6854 AAD

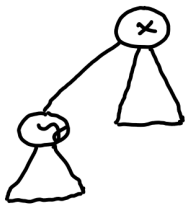
L06 Binomial Heaps

This Lecture: Ch 3.2

Binomial Trees, Heaps with higher arities.

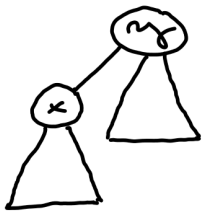
Link Operation

Case 1 $x \leq y$



Heap Property Satisfied

Case 2 $x > y$

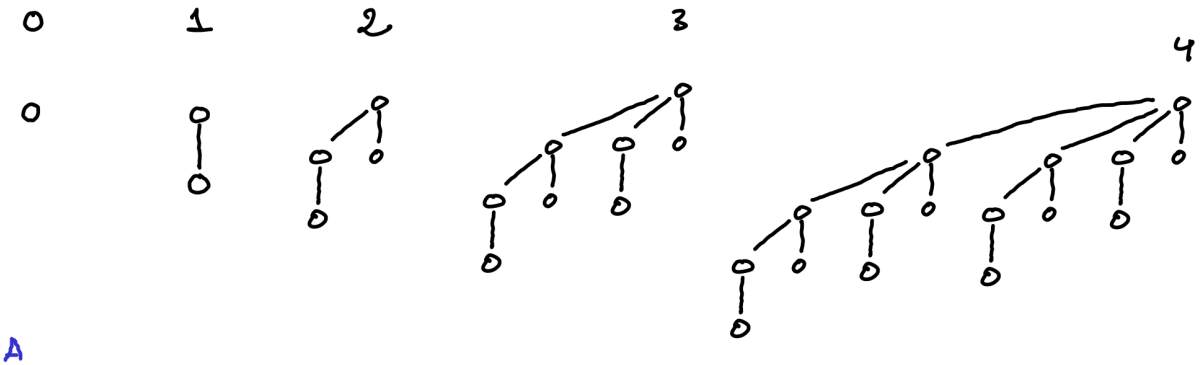


Heap Property Satisfied

$O(1)$ time!

Def: A Binomial tree of rank r is

- a singleton node if $r = 0$, and
- composed of two binomial trees of ranked $r-1$, where one tree is the leftmost child of the other tree.



LEMMA

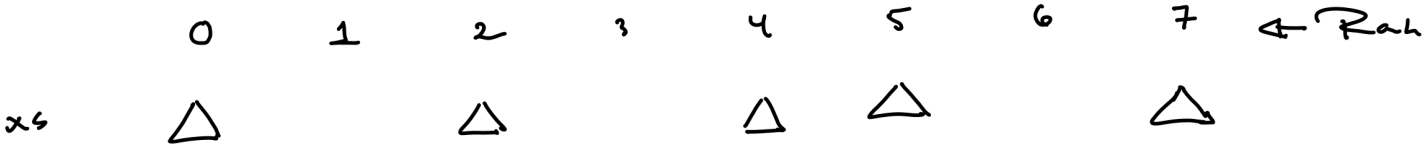
The tree of rank r has

- size 2^r
- depth r
- r children

Proof: By induction

Def A Binomial Heap is a collection of Binomial Trees s.t.

- each tree is a min heap
- for any $r \in \mathbb{N}$, there exists at most one tree of rank r



Lemma

In a Binomial Heap with n elements

- the largest rank is no more than $\log n$
- the largest depth d is no more than $\log n$

Proof

Clearly for any tree, including the largest, we have

$2^i \leq n$, so $i \leq \log n$, and

$d \leq \log n$.

Operations

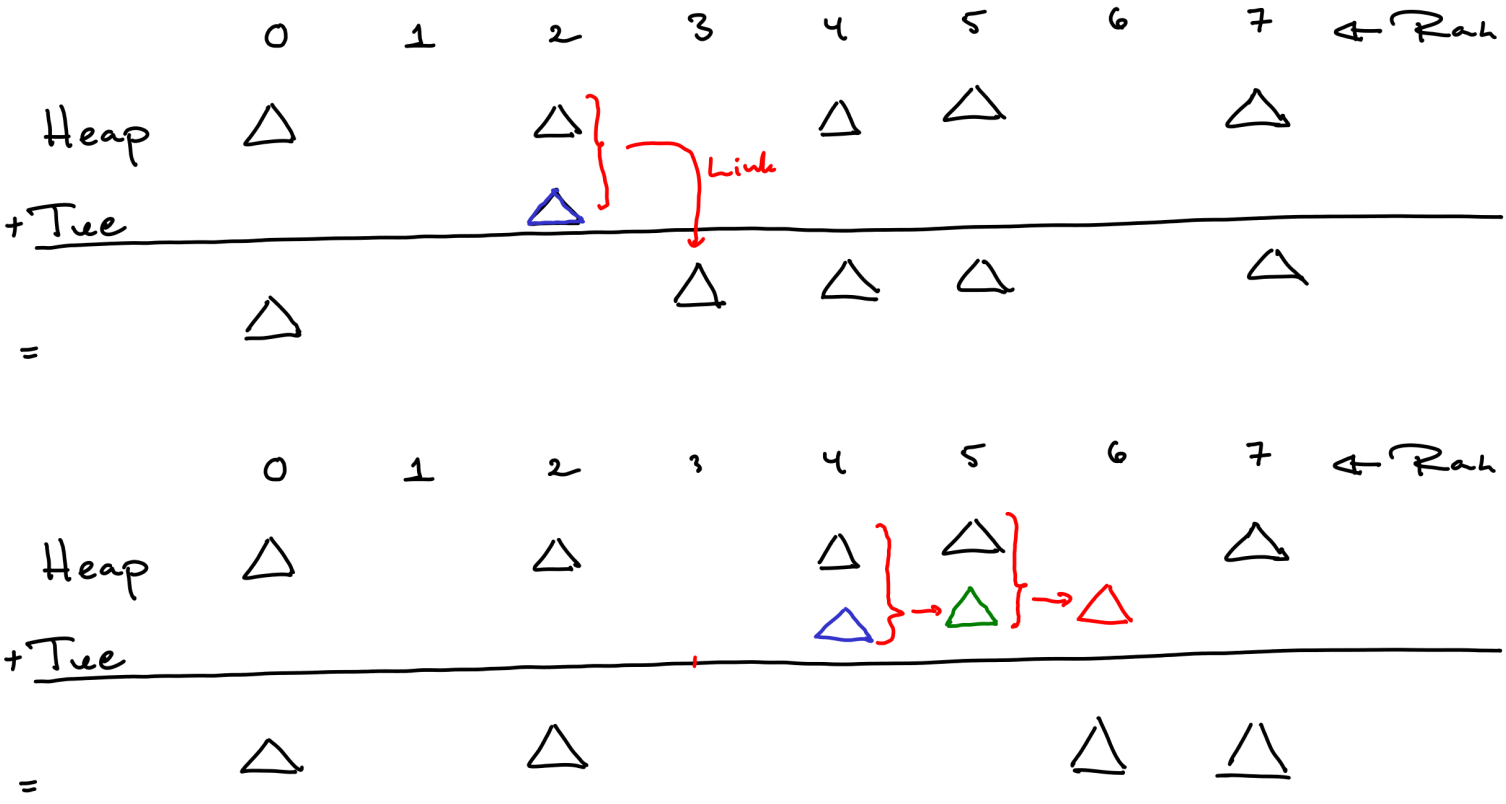
insTree – insert a new tree into the heap

merge – merge two heaps

findMin – find minimal value in heap

deleteMin – delete the minimal value from heap

insTree :: BinTree a → BinHeap a → BinHeap a



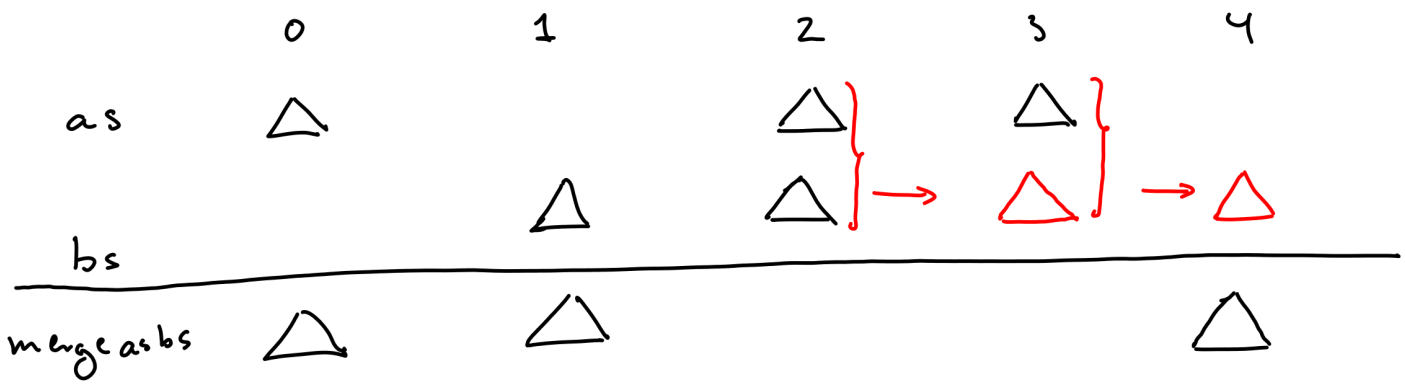

```
insTree t [] = t
insTree t (h:hs)
  | rank t < rank h  =  t : h : hs
  | rank t ≡ rank h  =  insTree (link t h) hs
  | rank t > rank h  =  h : (insTree t hs)
```

The runtime of insTree t hs when length hs = k is

$$T(k) = \begin{cases} O(1) & \text{if } k = 0 \\ T(k-1) + O(1) & \text{if } k > 0 \end{cases}$$

Runtime independent of tree depths!

merge :: BinHeap a → BinHeap a → BinHeap a



If length as ≤ k and length bs ≤ k, then

$$T(n) = O(k) = O(\log n)$$

```
merge bs [] = bs
merge [] bs = bs
merge (x:xs) (y:ys)
  | rank x < rank y  =  x : (merge xs (y:ys))
  | rank x > rank y  =  y : (merge (x:xs) ys)
  | rank x = rank y  =  insTree (join x y) (merge xs ys)
```

`removeMinTree` :: BinHeap a → (BinTree a, BinHeap a)

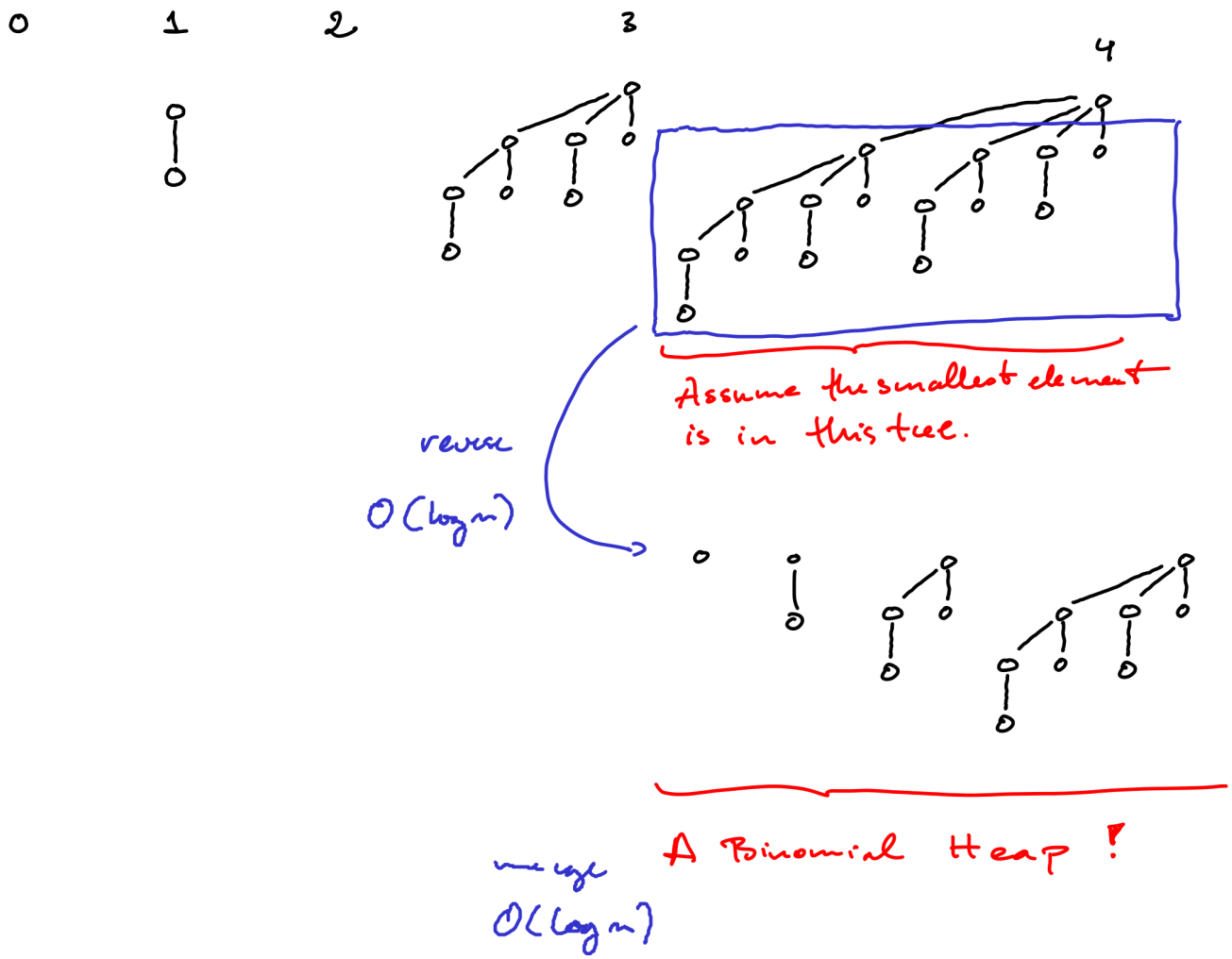
```
removeMinTree [t] = (t, [])
removeMinTree (t:ts) =
    if minTree t ≤ minTree x then (t, ts) else (x, t:xs)
    where (x, xs) = removeMinTree ts
```

$T(k) = T(k-1) + O(1)$
$T(1) = O(1)$
$\Big\} \; T(k) = O(k) = O(\log n)$

`findMin` :: BinHeap a → a

```
findMin ts = minTree t where (t, _) = removeMinTree ts.
```

`deleteMin` :: BinHeap a → BinHeap a




```
deleteMin hs =
    merge (reverse children) rest
    where (B _ _ children, rest) = removeMinTree hs
```

	Leftist Heaps	Binomial Heaps	
findMin	$O(1)$	$O(\log n)$	(*)
merge	$O(\log n)$	$O(\log n)$	
insert	$O(\log n)$	$O(\log n)$	(**)
delMin	$O(\log n)$	$O(\log n)$	

(*) Can be implemented in $O(1)$ time. Store min-value separately.

(**) Has $O(1)$ amortized time.