

# Übung 15

## Asymptotischer Aufwand

Einführung

Aufwände in Haskell

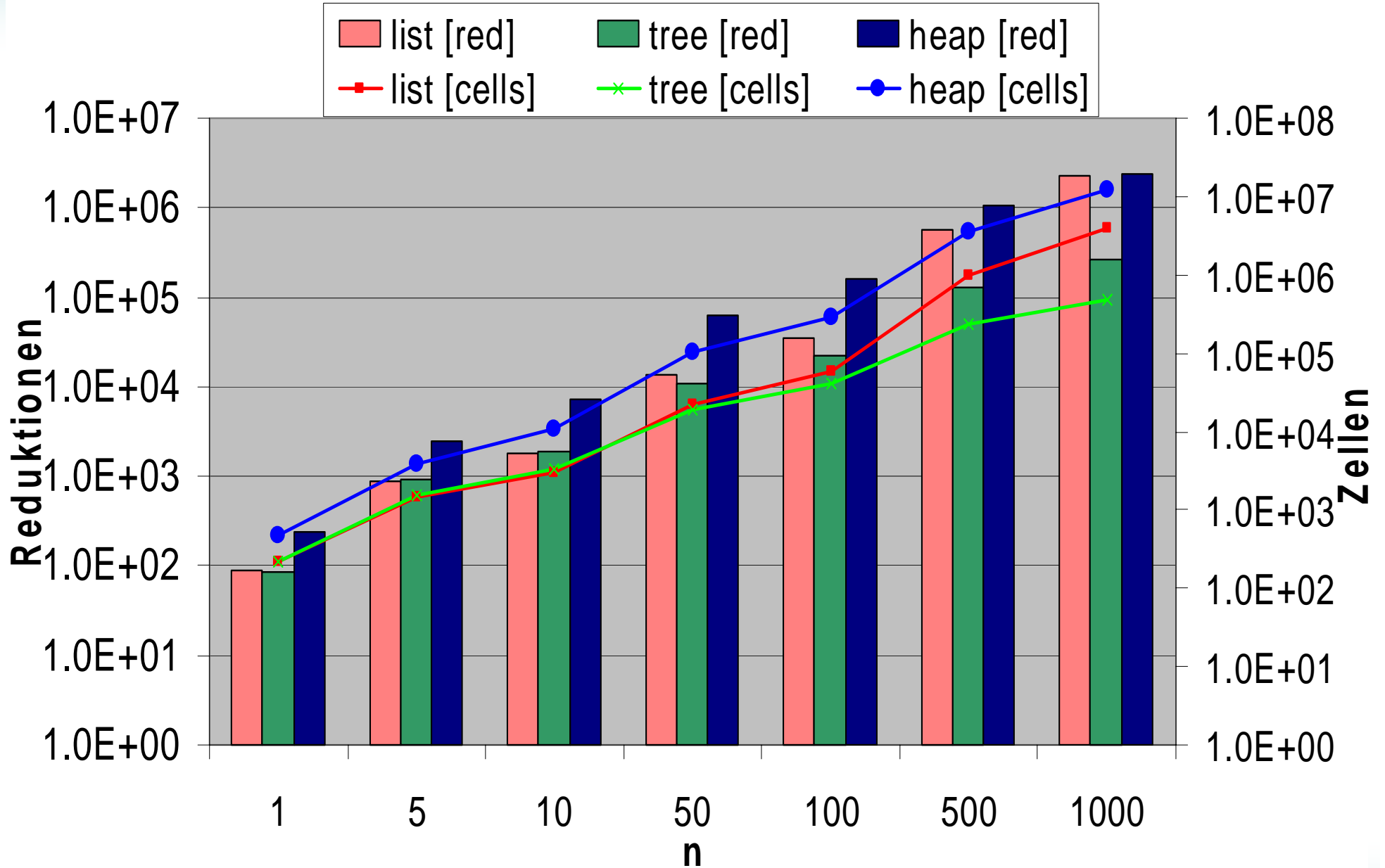
Beispiele

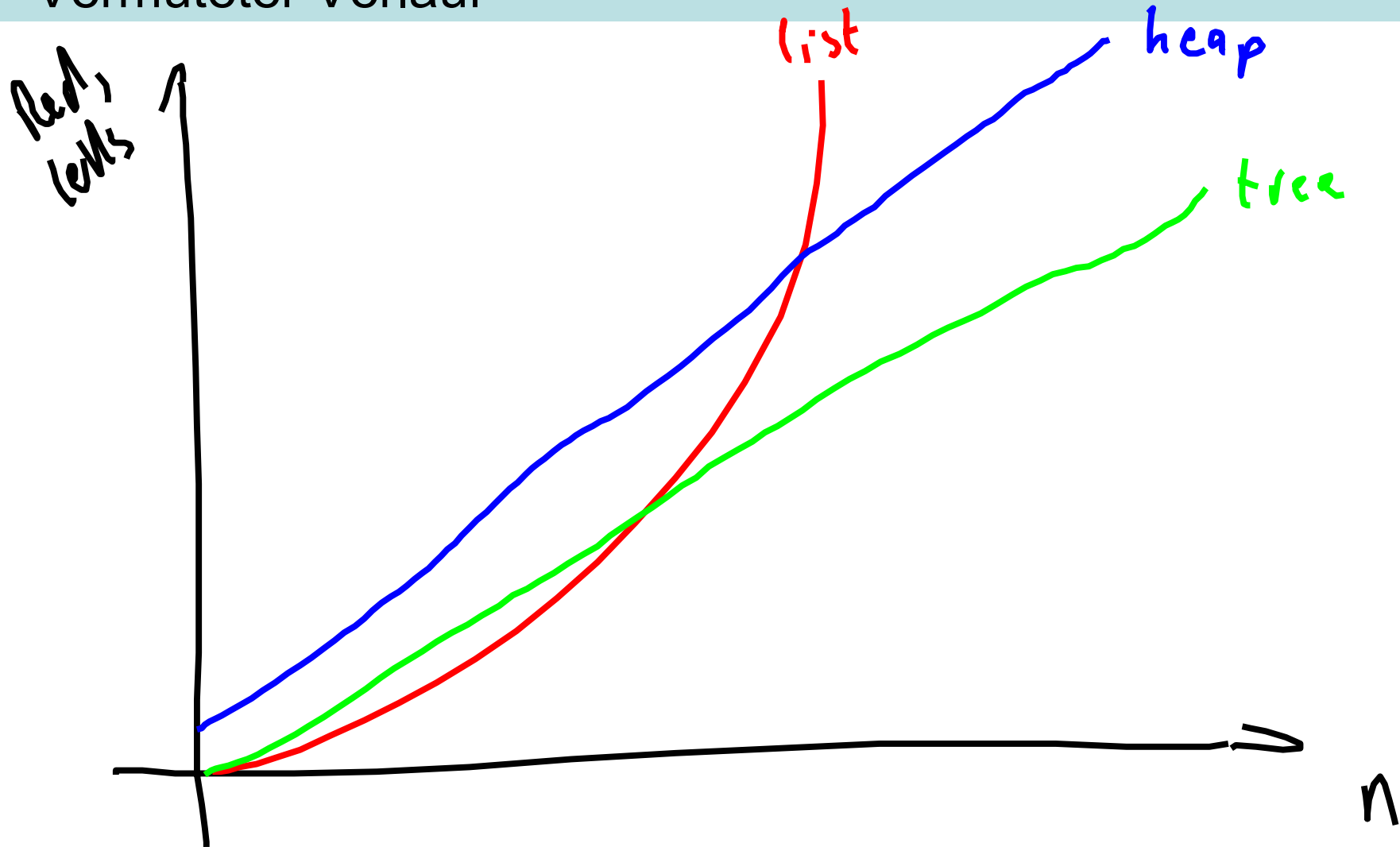
Vergleich:

Implementierungen der  
Prioritätsschlange

Heapsort







- $O(1)$  für
  - Elementaroperationen auf Int, Float, Double, Char, Boolean
    - Integer problematisch, da unbegrenzte Länge
  - Listenoperator :  $(x : xs)$
  - Tupelbildung  $(1, 2)$   $( 'a', 'b', 'c' )$
  - Konstruktoren
  - Mustererkennung
  - Arrayzugriff (einen Wert lesen / schreiben)
  - Wächter
  - let, where
  - Funktionsaufruf

- Stets Aufwände der inneren Ausdrücke berücksichtigen!



▪  $\text{head } (x:xs) = x \quad O(1)$

$\underbrace{\begin{matrix} O(1) \\ O(1) \end{matrix}}_{O(1)}$

▪  $\text{tail } (x:xs) = xs \quad O(1)$

▪  $\text{length } [] = 0 \quad O(1)$

$O(1)$

$\text{length } (x:xs) = 1 + \text{length } xs$

$\underbrace{\hspace{2em}}_{O(1)} \quad \underbrace{\hspace{2em}}_{O(1)}$

$n = \text{Länge der Liste}$

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

$= \sum_{i=0}^n c = c \cdot n \in O(n)$

▪  $[] ++ ys = ys \quad O(1)$

$(x:xs) ++ ys = x : (xs ++ ys)$

$\underbrace{\hspace{2em}}_{O(1)} \quad \underbrace{\hspace{2em}}_{O(1)}$

$n = \text{Länge der 1. Liste}$

$g(n) = O(1) + f(n-1) \in O(n)$



- $\text{take } 0 \text{ } \_ = []$   $O(1)$
- $\text{take } \_ [] = []$
- $\text{take } n (x:xs) = x : (\text{take } (n-1) xs)$   
 $\quad \quad \quad \underbrace{\quad}_{O(1)} \quad \quad \quad \underbrace{\quad}_{O(1)}$   
 $\quad \quad \quad O(1)$

$$f(n) = O(1) + f(n-1) \in O(n)$$

- $\text{drop } 0 \text{ } xs = xs$
- $\text{drop } \_ [] = []$
- $\text{drop } n (x:xs) = \text{drop } (n-1) \text{ } xs$



~~$O(\log n)$ ?~~

▪ bsearch [] \_ = False  $O(1)$

$O(1)$ 
  
 bsearch xs x | x < y = bsearch l s x  $F(n/2)$ 
  
                   | x == y = True  $O(1)$ 
  
                   | x > y = bsearch r s x

where l s = take h xs  $O(h) = O(n)$

(y:rs) = drop h xs  $O(h) = O(n)$

h = l `div` 2  $O(1)$

l = length xs  $O(n)$

$n = \text{länge } xs$

Op. auf Listenelem in konst. Zeit

$$F(n) = O(n) + 1 \times F(n/2)$$

$$1 < 2 \xrightarrow{(\text{HS})} \in O(n)$$





$$O(2n) = O(n)$$



▪  $\text{map } f [] = [] \quad O(1)$

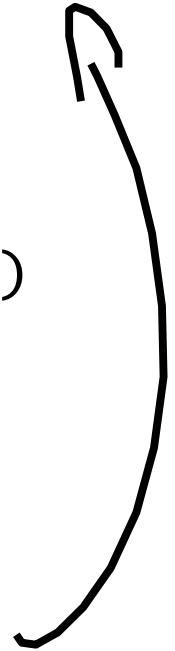
$\text{map } f (x:xs) = (f x) : (\text{map } f xs)$   
 $O(1) \quad O(1) \quad O(n-1)$

$g := \text{Länge von } xs$   
 $h = \text{Aufwand } f$

$O(n)$   
 $O(n) * O(h)$

▪  $\text{filter } f [] = []$

$\text{filter } f (x:xs) \mid f x = x : (\text{filter } f xs)$   
 $\mid \text{otherwise} = \text{filter } f xs$



- Listenfunktionen
  - $O(1)$ 
    - head, tail
  - $O(n)$ 
    - length, take, drop, ++
- Funktionen höherer Ordnung
  - $O(n)*f$  (f Aufwand der inneren Funktion)
- Binärsuche
  - $O(n)$  auf Listen
  - $O(\log n)$  auf Arrays



type PQueue a = [(a, Int)] *Möchste Prio. vorne*

create = []

dequeue (a:as) = as

front (a:as) = fst a

}  $O(1)$

enqueue [] (x, p) = [(x, p)]

enqueue (a:as) (x, p)

| snd a < p = (x, p):a:as  $O(1)$

| otherwise = a:(enqueue as (x, p))

$n$  = Länge d. Liste  $O(1)$

*Annahme: Zufällig verteilte Prios  $\Rightarrow$  erwarteter asympt. Aufwand  $O(n)$*



# Einfügen von $n$ Zahlen.

$$w = [1, 2, 3, 4, 5]$$

$$\begin{array}{cccccc} 1 & & & & & \\ 2 & 1 & & & & \\ 3 & 2 & 1 & & & \\ \vdots & & & & & \\ 5 & 4 & 3 & 2 & 1 & \end{array}$$
$$\sum_{i=1}^n 1 \in O(n)$$

$$w = [5, 4, 3, 2, 1]$$

$$\begin{array}{cccccc} 5 & & & & & \\ 5 & 4 & & & & \\ 5 & 4 & 3 & & & \\ \vdots & & & & & \\ 5 & 4 & 3 & 2 & 1 & \end{array}$$
$$\sum_{i=1}^n i \in O(n^2)$$



data PQueue a

= PQInner (PQueue a) a Int (PQueue a) | PQLeaf  
 deriving Show

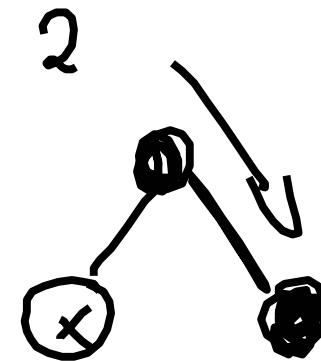
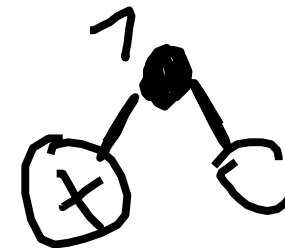
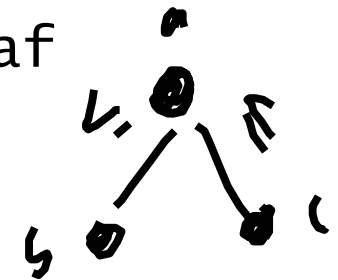
create = PQLeaf  $O(1)$

1 front (PQInner \_ a \_ PQLeaf) = a  $O(1)$

2 front (PQInner \_ \_ \_ r) = front r  
 $O(1)$

$h =$  Höhe d. Baumes.

$$f(h) = 1 + f(h-1) \in O(h)$$



enqueue PQLeaf (a,p)  $O(h)$   
 = PQInner PQLeaf a p PQLeaf  
 enqueue (PQInner l a' p' r) (a,p)  
 | p < p' = PQInner (enqueue l (a,p)) a' p' r  
 | otherwise = PQInner l a' p' (enqueue r (a,p))

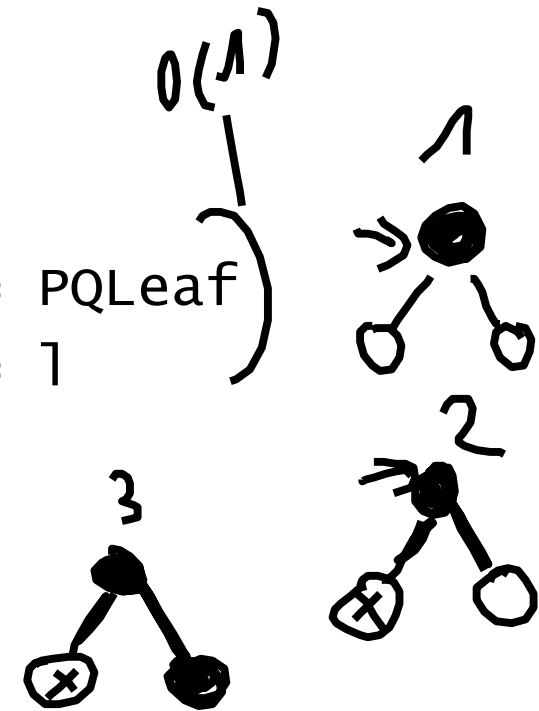
Prio. zufällig verteilt, erwarteter Aufwand?

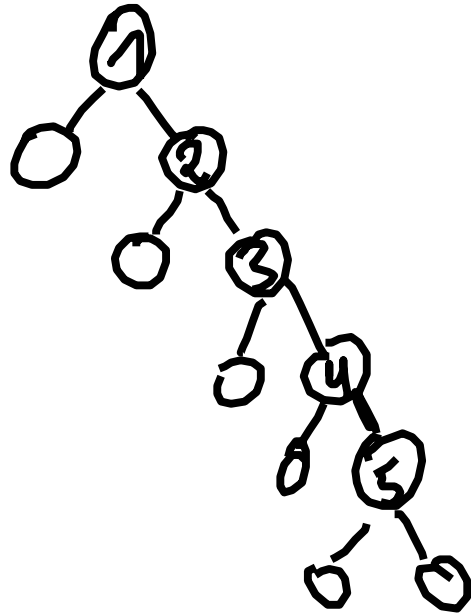
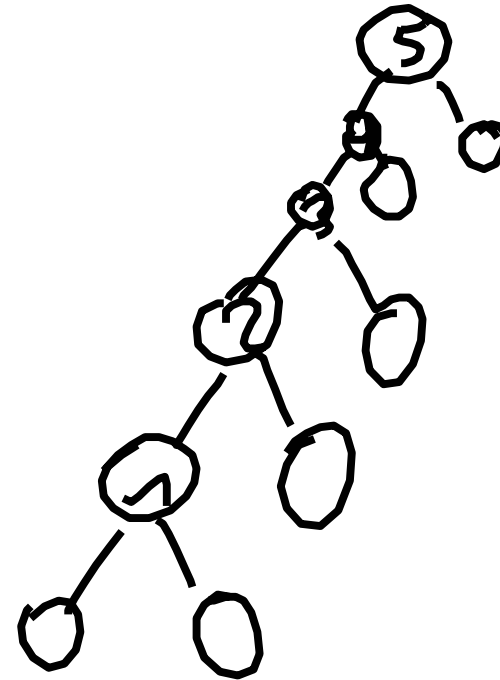
$$g(h) = O(1) + g(h-1) \in O(h)$$

- 1 dequeue (PQInner PQLeaf \_ \_ PQLeaf) = PQLeaf
- 2 dequeue (PQInner l \_ \_ PQLeaf) = l
- 3 dequeue (PQInner l a p r) = PQInner l a p (dequeue r)

$h :=$  Höhe Baum

$$f(h) = O(1) + f(h-1) \in O(h)$$



$[1, 2, 3, 4, 5]$ 

 $[5, 4, 3, 2, 1]$ 


Zufällige Verteilung  
der Prioritäten



$n$  Knoten im Baum  
 $h = \log(n)$



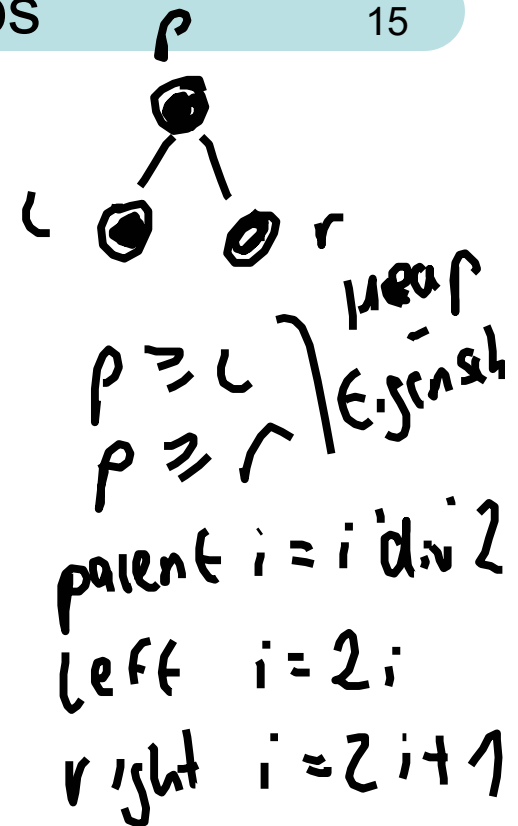
type PQueue a = ((Array Int (a,Int)), Int) *#Elemente im Heap*

create = (array (1,8) [], 0)  $O(1)$

front (a, size) | size >= 1 = fst (a ! 1)  $O(1)$

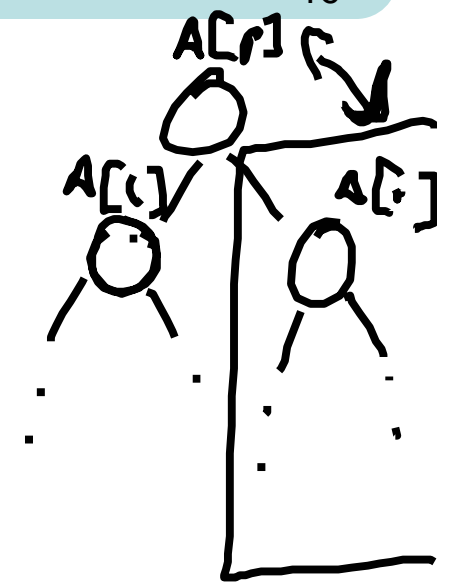
dequeue (a, size)  
 = heapify (newA, (size-1)) 1  
 where newA = a // [(1, a ! size)]  
 $a[1] = a[size]$   $O(1)$

$O(\log n)$



# Implementierung mit Heaps: heapify

```
heapify (a,size) i
| i /= max2 = heapify (newA,size) max2
| otherwise = (a,size)
where l = left i
      r = right i
      max1 = larger (a,size) i l
      max2 = larger (a,size) max1 r
      newA = a // [(i,a!max2),(max2,a!i)]
```

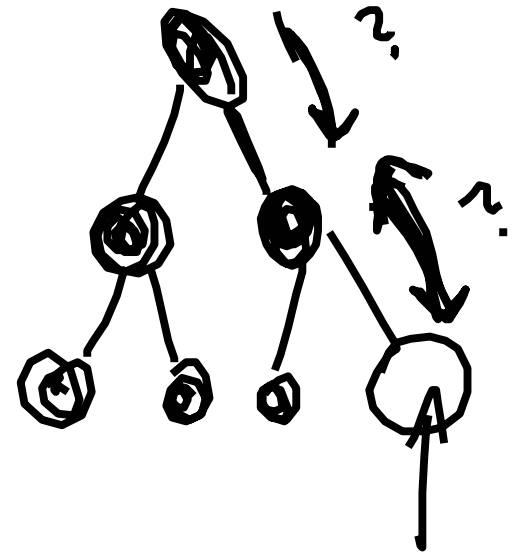


$max2 = j \in \{i, l, r\} : A[j] = \max.$

$$f(n) = O(1) + f(n/2) \in O(\log n)$$

```
larger (a,size) i x
| (x <= size) && (snd (a!x) > snd (a!i)) = x
| otherwise = i
```

$O(\log n)$ 
  
 enqueue (a, size) x
   
 = (grown // (updates sz), sz)
   
 where grown = grow a size  $O(1)$ 
  
 $O(1)$  sz = size + 1  $O(1)$ 
  
 updates = updates a sz x  $O(\log n)$



updates a i x
  
 |  $i > 1$  &&  $\text{snd } a_p < \text{snd } x = (i, a_p) : \text{updates } @ p x$ 
  
 | otherwise = (i, x) : []

where p = parent i )  $O(1)$ 
  
 ap = a ! p )  $O(1)$

$O(\log n)$ 
  
 wobei  $n = \text{size}$

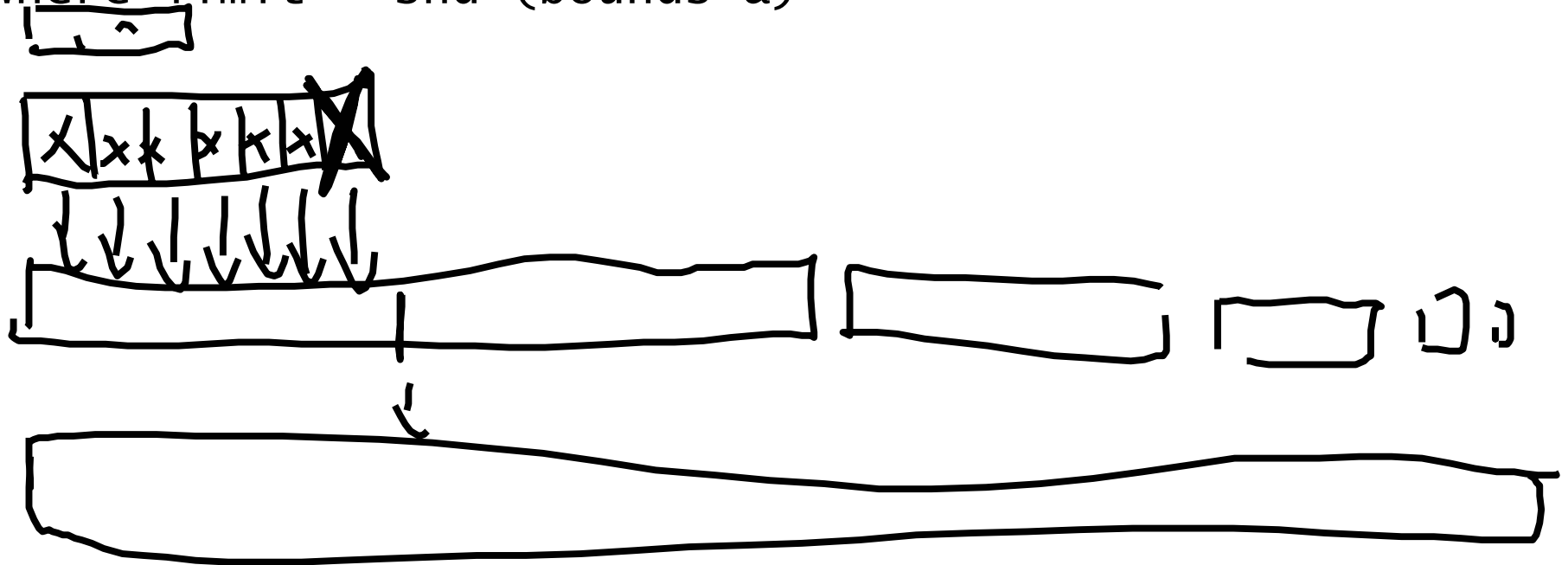


grow a size

| size < limit = a

| otherwise = listArray (1,2\*limit) (elems a)

where limit = snd (bounds a)



Kopieren ist amortisiert  $O(1)$



- Sortieren mit Heaps
  - Keine Unterscheidung zwischen Priorität und Nutzdaten mehr
- Erzeuge Heap aus bestehendem Array
  - 1 Aufruf an create  $O(1)$
  - n Aufrufe an enqueue  $n \cdot O(\log n)$
- Erzeuge sortiertes Ergebnis
  - n Aufrufe an dequeue  $n \cdot O(\log n)$
- Gesamtaufwand:  
 $O(n \cdot \log n)$

