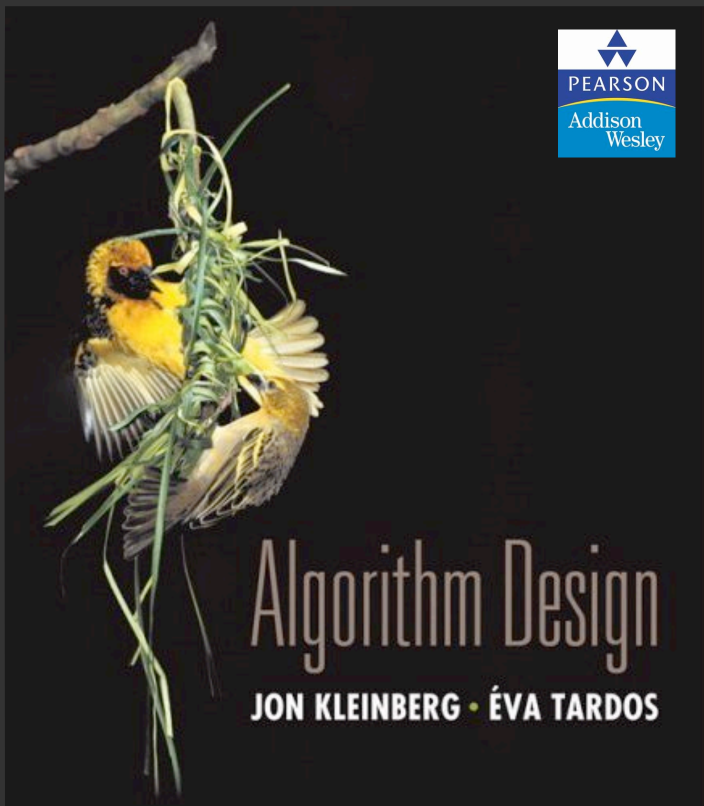


Priority Queues

- *binary heaps*
- *d-ary heaps*
- *binomial heaps*
- *Fibonacci heaps*

Lecture slides by Kevin Wayne

http://www.cs.princeton.edu/~wayne/kleinberg-tardos

Last updated on Apr 10, 2013 5:50 AM

Priority queue data type

A min-oriented priority queue supports the following core operations:

- MAKE-HEAP(): create an empty heap.
- INSERT(H, x): insert an element x into the heap.
- EXTRACT-MIN(H): remove and return an element with the smallest key.
- DECREASE-KEY(H, x, k): decrease the key of element x to k.

The following operations are also useful:

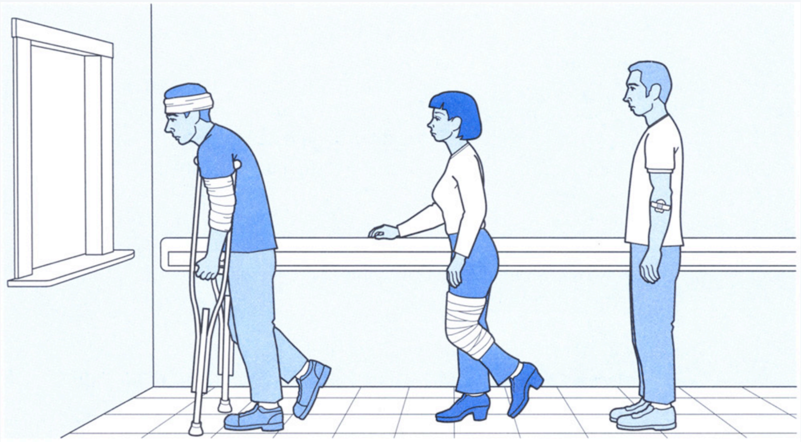
- IS-EMPTY(H): is the heap empty?
- FIND-MIN(H): return an element with smallest key.
- DELETE(H, x): delete element x from the heap.
- UNION(H_1, H_2): replace heaps H_1 and H_2 with their union.

Note. Each element contains a key (duplicate keys are permitted) from a totally-ordered universe.

Priority queue applications

Applications.

- A* search.
- Heapsort.
- Online median.
- Huffman encoding.
- Prim's MST algorithm.
- Discrete event-driven simulation.
- Network bandwidth management.
- Dijkstra's shortest-paths algorithm.
- ...



http://younginc.site11.com/source/5895/fos0092.html

SECTION 2.4

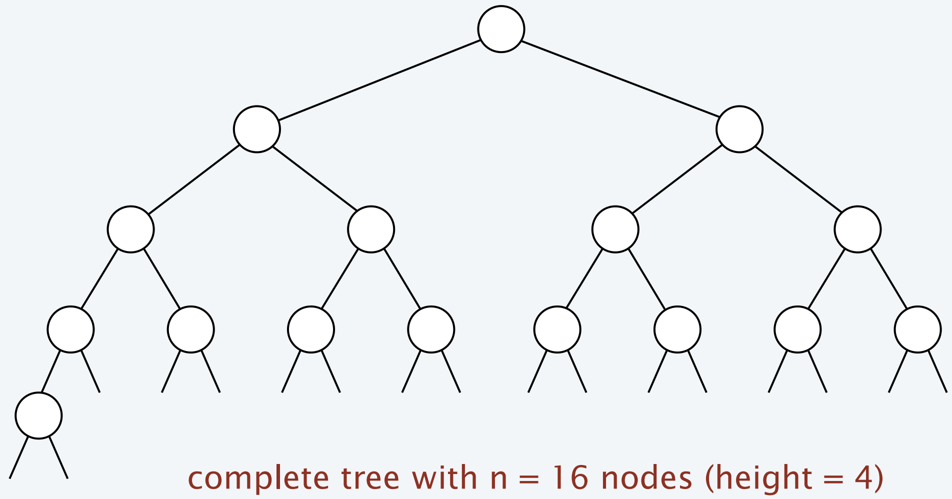
PRIORITY QUEUES

- *binary heaps*
- *d-ary heaps*
- *binomial heaps*
- *Fibonacci heaps*

Complete binary tree

Binary tree. Empty or node with links to two disjoint binary trees.

Complete tree. Perfectly balanced, except for bottom level.



complete tree with n = 16 nodes (height = 4)

Property. Height of complete binary tree with n nodes is $\lfloor \log_2 n \rfloor$.

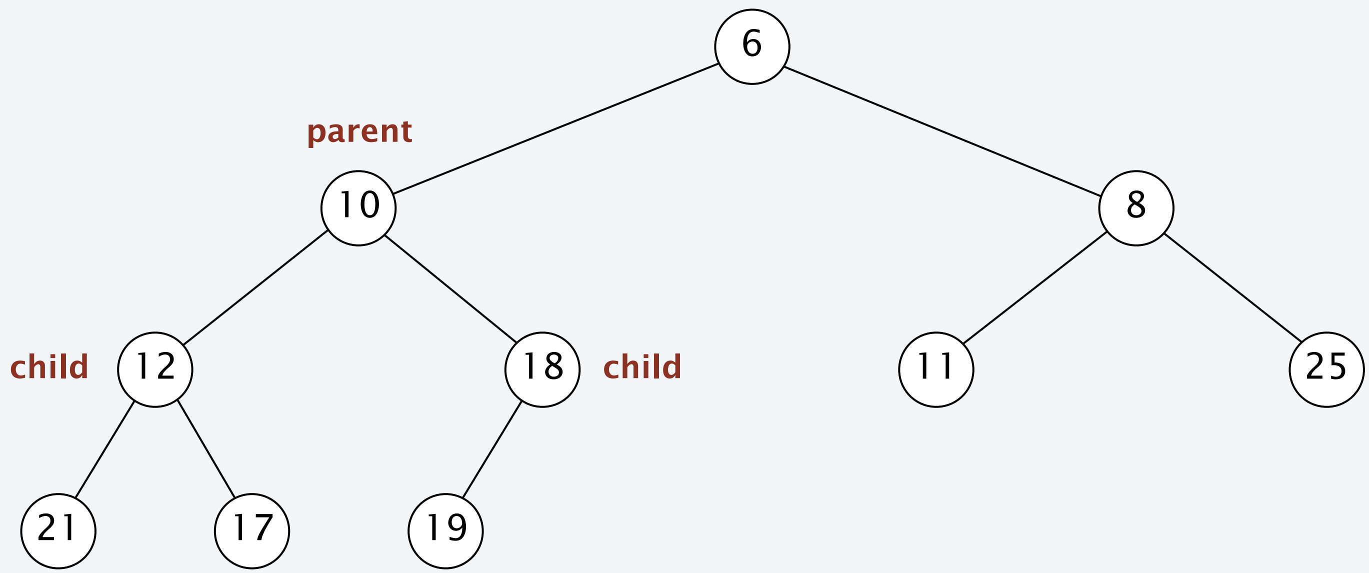
Pf. Height increases (by 1) only when n is a power of 2. ▪

A complete binary tree in nature

Binary heap

Binary heap. Heap-ordered complete binary tree.

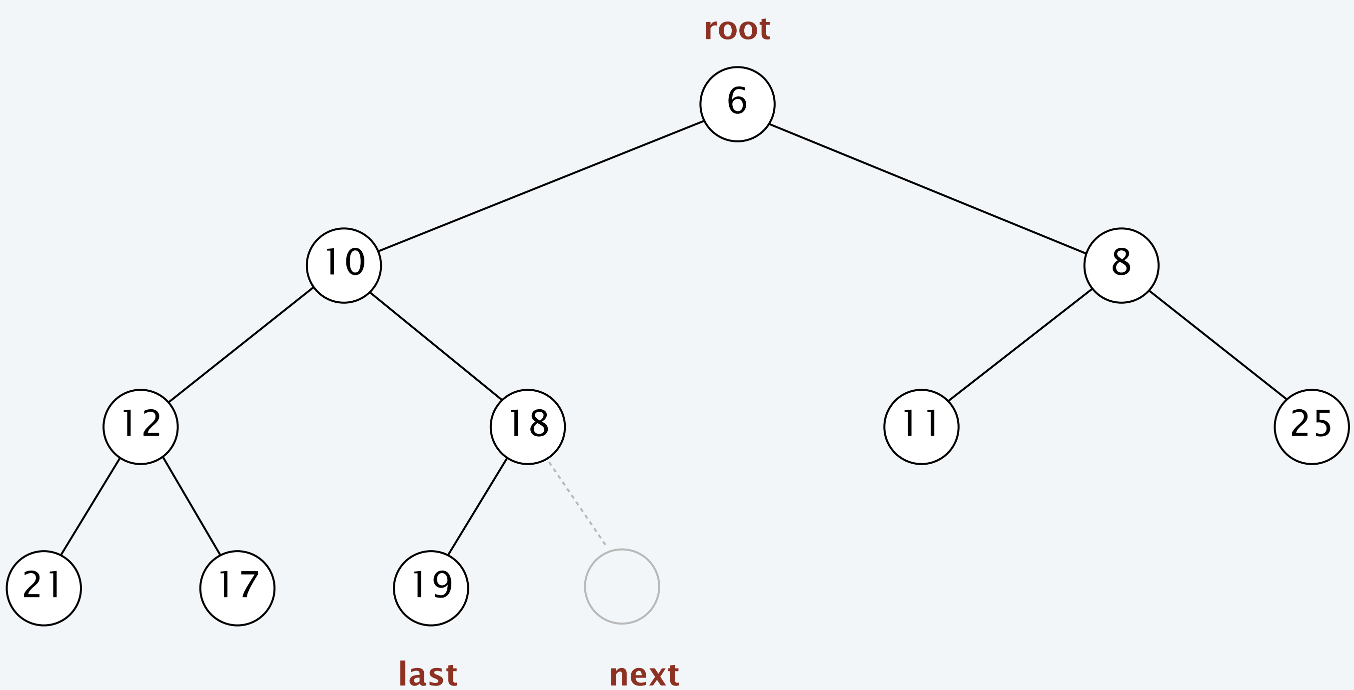
Heap-ordered. For each child, the key in child $\leq$ key in parent.



Explicit binary heap

Pointer representation. Each node has a pointer to parent and two children.

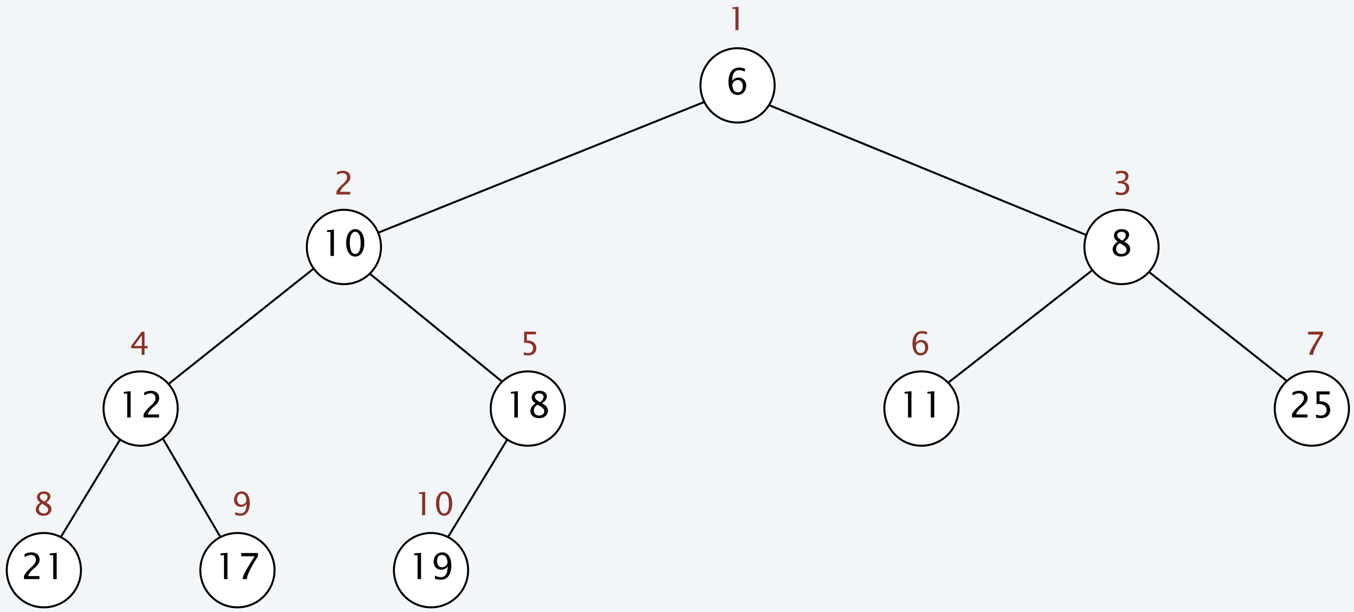
- Maintain number of elements n.
- Maintain pointer to root node.
- Can find pointer to last node or next node in $O(\log n)$ time.



Implicit binary heap

Array representation. Indices start at 1.

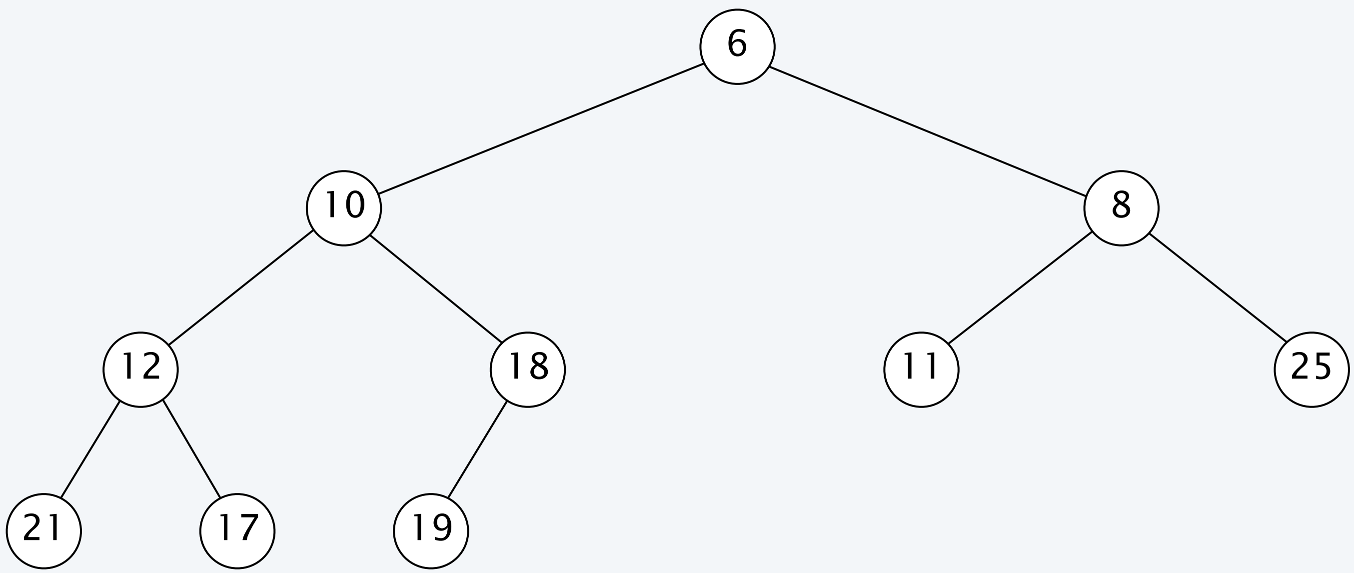
- Take nodes in level order.
- Parent of node at k is at $\lfloor k / 2 \rfloor$.
- Children of node at k are at $2k$ and $2k + 1$.



1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
6	10	8	12	18	11	25	21	17	19						

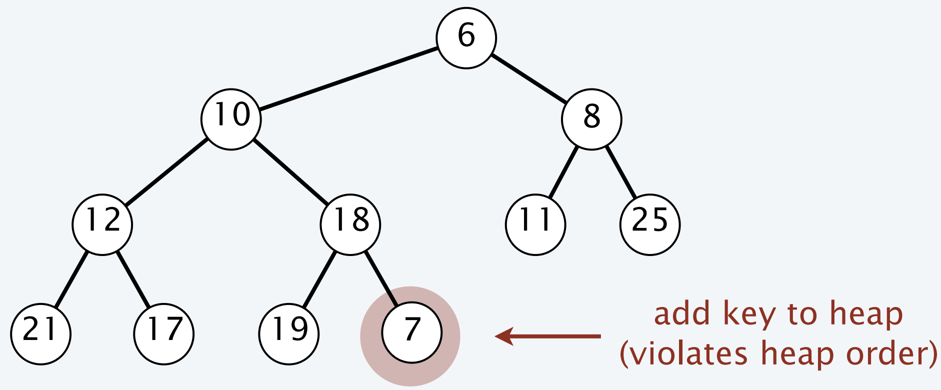
Binary heap demo

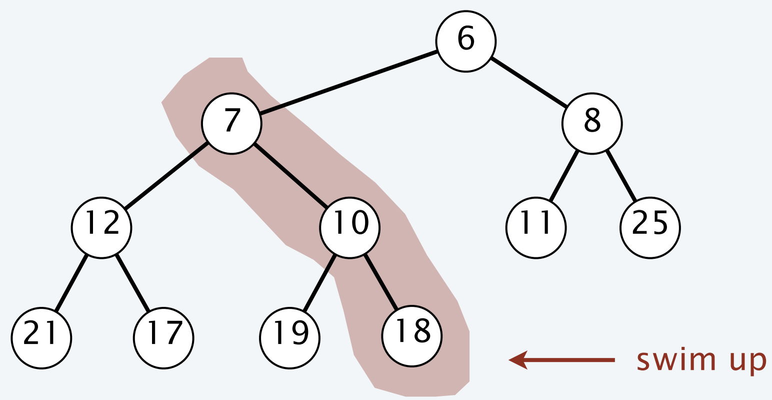
heap ordered



Binary heap: insert

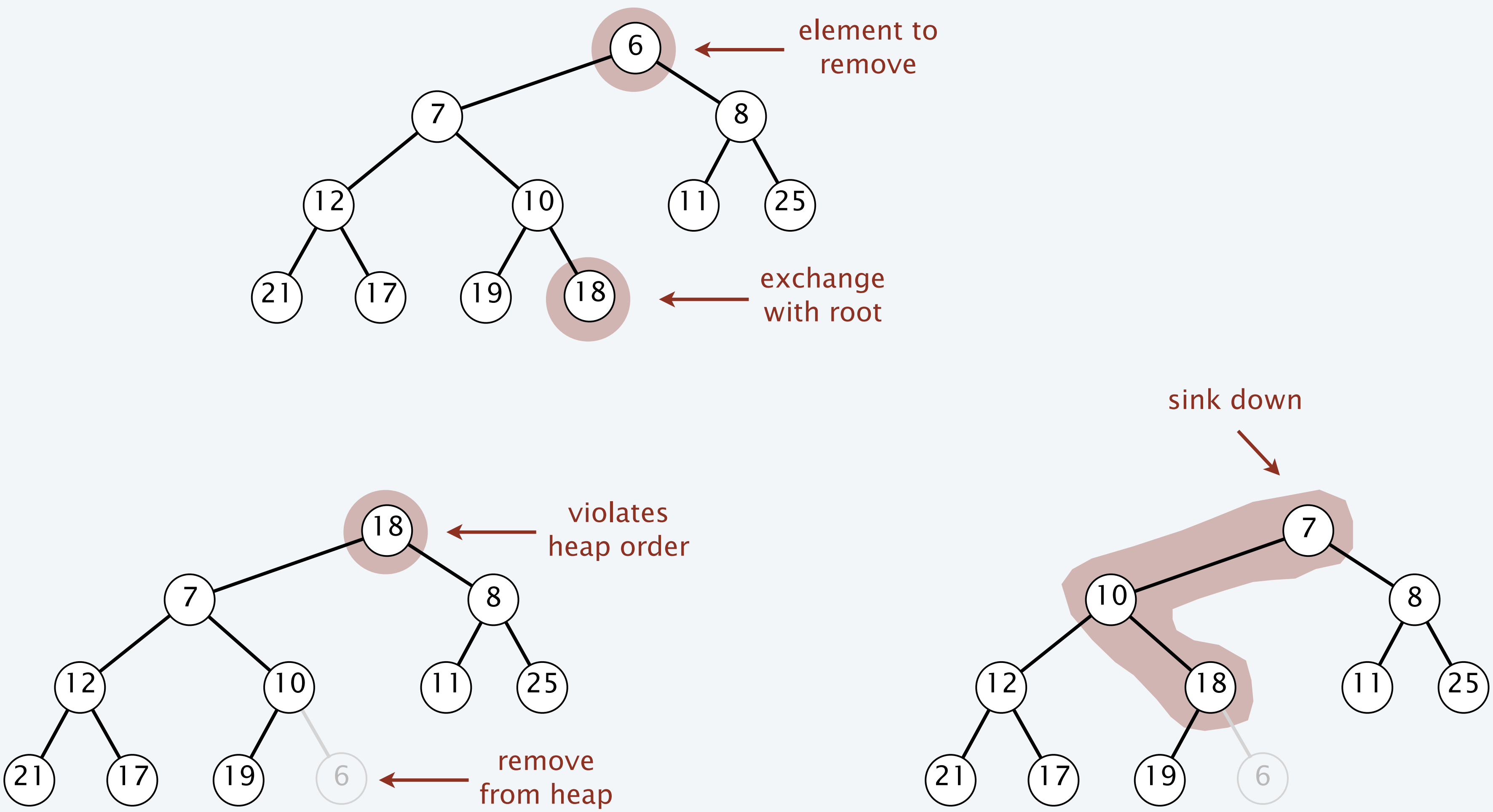
Insert. Add element in new node at end; repeatedly exchange new element with element in its parent until heap order is restored.





Binary heap: extract the minimum

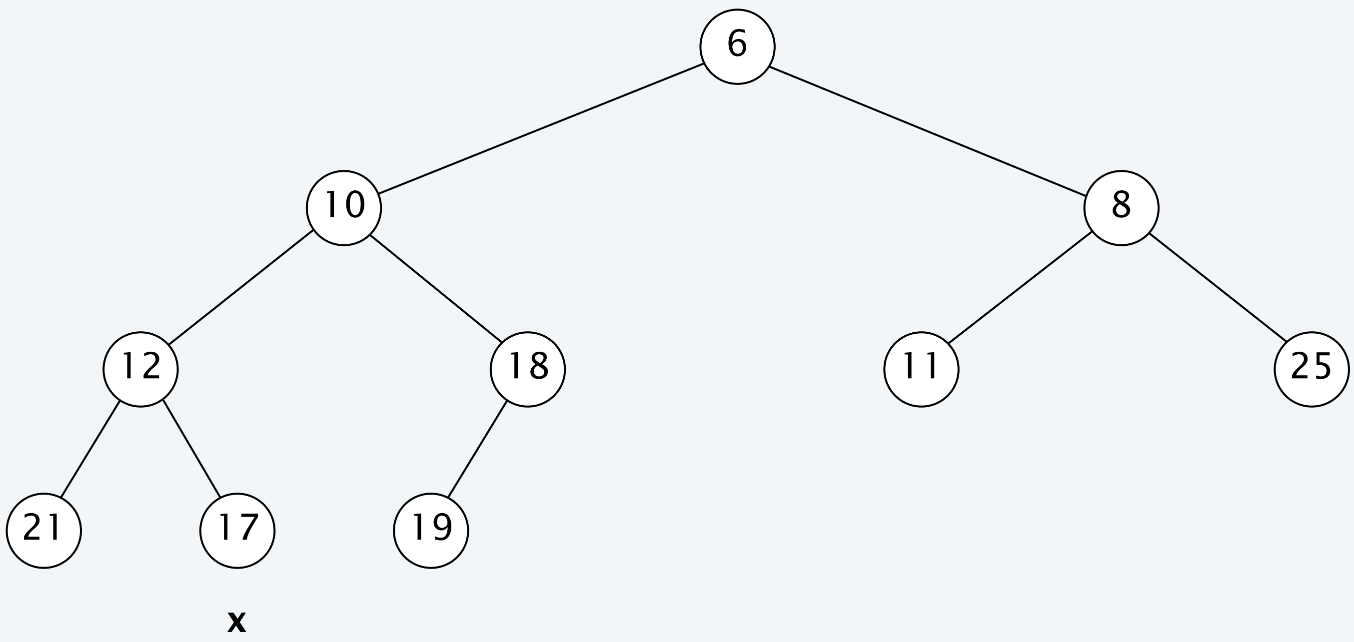
Extract min. Exchange element in root node with last node; repeatedly exchange element in root with its smaller child until heap order is restored.



Binary heap: decrease key

Decrease key. Given a handle to node, repeatedly exchange element with its parent until heap order is restored.

decrease key of node x to 11



Binary heap: analysis

Theorem. In an implicit binary heap, any sequence of m INSERT, EXTRACT-MIN, and DECREASE-KEY operations with n INSERT operations takes $O(m \log n)$ time.

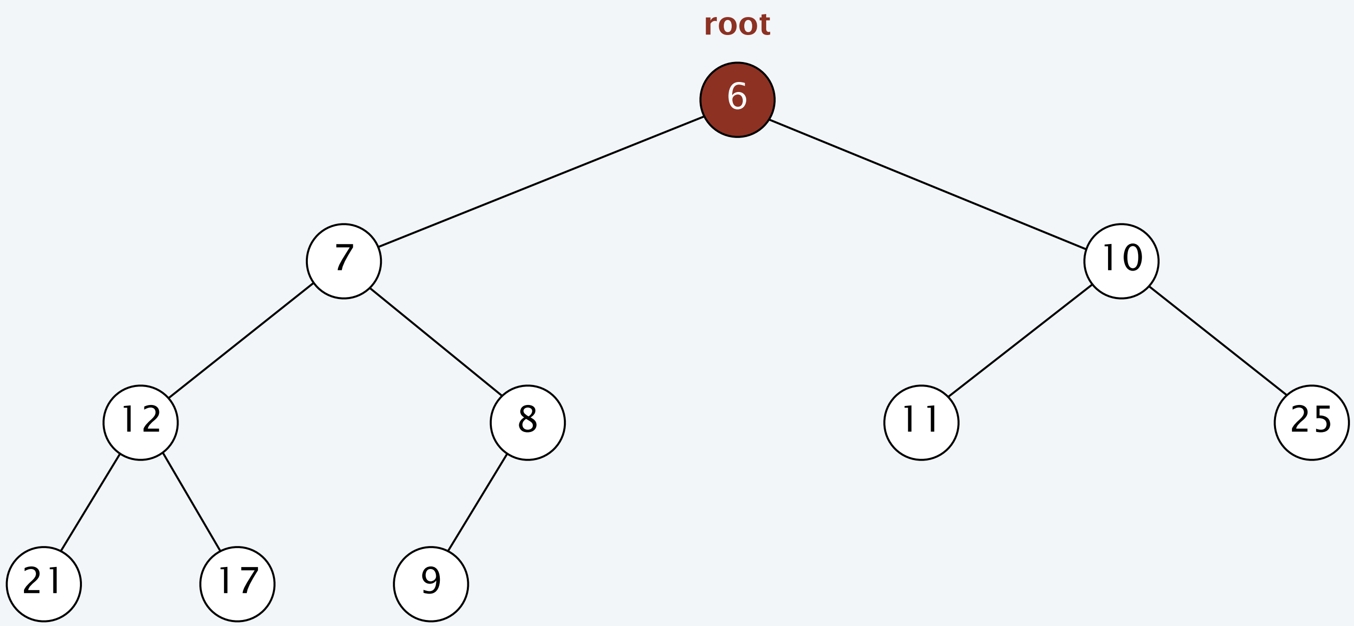
Pf.

- Each heap op touches nodes only on a path from the root to a leaf; the height of the tree is at most $\log_2 n$.
- The total cost of expanding and contracting the arrays is $O(n)$. ▪

Theorem. In an explicit binary heap with n nodes, the operations INSERT, DECREASE-KEY, and EXTRACT-MIN take $O(\log n)$ time in the worst case.

Binary heap: find-min

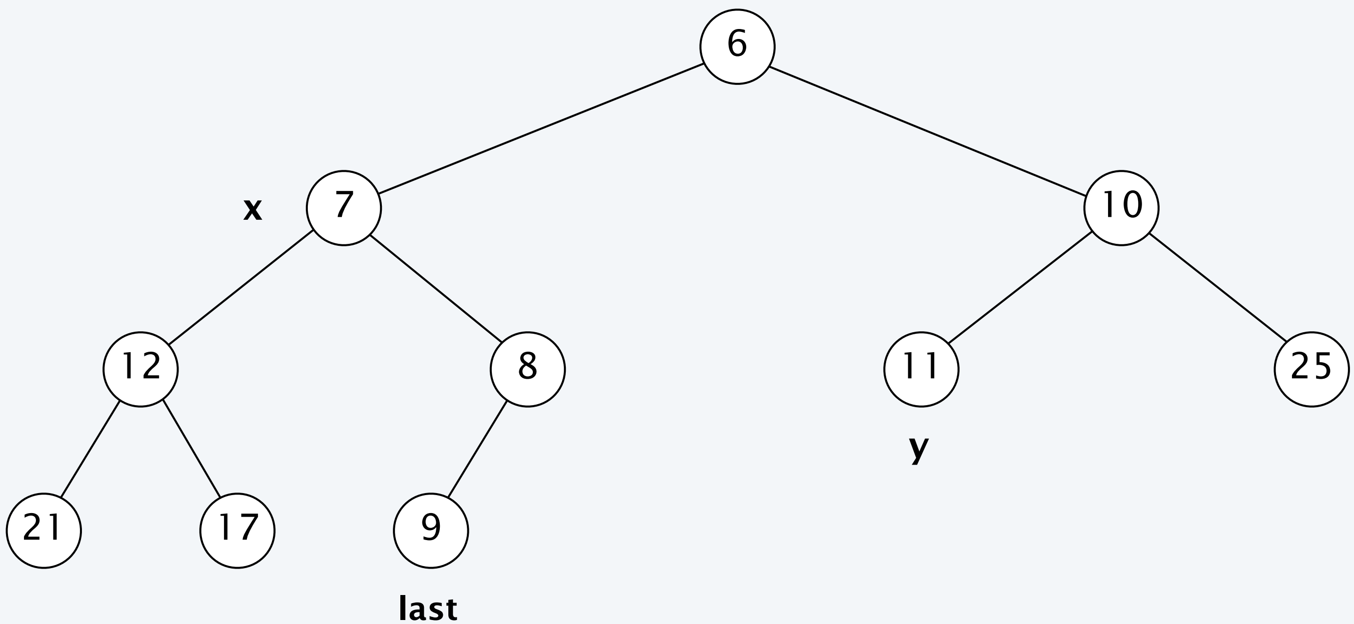
Find the minimum. Return element in the root node.



Binary heap: delete

Delete. Given a handle to a node, exchange element in node with last node; either swim down or sink up the node until heap order is restored.

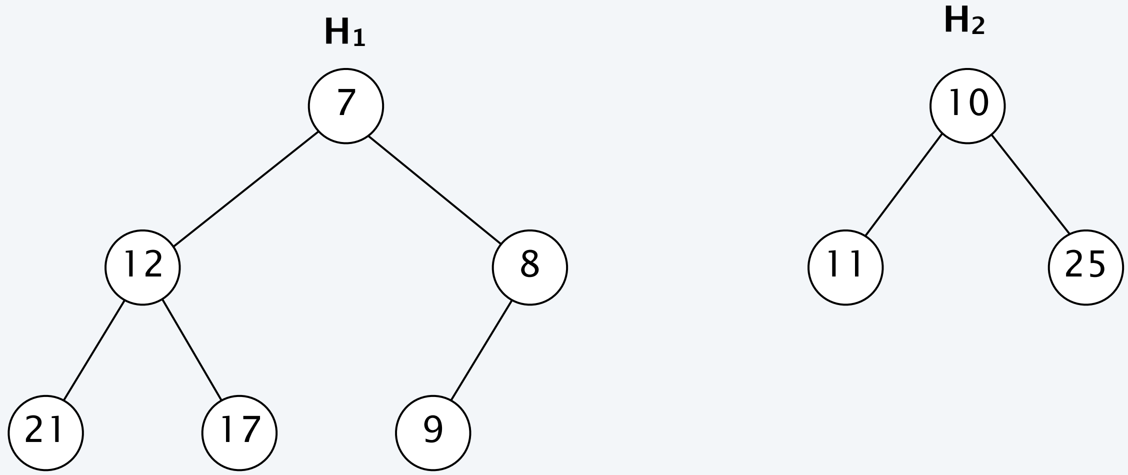
delete node x or y



Binary heap: union

Union. Given two binary heaps H_1 and H_2, merge into a single binary heap.

Observation. No easy solution: $\Omega(n)$ time apparently required.

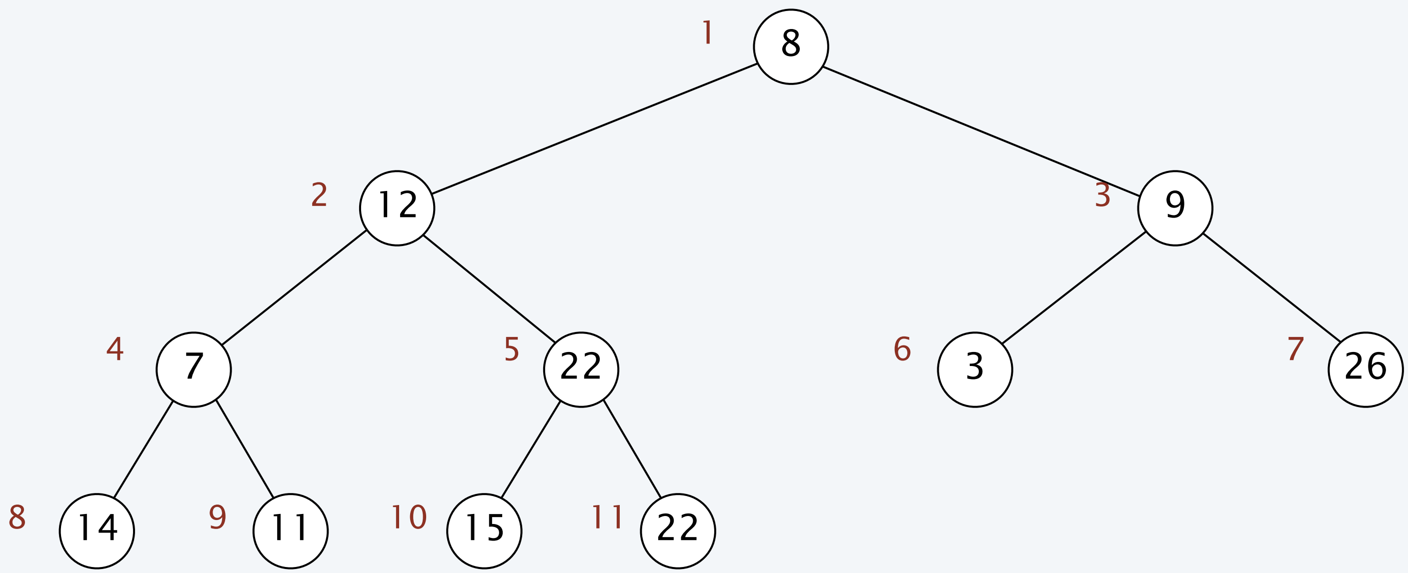


Binary heap: heapify

Heapify. Given n elements, construct a binary heap containing them.

Observation. Can do in $O(n \log n)$ time by inserting each element.

Bottom-up method. For $i = n$ to 1, repeatedly exchange the element in node i with its smaller child until subtree rooted at i is heap-ordered.



8	12	9	7	22	3	26	14	11	15	22
1	2	3	4	5	6	7	8	9	10	11

Binary heap: heapify

Theorem. Given n elements, can construct a binary heap containing those n elements in $O(n)$ time.

Pf.

- There are at most $\lceil n / 2^{h+1} \rceil$ nodes of height h.
- The amount of work to sink a node is proportional to its height h.
- Thus, the total work is bounded by:

$$\sum_{h=0}^{\lfloor \log_2 n \rfloor} \lceil n / 2^{h+1} \rceil \, h \;\leq\; \sum_{h=0}^{\lfloor \log_2 n \rfloor} n\, h / 2^h$$

$$\leq \; 2n \;\blacksquare$$

$$\sum_{i=1}^{k} \frac{i}{2^i} = 2 - \frac{k}{2^k} - \frac{1}{2^{k-1}}$$

$$\leq 2$$

Corollary. Given two binary heaps H_1 and H_2 containing n elements in total, can implement UNION in $O(n)$ time.

Priority queues performance cost summary

operation	linked list	binary heap
MAKE-HEAP	$O(1)$	$O(1)$
ISEMPTY	$O(1)$	$O(1)$
INSERT	$O(1)$	$O(\log n)$
EXTRACT-MIN	$O(n)$	$O(\log n)$
DECREASE-KEY	$O(1)$	$O(\log n)$
DELETE	$O(1)$	$O(\log n)$
UNION	$O(1)$	$O(n)$
FIND-MIN	$O(n)$	$O(1)$

Priority queues performance cost summary

Q. Reanalyze so that EXTRACT-MIN and DELETE take $O(1)$ amortized time?

operation	linked list	binary heap	binary heap †
MAKE-HEAP	$O(1)$	$O(1)$	$O(1)$
ISEMPTY	$O(1)$	$O(1)$	$O(1)$
INSERT	$O(1)$	$O(\log n)$	$O(\log n)$
EXTRACT-MIN	$O(n)$	$O(\log n)$	$O(1)$ †
DECREASE-KEY	$O(1)$	$O(\log n)$	$O(\log n)$
DELETE	$O(1)$	$O(\log n)$	$O(1)$ †
UNION	$O(1)$	$O(n)$	$O(n)$
FIND-MIN	$O(n)$	$O(1)$	$O(1)$

† amortized

SECTION 2.4

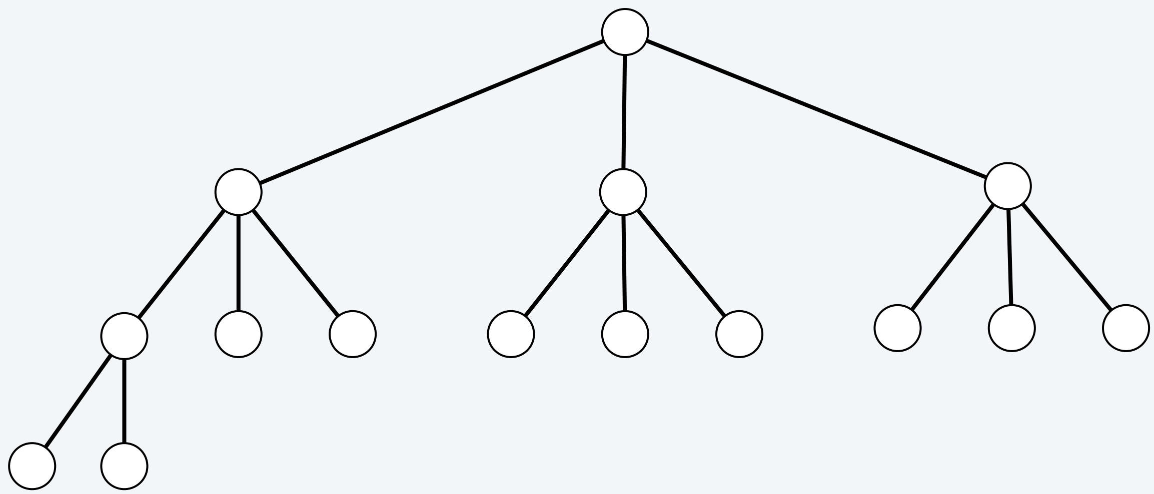
PRIORITY QUEUES

- *binary heaps*
- *d-ary heaps*
- *binomial heaps*
- *Fibonacci heaps*

Complete d-ary tree

Binary tree. Empty or node with links to d disjoint d-ary trees.

Complete tree. Perfectly balanced, except for bottom level.

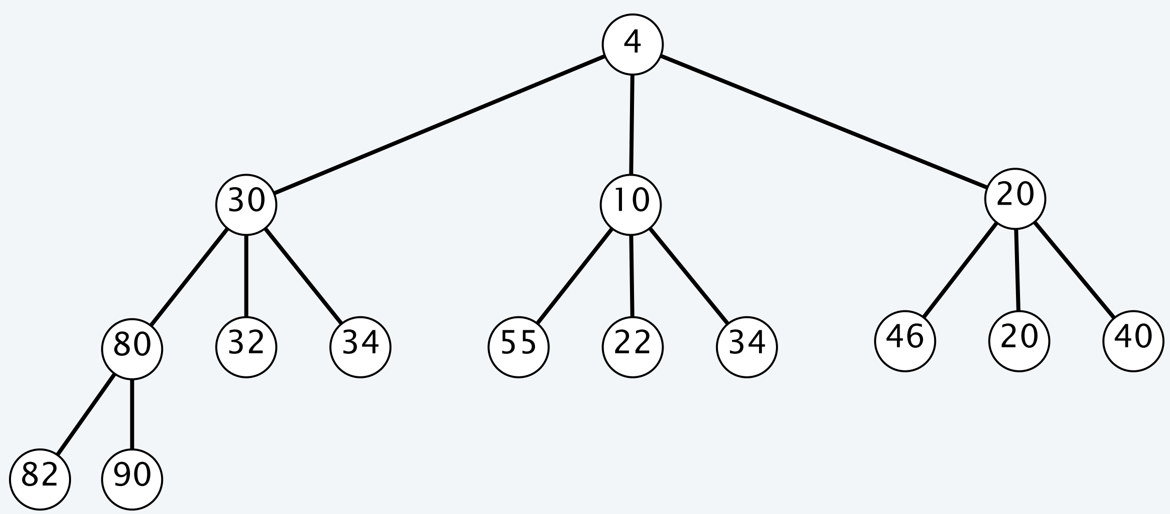


Fact. The height of a complete d-ary tree with n nodes is ≤ $\lceil \log_d n \rceil$.

Multiway heap: insert

Insert. Add node at end; repeatedly exchange element in child with element in parent until heap order is restored.

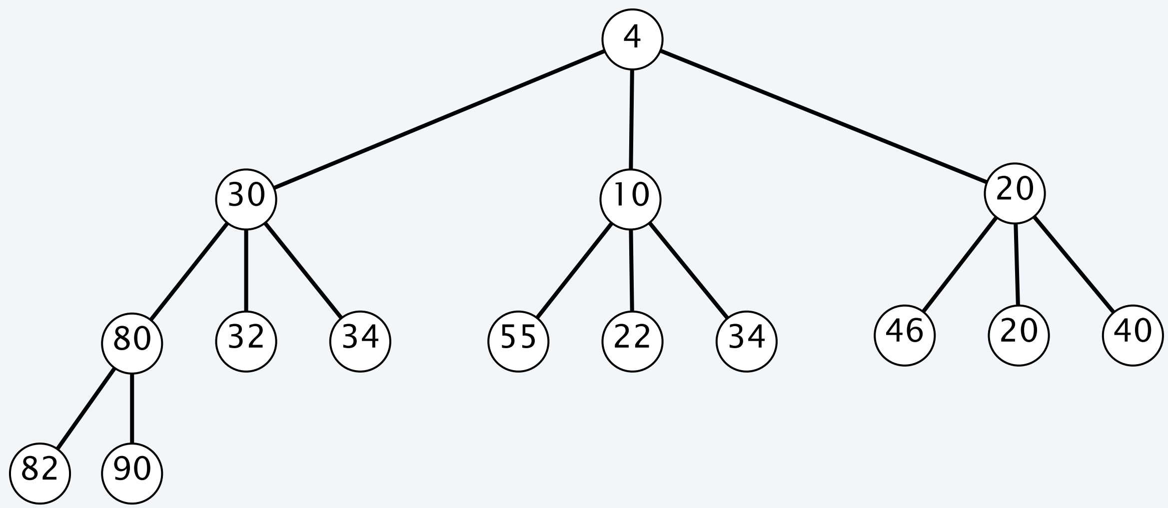
Running time. Proportional to height = $O(\log_d n)$.



Multiway heap: extract the minimum

Extract min. Exchange root node with last node; repeatedly exchange element in parent with element in largest child until heap order is restored.

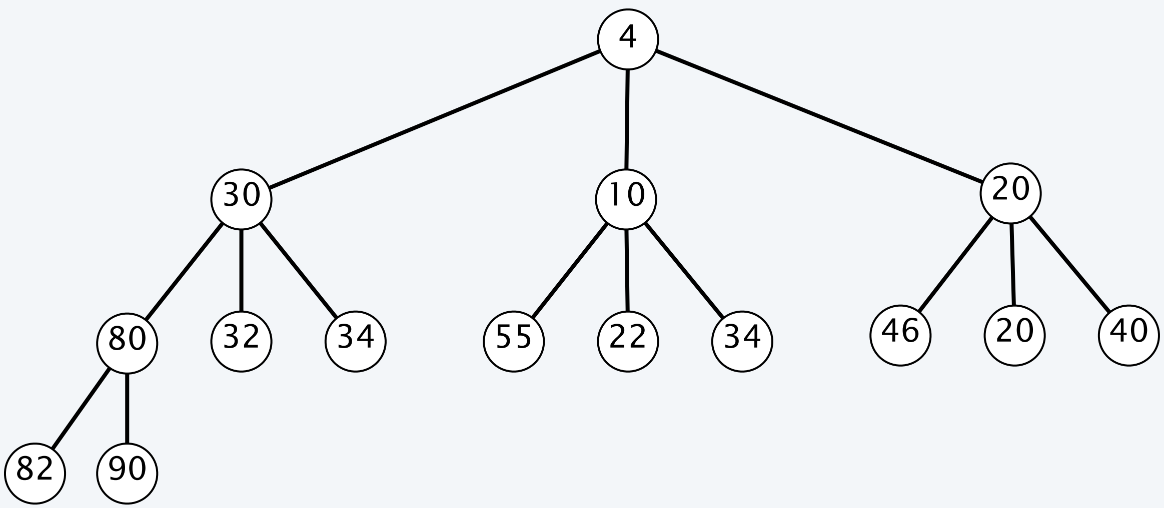
Running time. Proportional to $d \times$ height $= O(d \log_d n)$.



Multiway heap: decrease key

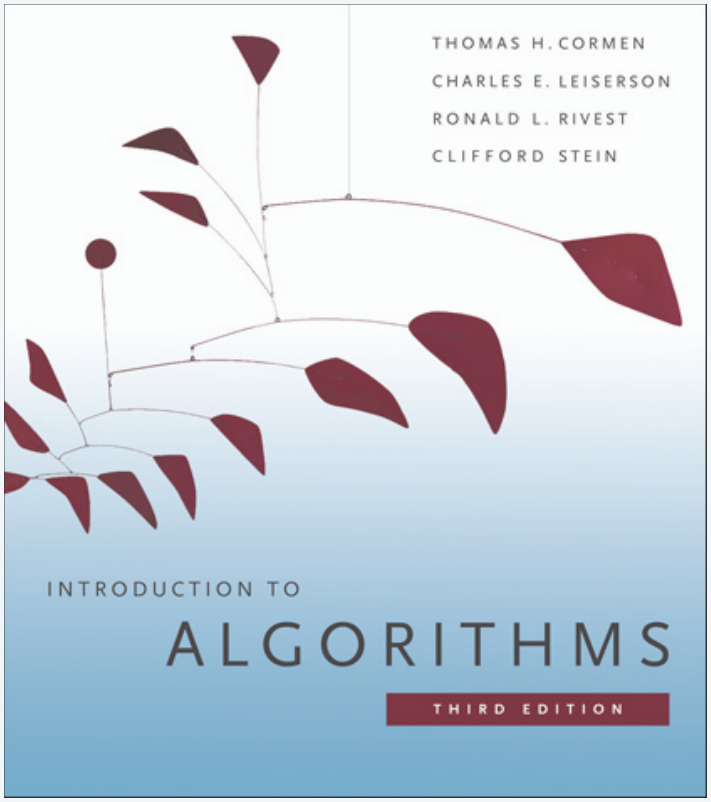
Decrease key. Given a handle to an element x, repeatedly exchange it with its parent until heap order is restored.

Running time. Proportional to height = $O(\log_d n)$.



Priority queues performance cost summary

operation	linked list	binary heap	d-ary heap
MAKE-HEAP	$O(1)$	$O(1)$	$O(1)$
ISEMPTY	$O(1)$	$O(1)$	$O(1)$
INSERT	$O(1)$	$O(\log n)$	$O(\log_d n)$
EXTRACT-MIN	$O(n)$	$O(\log n)$	$O(d \log_d n)$
DECREASE-KEY	$O(1)$	$O(\log n)$	$O(\log_d n)$
DELETE	$O(1)$	$O(\log n)$	$O(d \log_d n)$
UNION	$O(1)$	$O(n)$	$O(n)$
FIND-MIN	$O(n)$	$O(1)$	$O(1)$



CHAPTER 6 (2ND EDITION)

PRIORITY QUEUES

- *binary heaps*
- *d-ary heaps*
- *binomial heaps*
- *Fibonacci heaps*

Priority queues performance cost summary

operation	linked list	binary heap	d-ary heap
MAKE-HEAP	$O(1)$	$O(1)$	$O(1)$
ISEMPTY	$O(1)$	$O(1)$	$O(1)$
INSERT	$O(1)$	$O(\log n)$	$O(\log_d n)$
EXTRACT-MIN	$O(n)$	$O(\log n)$	$O(d \log_d n)$
DECREASE-KEY	$O(1)$	$O(\log n)$	$O(\log_d n)$
DELETE	$O(1)$	$O(\log n)$	$O(d \log_d n)$
UNION	$O(1)$	$O(n)$	$O(n)$
FIND-MIN	$O(n)$	$O(1)$	$O(1)$

Goal. $O(\log n)$ INSERT, DECREASE-KEY, EXTRACT-MIN, and UNION.

mergeable heap

Binomial heaps

Programming Techniques

S.L. Graham, R.L. Rivest
Editors

A Data Structure for Manipulating Priority Queues

Jean Vuillemin
Université de Paris-Sud

A data structure is described which can be used for representing a collection of priority queues. The primitive operations are insertion, deletion, union, update, and search for an item of earliest priority.

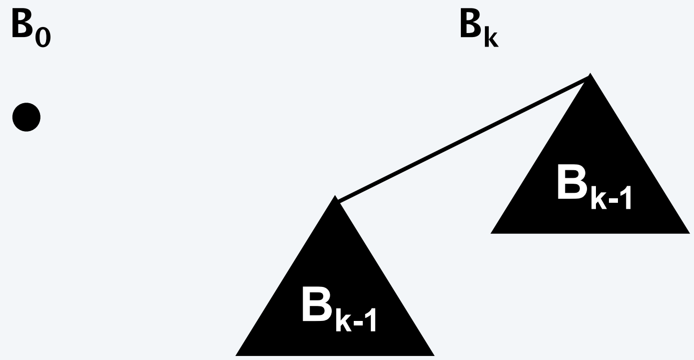
Key Words and Phrases: data structures, implementation of set operations, priority queues, mergeable heaps, binary trees

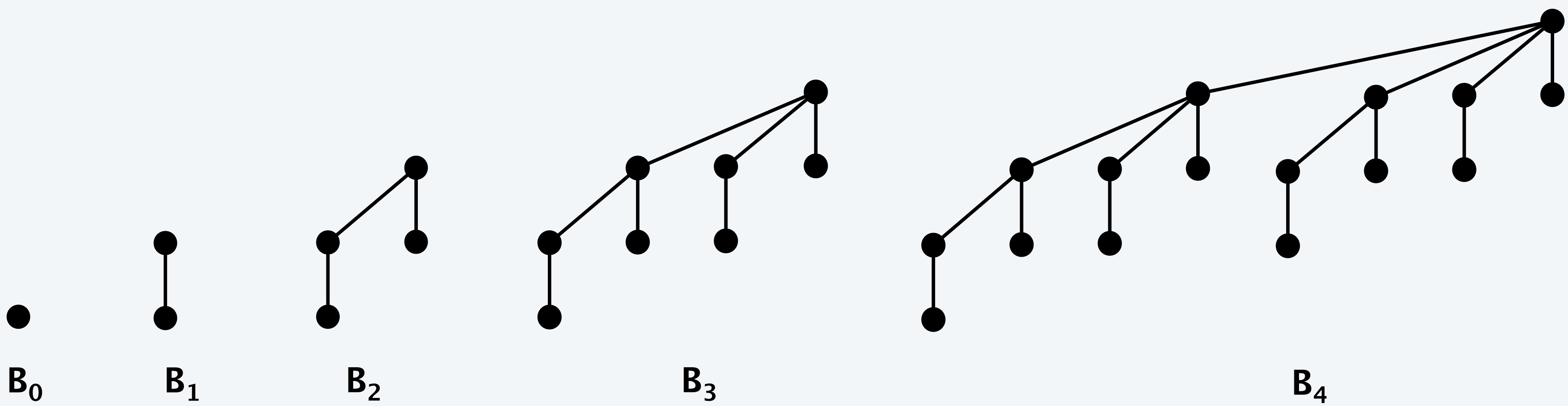
CR Categories: 4.34, 5.24, 5.25, 5.32, 8.1

Binomial tree

Def. A binomial tree of order k is defined recursively:

- Order 0: single node.
- Order k: one binomial tree of order $k-1$ linked to another of order $k-1$.



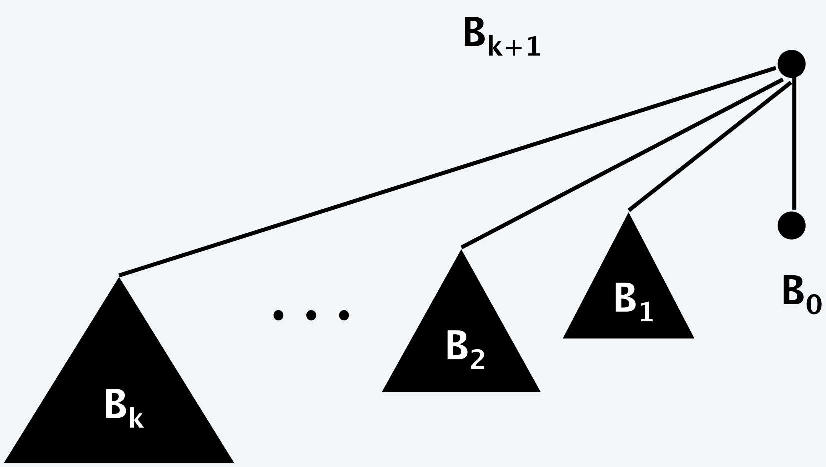


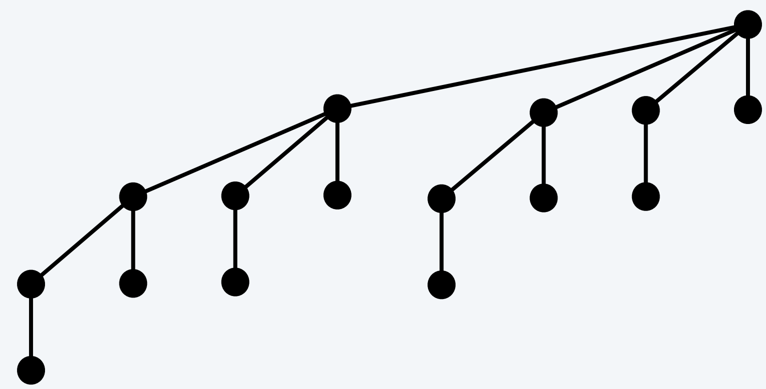
Binomial tree properties

Properties. Given an order k binomial tree B_k,

- Its height is k.
- It has 2^k nodes.
- It has $\binom{k}{i}$ nodes at depth i.
- The degree of its root is k.
- Deleting its root yields k binomial trees $B_{k-1}, \ldots, B_0$.

Pf. [by induction on k]

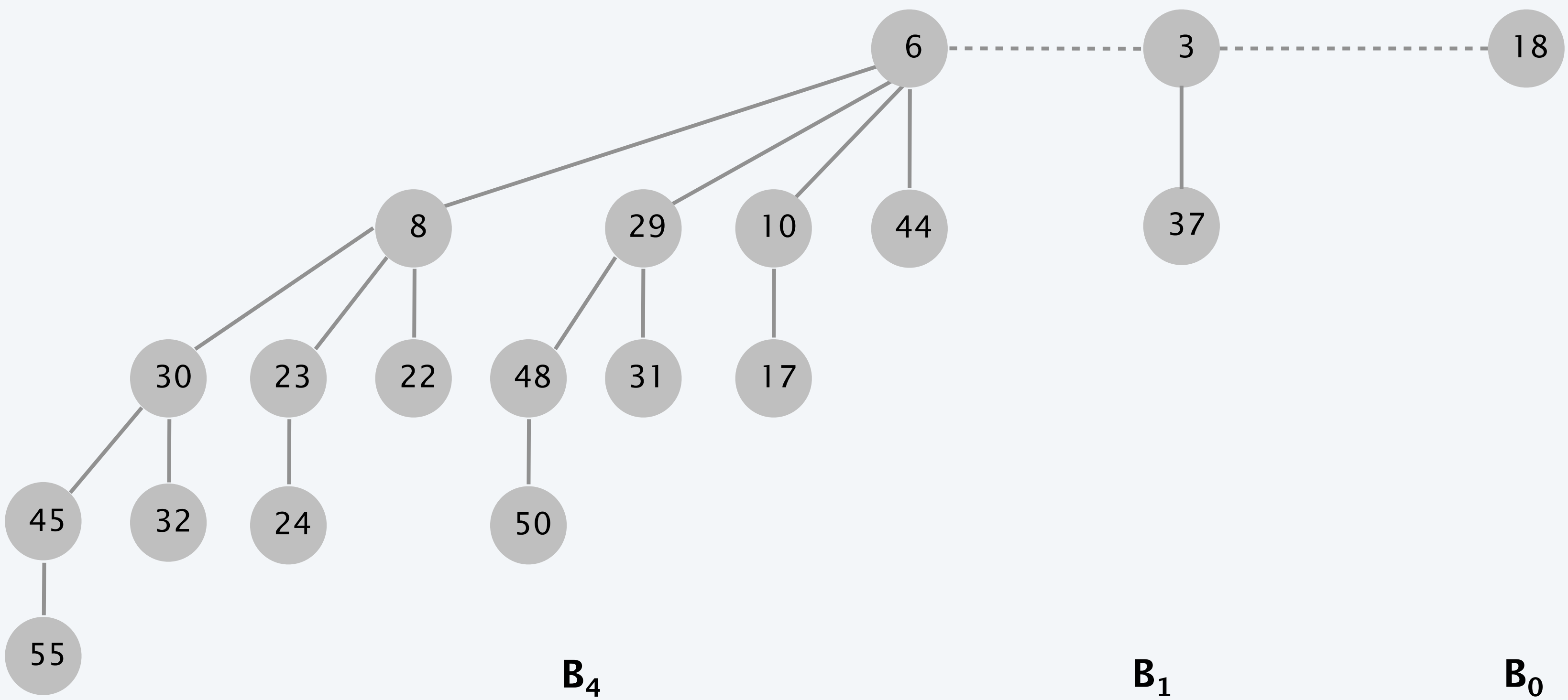




Binomial heap

Def. A binomial heap is a sequence of binomial trees such that:

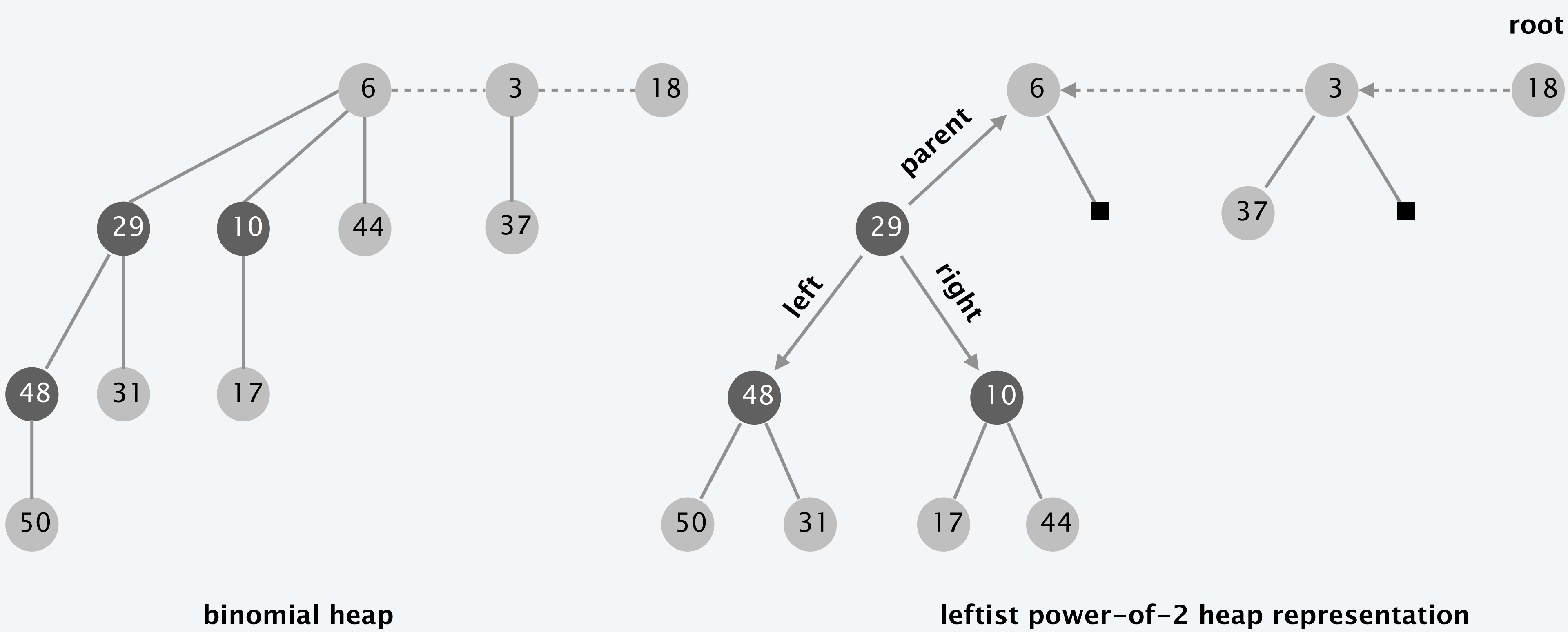
- Each tree is min-heap ordered.
- There is either 0 or 1 binomial tree of order k.



Binomial heap representation

Binomial trees. Represent trees using left-child, right-sibling pointers.

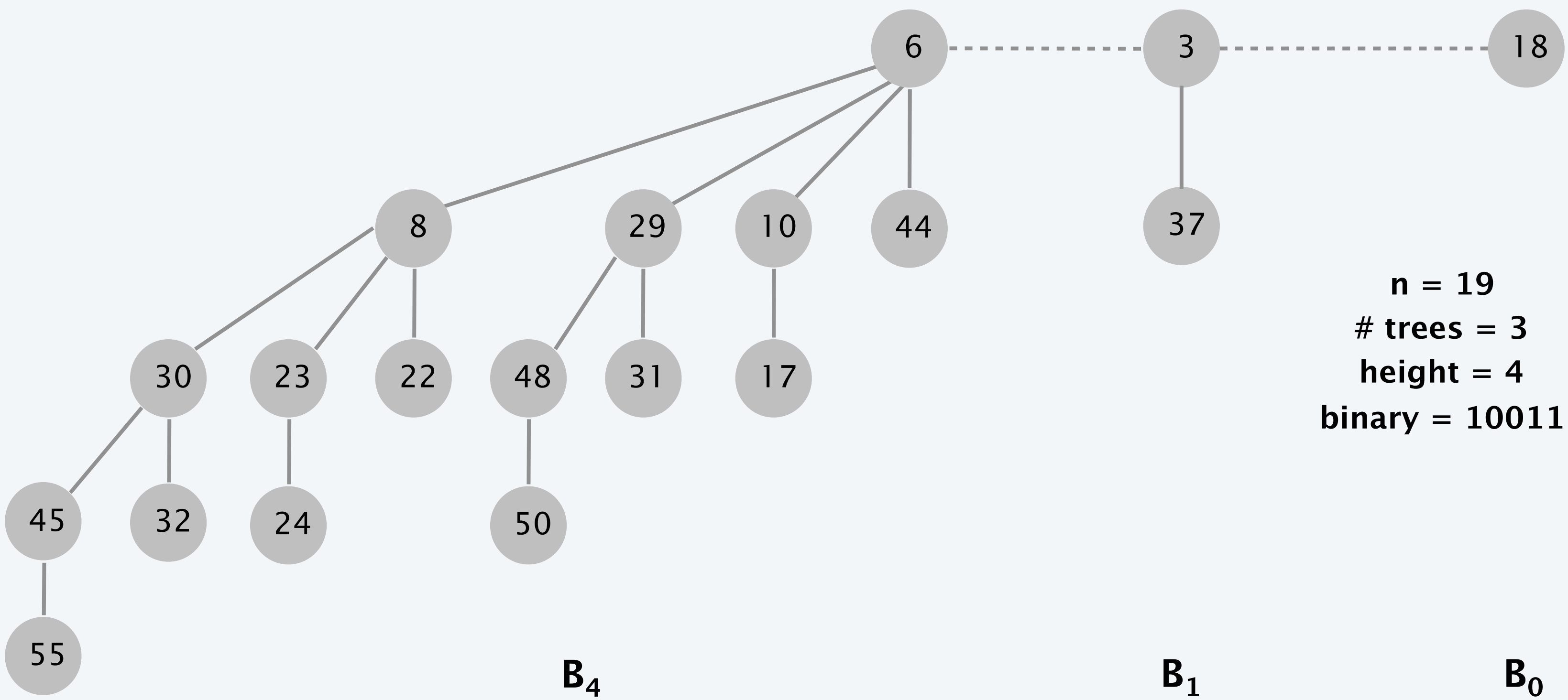
Roots of trees. Connect with singly-linked list, with degrees decreasing from left to right.



Binomial heap properties

Properties. Given a binomial heap with n nodes:

- The node containing the min element is a root of B_0, B_1, …, or B_k.
- It contains the binomial tree B_i iff $b_i = 1$, where $b_k \cdot b_2 b_1 b_0$ is binary representation of n.
- It has $\leq \lfloor \log_2 n \rfloor + 1$ binomial trees.
- Its height $\leq \lfloor \log_2 n \rfloor$.



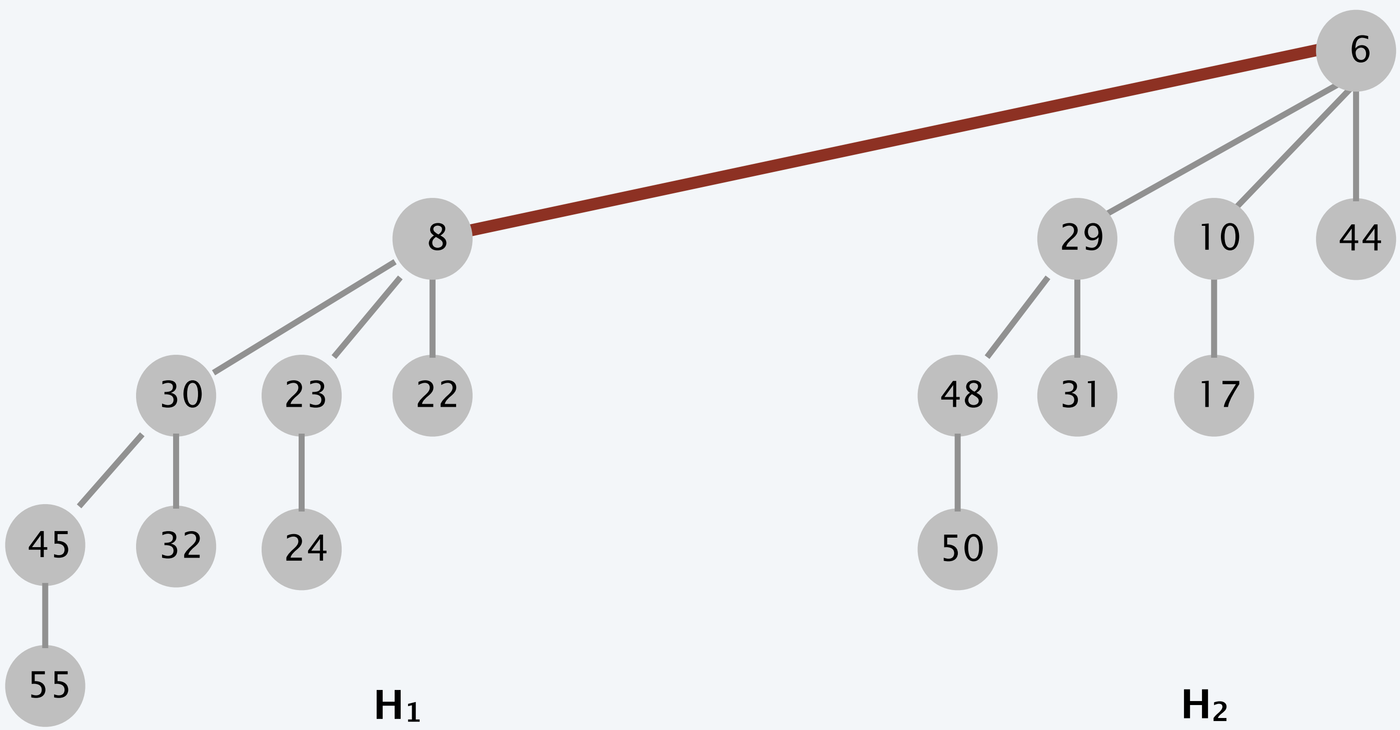
n = 19
# trees = 3
height = 4
binary = 10011

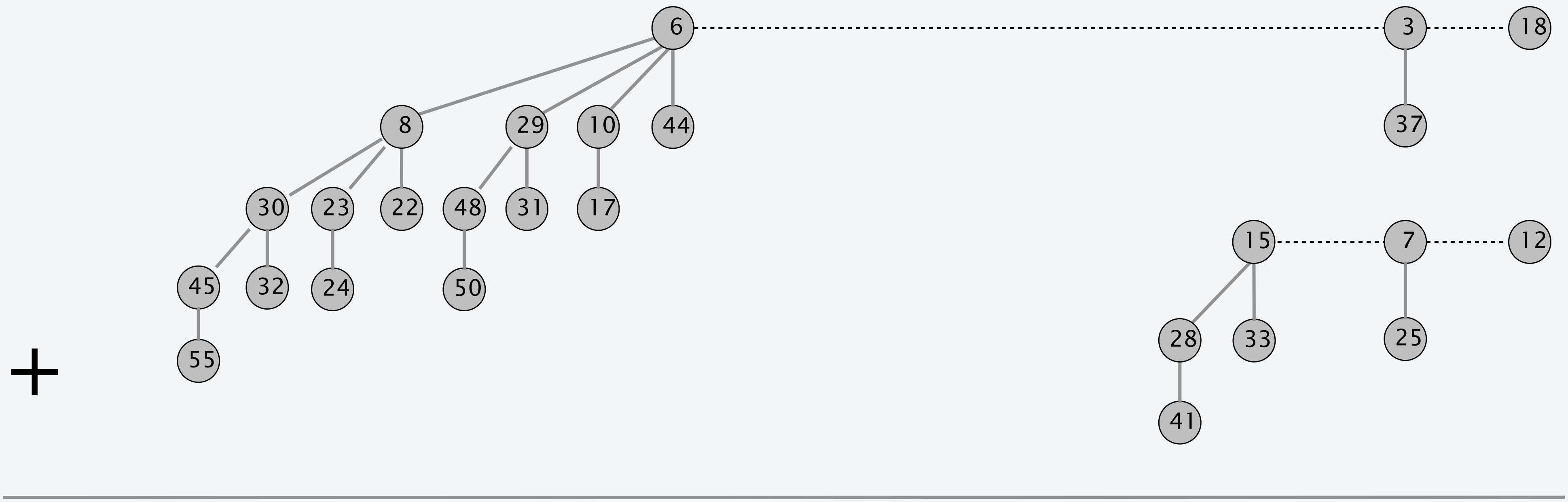
Binomial heap: union

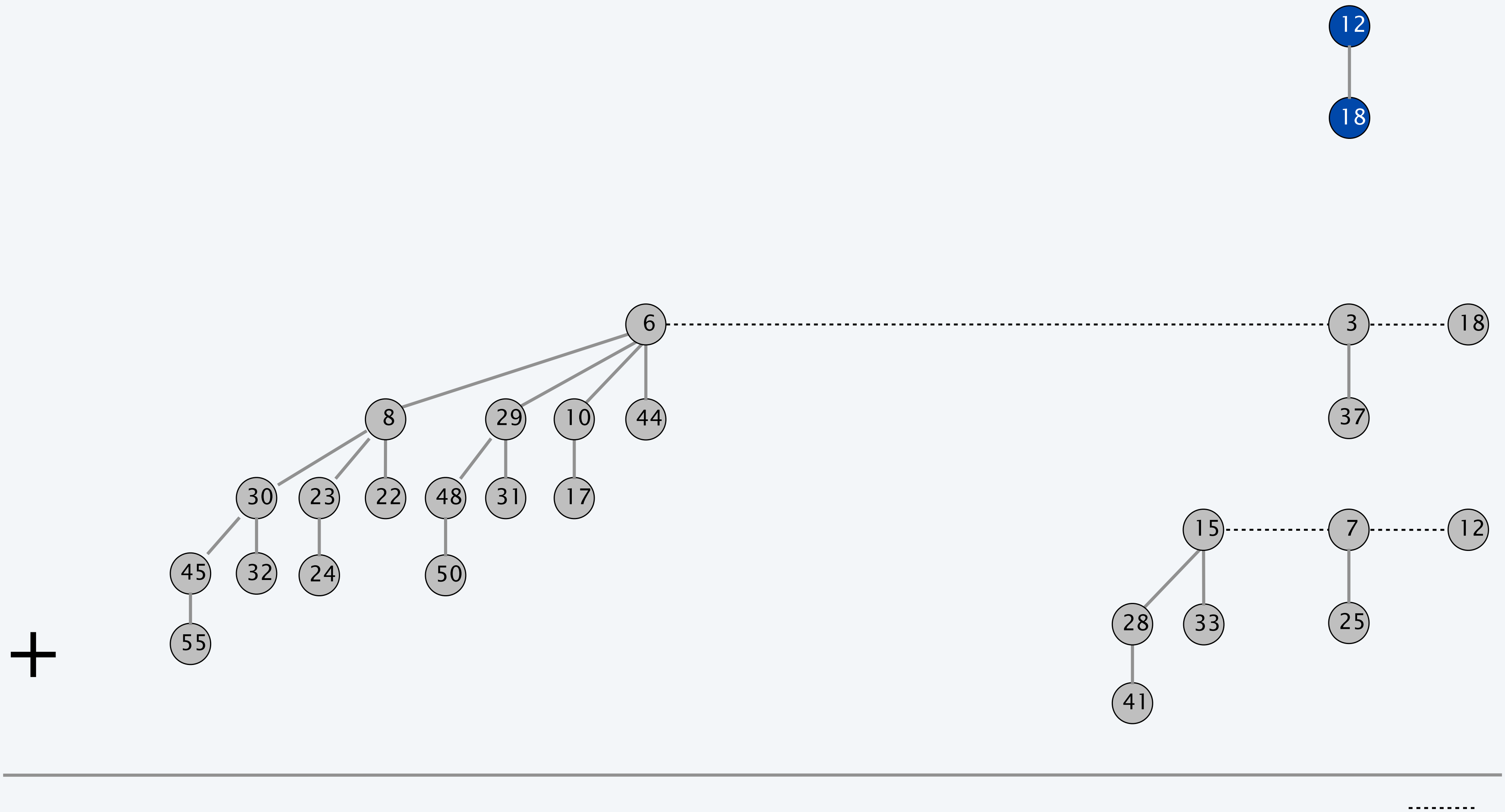
Union operation. Given two binomial heaps H_1 and H_2, (destructively) replace with a binomial heap H that is the union of the two.

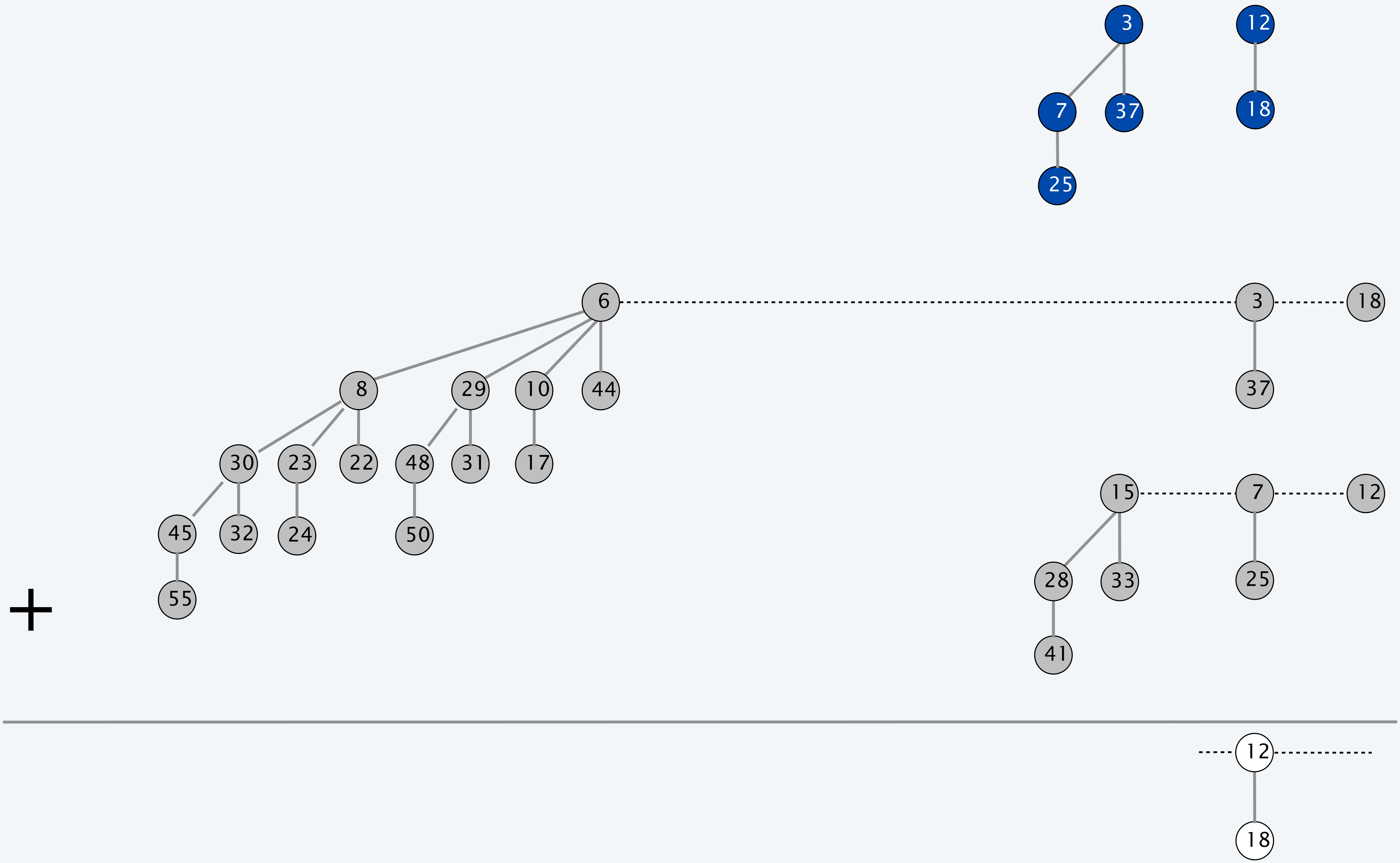
Warmup. Easy if H_1 and H_2 are both binomial trees of order k.

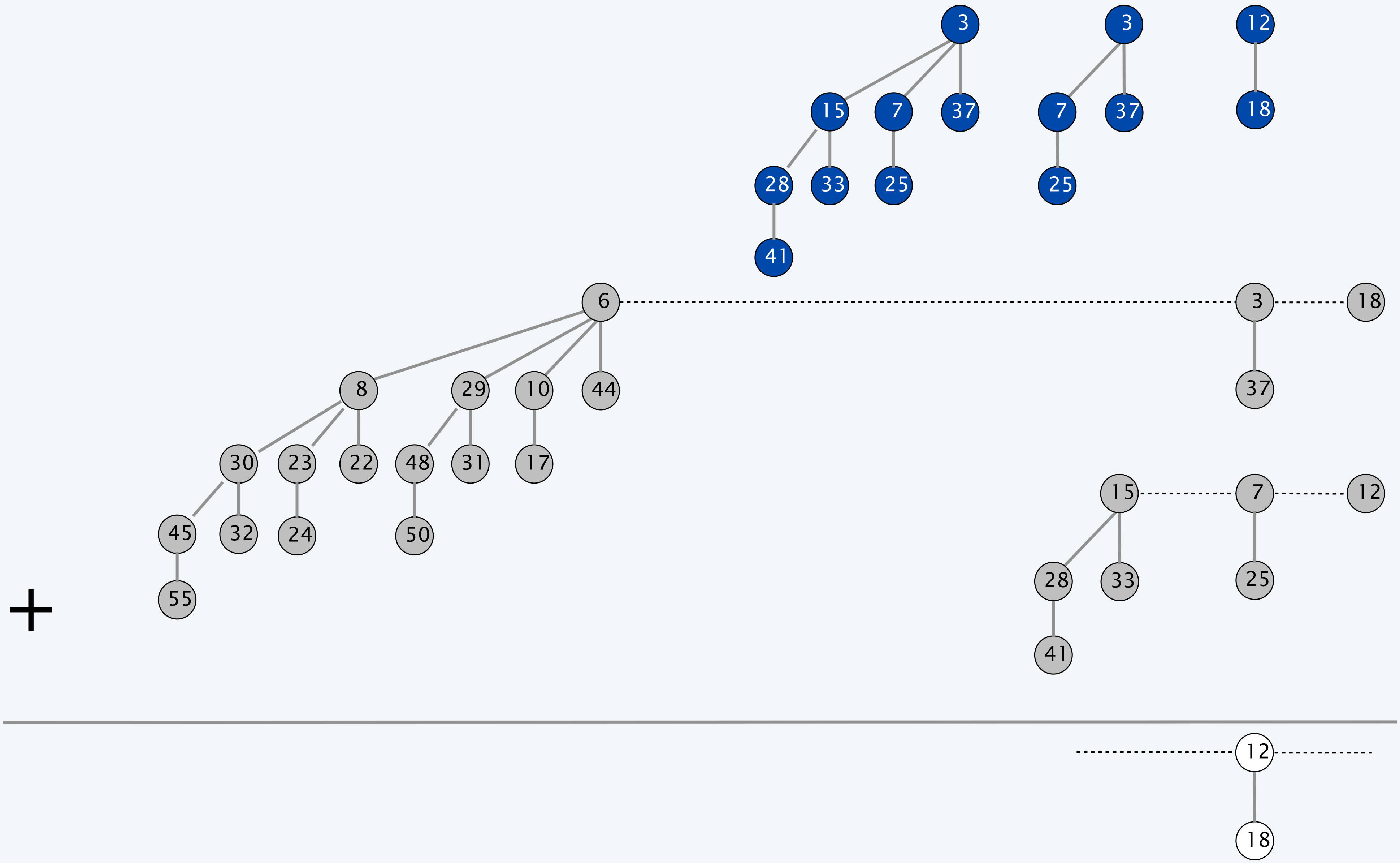
- Connect roots of H_1 and H_2.
- Choose node with smaller key to be root of H.

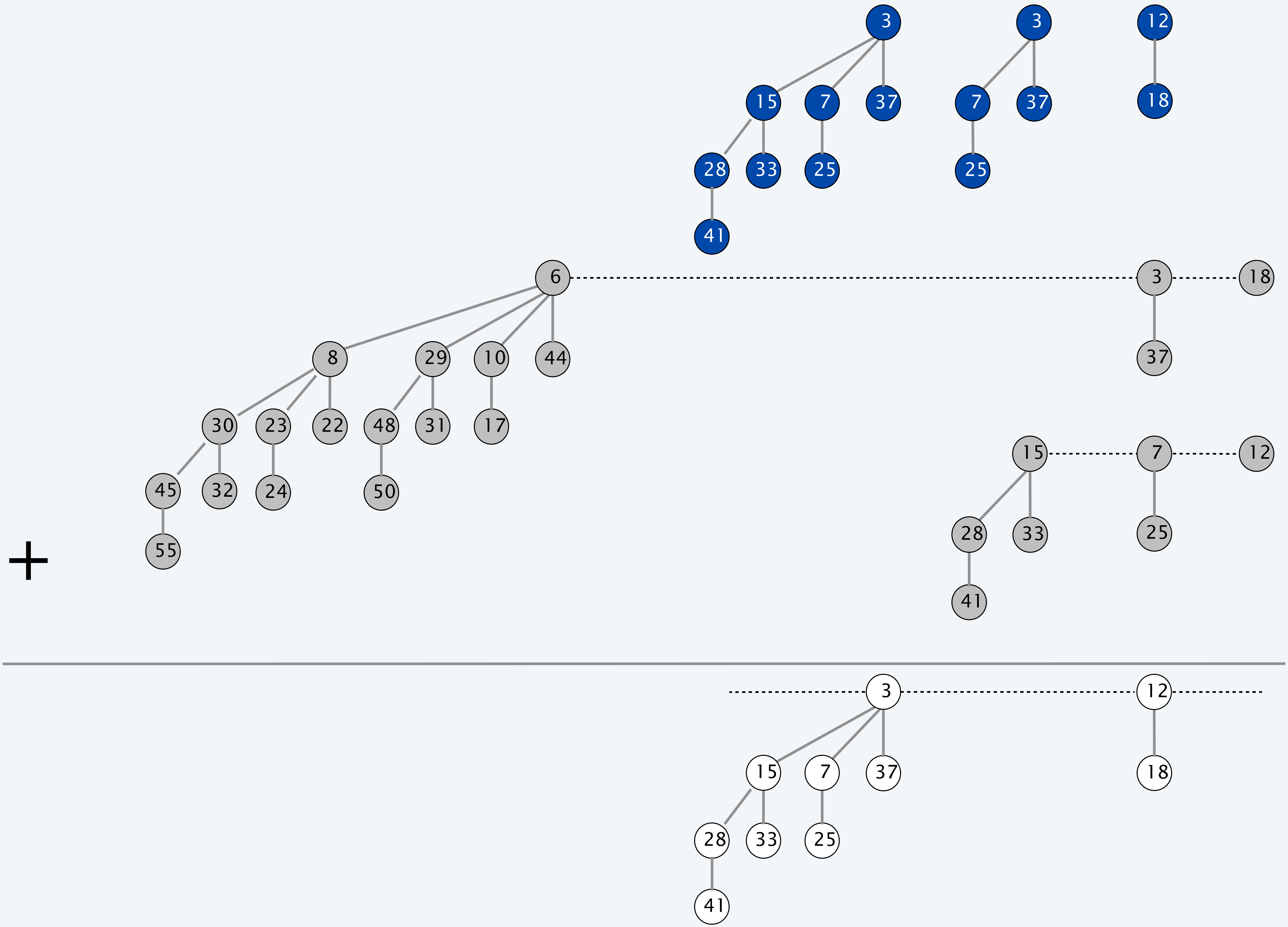


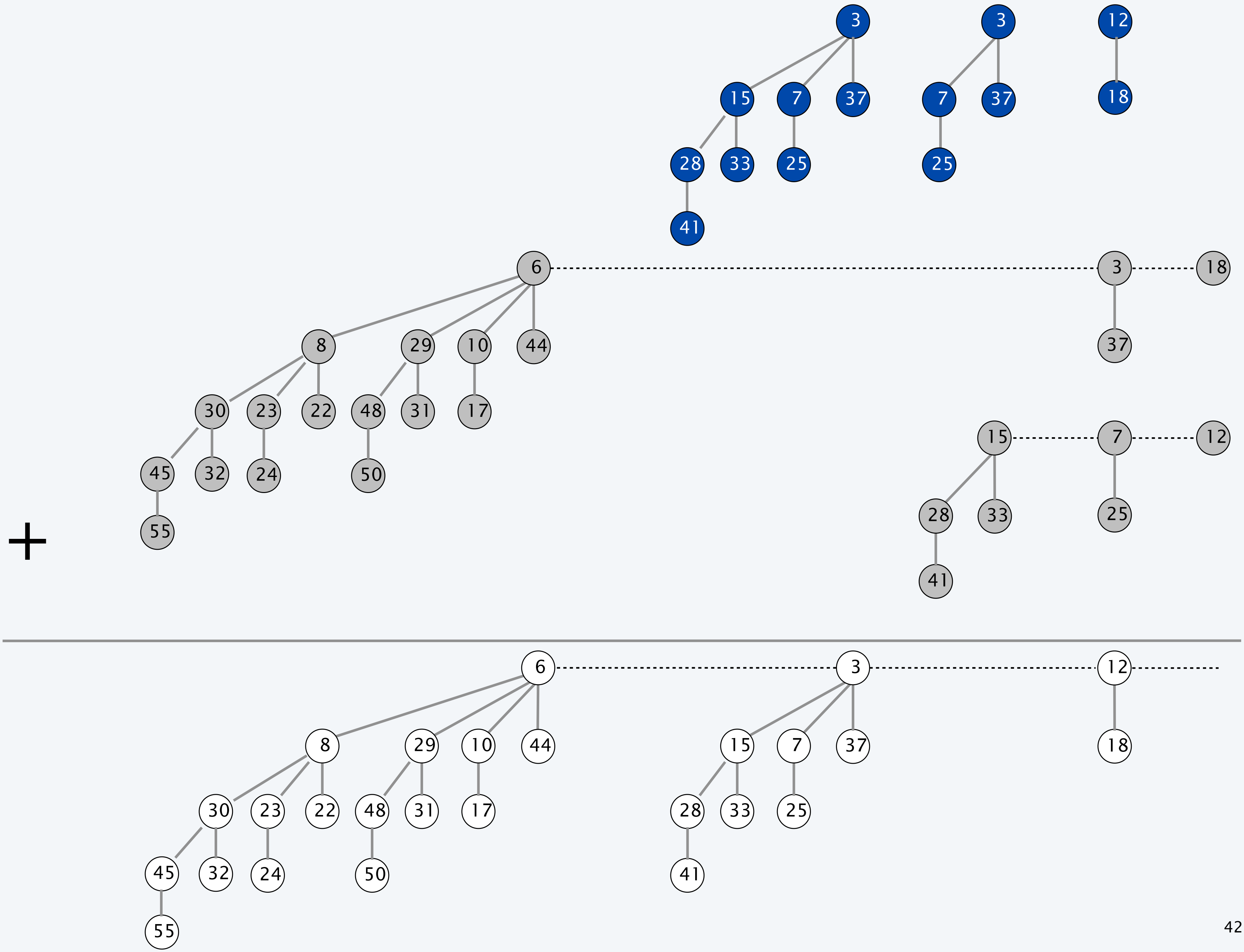

6
3
18
8
29
10
44
37
30
23
22
48
31
17
15
7
12
45
32
24
50
28
33
25
55
41

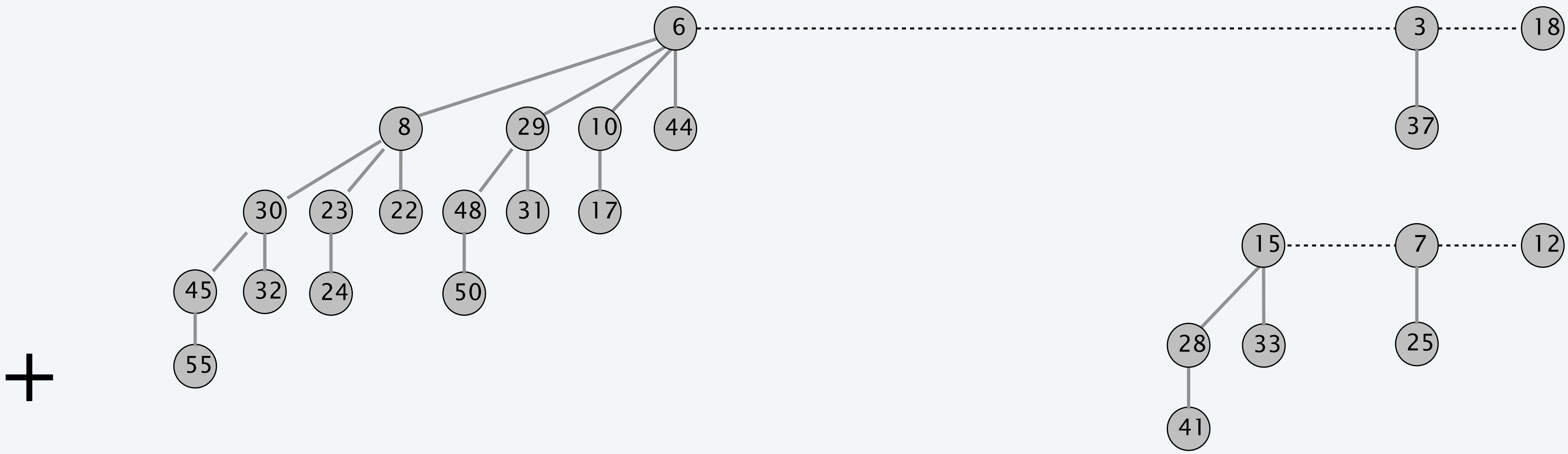

12
18
6
3
18
8
29
10
44
37
30
23
22
48
31
17
15
7
12
45
32
24
50
28
33
25
55
41


3
12
7
37
18
25
6
3
18
8
29
10
44
37
30
23
22
48
31
17
15
7
12
45
32
24
50
28
33
25
55
41
12
18


3
15
7
37
28
33
25
41
3
7
37
25
12
18
6
8
29
10
44
30
23
22
48
31
17
45
32
24
50
55
3
18
37
15
7
12
28
33
25
41
+
12
18


3
15
7
37
28
33
25
41
3
7
37
25
12
18
6
8
29
10
44
30
23
22
48
31
17
45
32
24
50
55
3
18
37
15
7
12
28
33
25
41
+
3
12
15
7
37
18
28
33
25
41


3
15
7
37
28
33
25
41
3
7
37
25
12
18
6
8
29
10
44
30
23
22
48
31
17
45
32
24
50
55
3
18
37
15
7
12
28
33
25
41
+
6
3
12
8
29
10
44
15
7
37
18
30
23
22
48
31
17
28
33
25
45
32
24
50
41
55



19 + 7 = 26

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

Binomial heap: union

Union operation. Given two binomial heaps H_1 and H_2, (destructively) replace with a binomial heap H that is the union of the two.

Solution. Analogous to binary addition.

Running time. $O(\log n)$.

Pf. Proportional to number of trees in root lists $\leq 2(\lfloor \log_2 n \rfloor + 1)$. ▪

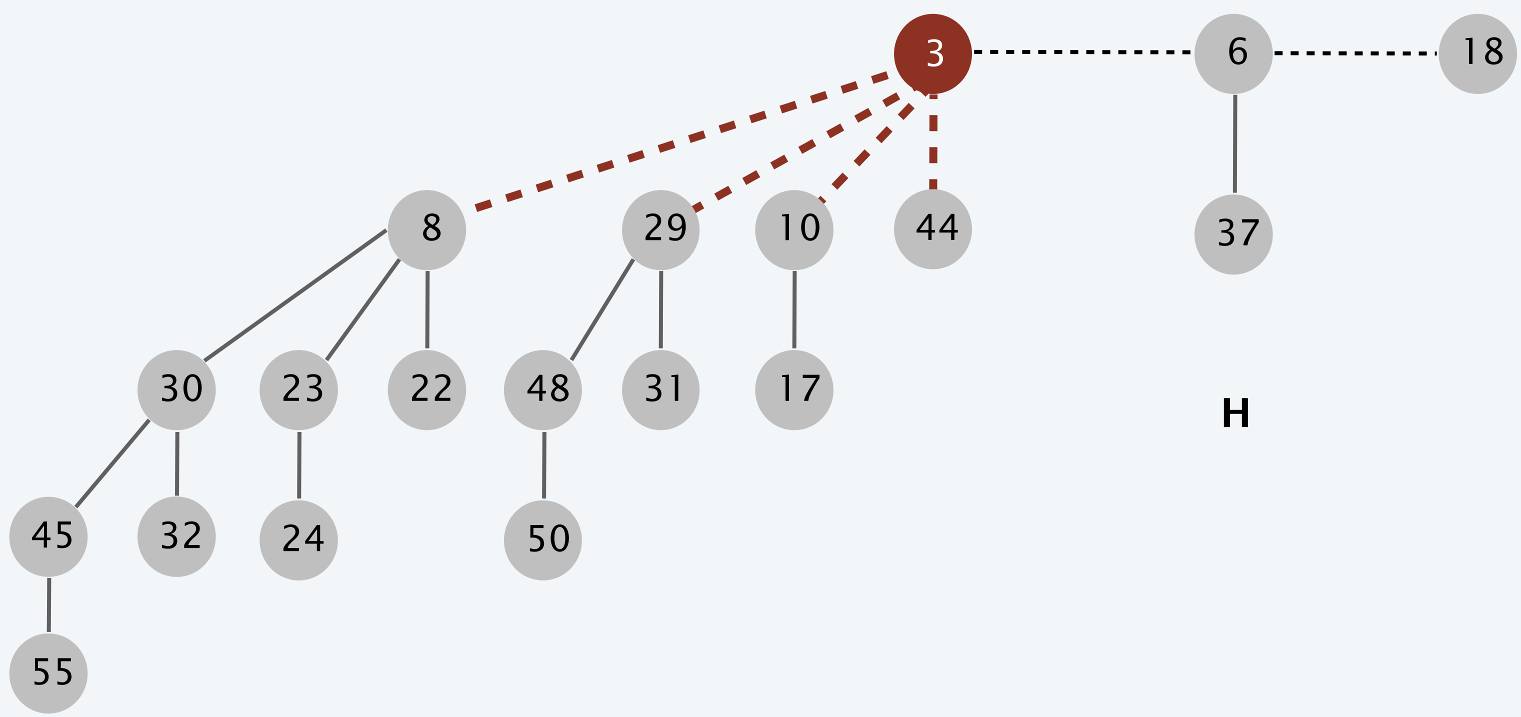
19 + 7 = 26

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

Binomial heap: extract the minimum

Extract-min. Delete the node with minimum key in binomial heap H.

- Find root x with min key in root list of H, and delete.

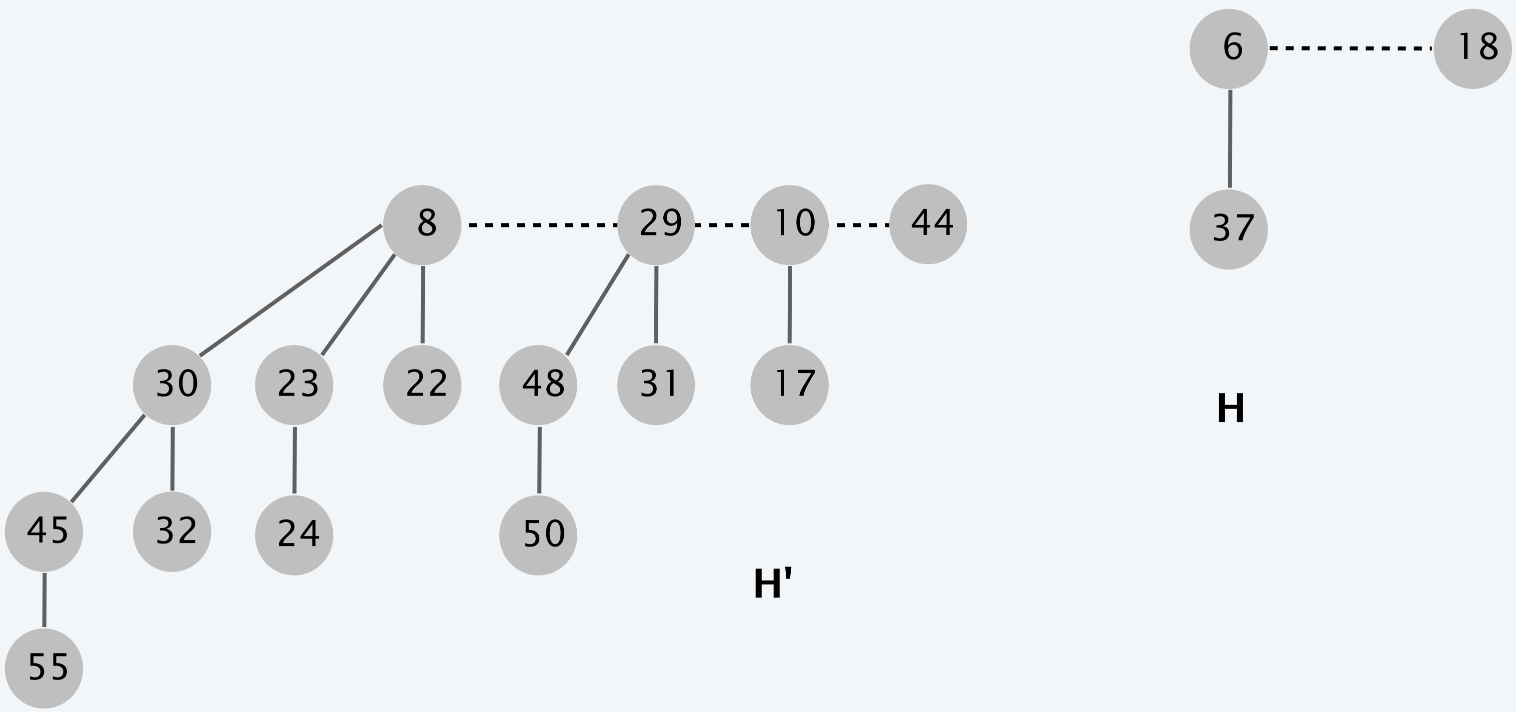


Binomial heap: extract the minimum

Extract-min. Delete the node with minimum key in binomial heap H.

- Find root x with min key in root list of H, and delete.
- $H' \leftarrow$ broken binomial trees.
- $H \leftarrow$ UNION(H', H).

Running time. $O(\log n)$.

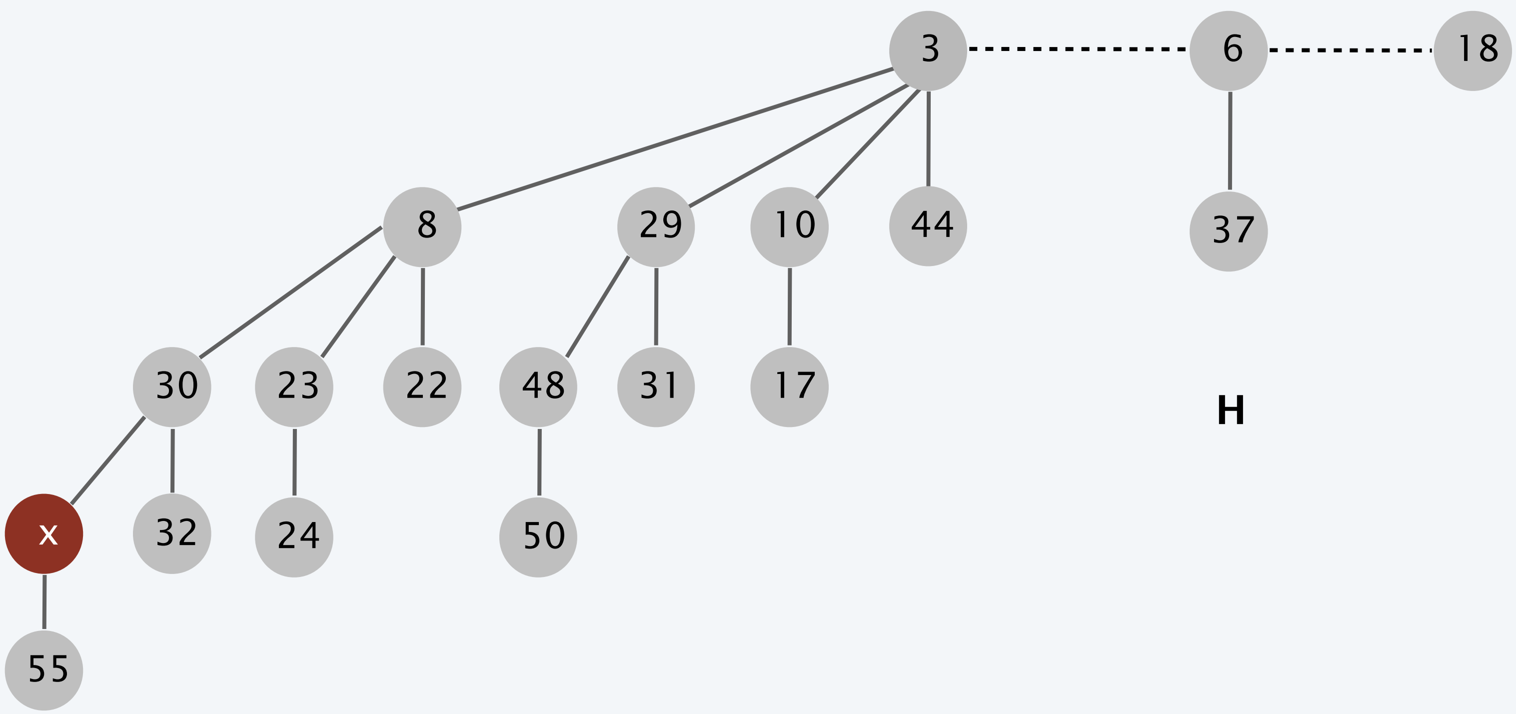


Binomial heap: decrease key

Decrease key. Given a handle to an element x in H, decrease its key to k.

- Suppose x is in binomial tree B_k.
- Repeatedly exchange x with its parent until heap order is restored.

Running time. $O(\log n)$.

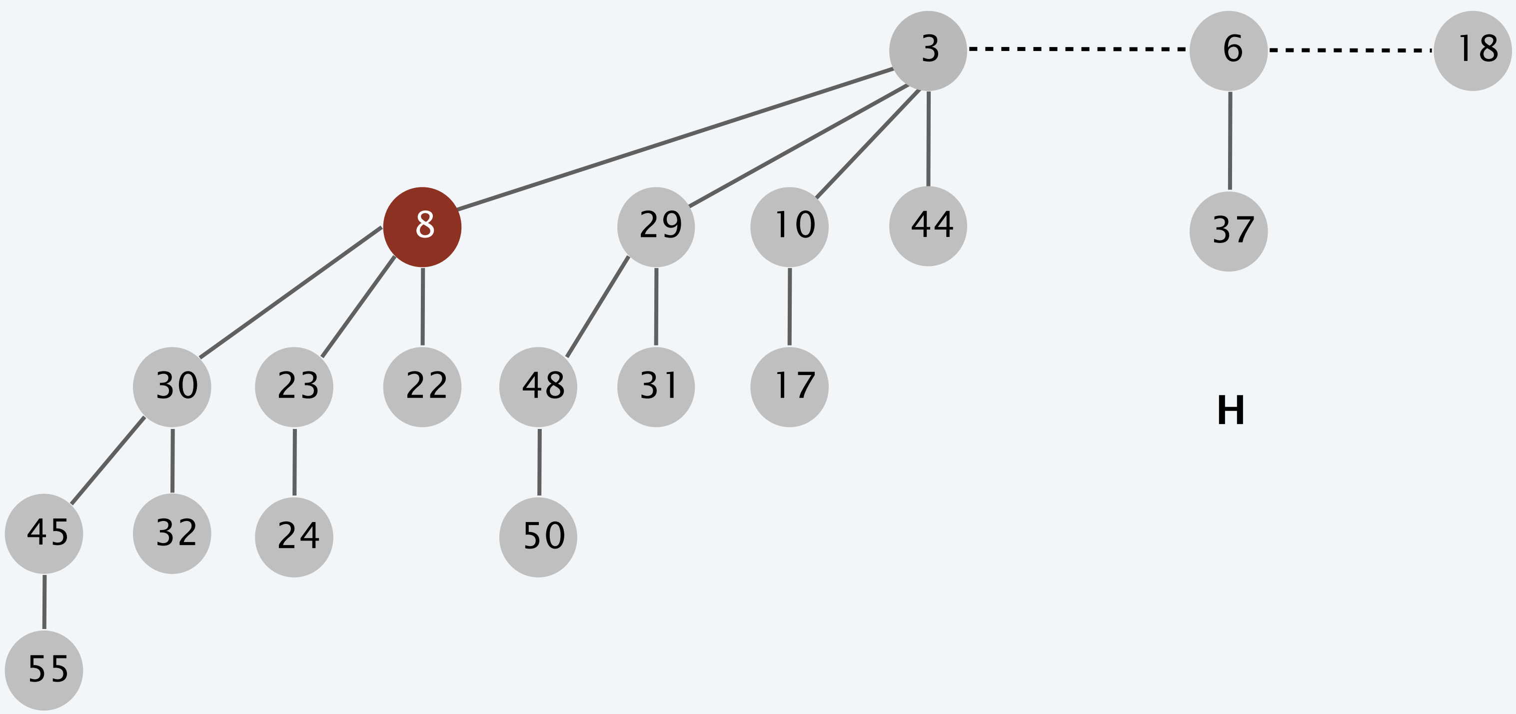


Binomial heap: delete

Delete. Given a handle to an element x in a binomial heap, delete it.

- DECREASE-KEY(H, x, $-\infty$).
- DELETE-MIN(H).

Running time. $O(\log n)$.

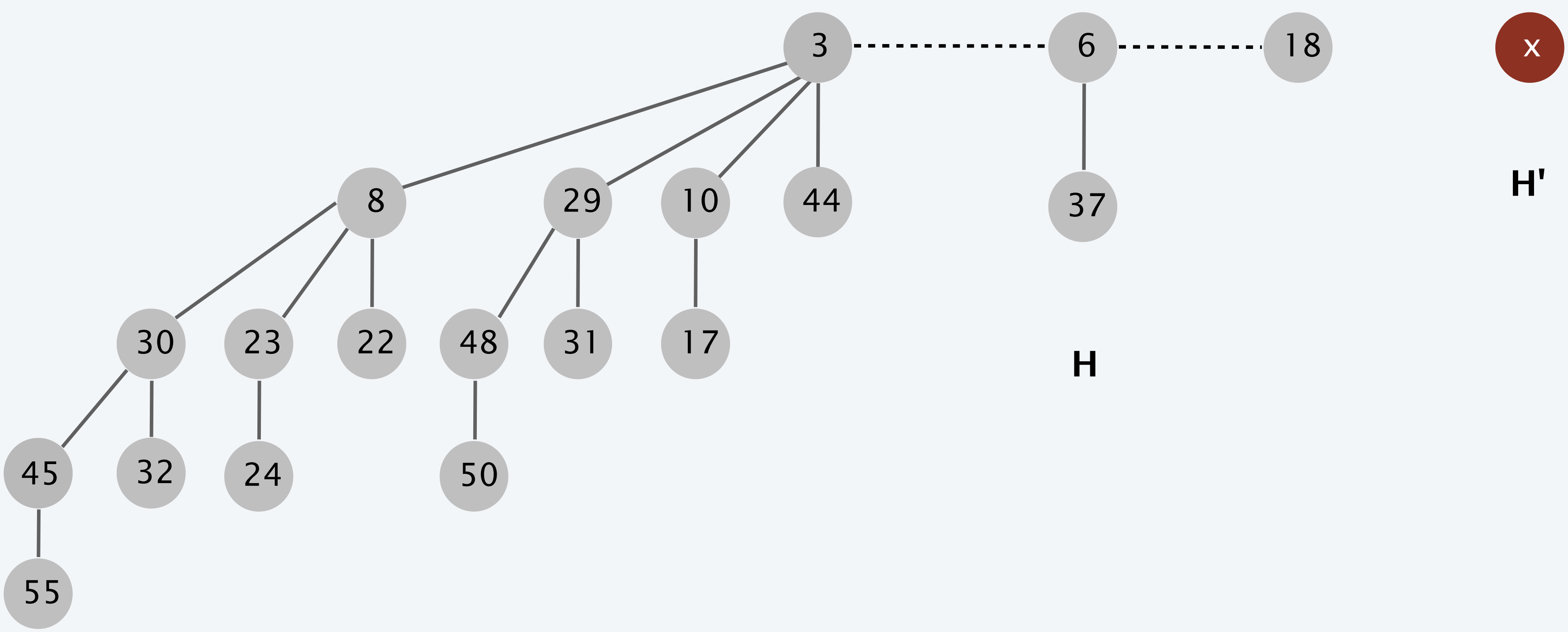


Binomial heap: insert

Insert. Given a binomial heap H, insert an element x.

- $H' \leftarrow$ MAKE-HEAP( ).
- $H' \leftarrow$ INSERT(H', x).
- $H \leftarrow$ UNION(H', H).

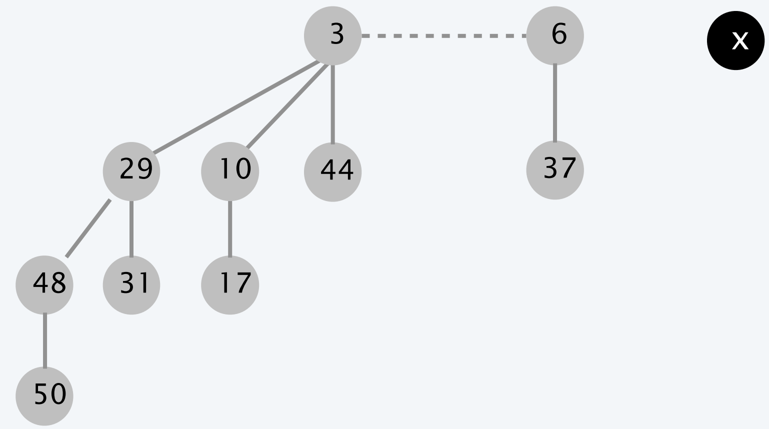
Running time. $O(\log n)$.



Binomial heap: sequence of insertions

Insert. How much work to insert a new node x ?

- If $n =$ ....0, then only 1 credit.
- If $n =$ ....01, then only 2 credits.
- If $n =$ ....011, then only 3 credits.
- If $n =$ ....0111, then only 4 credits.



Observation. Inserting one element can take $\Omega(\log n)$ time.

if n = 11...111

Theorem. Starting from an empty binomial heap, a sequence of n consecutive INSERT operations takes $O(n)$ time.

Pf. $(n / 2)\,(1) + (n / 4)(2) + (n / 8)(3) + \ldots \;\leq\; 2\,n.$ ▪

$$\sum_{i=1}^{k} \frac{i}{2^i} \;=\; 2 - \frac{k}{2^k} - \frac{1}{2^{k-1}}$$
$$\leq 2$$

Binomial heap: amortized analysis

Theorem. In a binomial heap, the amortized cost of INSERT is $O(1)$ and the worst-case cost of EXTRACT-MIN and DECREASE-KEY is $O(\log n)$.

Pf. Define potential function $\Phi(H_i) = trees(H_i) = \#$ trees in binomial heap H_i.

- $\Phi(H_0) = 0$.
- $\Phi(H_i) \geq 0$ for each binomial heap H_i.

Case 1. [INSERT]

- Actual cost $c_i =$ number of trees merged $+ 1$.
- $\Delta\Phi = \Phi(H_i) - \Phi(H_{i-1}) =$ number of trees merged $- 1$.
- Amortized cost $= \hat{c}_i = c_i + \Phi(H_i) - \Phi(H_{i-1}) = 2$.

Binomial heap: amortized analysis

Theorem. In a binomial heap, the amortized cost of INSERT is $O(1)$ and the worst-case cost of EXTRACT-MIN and DECREASE-KEY is $O(\log n)$.

Pf. Define potential function $\Phi(H_i) = trees(H_i) = \#$ trees in binomial heap H_i.

- $\Phi(H_0) = 0$.
- $\Phi(H_i) \geq 0$ for each binomial heap H_i.

Case 2. [DECREASE-KEY]

- Actual cost $c_i = O(\log n)$.
- $\Delta\Phi = \Phi(H_i) - \Phi(H_{i-1}) = 0$.
- Amortized cost $= \hat{c}_i = c_i = O(\log n)$.

Binomial heap: amortized analysis

Theorem. In a binomial heap, the amortized cost of INSERT is $O(1)$ and the worst-case cost of EXTRACT-MIN and DECREASE-KEY is $O(\log n)$.

Pf. Define potential function $\Phi(H_i) = trees(H_i) = \#$ trees in binomial heap H_i.

- $\Phi(H_0) = 0$.
- $\Phi(H_i) \geq 0$ for each binomial heap H_i.

Case 3. [EXTRACT-MIN or DELETE]

- Actual cost $c_i = O(\log n)$.
- $\Delta\Phi = \Phi(H_i) - \Phi(H_{i-1}) \leq \lfloor \log_2 n \rfloor$.
- Amortized cost $= \hat{c}_i = c_i + \Phi(H_i) - \Phi(H_{i-1}) = O(\log n)$. ▪

Priority queues performance cost summary

operation	linked list	binary heap	binomial heap	binomial heap
MAKE-HEAP	$O(1)$	$O(1)$	$O(1)$	$O(1)$
ISEMPTY	$O(1)$	$O(1)$	$O(1)$	$O(1)$
INSERT	$O(1)$	$O(\log n)$	$O(\log n)$	$O(1)$ †
EXTRACT-MIN	$O(n)$	$O(\log n)$	$O(\log n)$	$O(\log n)$
DECREASE-KEY	$O(1)$	$O(\log n)$	$O(\log n)$	$O(\log n)$
DELETE	$O(1)$	$O(\log n)$	$O(\log n)$	$O(\log n)$
UNION	$O(1)$	$O(n)$	$O(\log n)$	$O(1)$ † (homework)
FIND-MIN	$O(n)$	$O(1)$	$O(\log n)$	$O(1)$

† amortized

Hopeless challenge. $O(1)$ INSERT, DECREASE-KEY and EXTRACT-MIN. Why?

Challenge. $O(1)$ INSERT and DECREASE-KEY, $O(\log n)$ EXTRACT-MIN.