Lecture 15: Sequences

- **work**: index access is $O(1)$ work
  fast

- **span**: map is $O(1)$ span

Array-backed sequences

$A[i]$ gets elt at pos. i

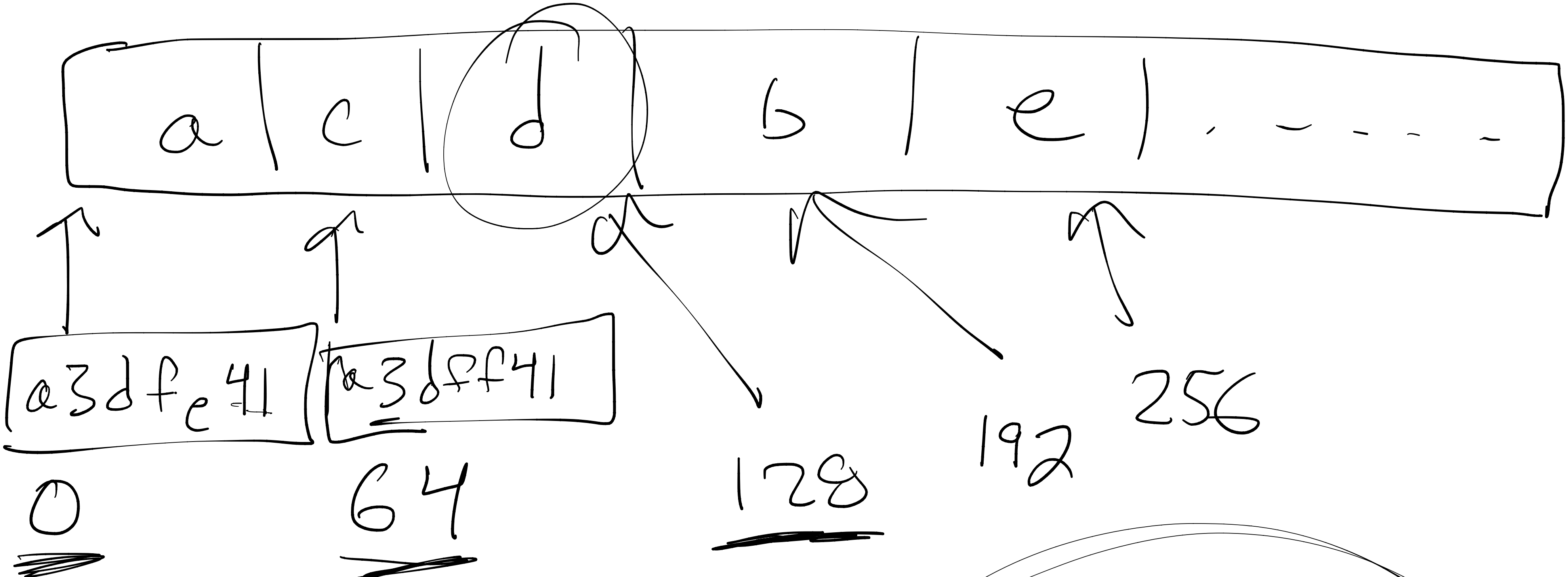
index calculations

$A[i] = f$ } update

$$\overbrace{[\,a,\ c,\ d,\ b,\ e\,]}^{A}$$

elements	a	c	d	b	e
index	0	1	2	3	4

"0-indexed"



$A[2]$

work of $A[i]$ is $O(1)$

bounds:

$A[i]$ $\quad 0 \le i < \text{length } A$

```
fun nth (l : 'a list, n : int) : 'a =
    case (n, l)
        (0, x::_) => x
      | (_, x::xs) => nth(xs, n-1)
```

$O(n)$ work

n either or length(l)

assume $0 \le n < \text{length}(l)$

$\boxed{\text{accessing elt. at position } i}$

arrays: $O(1)$

lists: $O(n)$

trees: $O(\log n)$

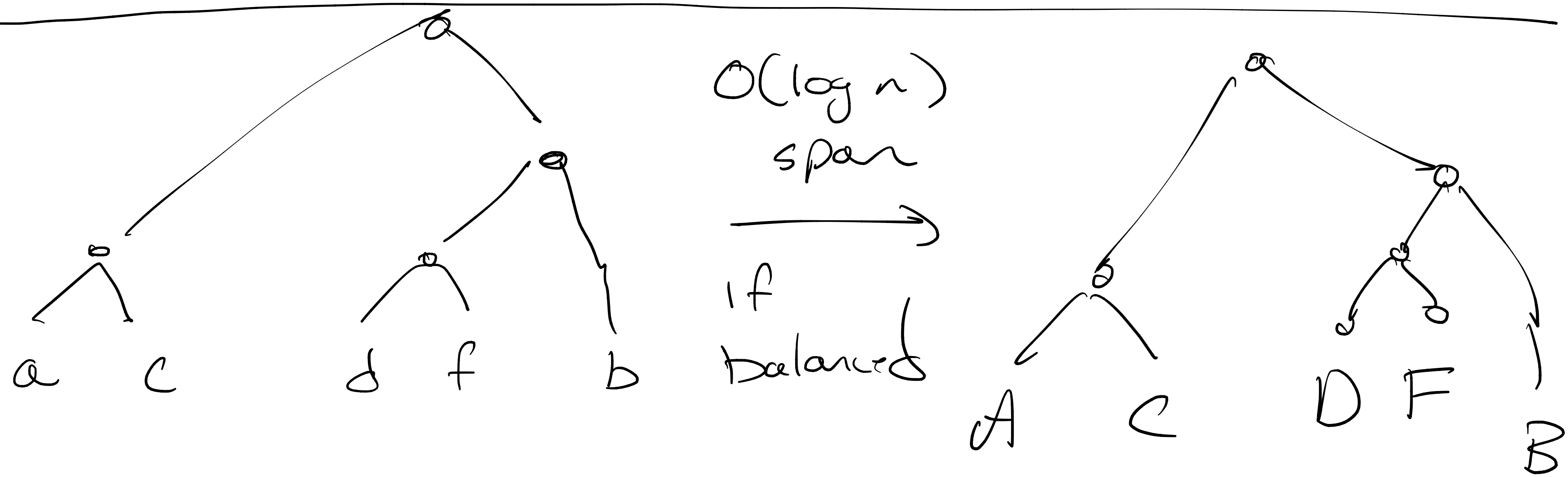
(balanced)

[a, c, d, f, b]

↓ map capitalize

[A, C, D, F, B]

list:
span is
$O(n)$



Conceptually:

$$\langle a, c, d, f, b \rangle$$

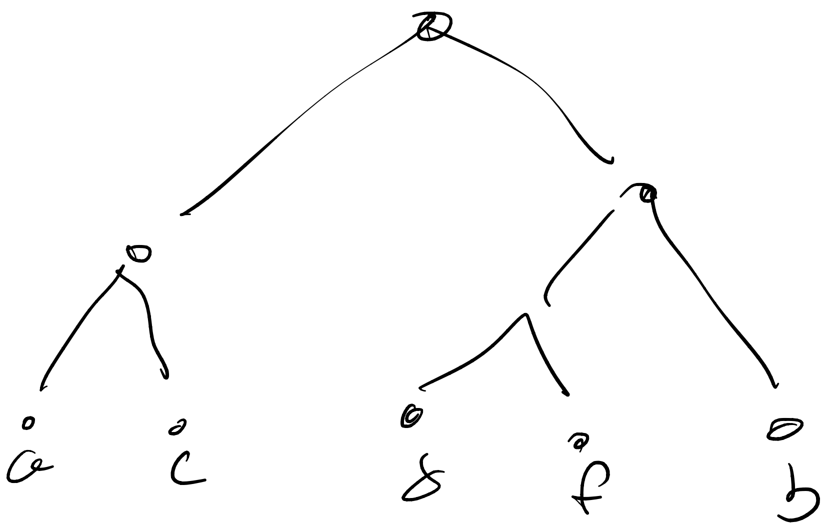
$$\downarrow$$

$$\langle A, C, D, F, B \rangle$$

should be $O(1)$ span!

"ordered collection of elts"

1. [a, c, d, f, b]

2. 

3.

a	c	d	f	b

type 'a Seq.seq → array-backed

val Seq.map : ('a → 'b * 'a Seq.seq) → 'b Seq.seq

val Seq.length : 'a Seq.seq → int

val Seq.nth : int * 'a Seq.seq → 'a

val Seq.reduce : (('a * 'a → 'a) *
'a *
'a Seq.seq)
→ 'a

⋮

Abstract specification of

1. behavior

2. work and span

1. Behavior

$map(f, \langle x_0, x_1 \ldots\ldots x_{n-1} \rangle)$

$= \langle f\,x_0, f\,x_1, \ldots\ldots, f\,x_{n-1} \rangle$

seq:

'a list

→ 'a Seq.seq

2. work: sum of W_f on x_i for all i

span: max of S_f on x_i for all i

If f is constant time Work is $O(n)$ n = length of input

span is $O(1)$

upper : char $\rightarrow$ char

$$\text{Seq.Map}\left(\text{upper}, \overbrace{\langle a, c, d, f, b \rangle}^{\text{length } n}\right)$$

$O(n)$ work

$O(1)$ span

$\boxed{\text{Seq. length}}$

① Behavior:

$$\text{Seq.length}\ \langle x_0, x_1, \ldots, x_{n-1} \rangle = n$$

② work
span

$\left.\begin{array}{l}\text{work}\\\text{span}\end{array}\right]\ \underline{\underline{O(1)}}$

Seq.nth "A[i]"

(1) Behaviour

$$\text{Seq.nth}(i, \langle x_0, x_1, \ldots, x_{n-1} \rangle)$$

$$= x_i \quad \text{if } 0 \le i < n$$

or raise Range otherwise

(2) Work, Span $\Big\}$ $O(1)$

$\boxed{\text{Reduce}}$: (('a * 'a → 'a) *   combiner
'a   base case
'a Seq.seq)
→ 'a Seq.seq

① Behaviour:

$$\text{reduce}(\overset{c}{\odot}, b, \langle x_0, x_1, \ldots, x_{n-1} \rangle)$$

$$= (((x_0 \odot x_1) \odot (x_2 \odot x_3 \odot)) \ldots (\ldots \odot x_{n-1}))$$

balanced tree parenthesization

or = b if ⟨⟩

$$\left.\begin{array}{l} x_1 + (x_2 + (x_3 + x_4)) \\[2em] (x_1 + x_2) + (x_3 + x_4) \\[2em] ((x_1 + x_2) + x_3) + x_4 \end{array}\right\}$$

associativity

① assume associative; all $=$
(but: floating point)

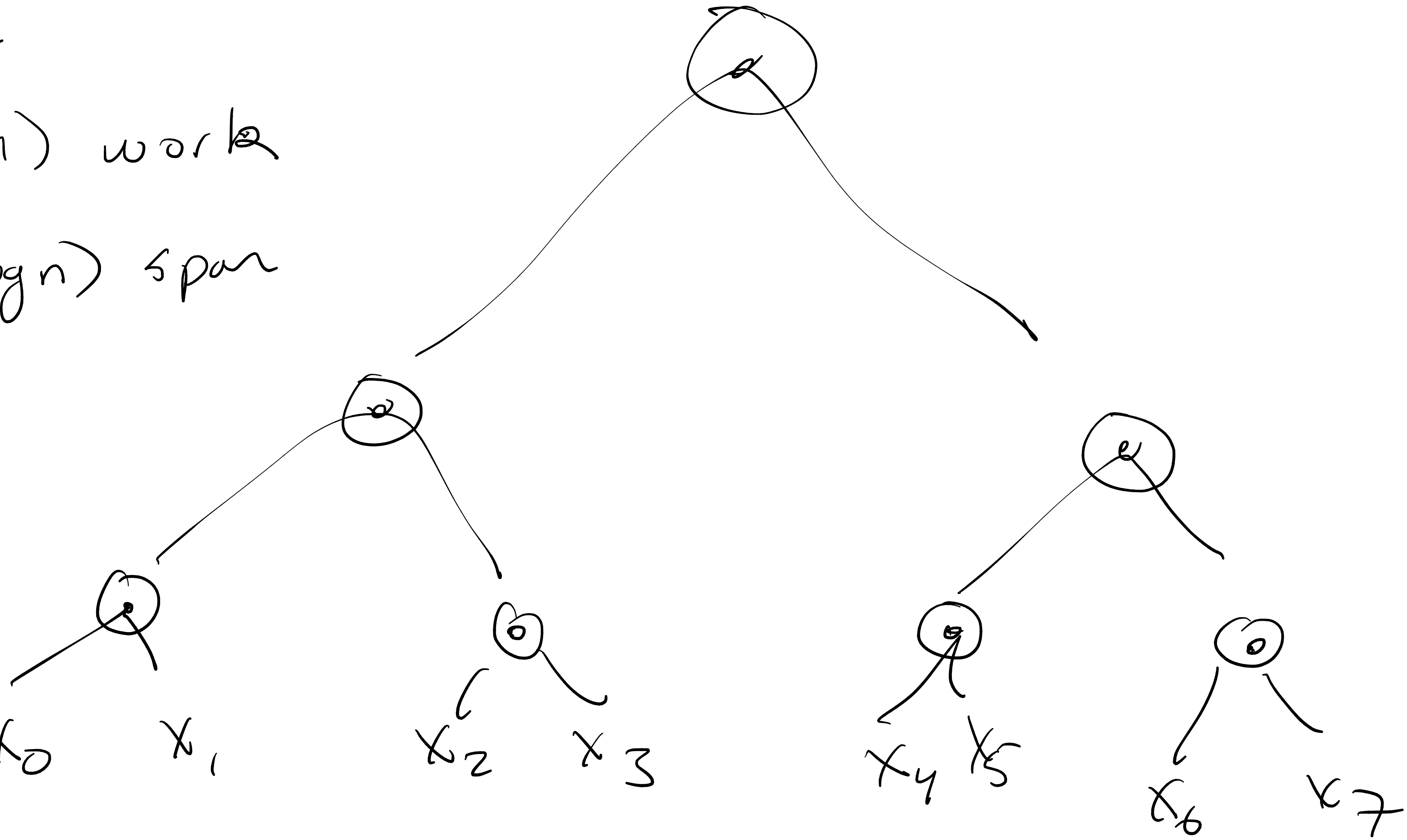
② fix a parenthesization

If $\odot$ constant time

then

$O(n)$ work

$O(\log n)$ span



Analyze reduce

: write + solve recurrence

$\text{reduce}(\odot, b, \langle\rangle) = b$

$\text{reduce}(\odot, b, s) =$

$\odot($ reduce on left, reduce on right $)$

first half of S

second half of S

calculating left + right $O(1)$

```
fun sum (s : int Seq.seq) : int =
  Seq.reduce (fn (x, y) => x + y,
              0,
              s)
```

$O(n)$ work

$O(\log n)$ span

sum $\langle 1, 2, 4, 8 \rangle = 15$

$1 + 2 + 4 + 8 = 15$

sum $\langle \rangle = 0$

count

$$\langle \langle 1, 0, 1, 0, 0 \rangle, \langle 0, 1, 0, 0, 0 \rangle, \ldots \rangle \xrightarrow{map} \langle 2, 1, 0, 0 \rangle \xrightarrow{sum} 3$$

```
fun count(s: (int Seq.seq) Seq.seq) : int =
    sum(Seq.map(sum, s))
```

count

$$\langle \langle 1, 0, 1, 0, 0 \rangle, \langle 0, 1, 0, 0, 0 \rangle, \ldots \rangle$$

$$\mapsto \text{sum} \langle \text{sum} \langle 1, 0, 1, 0, 0 \rangle, \text{sum} \langle 0, 1, 0, 0, 0 \rangle, \ldots \rangle$$

$$\mapsto \text{sum} \langle 2, 1, 0, 0 \rangle$$

$$\mapsto 3$$

Assume $n \times n$

	input size	work	span
inner sum	always n	$O(n)$	$O(\log n)$
map	$n \times n$	$O(n^2)$ +	$O(\log n)$ +
outer sum	n	$O(n)$	$O(\log n)$
Overall		$O(n^2)$	$O(\log n)$

Brent's principle:

time on p processors

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

can use $\approx \frac{W}{S} \approx \frac{n^2}{\log n}$

processors