Pattern Matching

```
case n of
    0 => ________
  | _ => ________
```

```
case l of
    [] => ____
  | x::xs => ____
```

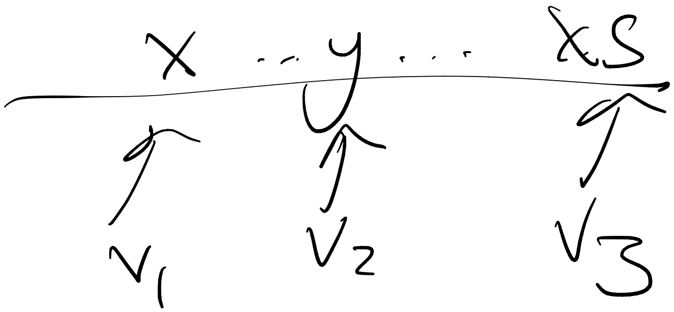
```
case b of
    true =>
  | false =>
```

```
case 1 of
    0 => ________
  | 1 => ____
  | _ => ________
```

```
case l of
    [] => ______
  | [x] => ________
  | x::y::xs => ______
```

Patterns	Type	Values match
0 1 2	int	that value
true false	bool	that value
x _	any	any value
[ ] $P_1 :: P_2$	int list	[ ] $v_1 :: v_2$ where P_1 matches v_1, P_2 matches v_2
(P_1, P_2)	$T_1 * T_2$	(v_1, v_2) P_1 match v_1, P_2 match v_2

$$\underbrace{x :: (y :: xs)}_{\text{pattern}} \Longrightarrow$$



$P_1 :: P_2$

$P_1 = \underline{x}$

$P_2 = \underline{y} :: \underline{xs}$

$\underline{v_1} :: (\underline{v_2} :: \underline{v_3})$

case e of

$p_1 \Rightarrow e_1$

$| \ p_2 \Rightarrow e_2$

$| \ p_3 \Rightarrow e_3$

⋮

① step e to a value v

② try matching v with p_1

success! step e_1

fails try p_2

⋮

first-match

case 0 of

_ => true "$\longmapsto$" true

| 0 => false "redundant" → error

case 0 of

0 => true $\longmapsto$ true

| _ => false

```
case x of
    y => true
  | ... => - - - -
```

not test

x = y

```
case x = y of
    true => ——
  | false => ——
```

exhaustiveness: patterns cover all possible values of that type

```
fun f(l) =
   case l of
      [x] => [x]
    | x::xs => ---recur...
```

no case for []

non-exhaustive warning → Match exception

```
fun merge(l1, l2) =
   case (l1, l2) of
        ([], _) =>              ________
      | (_, []) =>              ________
      | (x::xs, y::ys) =>       ________
                  ↑
             y1::y2::ys
                    ⋮
```

Lect 9 : span of mergesort

```
fun mergesort(l) =
    case l of
       [] => []
     | [x] => [x]
     | _ => let val (p1, p2) = split(l)
               in
              merge(mergesort p1 , mergesort p2)
          end
```

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

$$\underline{W_{split}(n)} = 1 + \underline{W_{split}}(n-2) \qquad O(n)$$

$$\underline{S_{split}(n)} = 1 + \underline{S_{split}}(n-2) \qquad O(n)$$

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

```
fun merge(l1, l2) =
    case (l1, l2) of
        ([], _) => l2
      | (_, []) => l1
      | (x::xs, y::ys) =>
          (case x ≤ y of
              true => x :: merge(xs, l2)
            | false => y :: merge(l1, ys)))
```

length l_1 + length l_2

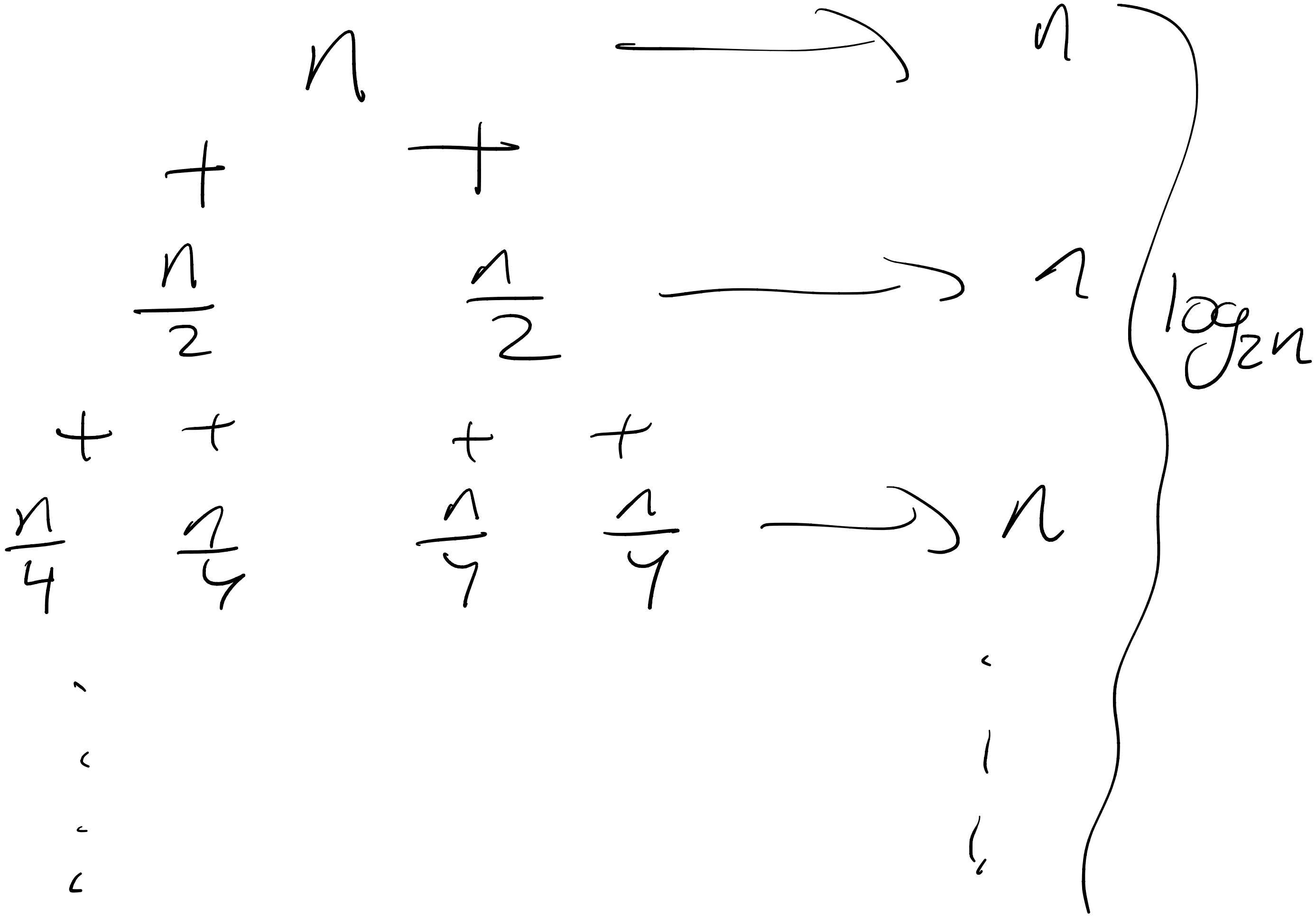
$W_{merge}(s) = 1 + W_{merge}(s-1)$ $\quad O(s)$

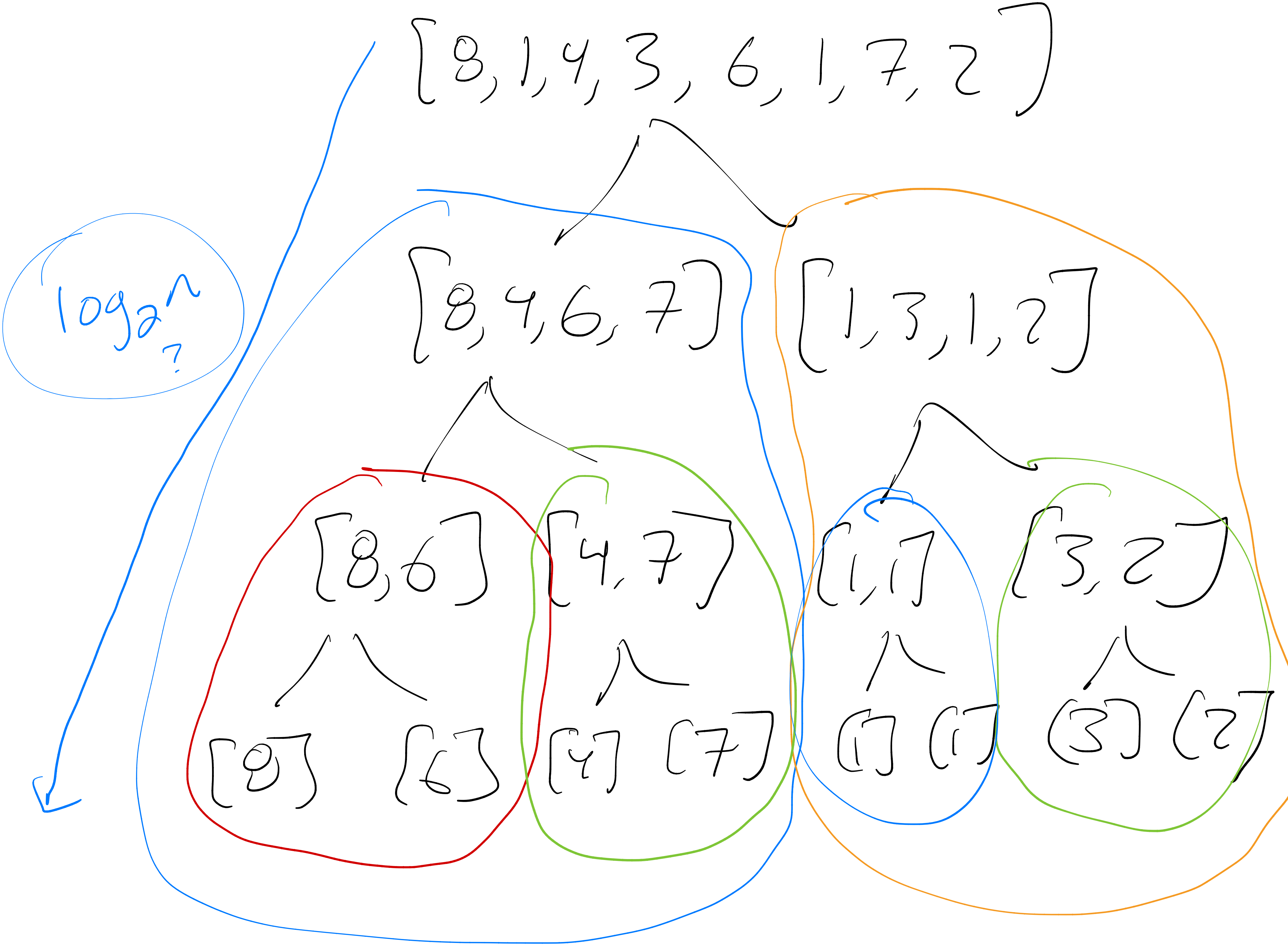
$S_{merge}(s) = 1 + S_{merge}(s-1)$ $\quad O(s)$

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

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

$O(n\log n)$




[8, 1, 4, 3, 6, 1, 7, 2]
log₂n ?
[8, 4, 6, 7]
[1, 3, 1, 2]
[8, 6]
[4, 7]
[1, 1]
[3, 2]
[8]
[6]
[4]
[7]
[1]
[1]
[3]
[2]

Span of mergesort

Brent
$O(\max(\frac{w}{p}, S))$

- time with "enough" processors to do as much as possible in parallel

- # steps for the slowest part (when done in parallel)

$$S_{mergesort}(n) = S_{split}(n) + S_{merge}(n)$$

$$+ \max\left(S_{mergesort}\left(\frac{n}{2}\right), S_{mergesort}\left(\frac{n}{2}\right)\right)$$

$$= \underline{n} + 1 \underline{S_{mergesort}\left(\frac{n}{2}\right)}$$

<u>Closed form:</u>

$$n + \frac{n}{2} + \frac{n}{4} + \frac{n}{8} + \cdots\cdots$$

$$n\left(1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \cdots\cdots\right)$$

$\leq n * 2$ "Zeno's paradox" $O(n)$

$n = 1$ billion

$\log_2 n \approx 30$

$n \log_2 n \approx 30$ billion

Span is ≈ 1 billion

$O(n)$

want:

span

is

$O((\log_2 n)^k)$

$[k = 3]$

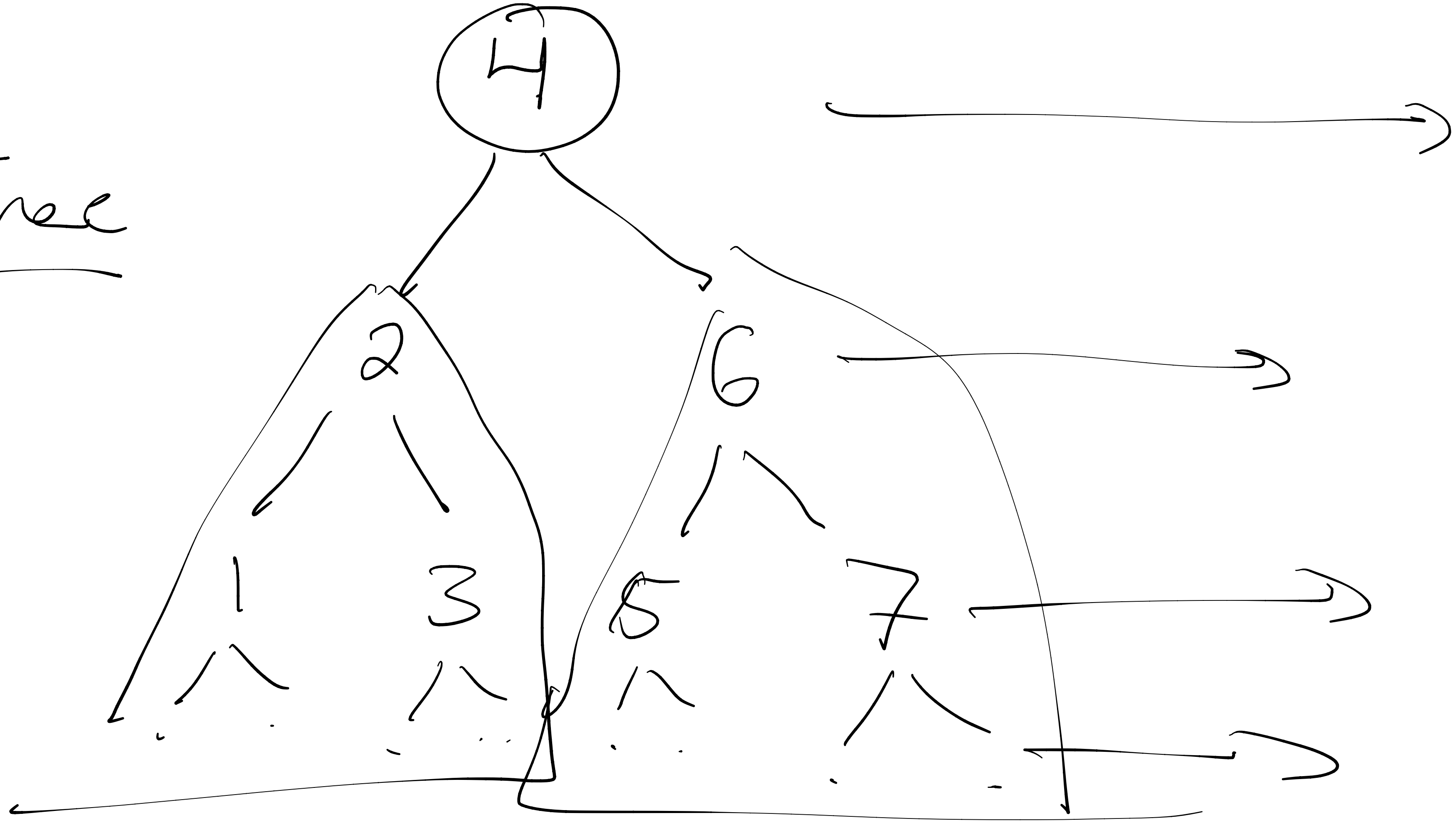
Problem: lists are too sequential

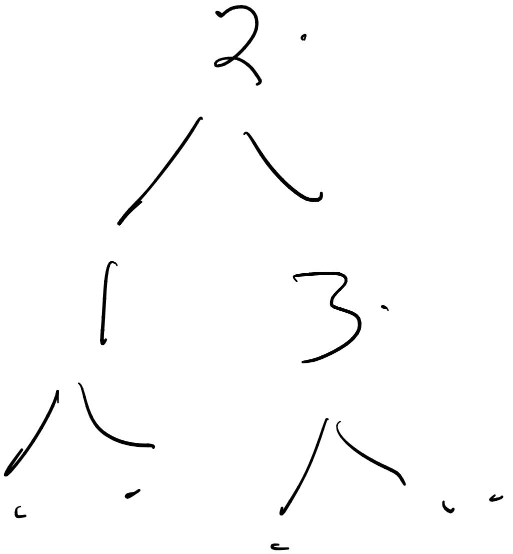
Sol: trees

Trees

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

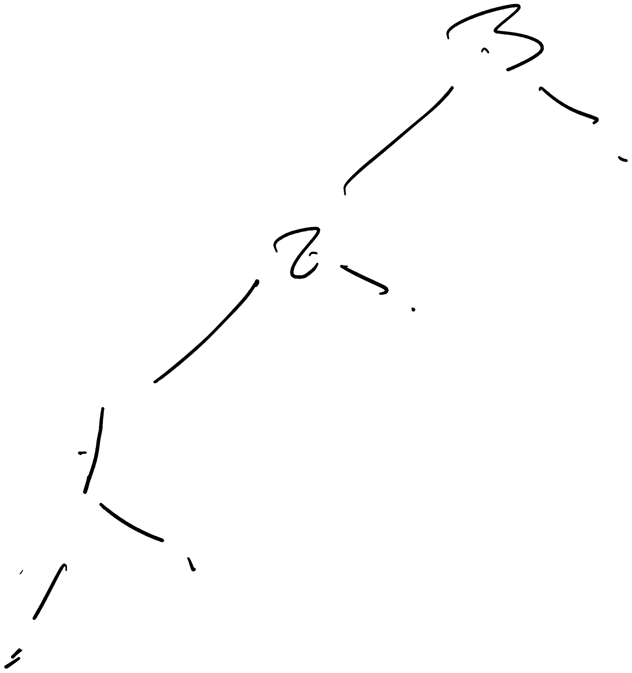
Tree

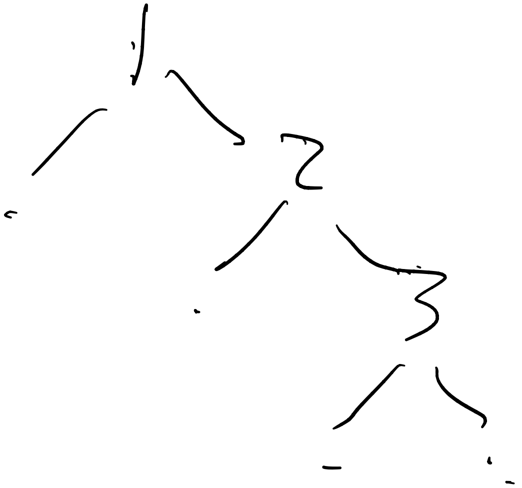




balanced

#levels is $O(\log_2(\#\text{elts}))$





not balanced

A tree is either

Empty, or

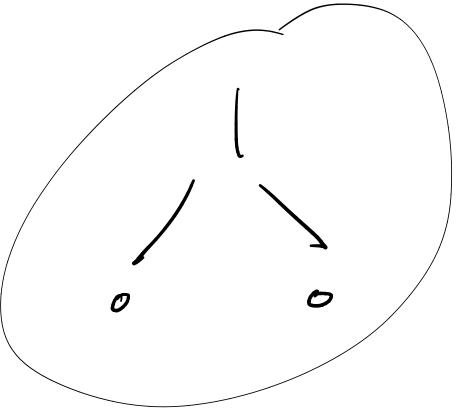
Node(l, x, r) where

l : tree
x : int
r : tree

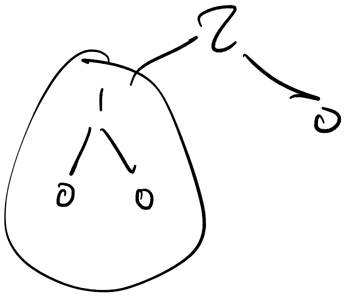
→ and that's it!

```
datatype
 tree
 = Empty
 | Node of
  tree * int * tree
```

Empty



Node(Empty, 1, Empty)



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

```
fun f(t) =
 case t : tree of
   Empty => ________
 | Node(l, x, r) =>
      ....  x  ....  l ... r ..
             f(l)
             f(r)
```

Structural induction on trees

case for Empty:

Case for Node(l, x, r):

IH for l

IH for r

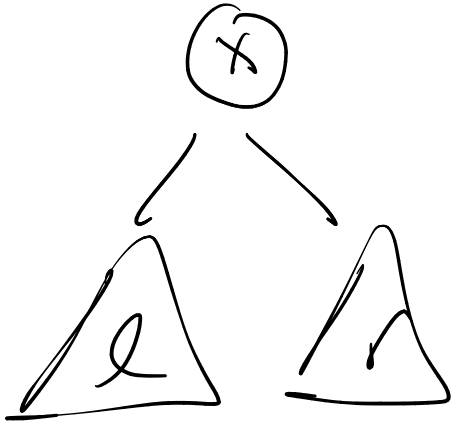
To show for $Node(l, x, r)$

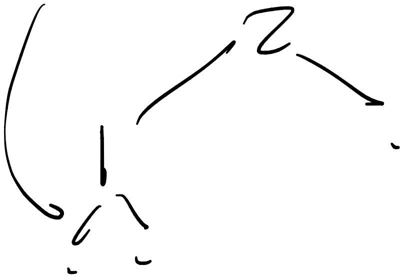
(* compute the # elts in a tree *)

(* size  = 2 *)

```
fun size (t : tree) : int =
    case t of
      Empty => 0

    | Node(l, x, r) =>
            1 + size(l) + size(r)
```



$\mathrm{size}\Big($  $\Big)$

$\mapsto 1 + \underline{\mathrm{size}\Big(}$  $\underline{\Big)} + \underline{\mathrm{size}(\cdot)}$

$\mapsto 1 + \underline{(1 + \mathrm{size}(\cdot) + \mathrm{size}(\cdot))} + \underline{\mathrm{size}(\cdot)}$

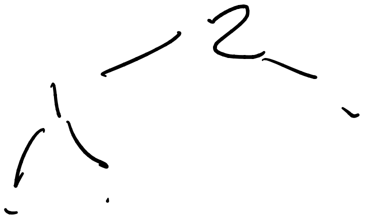
$\mapsto 1 + (1 + 0 + \mathrm{size}(\cdot)) + \mathrm{size}(\cdot)$

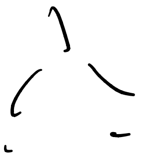
$\mapsto 1 + (1 + \cancel{(0} + \cancel{0)}) + \mathrm{size}(\cdot)$

$\mapsto 1 + (1 + 0 \quad) + \mathrm{size}(\cdot)$

$\mapsto 1 + (1 + \mathrm{size}(\cdot))$

$\mapsto 1 + (1 + 0) \mapsto 1 + 1 \mapsto 2$

size()

$\longmapsto$ 1 + size() + size($\cdot$)

$$\longmapsto 1 + (1 + \mathrm{size}(\cdot) + \mathrm{size}(\cdot)) + 0$$

$$\longmapsto 1 + ((1 \underset{\uparrow}{+} (0 \underset{\uparrow}{+} 0)) \underset{\uparrow}{+} 0)$$

$$\longmapsto 1 + (1 + 0) + 0$$

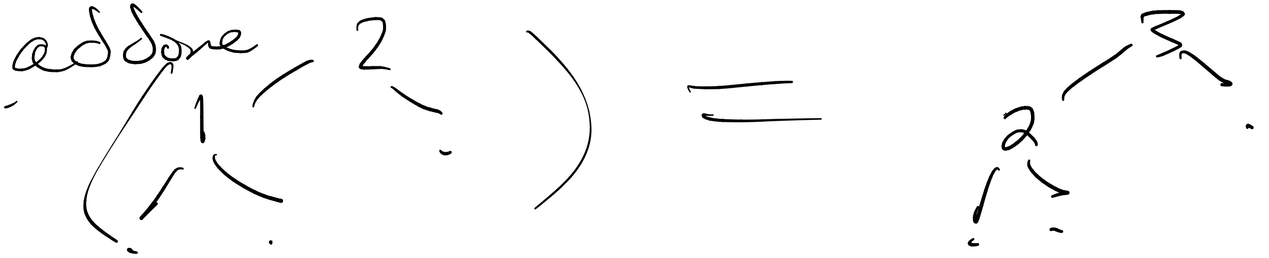
$$\longmapsto 1 + (1 + 0)$$

$$\longmapsto 1 + 1$$

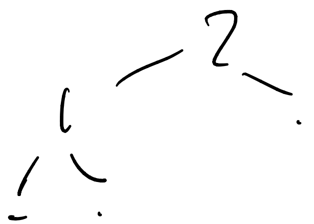
$$\longmapsto 2$$

add 1 to each elt

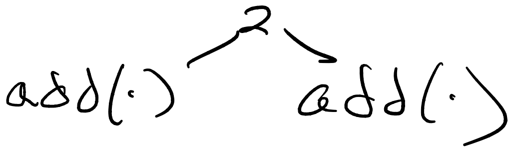
E.g.



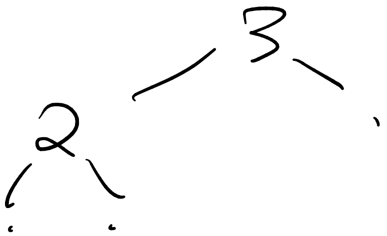
```
fun addone(t:tree):tree =
    case t of
        Empty => Empty
      | Node(l,x,r) =>
          Node(addone(l), x+1, addone(r))
```

addone()

⟼ Node(addone(), 2+1 = 3 , addone(·))

⟼ Node( , 3, Empty)

⟼ Node(Node(Empty, 2, Empty), 3, Empty)

= 

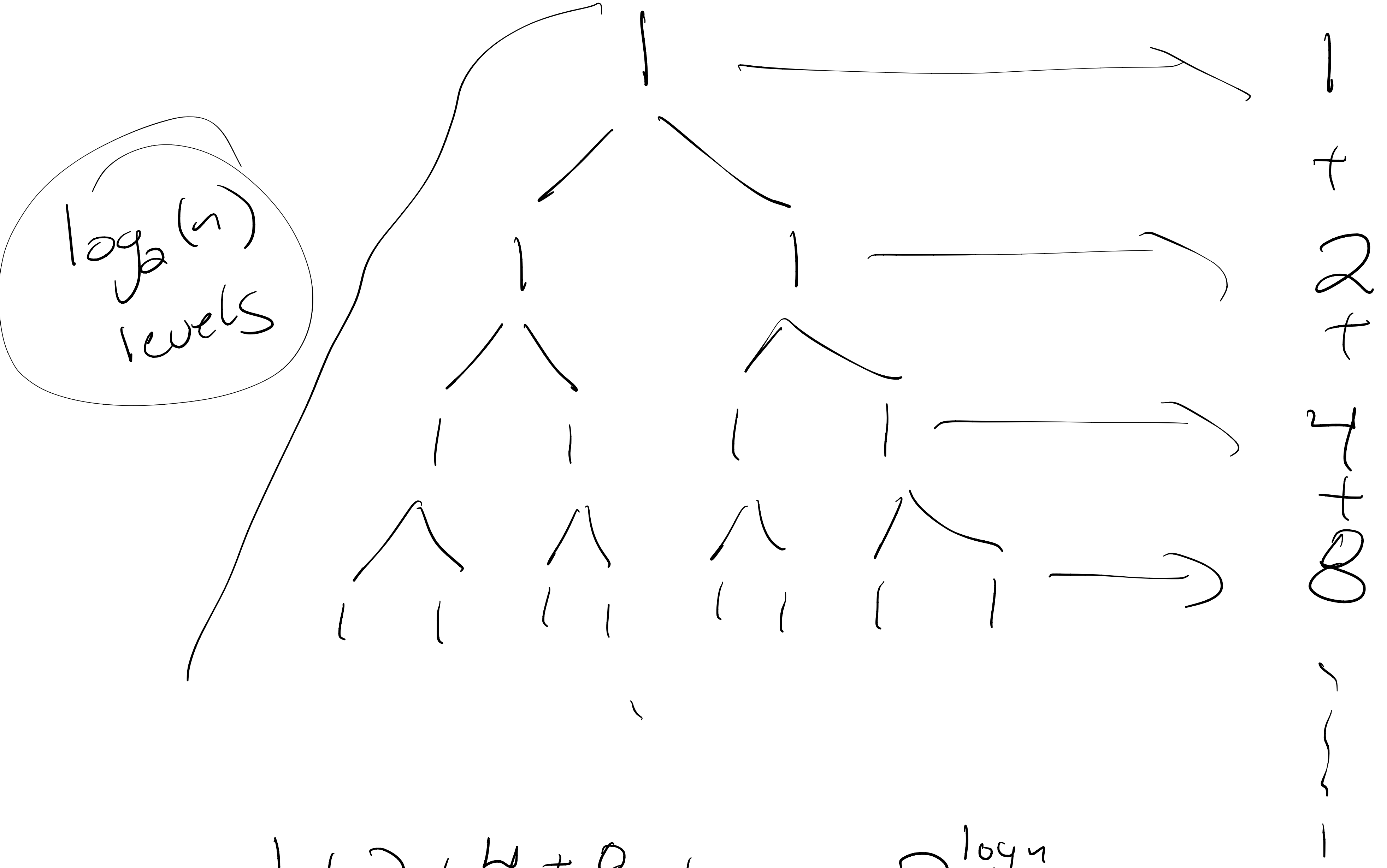
Work → assume tree is balanced

↳ size(l) is $\frac{n}{2}$

size(r) is $\frac{n}{2}$

$$W_{addone}(\underset{\underset{\text{size of } t}{\uparrow}}{n}) \leq 1 + W_{addone}(\underset{\underset{\text{size of } l}{\uparrow}}{\tfrac{n}{2}}) + W_{addone}(\underset{\underset{\text{size of } r}{\uparrow}}{\tfrac{n}{2}})$$

$$= 1 + 2\,W_{addone}\left(\frac{n}{2}\right)$$



$1+2+4+8+ \cdots 2^{\log n}$

$= 2 * 2^{\log_2 n} - 1$ is $2n-1$ is $O(n)$

$\boxed{\text{Span}}$ assume balanced

$$S_{addone}(n) = 1 + S_{addone}\left(\frac{n}{2}\right)$$

$$\left.\begin{array}{c}1\\+\\1\\+\\1\\+\\1\\ \vdots\end{array}\right\}\ \boxed{\log_2 n \text{ levels}}$$

$O(\log_2 n)$

span