Binary and Binomial Heaps

These lecture slides are adapted from CLRS, Chapters 6, 19.

Priority Queues

Supports the following operations.

- Insert element x.
- Return min element.
- Return and delete minimum element.
- Decrease key of element x to k.

Applications.

- Dijkstra's shortest path algorithm.
- Prim's MST algorithm.
- Event-driven simulation.
- Huffman encoding.
- Heapsort.
- . . .

Priority Queues in Action

Dijkstra's Shortest Path Algorithm

```
PQinit()
for each v ∈ V
   key(v) ← ∞
   PQinsert(v)

key(s) ← 0
while (!PQisempty())
   v = PQdelmin()
   for each w ∈ Q s.t (v,w) ∈ E
      if π(w) > π(v) + c(v,w)
         PQdecrease(w, π(v) + c(v,w))
```

Priority Queues

		Heaps			
Operation	**Linked List**	**Binary**	**Binomial**	**Fibonacci ***	**Relaxed**
make-heap	1	1	1	1	1
insert	1	log N	log N	1	1
find-min	N	1	log N	1	1
delete-min	N	log N	log N	log N	log N
union	1	N	log N	1	1
decrease-key	1	log N	log N	1	1
delete	N	log N	log N	log N	log N
is-empty	1	1	1	1	1

Dijkstra/Prim
1 make-heap
|V| insert
|V| delete-min
|E| decrease-key

Linked List: $O(|V|^2)$

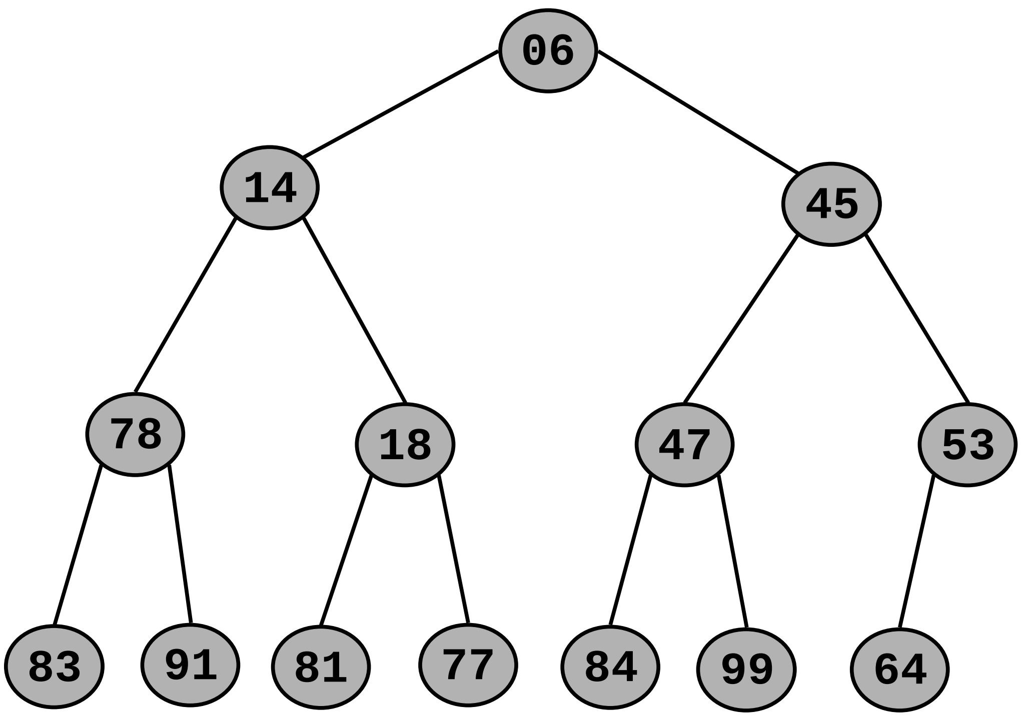
Binary: $O(|E| \log |V|)$

Fibonacci: $O(|E| + |V| \log |V|)$

Binary Heap: Definition

Binary heap.

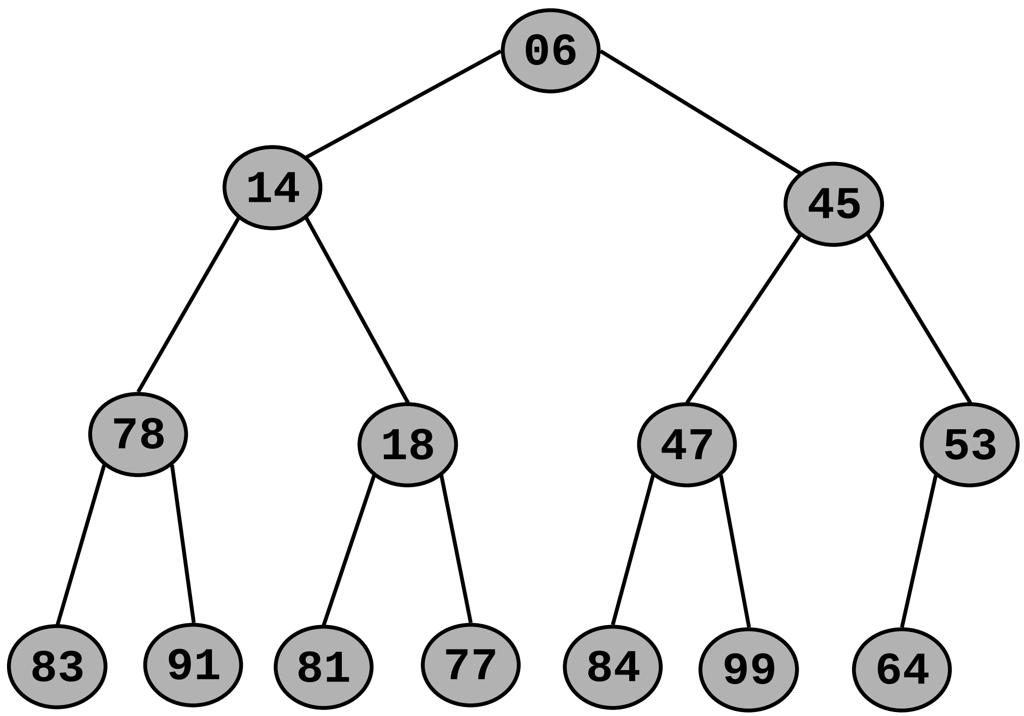
- **Almost complete binary tree.**
  - **filled on all levels, except last, where filled from left to right**
- **Min-heap ordered.**
  - **every child greater than (or equal to) parent**



Binary Heap: Properties

Properties.

- **Min element is in root.**
- **Heap with N elements has height = $\lfloor \log_2 N \rfloor$.**

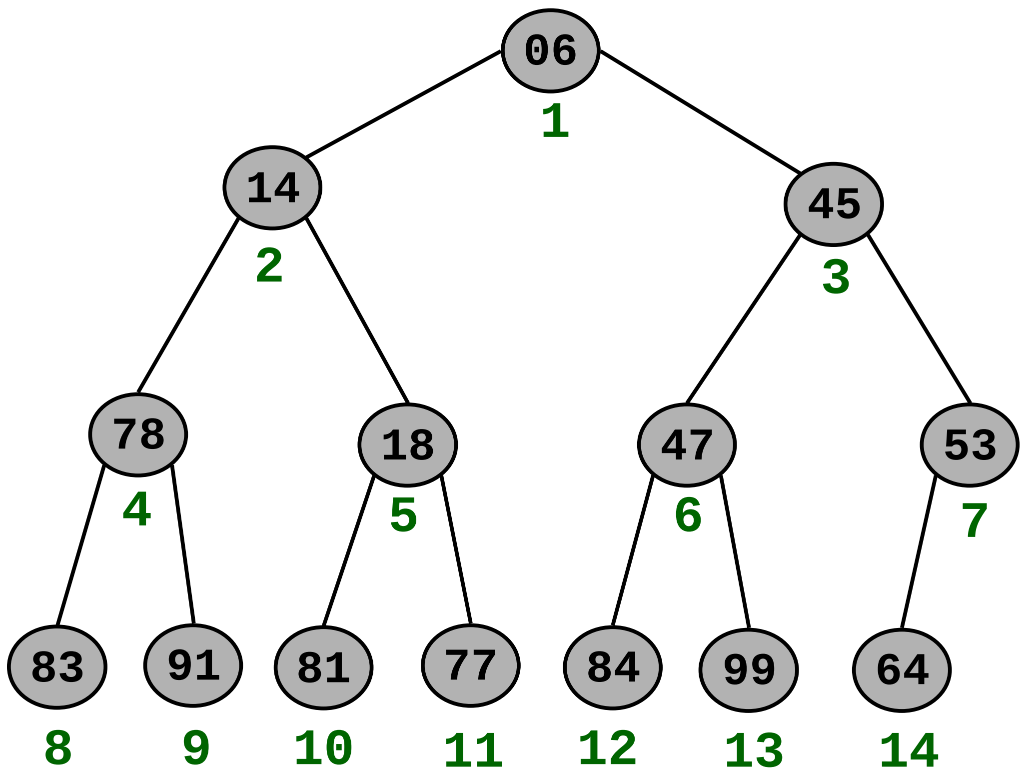


N = 14
Height = 3

Binary Heaps: Array Implementation

Implementing binary heaps.

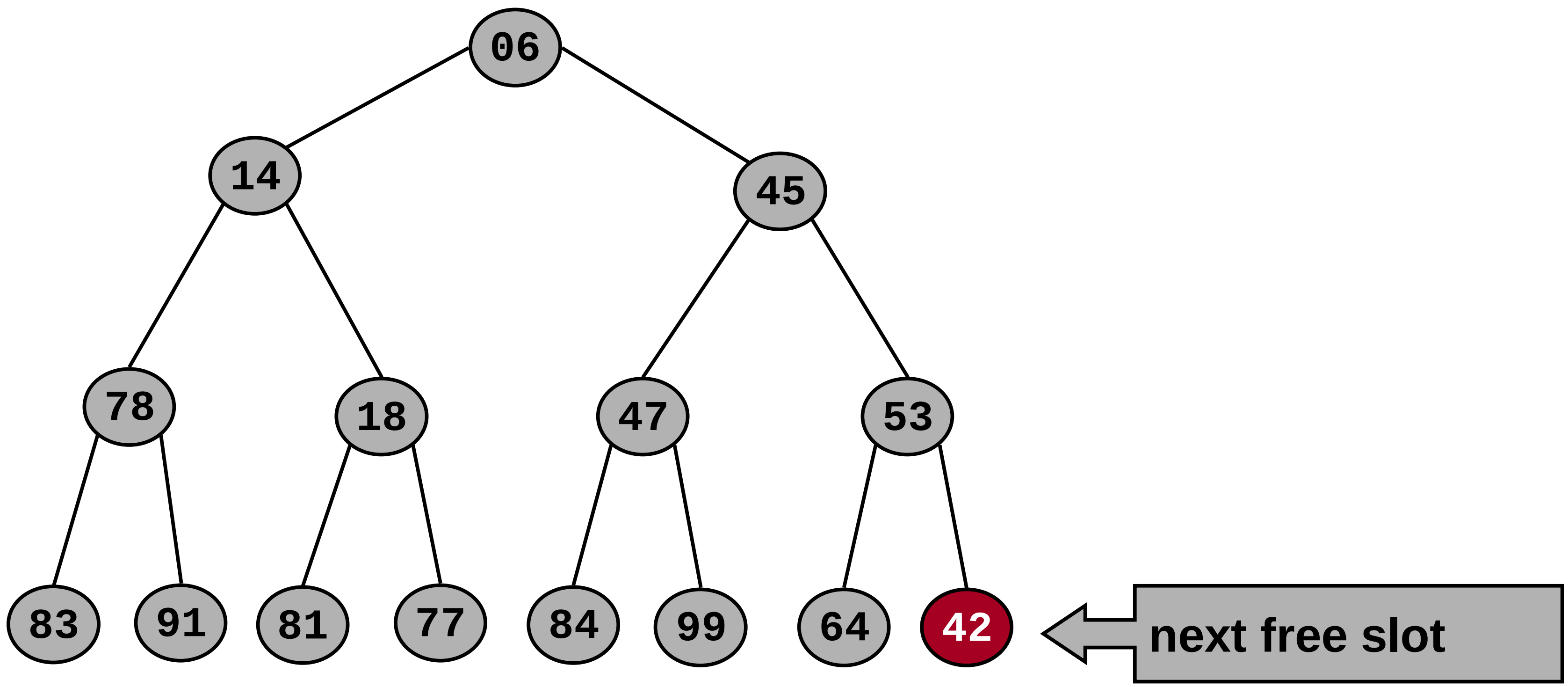
- **Use an array: no need for explicit parent or child pointers.**
  - **Parent(i) = ⌊i/2⌋**
  - **Left(i) = 2i**
  - **Right(i) = 2i + 1**



Binary Heap: Insertion

Insert element x into heap.

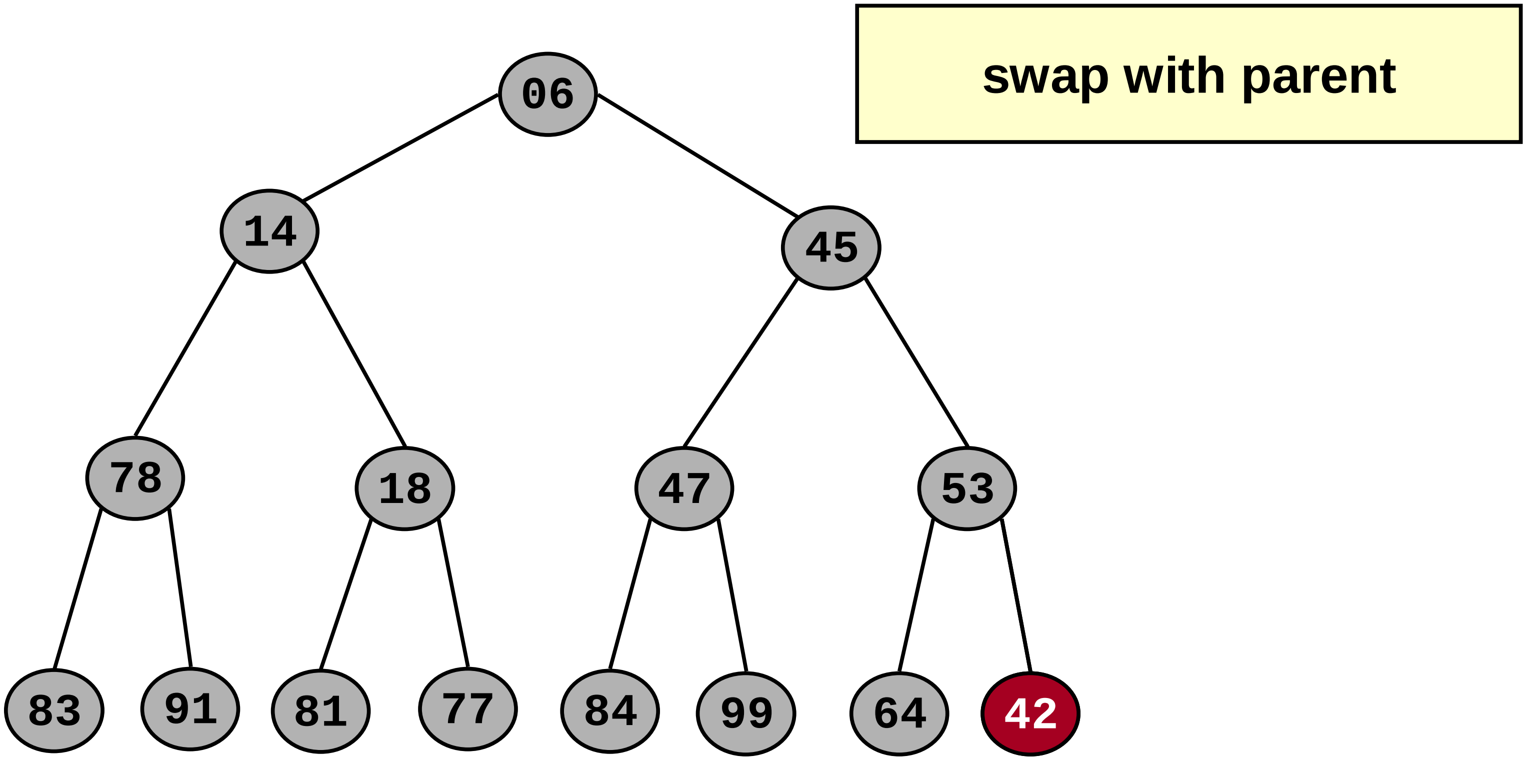
- **Insert into next available slot.**
- **Bubble up until it's heap ordered.**
  - **Peter principle: nodes rise to level of incompetence**



Binary Heap: Insertion

Insert element x into heap.

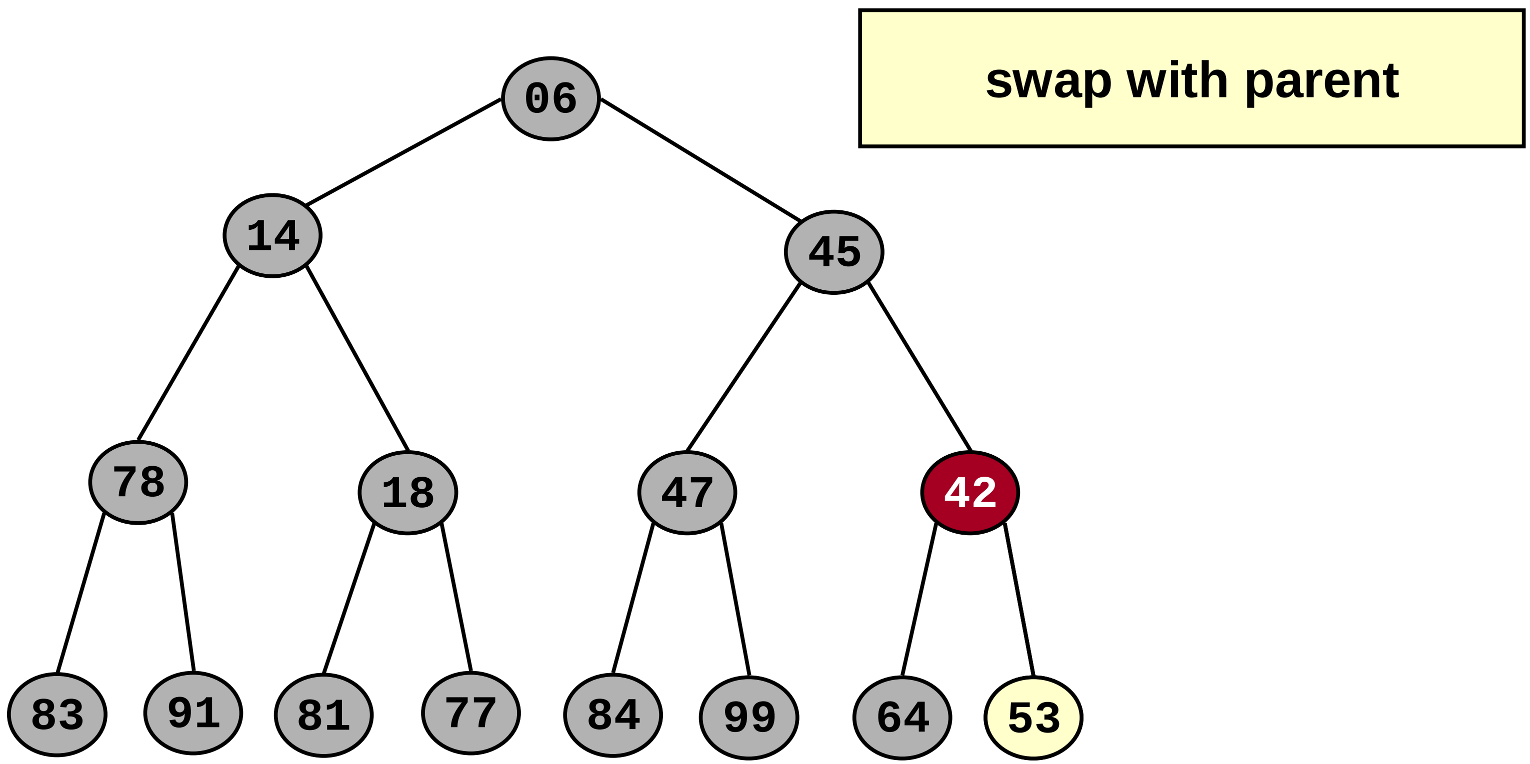
- Insert into next available slot.
- Bubble up until it's heap ordered.
  - Peter principle: nodes rise to level of incompetence



Binary Heap: Insertion

Insert element x into heap.

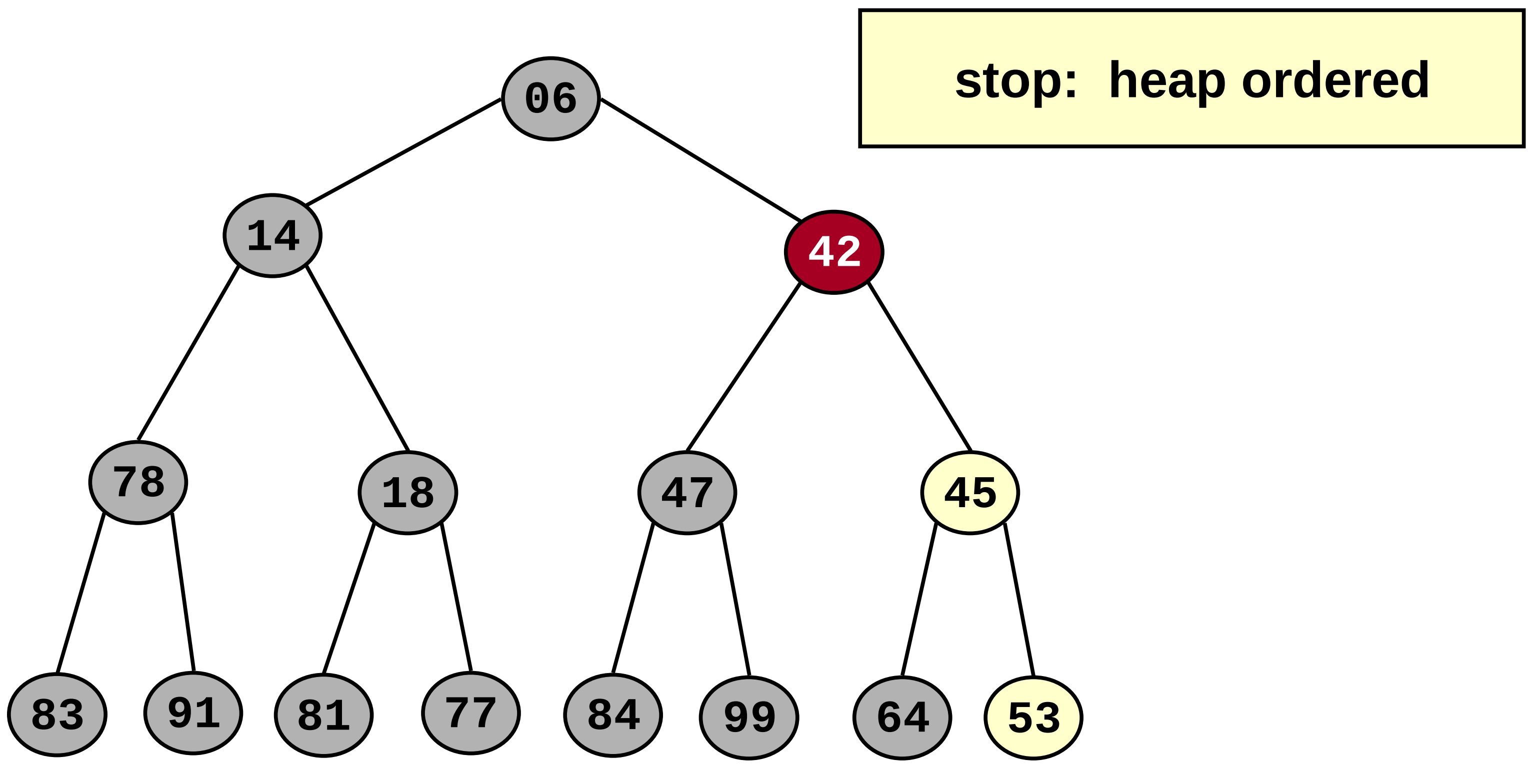
- **Insert into next available slot.**
- **Bubble up until it's heap ordered.**
  - **Peter principle: nodes rise to level of incompetence**



Binary Heap: Insertion

Insert element x into heap.

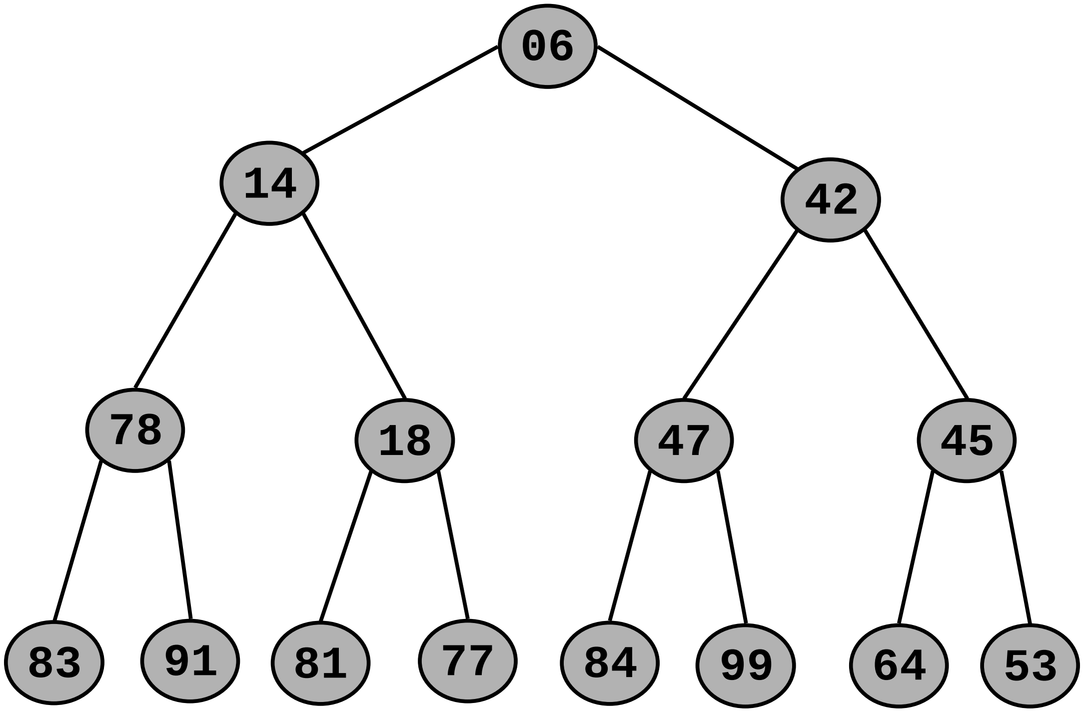
- Insert into next available slot.
- Bubble up until it's heap ordered.
  - Peter principle: nodes rise to level of incompetence
- O(log N) operations.



Binary Heap: Decrease Key

Decrease key of element x to k.

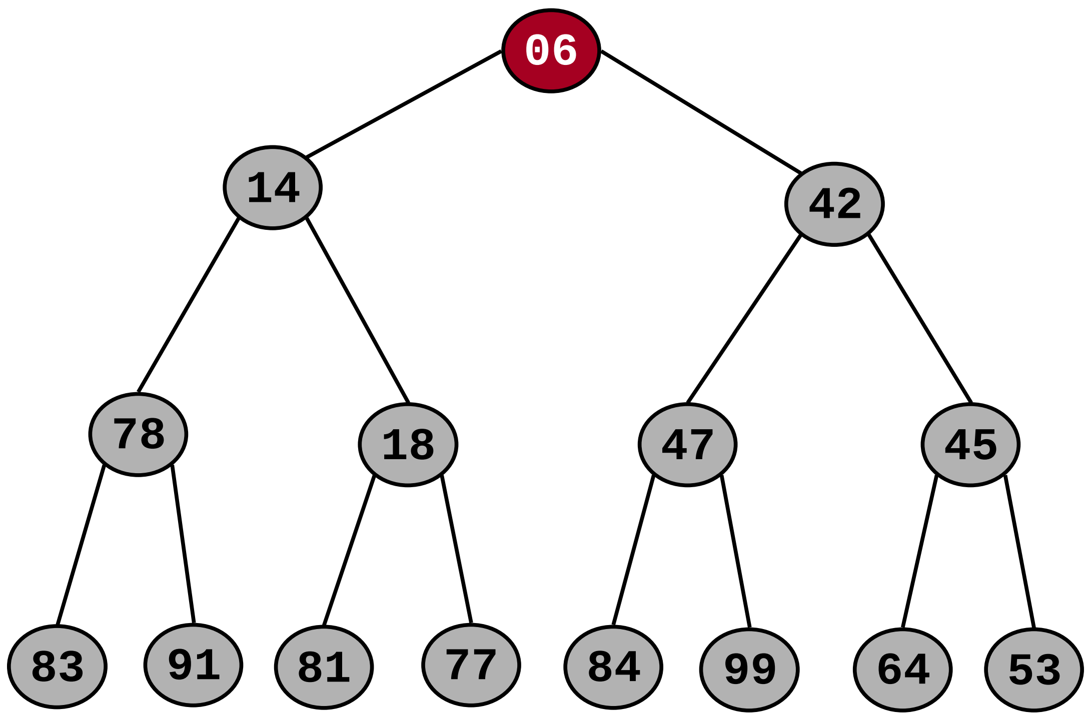
- **Bubble up until it's heap ordered.**
- **O(log N) operations.**



Binary Heap: Delete Min

Delete minimum element from heap.

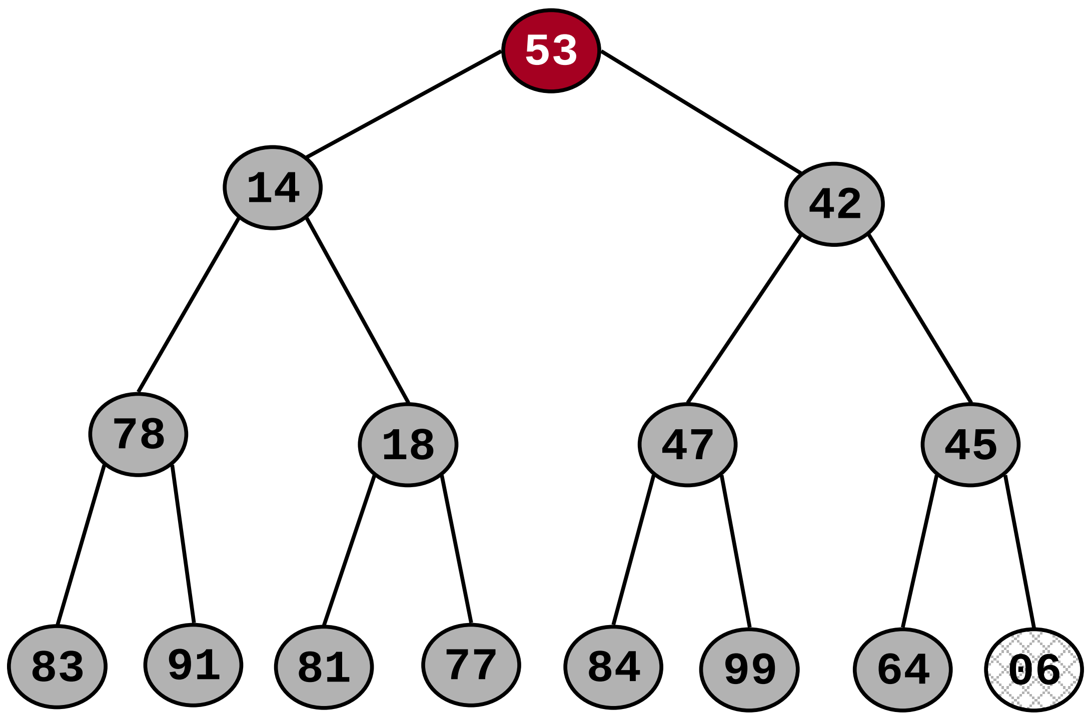
- **Exchange root with rightmost leaf.**
- **Bubble root down until it's heap ordered.**
  - **power struggle principle: better subordinate is promoted**



Binary Heap: Delete Min

Delete minimum element from heap.

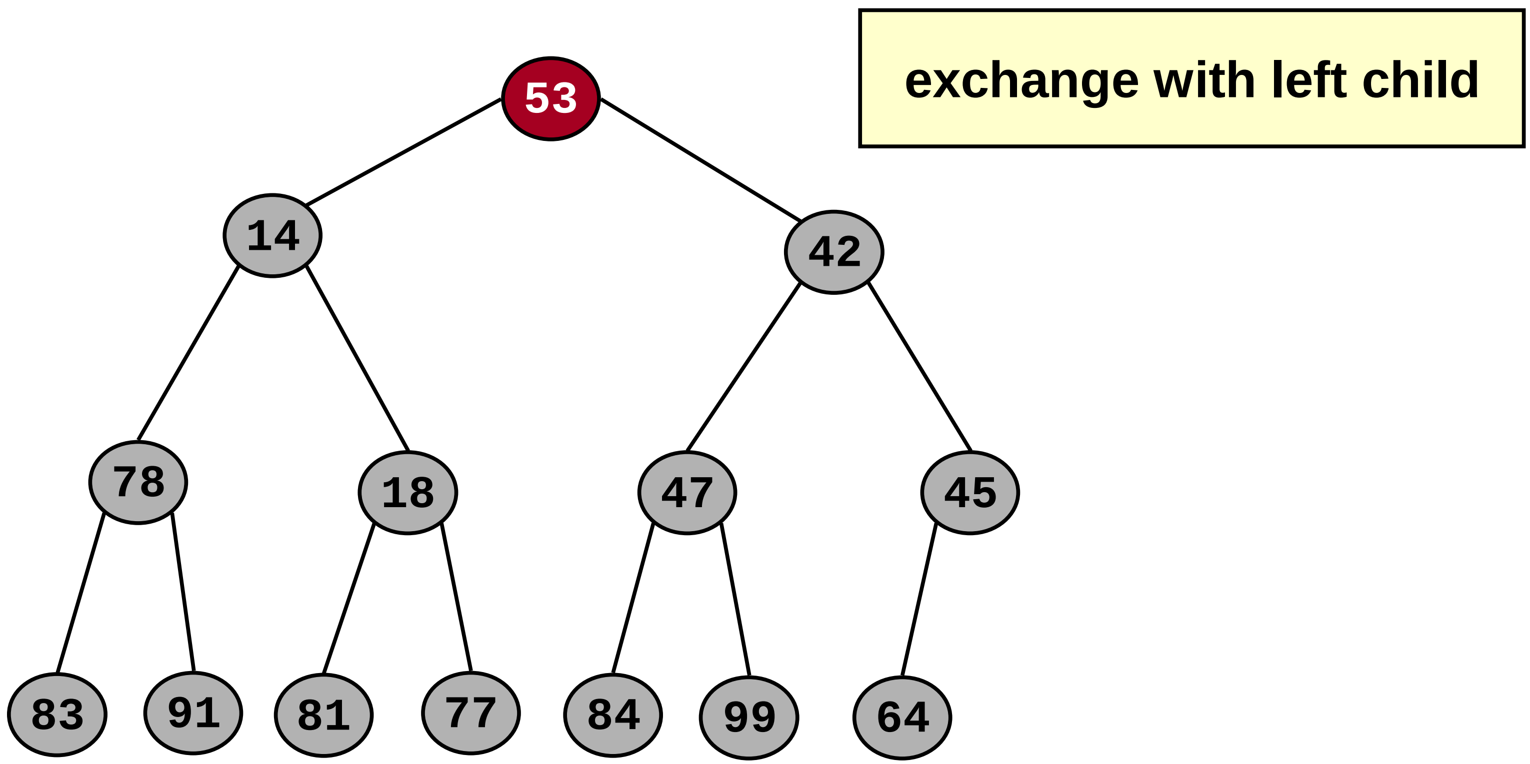
- **Exchange root with rightmost leaf.**
- **Bubble root down until it's heap ordered.**
  - **power struggle principle: better subordinate is promoted**



Binary Heap: Delete Min

Delete minimum element from heap.

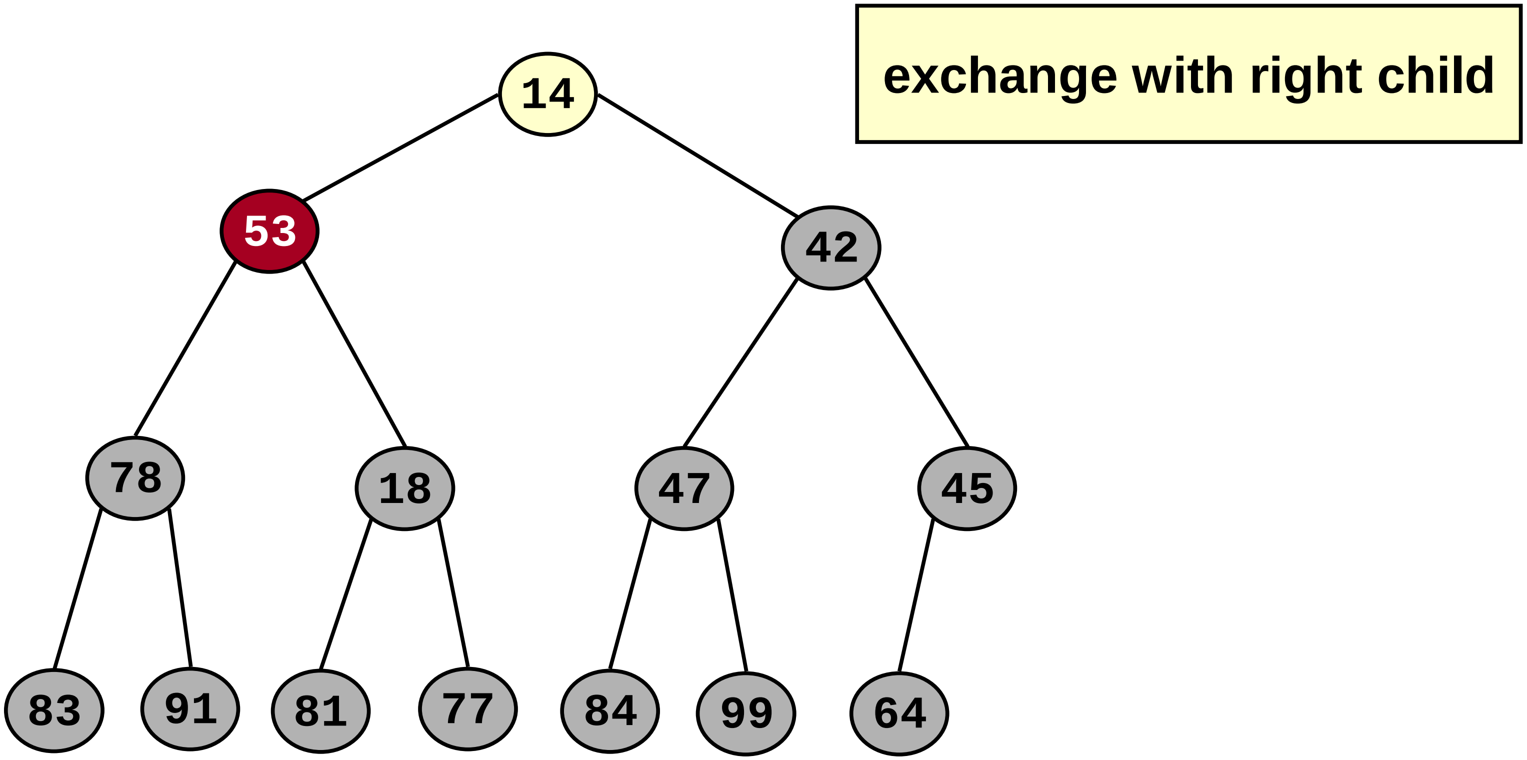
- Exchange root with rightmost leaf.
- Bubble root down until it's heap ordered.
  - power struggle principle: better subordinate is promoted



Binary Heap: Delete Min

Delete minimum element from heap.

- **Exchange root with rightmost leaf.**
- **Bubble root down until it's heap ordered.**
  - **power struggle principle: better subordinate is promoted**

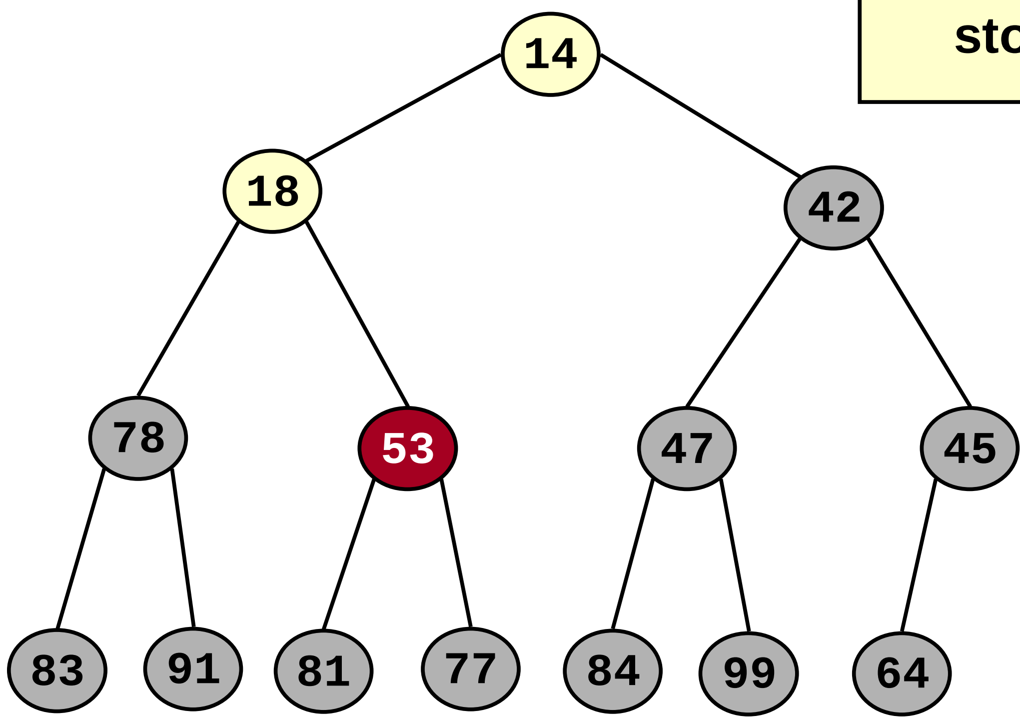


Binary Heap: Delete Min

Delete minimum element from heap.

- Exchange root with rightmost leaf.
- Bubble root down until it's heap ordered.
  - power struggle principle: better subordinate is promoted
- O(log N) operations.

stop: heap ordered



Binary Heap: Heapsort

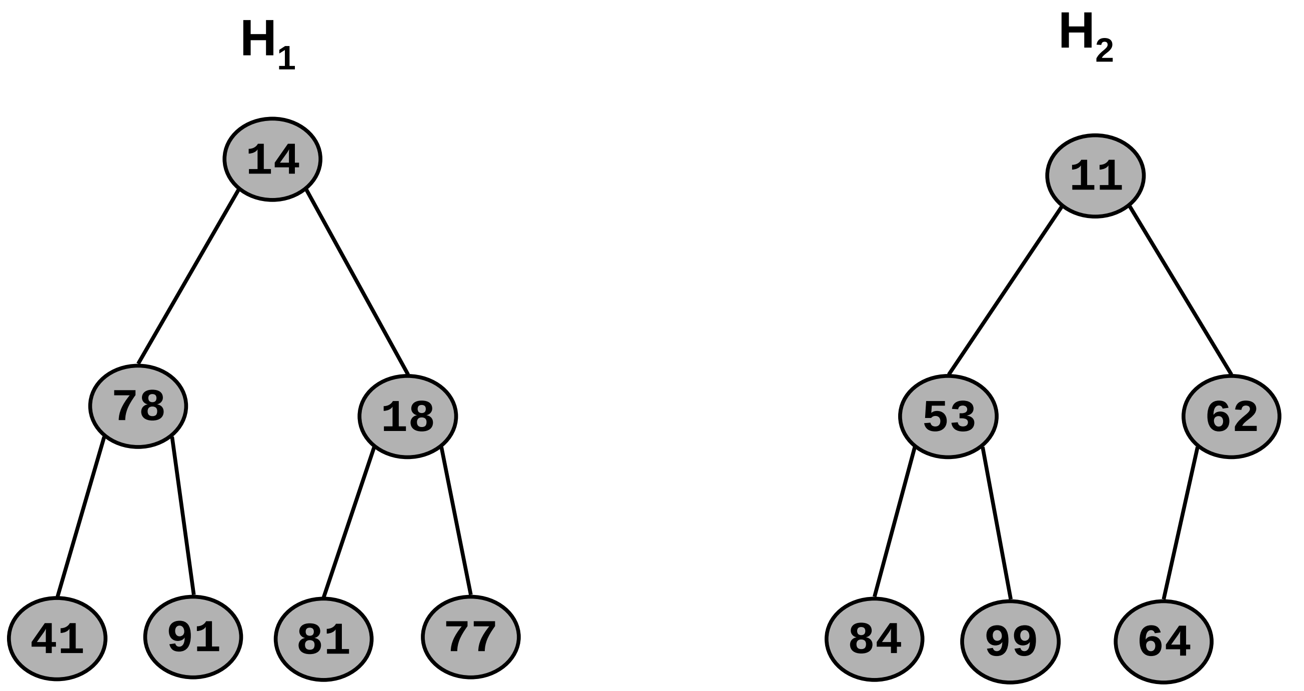
Heapsort.

- **Insert N items into binary heap.**
- **Perform N delete-min operations.**
- **O(N log N) sort.**
- **No extra storage.**

Binary Heap: Union

Union.

- Combine two binary heaps H_1 and H_2 into a single heap.
- No easy solution.
  - $\Omega(N)$ operations apparently required
- Can support fast union with fancier heaps.



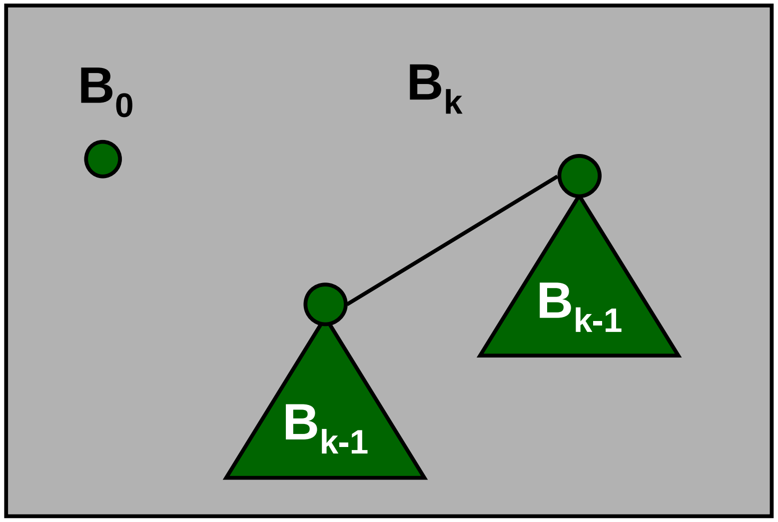
Priority Queues

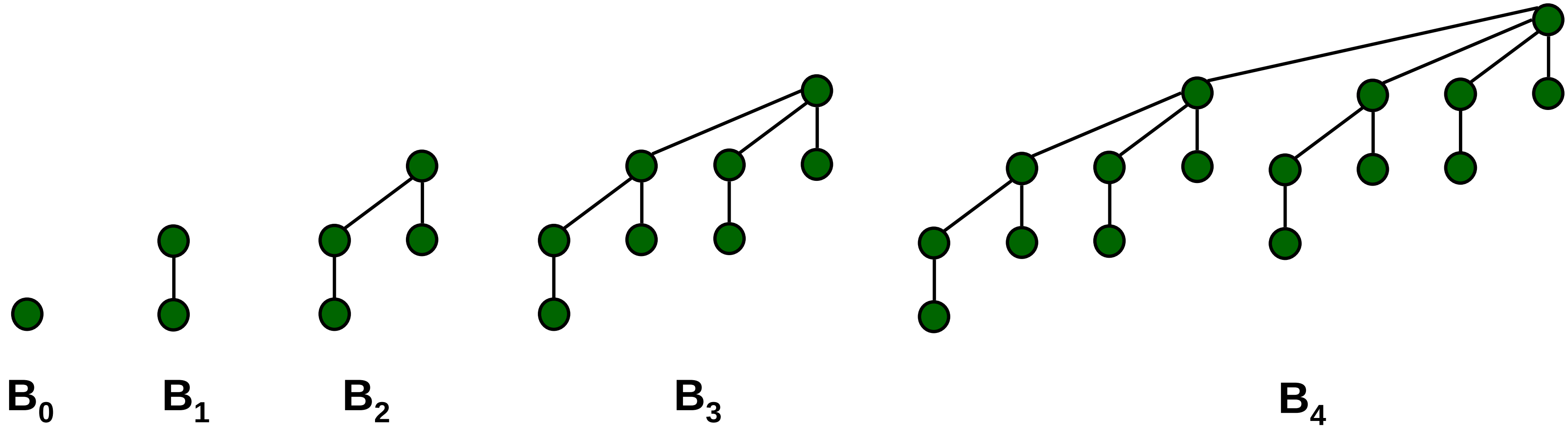
		Heaps			
Operation	**Linked List**	**Binary**	**Binomial**	**Fibonacci ***	**Relaxed**
make-heap	1	1	1	1	1
insert	1	log N	log N	1	1
find-min	N	1	log N	1	1
delete-min	N	log N	log N	log N	log N
union	1	N	log N	1	1
decrease-key	1	log N	log N	1	1
delete	N	log N	log N	log N	log N
is-empty	1	1	1	1	1

Binomial Tree

Binomial tree.

- **Recursive definition:**

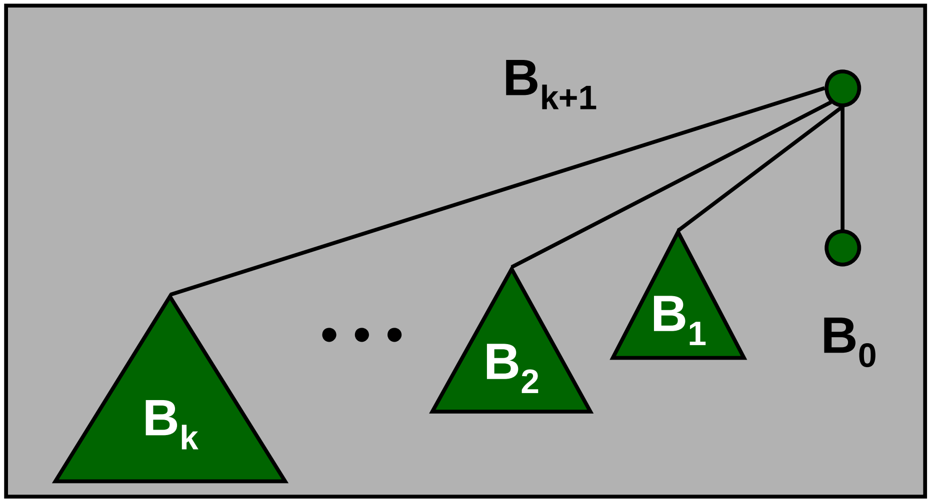




Binomial Tree

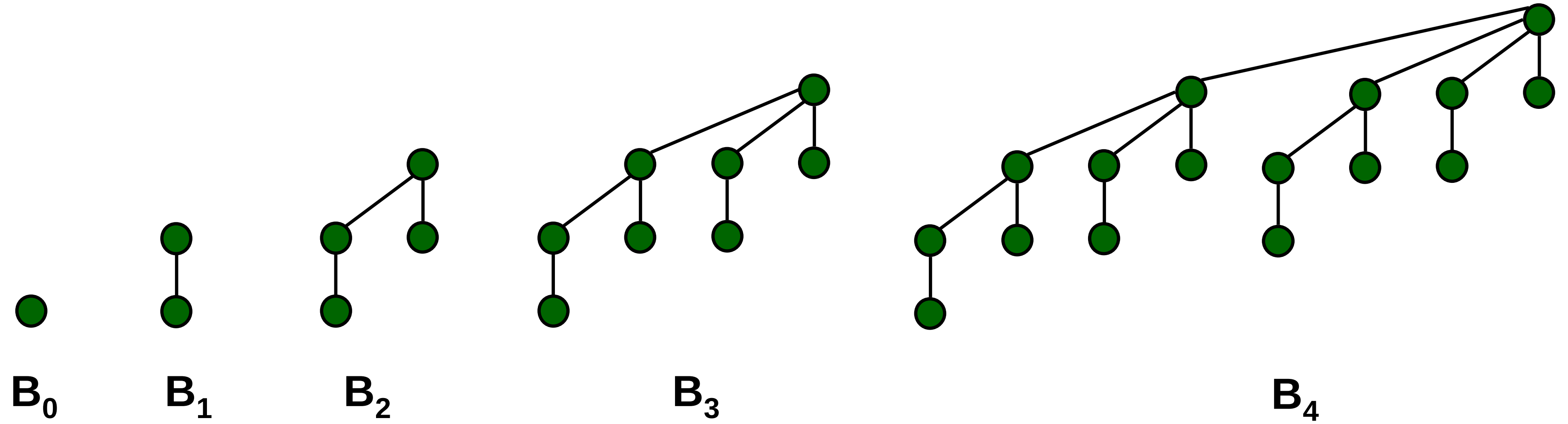
Useful properties of order k binomial tree B_k.

- Number of nodes = 2^k.
- Height = k.
- Degree of root = k.
- Deleting root yields binomial trees $B_{k-1}, \ldots, B_0$.



Proof.

- By induction on k.

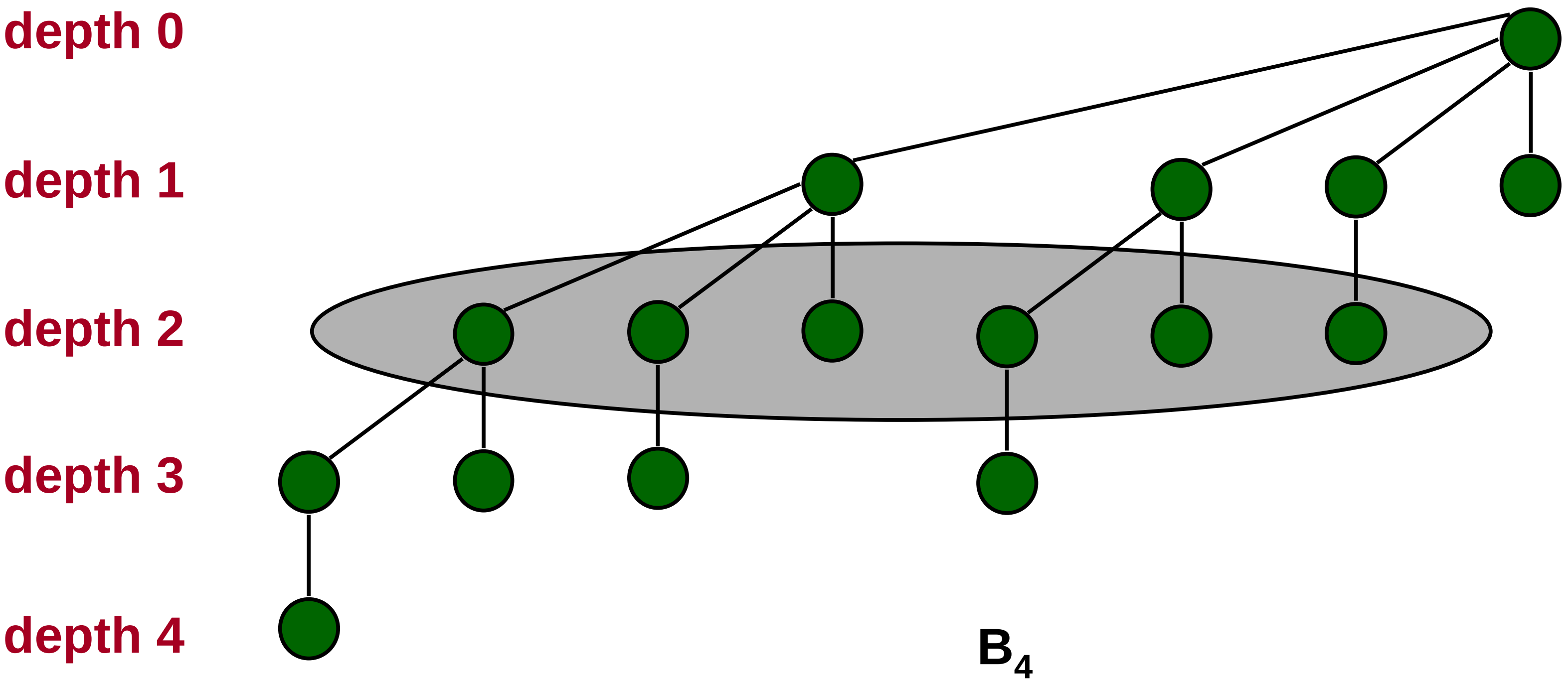


Binomial Tree

A property useful for naming the data structure.

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

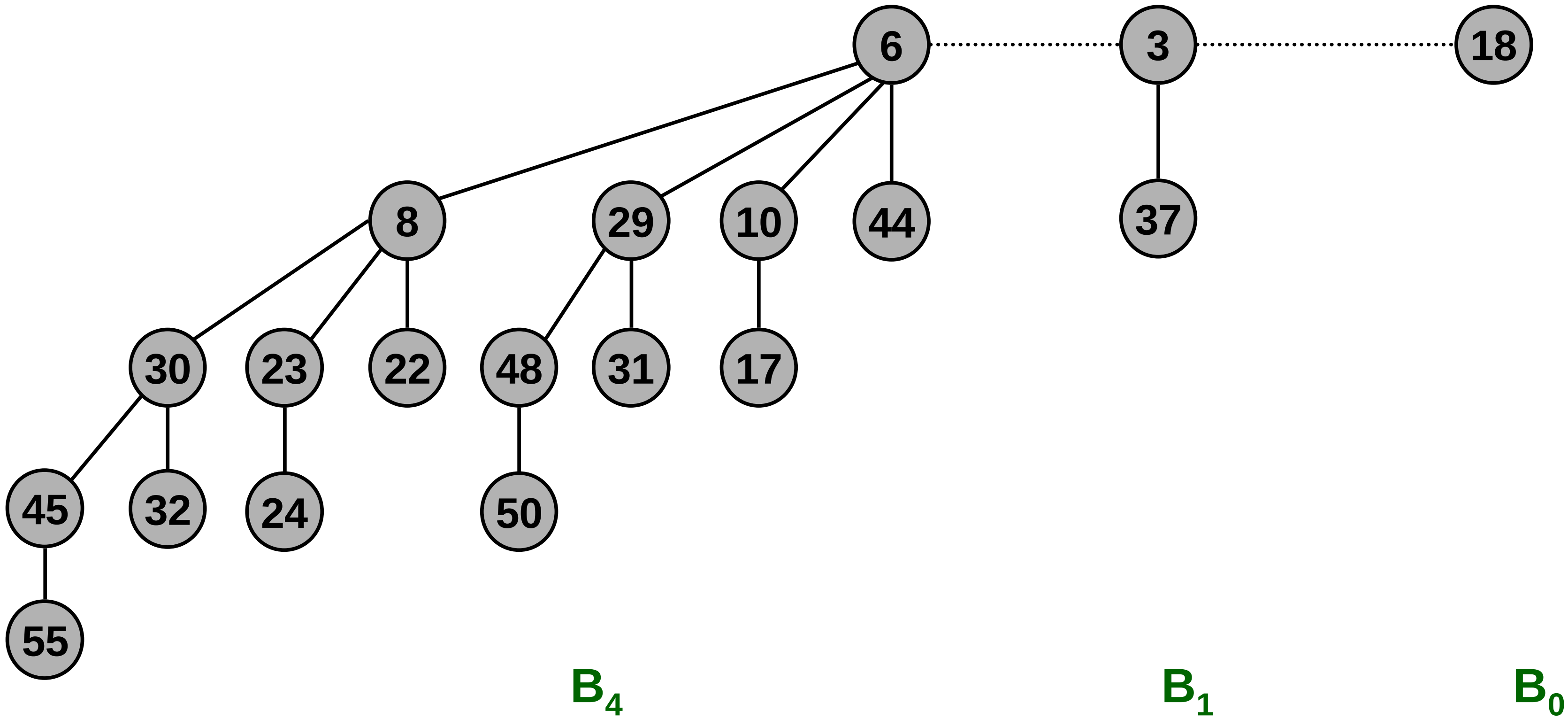
$$\binom{4}{2} = 6$$



Binomial Heap

Binomial heap. **Vuillemin, 1978.**

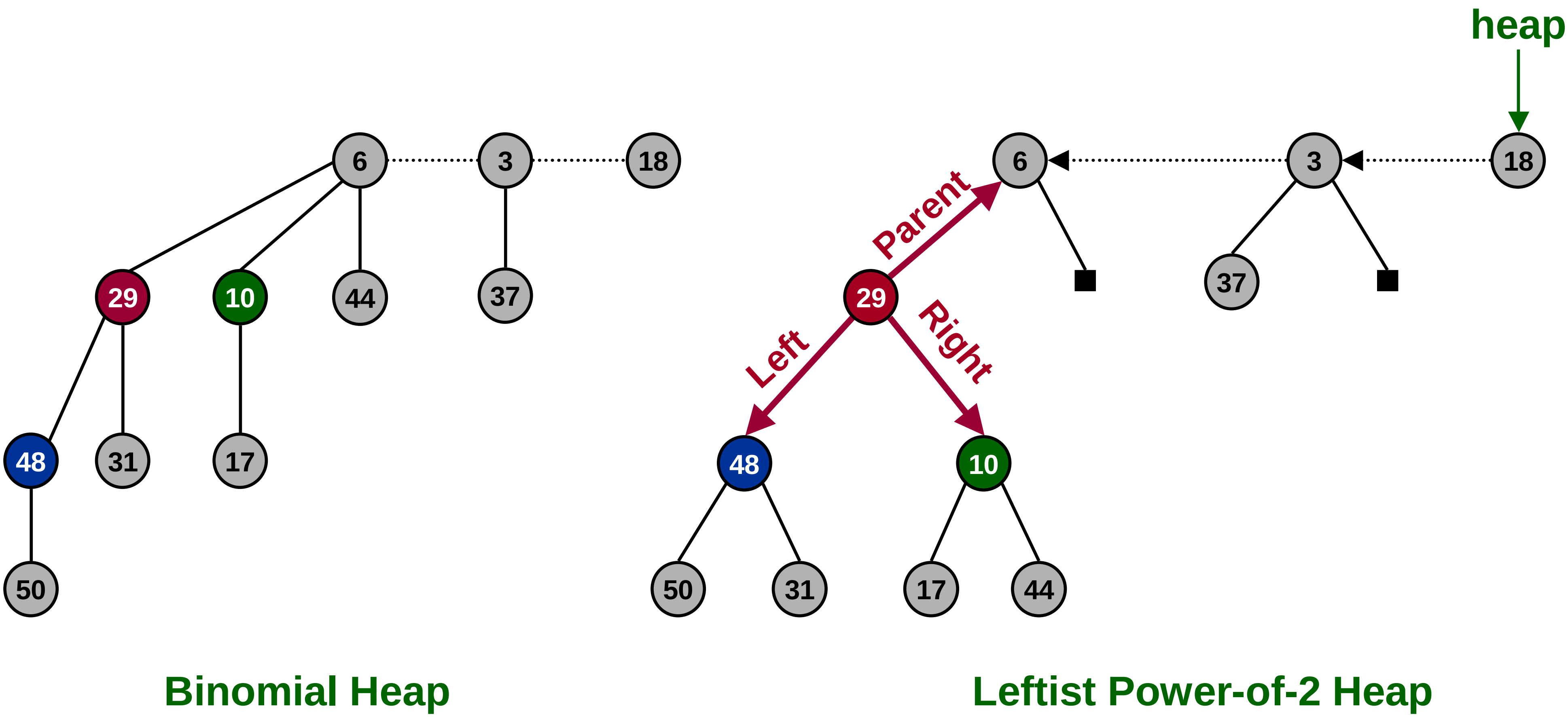
- **Sequence of binomial trees that satisfy binomial heap property.**
  - **each tree is min-heap ordered**
  - **0 or 1 binomial tree of order k**



Binomial Heap: Implementation

Implementation.

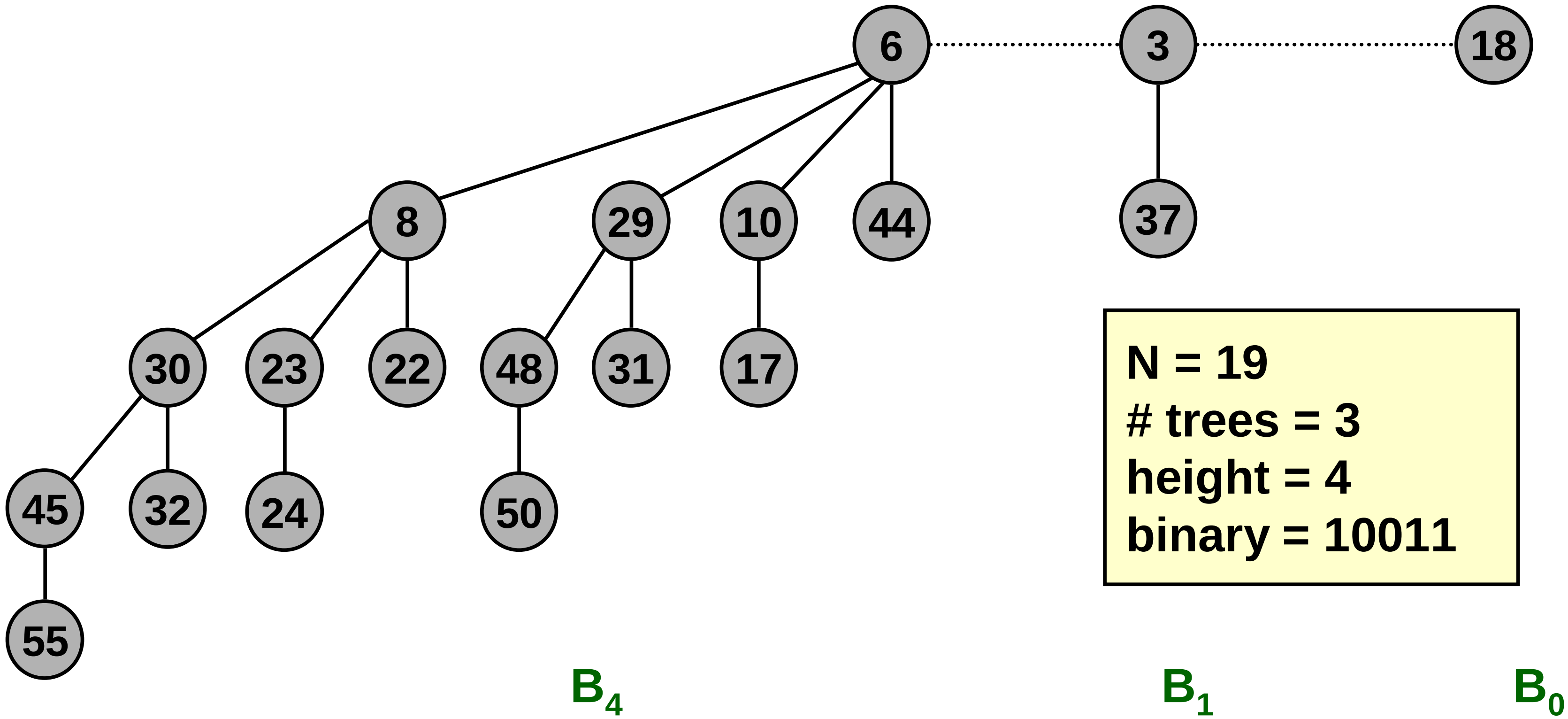
- Represent trees using left-child, right sibling pointers.
  - three links per node (parent, left, right)
- Roots of trees connected with singly linked list.
  - degrees of trees strictly decreasing from left to right



Binomial Heap: Properties

Properties of N-node binomial heap.

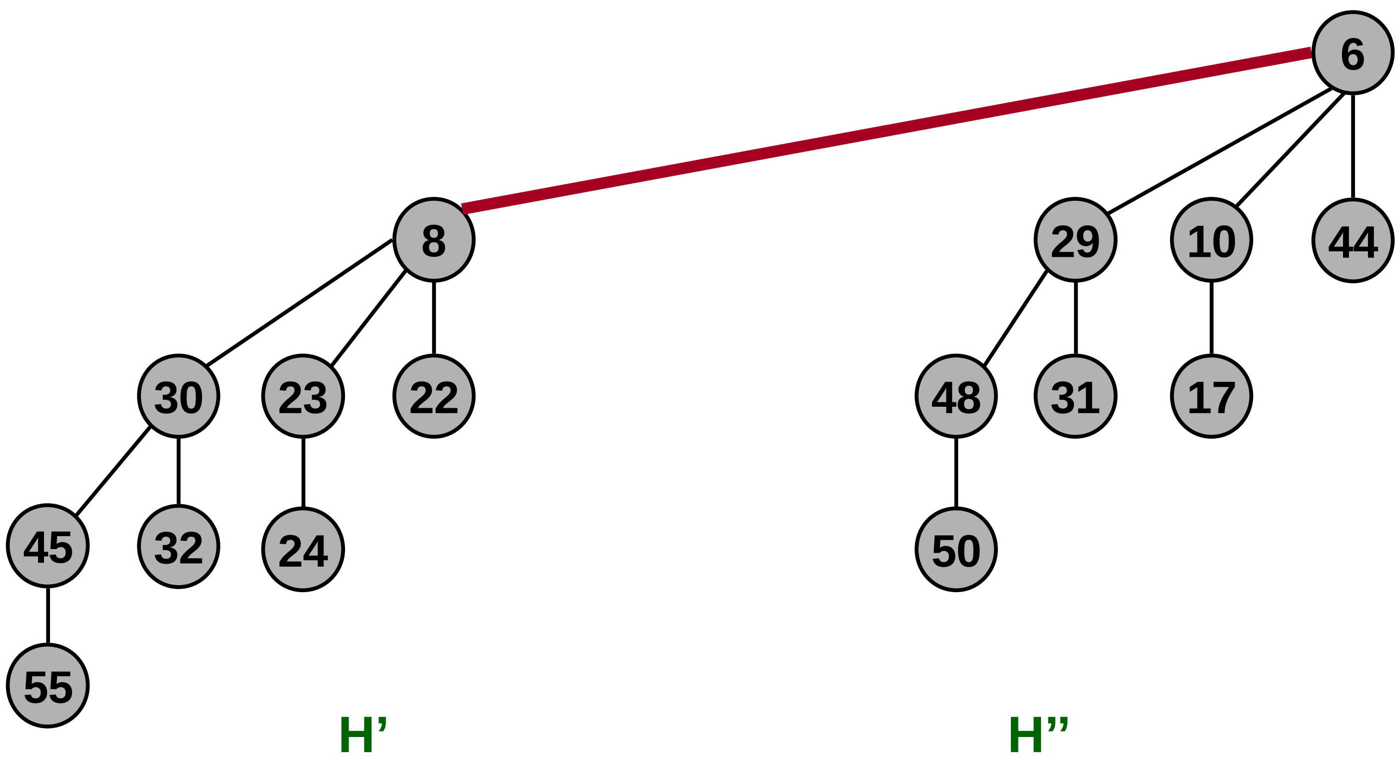
- Min key contained in root of $B_0, B_1, \ldots, B_k$.
- Contains binomial tree B_i iff $b_i = 1$ where $b_n \cdot b_2 b_1 b_0$ is binary representation of N.
- At most $\lfloor \log_2 N \rfloor + 1$ binomial trees.
- Height $\leq \lfloor \log_2 N \rfloor$.



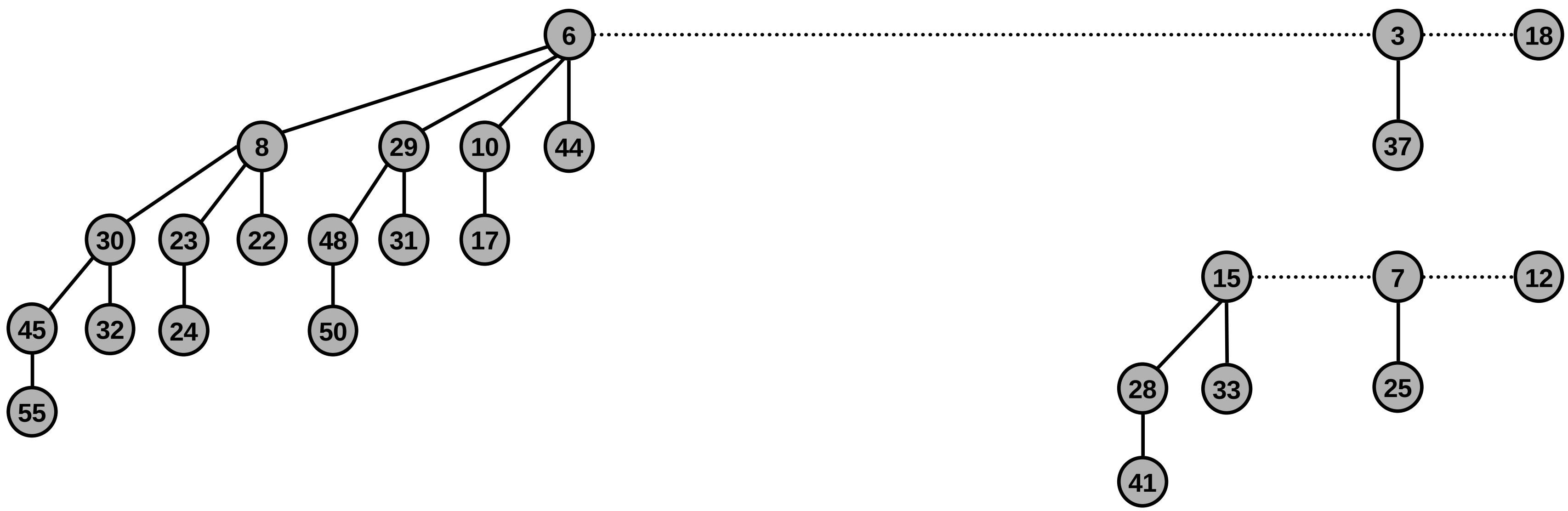
Binomial Heap: Union

Create heap H that is union of heaps H' and H''.

- "Mergeable heaps."
- Easy if H' and H'' are each order k binomial trees.
  - connect roots of H' and H''
  - choose smaller key to be root of H



Binomial Heap: Union

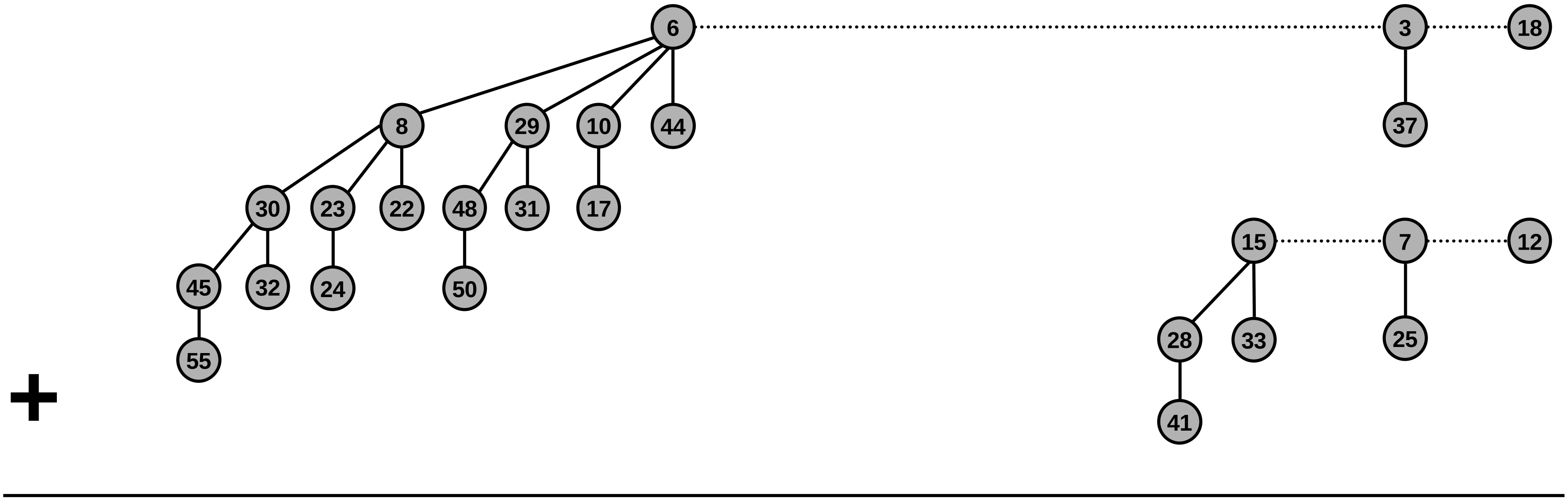


+

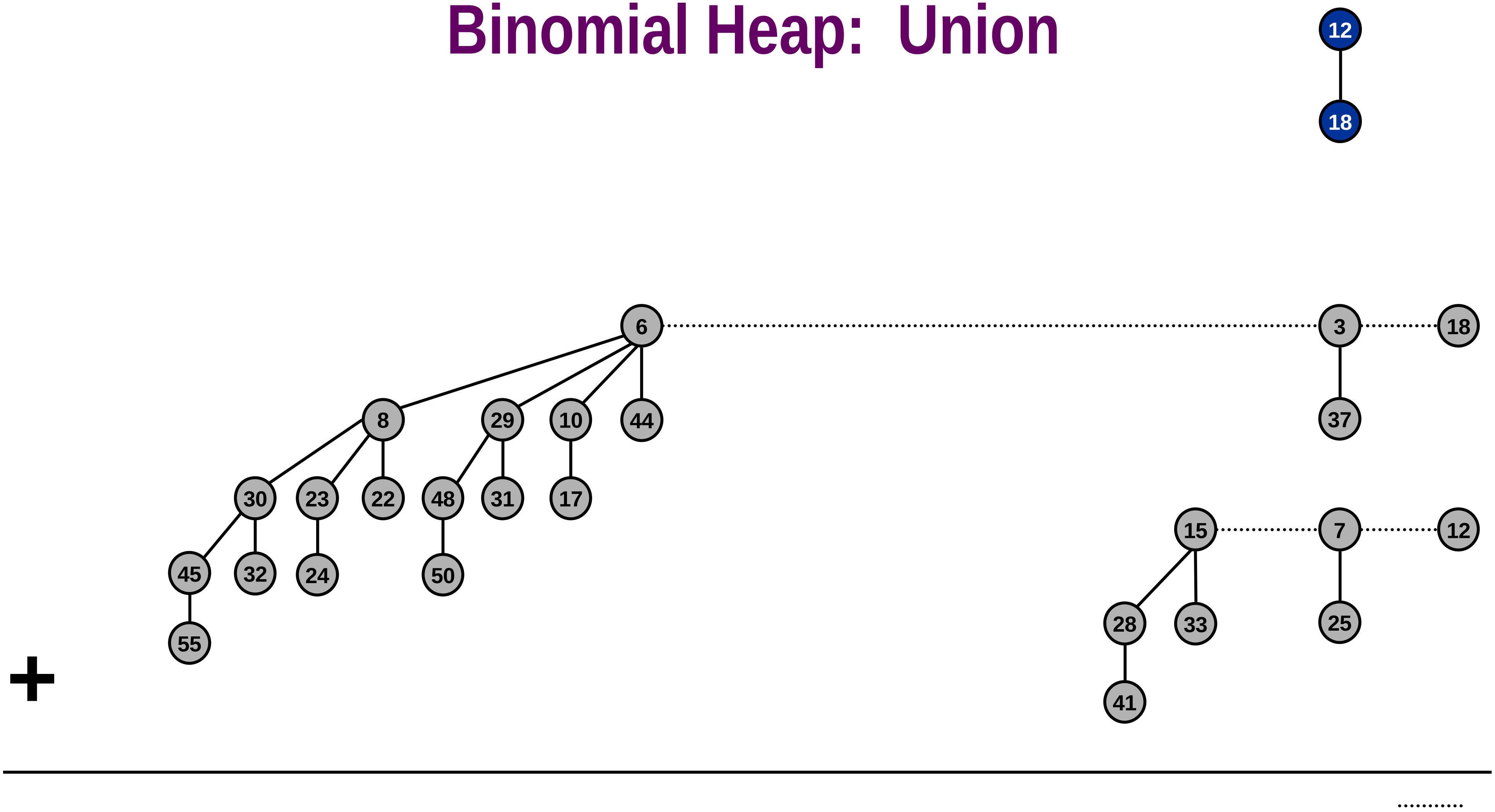
19 + 7 = 26

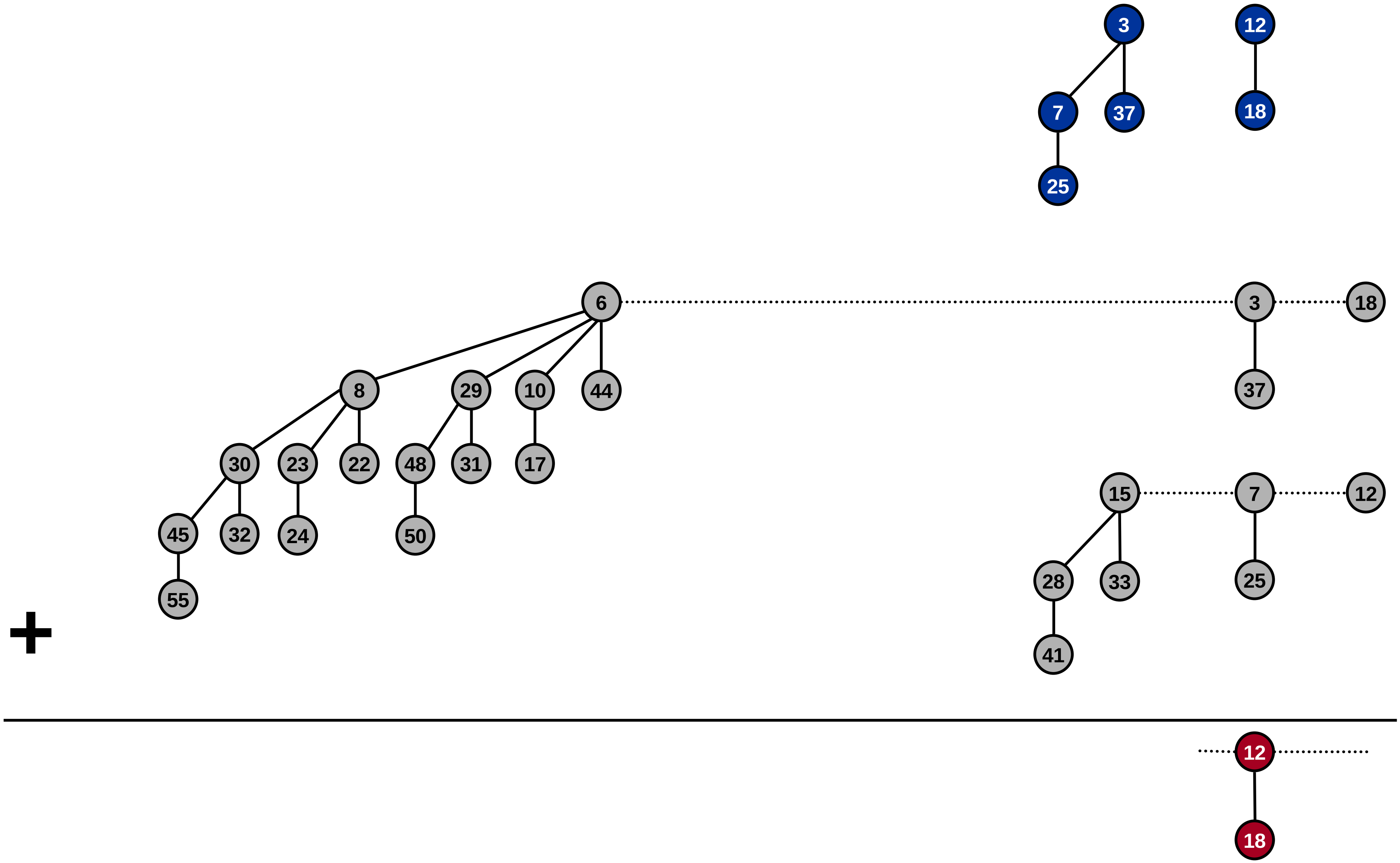
		1	1	1	
	1	0	0	1	1
+	0	0	1	1	1
	1	1	0	1	0

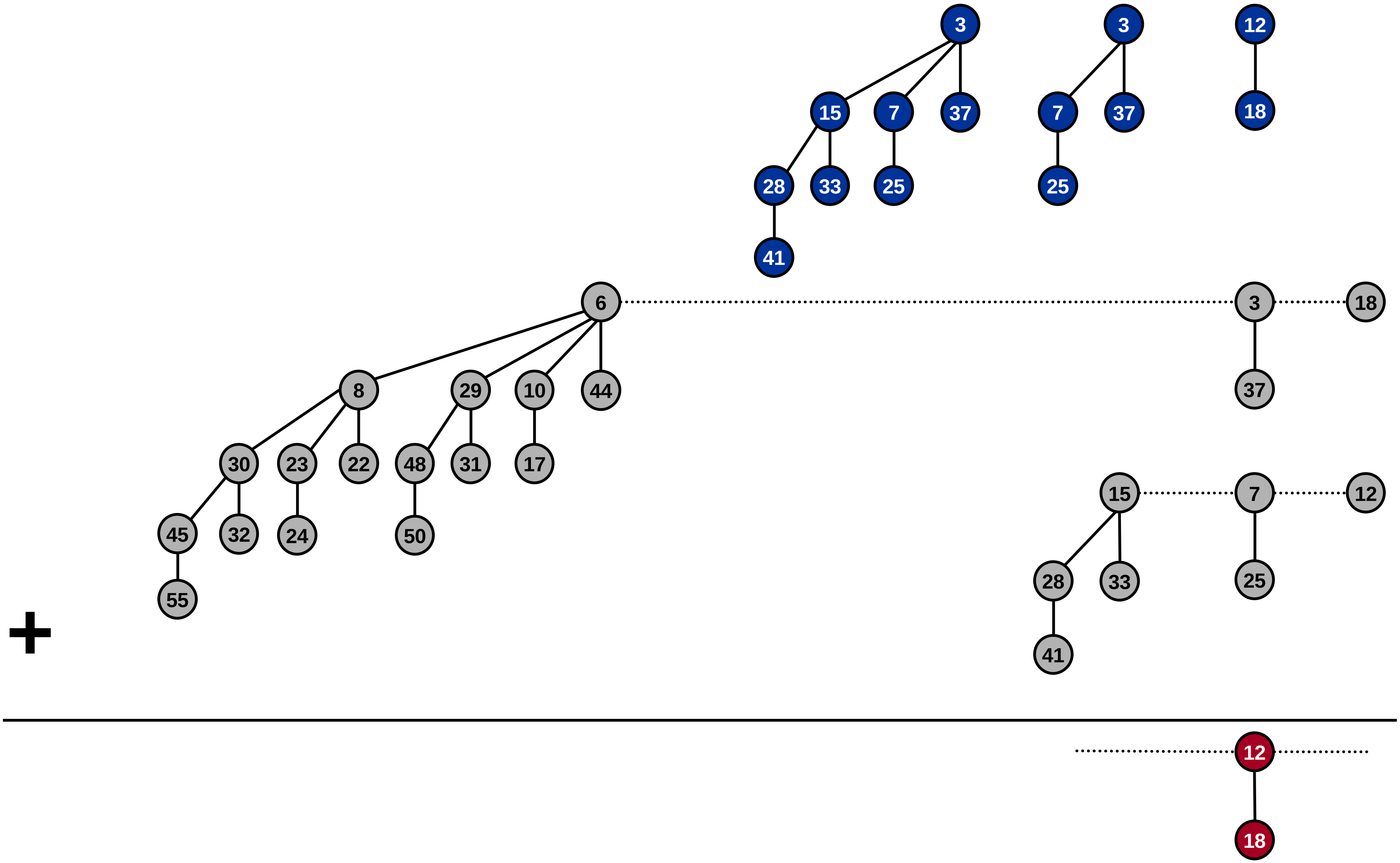
Binomial Heap: Union

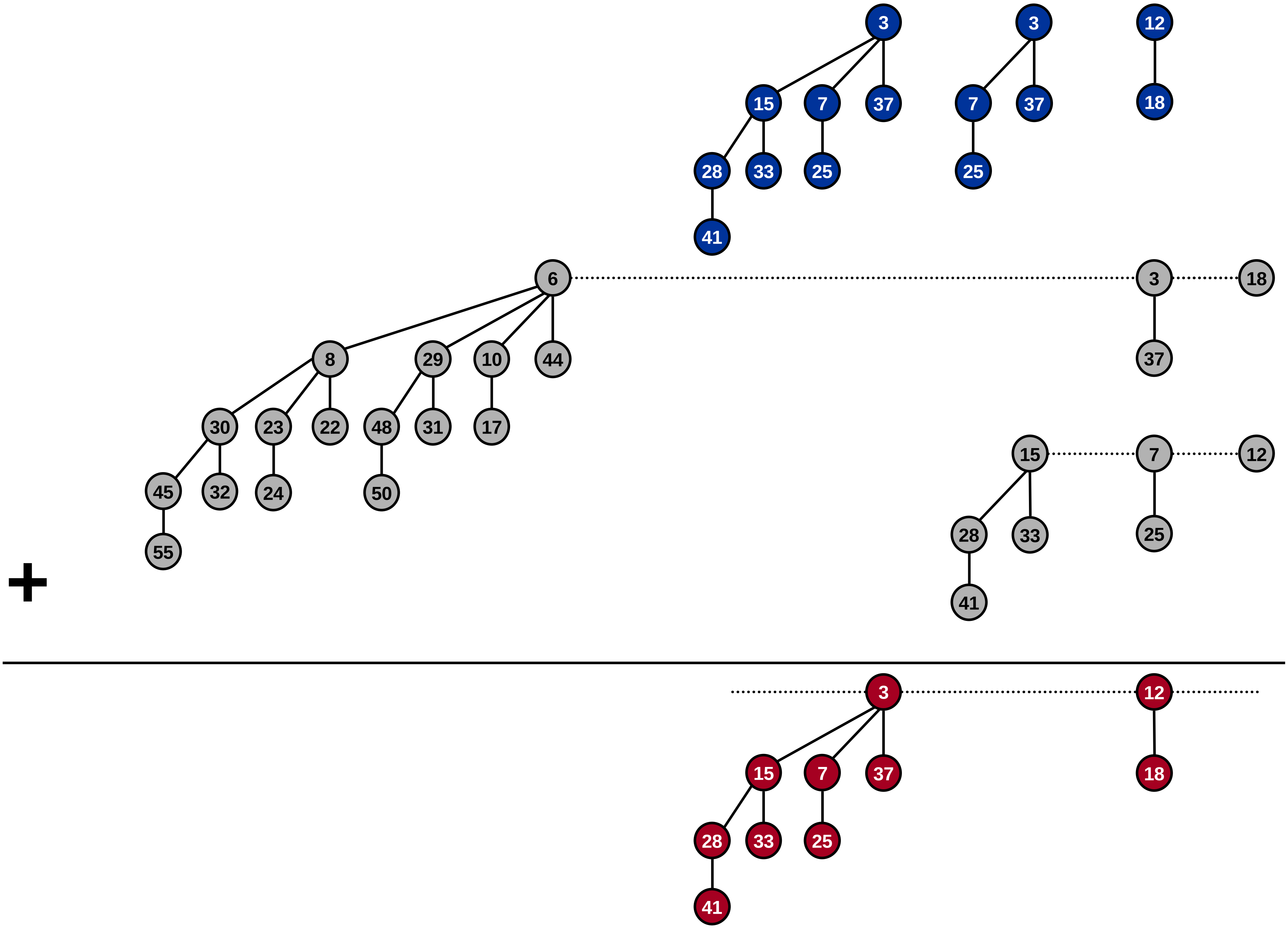


Binomial Heap: Union

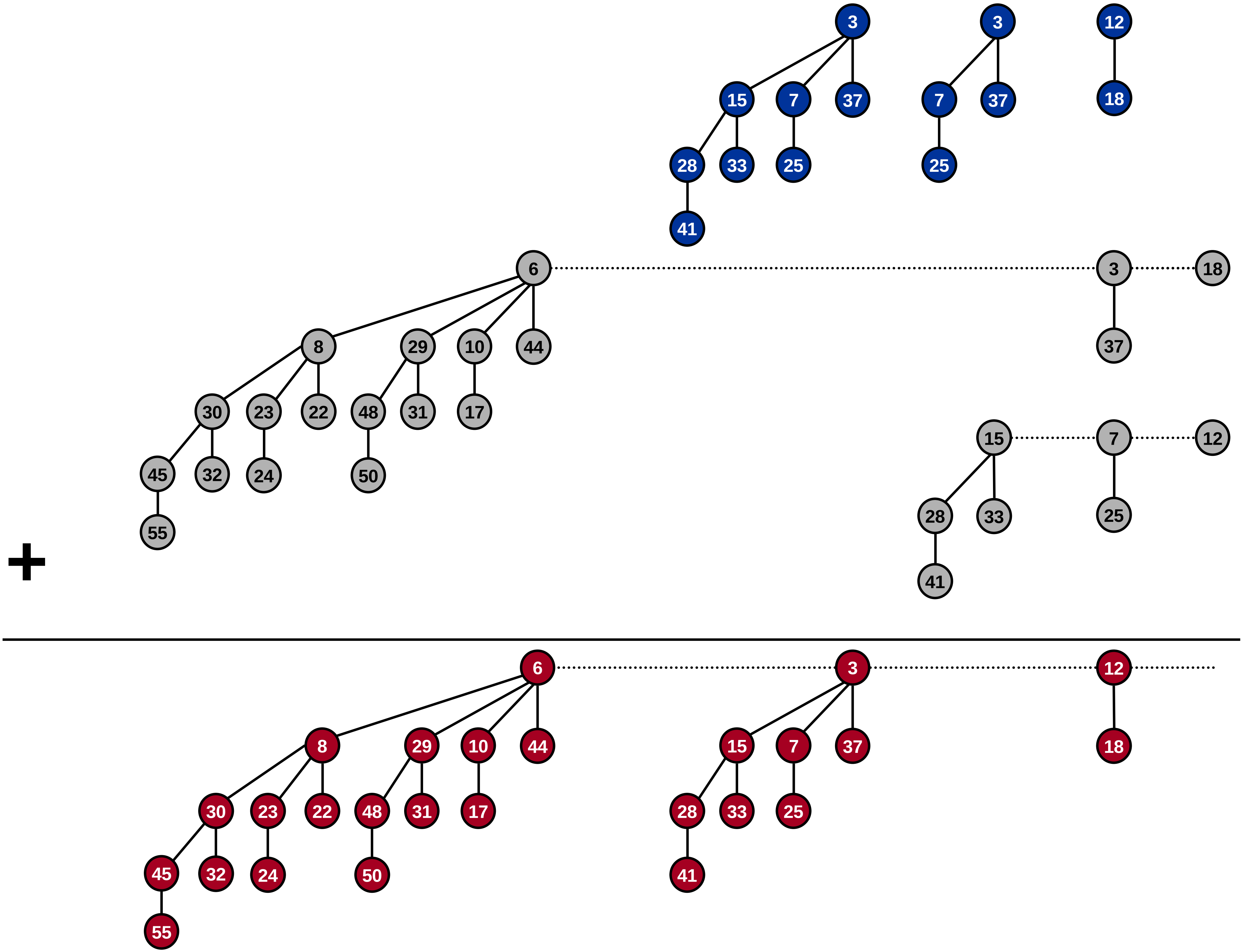



3
12
7
37
18
25
6
3
18
8
29
10
44
37
30
23
22
48
31
17
15
7
12
45
32
24
50
28
33
25
55
+
41
12
18


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

Binomial Heap: Union

Create heap H that is union of heaps H' and H''.

- Analogous to binary addition.

Running time. O(log N)

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

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

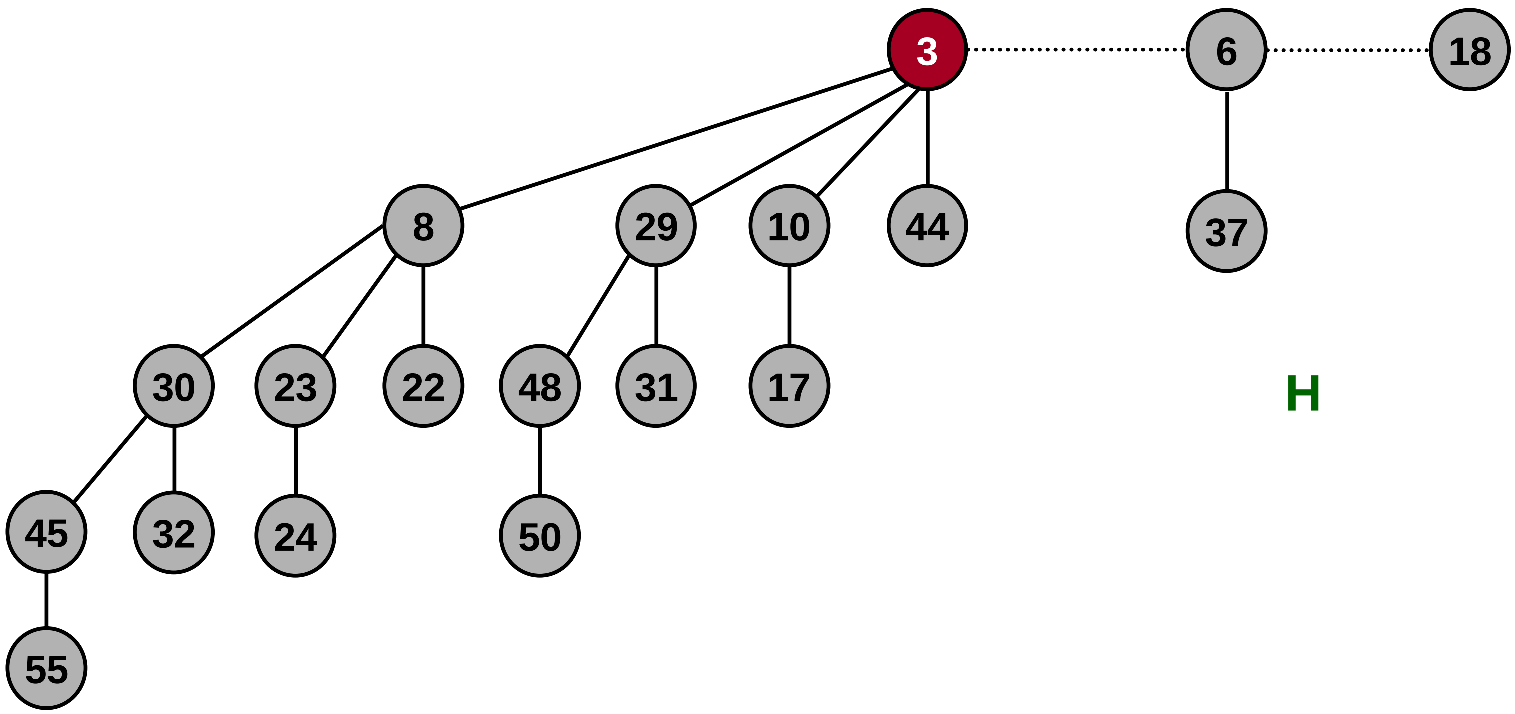
19 + 7 = 26

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' $\leftarrow$ broken binomial trees
- H $\leftarrow$ Union(H', H)

Running time. O(log N)

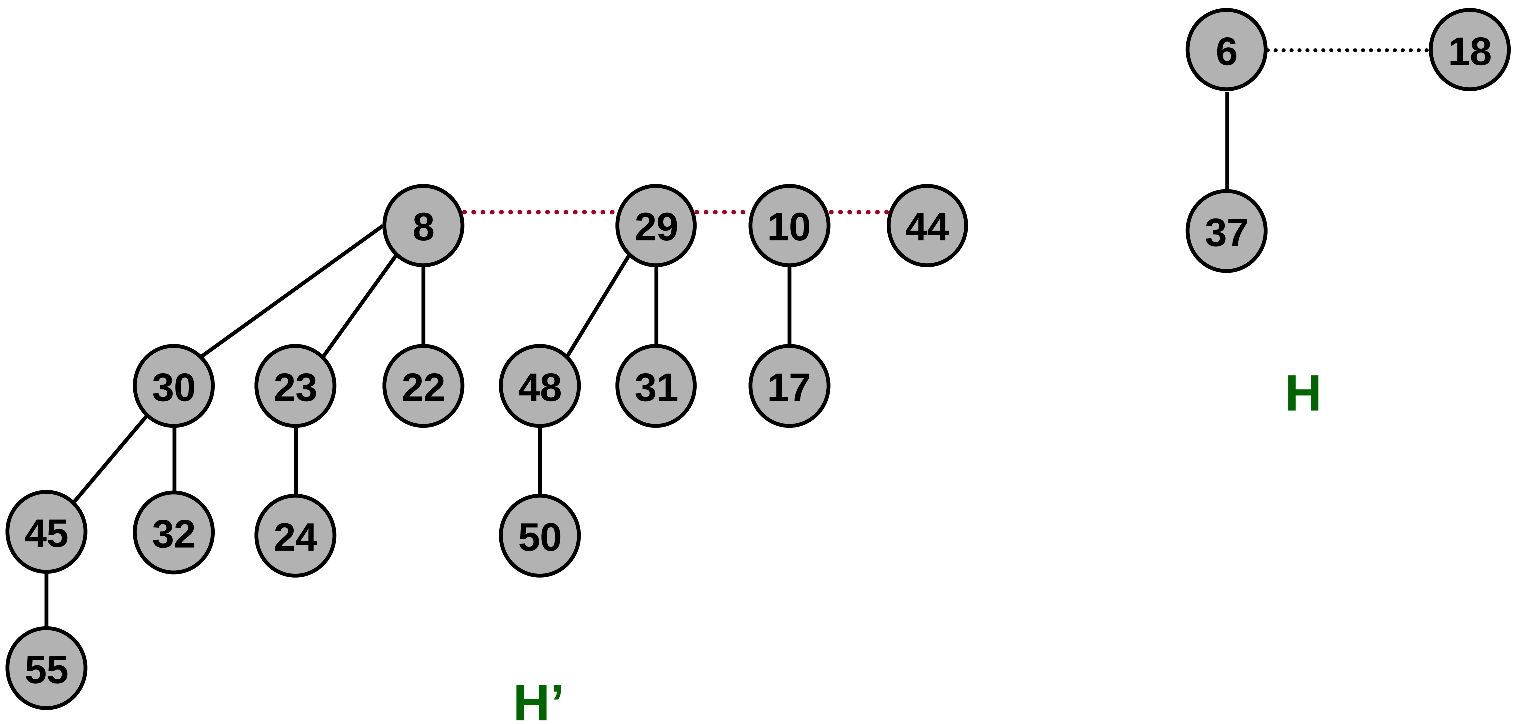


Binomial Heap: Delete Min

Delete node with minimum key in binomial heap H.

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

Running time. O(log N)



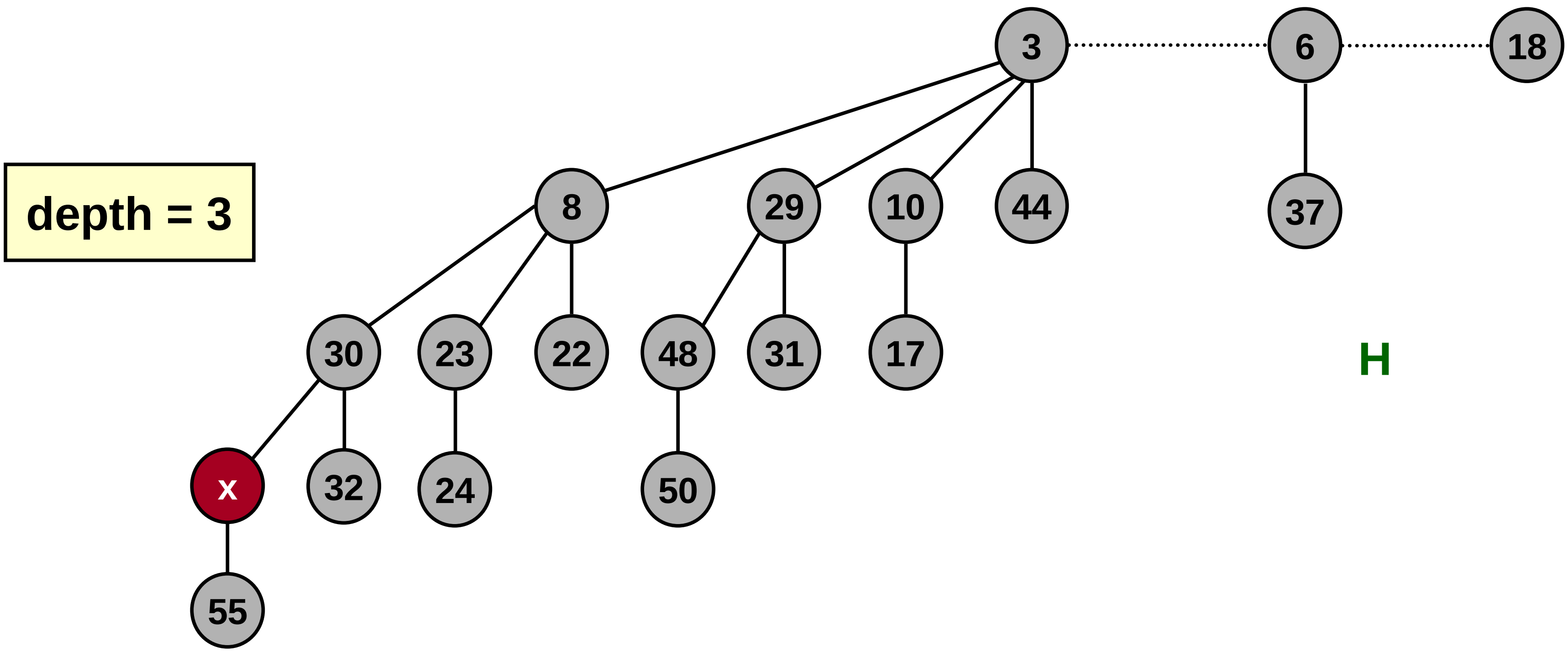
Binomial Heap: Decrease Key

Decrease key of node x in binomial heap H.

- **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 $\le \lfloor \log_2 N \rfloor$.**



Binomial Heap: Delete

Delete node x in binomial heap H.

- Decrease key of x to $-\infty$.
- Delete min.

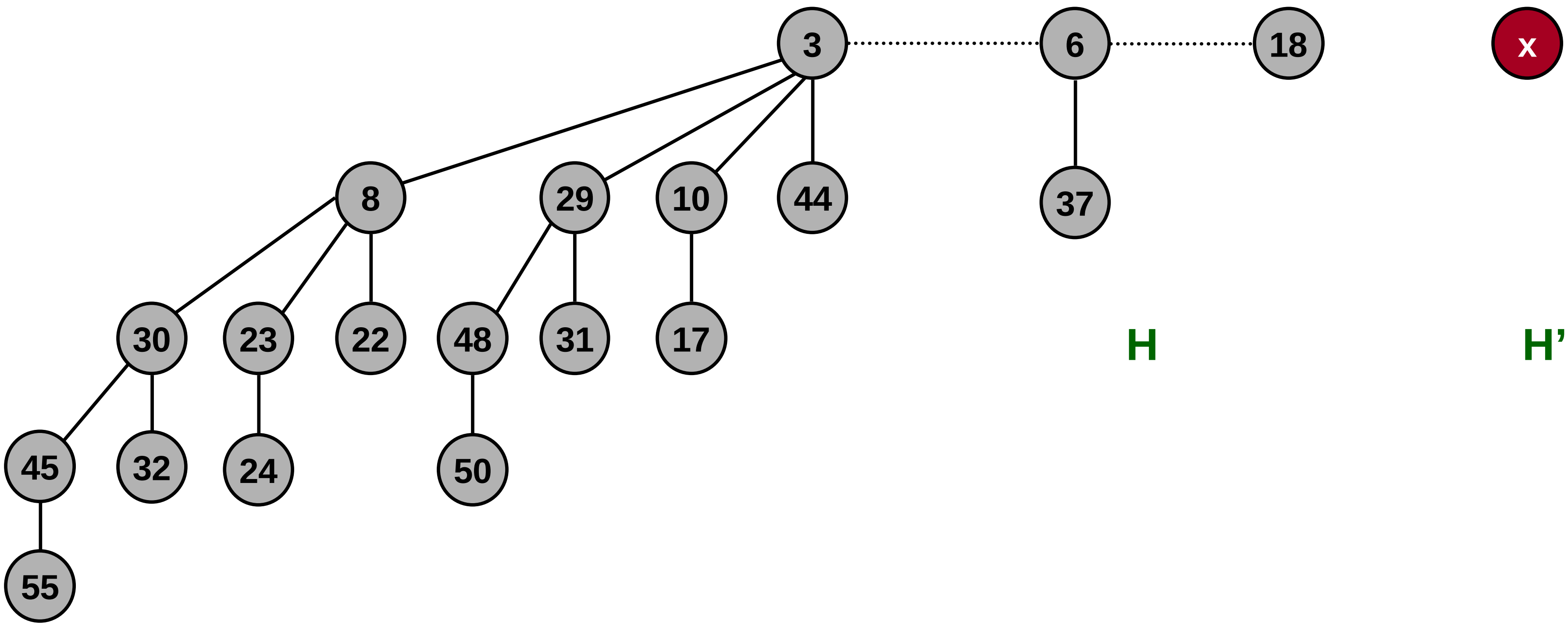
Running time. O(log N)

Binomial Heap: Insert

Insert a new node x into binomial heap H.

- **H' ← MakeHeap(x)**
- **H ← Union(H', H)**

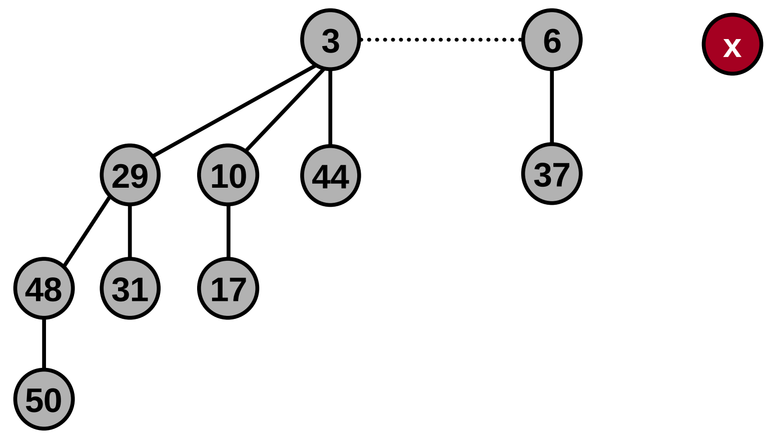
Running time. O(log N)



Binomial Heap: Sequence of Inserts

Insert a new node x into binomial heap H.

- If N = `.......0`, then only 1 steps.
- If N = `......01`, then only 2 steps.
- If N = `.....011`, then only 3 steps.
- If N = `....0111`, then only 4 steps.



Inserting 1 item can take Ω(log N) time.

- If N = `11...111`, then $\log_2 N$ steps.

But, inserting sequence of N items takes O(N) time!

- (N/2)(1) + (N/4)(2) + (N/8)(3) + . . . $\leq$ 2N
- Amortized analysis.
- Basis for getting most operations down to constant time.

$$\sum_{n=1}^{N} \frac{n}{2^n} = 2 - \frac{N}{2^N} - \frac{1}{2^{N-1}} \leq 2$$

Priority Queues

		Heaps			
Operation	**Linked List**	**Binary**	**Binomial**	**Fibonacci ***	**Relaxed**
make-heap	1	1	1	1	1
insert	1	log N	log N	1	1
find-min	N	1	log N	1	1
delete-min	N	log N	log N	log N	log N
union	1	N	log N	1	1
decrease-key	1	log N	log N	1	1
delete	N	log N	log N	log N	log N
is-empty	1	1	1	1	1

just did this