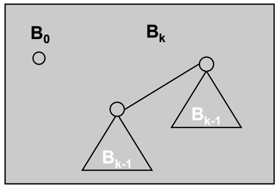
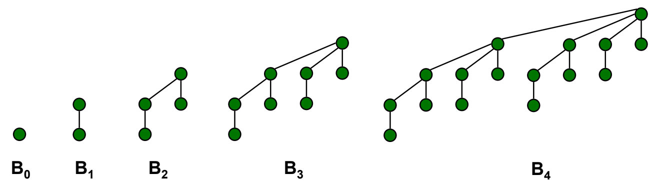
Binomial Tree

Adapted from: Kevin Wayne

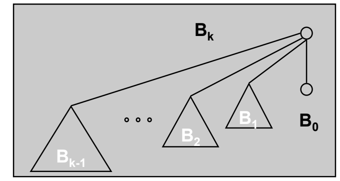
B_k is a binomial tree B_{k-1} with the addition of a left child with another binomial tree B_{k-1}



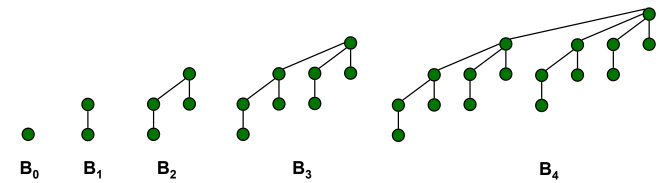


Binomial Tree

Number of nodes with respect to k?

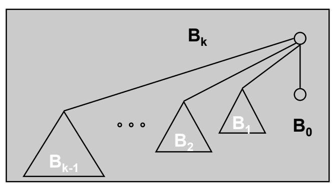


$N(B_o) = 1$
$N(B_k) = 2\ N(B_{k-1}) = 2^k$

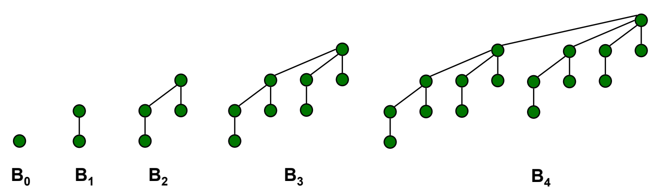


Binomial Tree

Height?

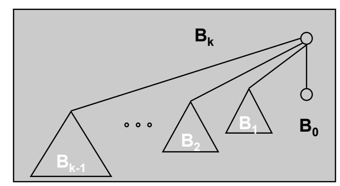


$H(B_o) = 1$
$H(B_k) = 1 + H(B_{k-1}) = k$

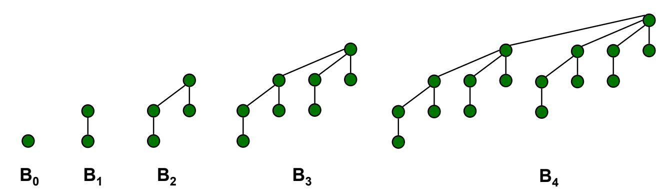


Binomial Tree

Degree of root node?

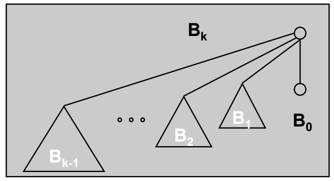


k, each time we add another binomial tree

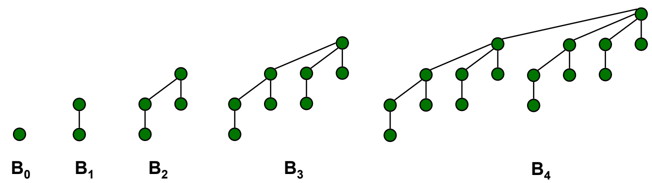


Binomial Tree

What are the children of the root?

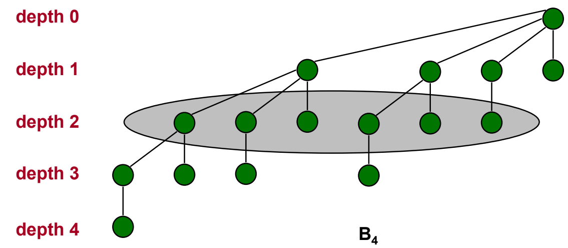


k-1 binomial trees:
B_{k-1}, B_{k-2}, …, B_0



Binomial Tree

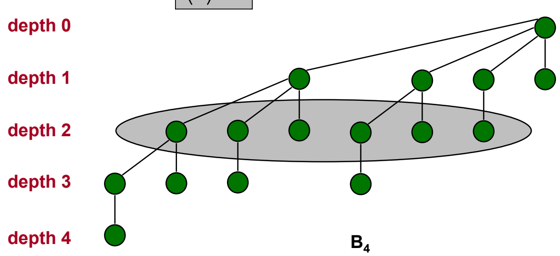
Why is it called a binomial tree?



Binomial Tree

B_k has $\binom{k}{i}$ nodes at depth i.

$\binom{4}{2} = 6$

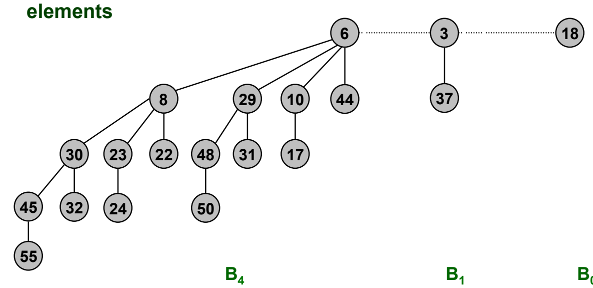


Binomial Heap

Binomial heap **Vuillemin, 1978.**

Sequence of binomial trees that satisfy binomial heap property:

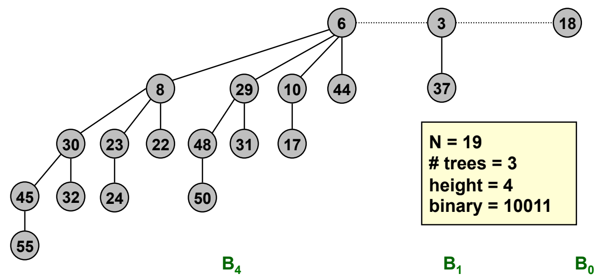
- **each tree is min-heap ordered**
- **top level: full or empty binomial tree of order k**
- **which are empty or full is based on the number of elements**



Binomial Heap

Like our "Kauchak"-set data structure from last time, except binomial tree heaps instead of arrays

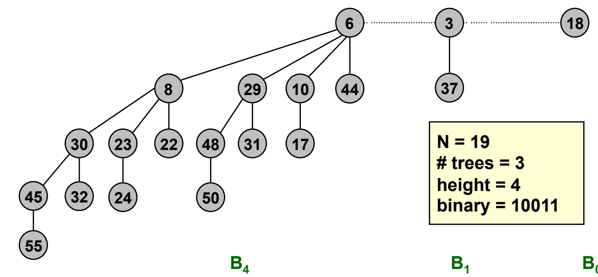
- A_0: [18]
- A_1: [3, 7]
- A_2: empty
- A_3: empty
- A_3: [6, 8, 29, 10, 44, 30, 23, 22, 48, 31, 17, 45, 32, 24, 55]



Binomial Heap: Properties

How many heaps?

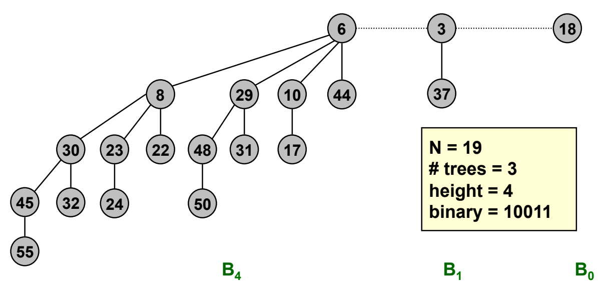
O(log n) – binary number representation



Binomial Heap: Properties

Where is the max/min?

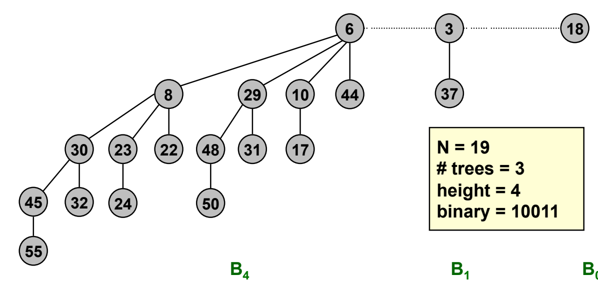
Must be one of the roots of the heaps



Binomial Heap: Properties

Runtime of max/min?

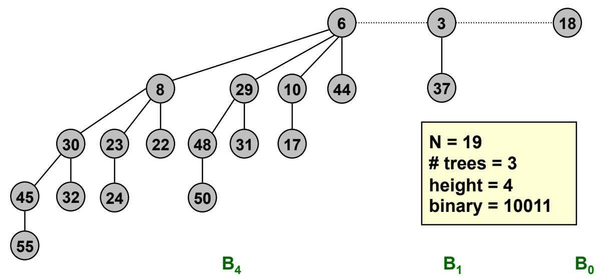
O(log n)



Binomial Heap: Properties

Height?

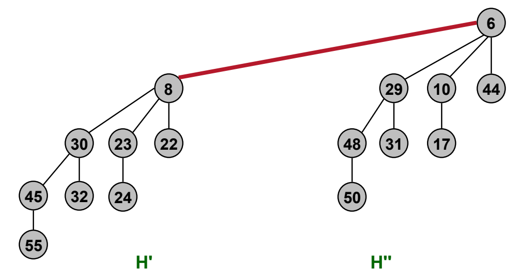
floor($\log_2$ n)
- largest tree = $B_{\log n}$
- height of that tree is log n



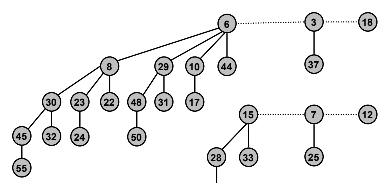
Binomial Heap: Union

How can we merge two binomial tree heaps of the same size (2^k)?
- connect roots of H' and H''
- choose smaller key to be root of H

Runtime? O(1)



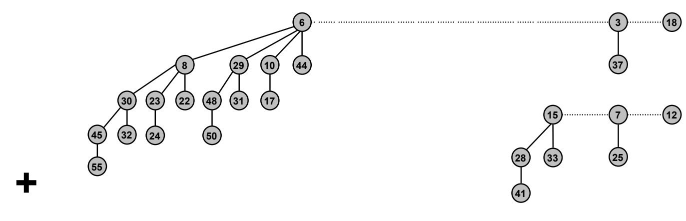
Binomial Heap: Union



What if they're not they're not the simple heaps of size 2^k?

Binomial Heap: Union

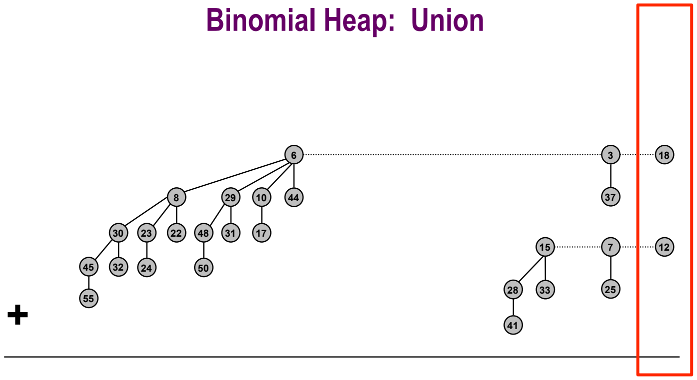
Go through each tree size starting at 0 and merge as we go



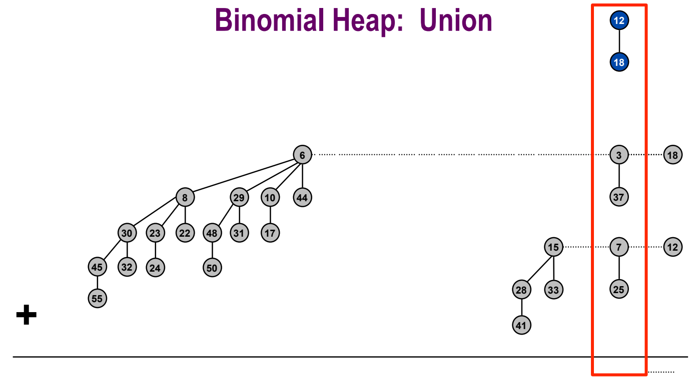
19 + 7 = 26

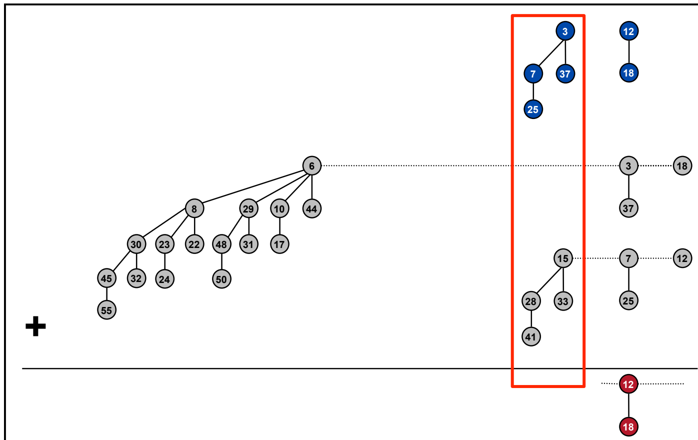
		1	1	1	
	1	0	0	1	1
+	0	0	1	1	1
	1	1	0	1	0

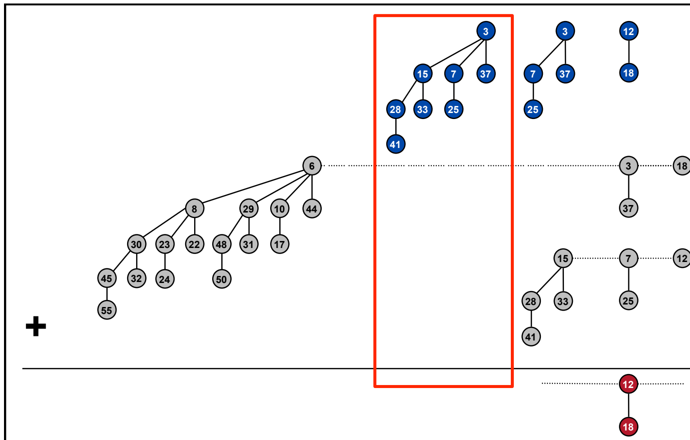
Binomial Heap: Union

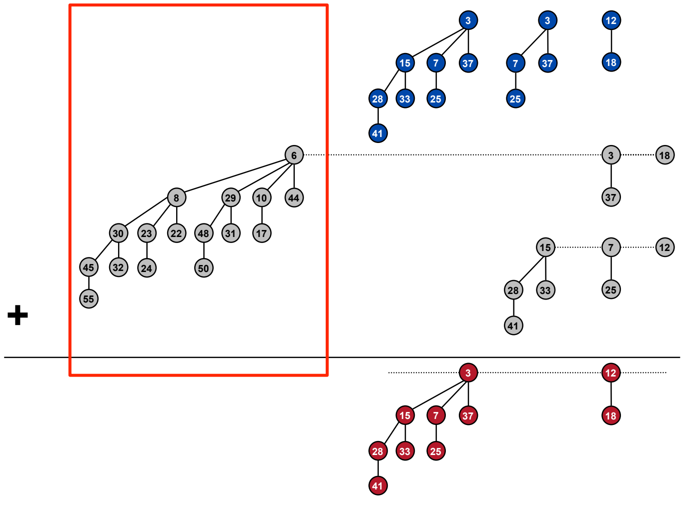


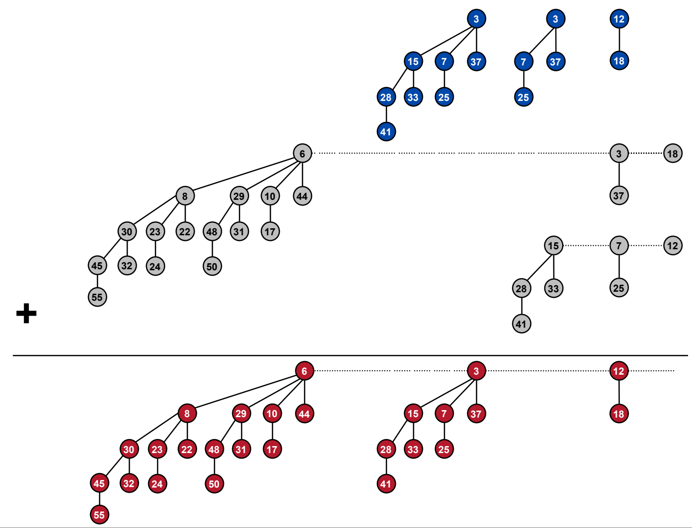
Binomial Heap: Union











Binomial Heap: Union

Analogous to binary addition

Running time?

- **Proportional to number of trees in root lists 2 $O(\log_2 N)$**
- **$O(\log N)$**

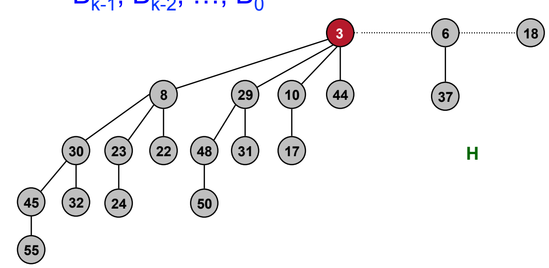
19 + 7 = 26

		1	1	1	
	1	0	0	1	1
+	0	0	1	1	1
	1	1	0	1	0

Binomial Heap: Delete Min/Max

We can find the min/max in $O(\log n)$.
How can we extract it?

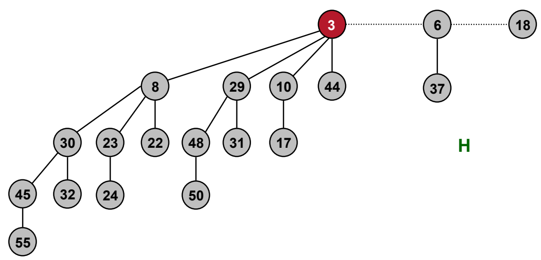
Hint: B_k consists of binomial trees:
$B_{k-1}, B_{k-2}, \ldots, B_0$



Binomial Heap: Delete Min

Delete node with minimum key in binomial heap H.

- Find root x with min key in root list of H, and delete
- H' ← broken binomial trees
- H ← Union(H', H)

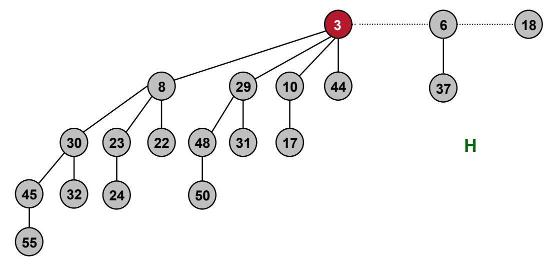


Binomial Heap: Delete Min

Delete node with minimum key in binomial heap H.

- Find root x with min key in root list of H, and delete
- H' ← broken binomial trees
- H ← Union(H', H)

Running time? O(log N)



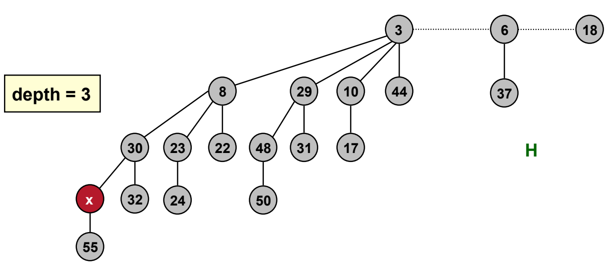
Binomial Heap: Decrease Key

Just call Decrease-Key/Increase-Key of Heap

- Suppose x is in binomial tree B_k
- Bubble node x up the tree if x is too small

Running time: O(log N)

- Proportional to depth of node x



Binomial Heap: Delete

Delete node x in binomial heap H

- Decrease key of x to -∞
- Delete min

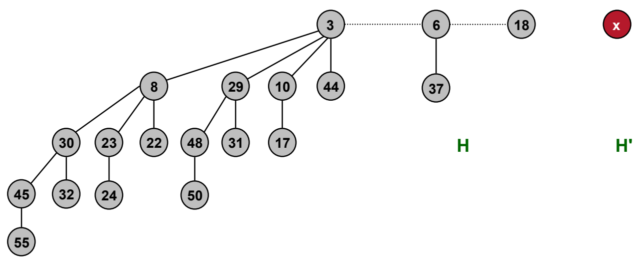
Running time: O(log N)

Binomial Heap: Insert

Insert a new node x into binomial heap H

- H' ← MakeHeap(x)
- H ← Union(H', H)

Running time. O(log N)



Build-Heap

Call insert n times

Runtime? O(n log n)

Can we get a tighter bound?

Build-Heap

Call insert n times

Consider inserting *n* numbers

	times	cost
how many times will B_0 be empty?		
how many times will we need to merge with B_0?		
how many times will we need to merge with B_1?		
how many times will we need to merge with B_2?		
…		
how many times will we need to merge with $B_{\log n}$?		

Build-Heap

Call insert n times

Consider inserting *n* numbers

	times	cost
how many times will B_0 be empty?	n/2	O(1)
how many times will we need to merge with B_0?	n/2	O(1)
how many times will we need to merge with B_1?	n/4	O(1)
how many times will we need to merge with B_2?	n/8	O(1)
…		
how many times will we need to merge with $B_{\log n}$?	1	O(1)

Runtime? Θ(n)

Heaps

Procedure	Binary heap (worst-case)	Binomial heap (worst-case)
BUILD-HEAP	$\Theta(n)$	$\Theta(n)$
INSERT	$\Theta(\log n)$	$O(\log n)$
MAXIMUM	$\Theta(1)$	$O(\log n)$
EXTRAC-MAX	$\Theta(\log n)$	$\Theta(\log n)$
UNION	$\Theta(n)$	$\Theta(\log n)$
INCREASE-ELEMENT	$\Theta(\log n)$	$\Theta(\log n)$
DELETE	$\Theta(\log n)$	$\Theta(\log n)$

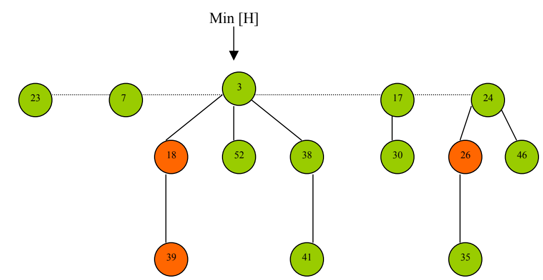
(adapted from Figure 19.1, pg. 456 [1])

Fibonacci Heaps

Similar to binomial heap
- **A Fibonacci heap consists of a sequence of heaps**

More flexible
- **Heaps do not have to be binomial trees**

More complicated ☺



Heaps

Procedure	Binary heap (worst-case)	Binomial heap (worst-case)	Fibonacci heap (amortized)
BUILD-HEAP	$\Theta(n)$	$\Theta(n)$	$\Theta(n)$
INSERT	$\Theta(\log n)$	$O(\log n)$	$\Theta(1)$
MAXIMUM	$\Theta(1)$	$O(\log n)$	$\Theta(1)$
EXTRAC-MAX	$\Theta(\log n)$	$\Theta(\log n)$	$O(\log n)$
UNION	$\Theta(n)$	$\Theta(\log n)$	$\Theta(1)$
INCREASE-ELEMENT	$\Theta(\log n)$	$\Theta(\log n)$	$\Theta(1)$
DELETE	$\Theta(\log n)$	$\Theta(\log n)$	$O(\log n)$

(adapted from Figure 19.1, pg. 456 [1])

Should you always use a Fibonacci heap?

Heaps

Procedure	Binary heap (worst-case)	Binomial heap (worst-case)	Fibonacci heap (amortized)
BUILD-HEAP	$\Theta(n)$	$\Theta(n)$	$\Theta(n)$
INSERT	$\Theta(\log n)$	$O(\log n)$	$\Theta(1)$
MAXIMUM	$\Theta(1)$	$O(\log n)$	$\Theta(1)$
EXTRAC-MAX	$\Theta(\log n)$	$\Theta(\log n)$	$O(\log n)$
UNION	$\Theta(n)$	$\Theta(\log n)$	$\Theta(1)$
INCREASE-ELEMENT	$\Theta(\log n)$	$\Theta(\log n)$	$\Theta(1)$
DELETE	$\Theta(\log n)$	$\Theta(\log n)$	$O(\log n)$

(adapted from Figure 19.1, pg. 456 [1])

- Extract-Max and Delete are O(n) worst case
- Constants can be large on some of the operations
- Complicated to implement

Heaps

Procedure	Binary heap (worst-case)	Binomial heap (worst-case)	Fibonacci heap (amortized)
BUILD-HEAP	$\Theta(n)$	$\Theta(n)$	$\Theta(n)$
INSERT	$\Theta(\log n)$	$O(\log n)$	$\Theta(1)$
MAXIMUM	$\Theta(1)$	$O(\log n)$	$\Theta(1)$
EXTRAC-MAX	$\Theta(\log n)$	$\Theta(\log n)$	$O(\log n)$
UNION	$\Theta(n)$	$\Theta(\log n)$	$\Theta(1)$
INCREASE-ELEMENT	$\Theta(\log n)$	$\Theta(\log n)$	$\Theta(1)$
DELETE	$\Theta(\log n)$	$\Theta(\log n)$	$O(\log n)$

(adapted from Figure 19.1, pg. 456 [1])

Can we do better?