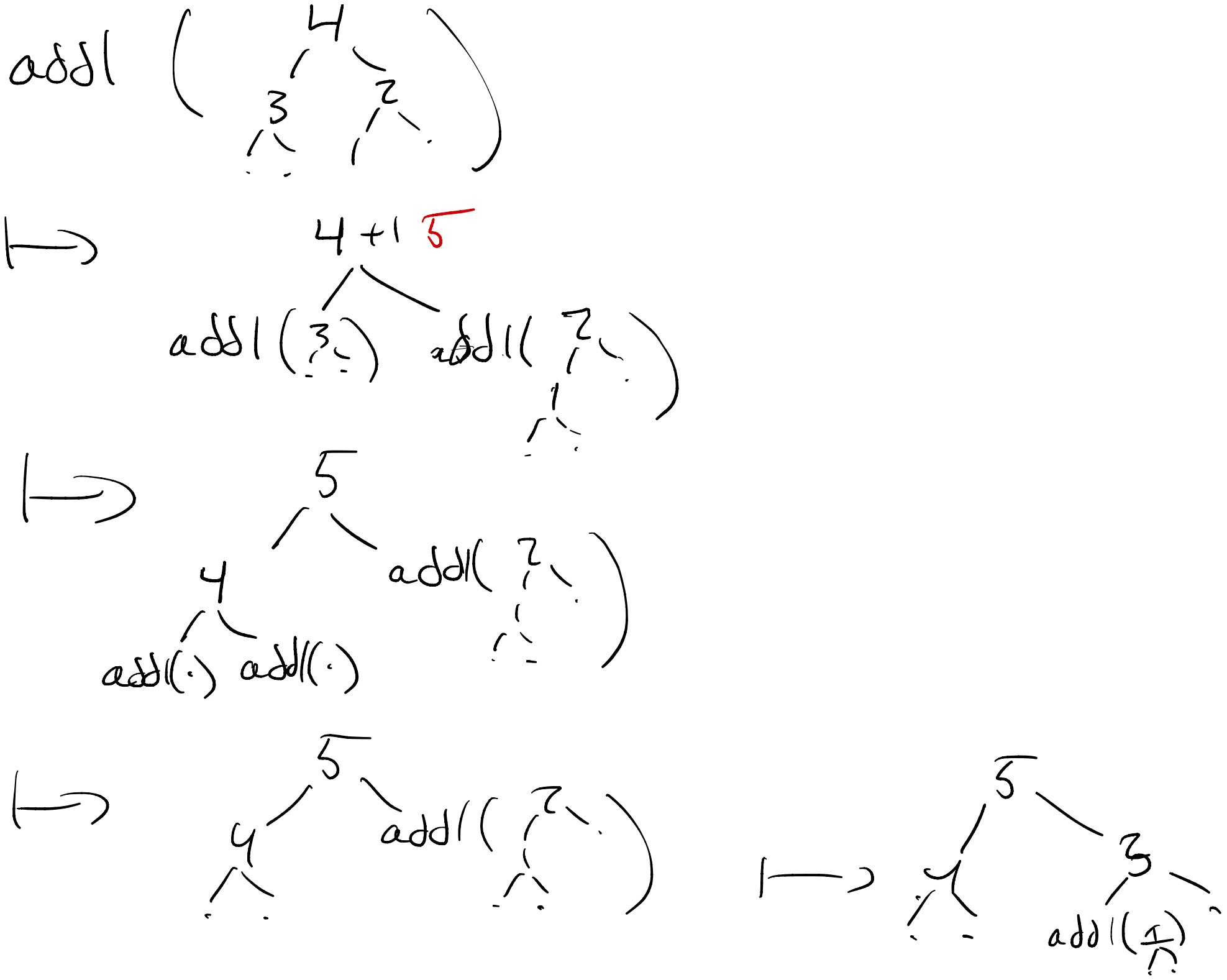
```
fun size(t:tree):int =
    Case t of
        Empty => 0
      | Node(l,x,r) =>
          1 + size(l) + size(r)
```

naturally get 2 subproblems

(* Purpose: add 1 to each number *)

(* add1  =  *)

```
fun add1 (t:tree): tree =
    case t of
      Empty => Empty
    | Node(l,x,r) =>
        Node(add1(l), x+1, add1(r))
```


add1
4
3
2
4+1
5
add1(3)
add1(2)
1
5
4
add1(2)
add1(·)
add1(·)
5
4
add1(2)
5
4
3
add1(1)

```
fun add1(t:tree): tree =
  case t of
    Empty => Empty
  | Node(l,x,r) =>
    Node(add1(l), x+1, add1(r))
```

$$W_{add1}(n) = 1 + 2W_{add1}\left(\frac{n}{2}\right) \text{ if balanced}$$

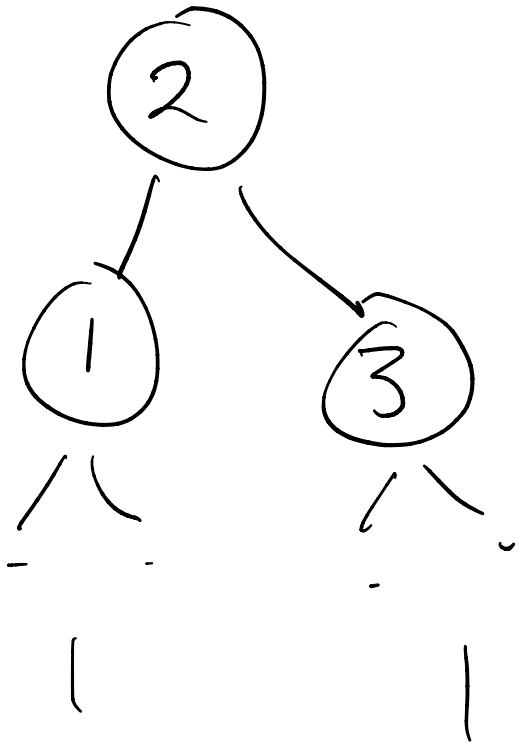
$O(n)$

size or depth

$$S_{add1}(n) = 1 + \max\left(S_{add1}\left(\frac{n}{2}\right), S_{add1}\left(\frac{n}{2}\right)\right) \quad \text{(if balanced)}$$

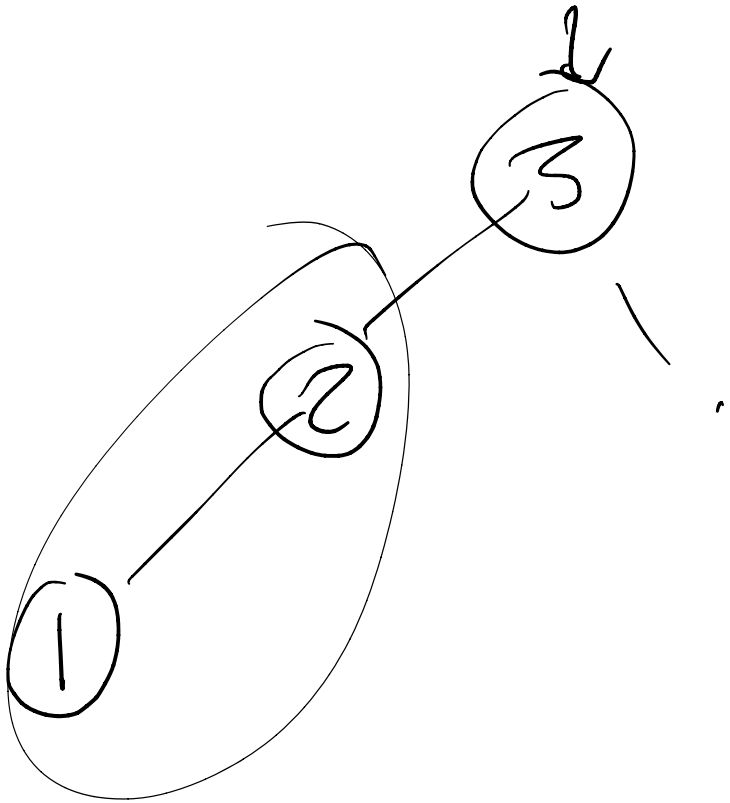
$$= 1 + S_{add1}\left(\frac{n}{2}\right) = \underbrace{1+1+1+1+\cdots+1}_{\log_2 n}$$

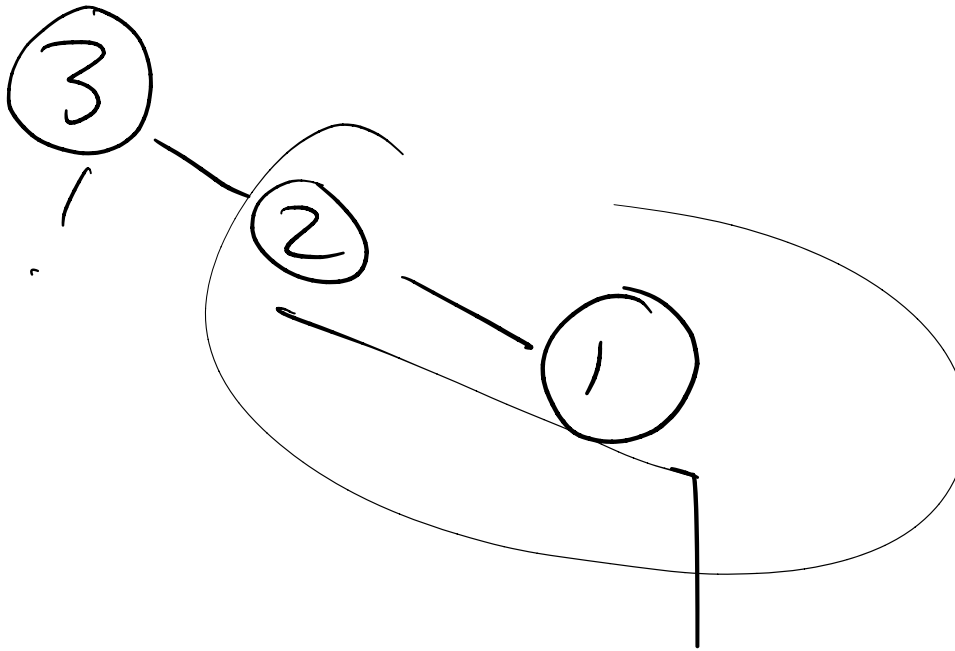
$O(\log n)$



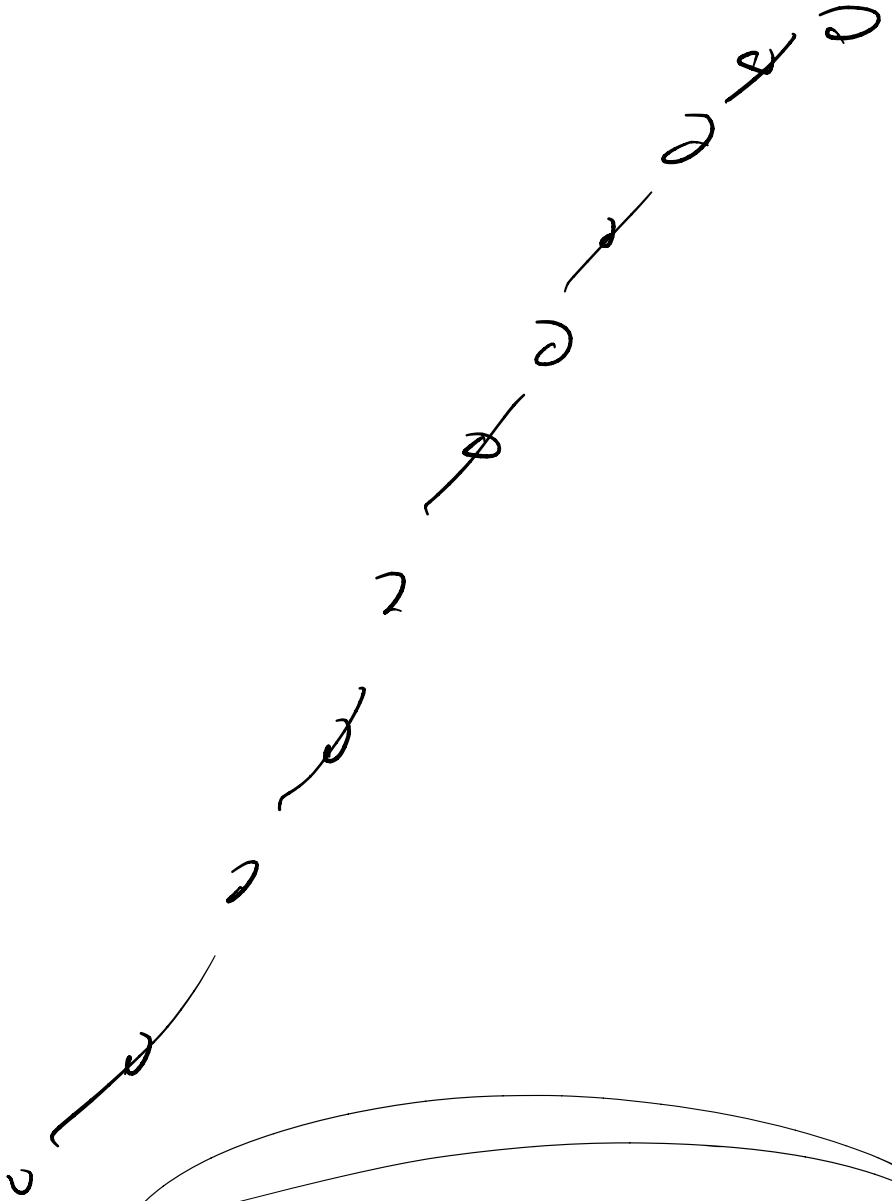
balanced:

Same size to the left + right of every Node!





balanced



unbalanced

$$\max\left(\frac{1\text{billion}}{p}, 30\right)$$