Analysis of Algorithms (Fall 2013) Istanbul Technical University Computer Eng. Dept.

Chapter 19

Binomial Heaps

Course slides from Kevin Wayne @Princeton have been used in preparation of these slides.

Last updated: December 23, 2009

Purpose

- Understand data structures known as mergeable heaps that can support
  - MAKE-HEAP(), INSERT(H, x), MINIMUM(H), EXTRACT-MIN(H), UNION(H_1, H_2)
  - DECREASE-KEY(H, x, k), DELETE(H, x)

Outline

- Binomial Trees and Binomial Heaps
- Operations on Binomial Heaps
  - Creating New Binomial Heap
  - Finding Minimum Key
  - Uniting Two Binomial Heaps
  - Inserting Node
  - Extracting Node with Minimum Key
  - Deleting Key
  - Decreasing Key

Priority Queues

- Supports 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, etc.

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

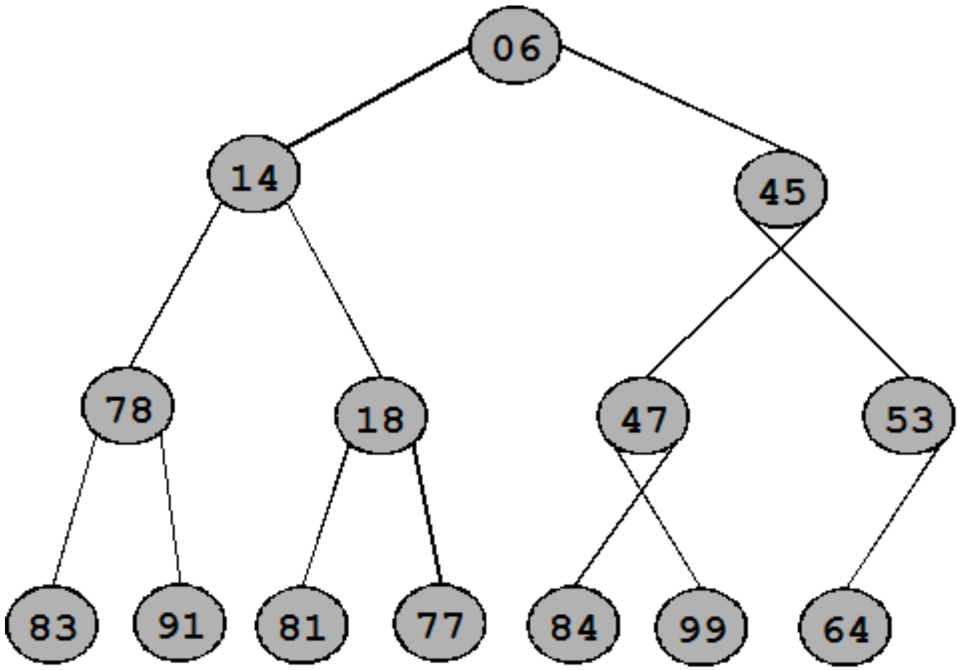
$O(|V|^2)$

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

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

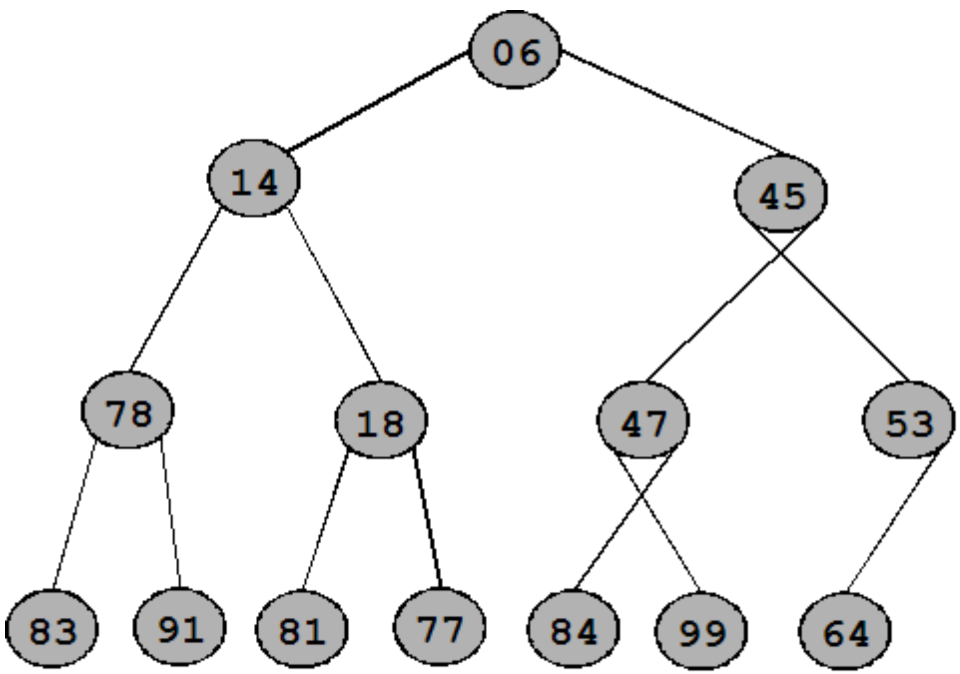
Binary Heap: Definition

- 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

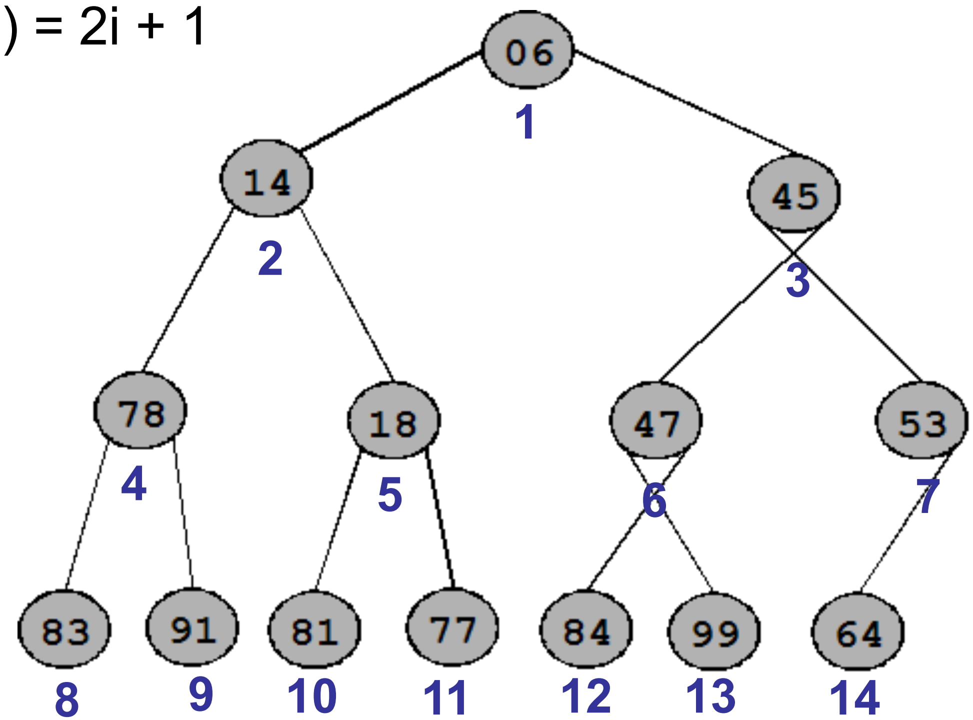
- Min element is in root
- Heap with N elements has height = $\lfloor \log_2 N \rfloor$



N = 14
Height = 3

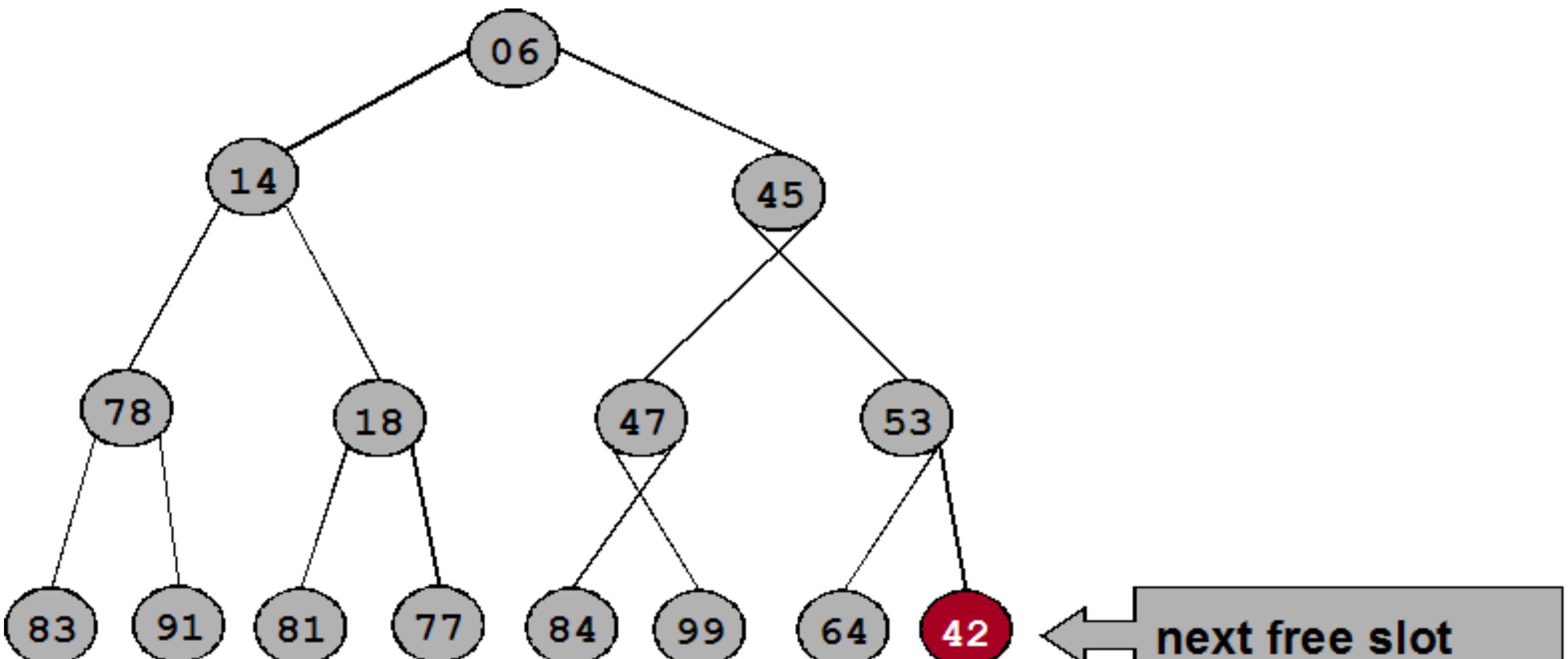
Implementing Binary Heaps: Array Implementation

- Use an array: no need for explicit parent or child pointers.
  - Parent(i) = $\lfloor i/2 \rfloor$
  - Left(i) = 2i
  - Right(i) = 2i + 1



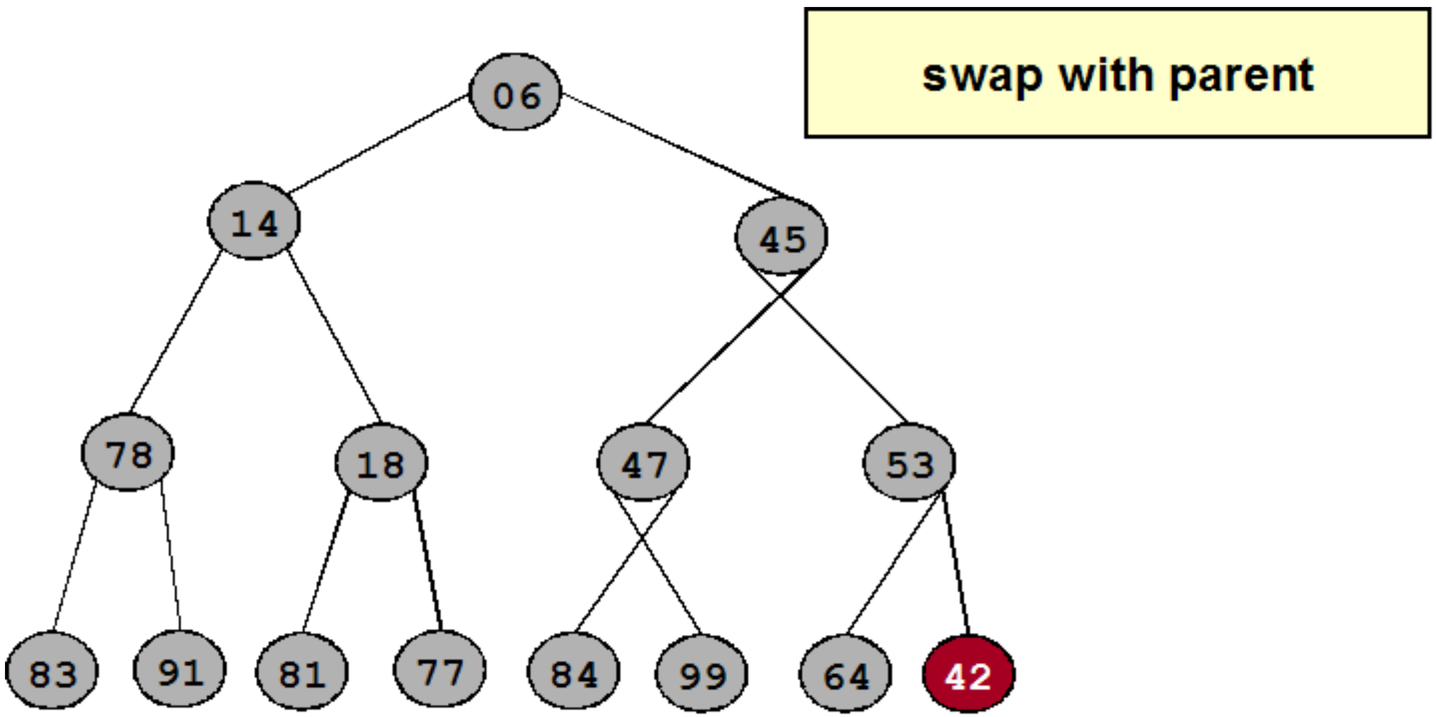
Binary Heaps: Insertion

- Insert element x into heap
  - Insert into next available slot
  - Bubble up until it is heap ordered



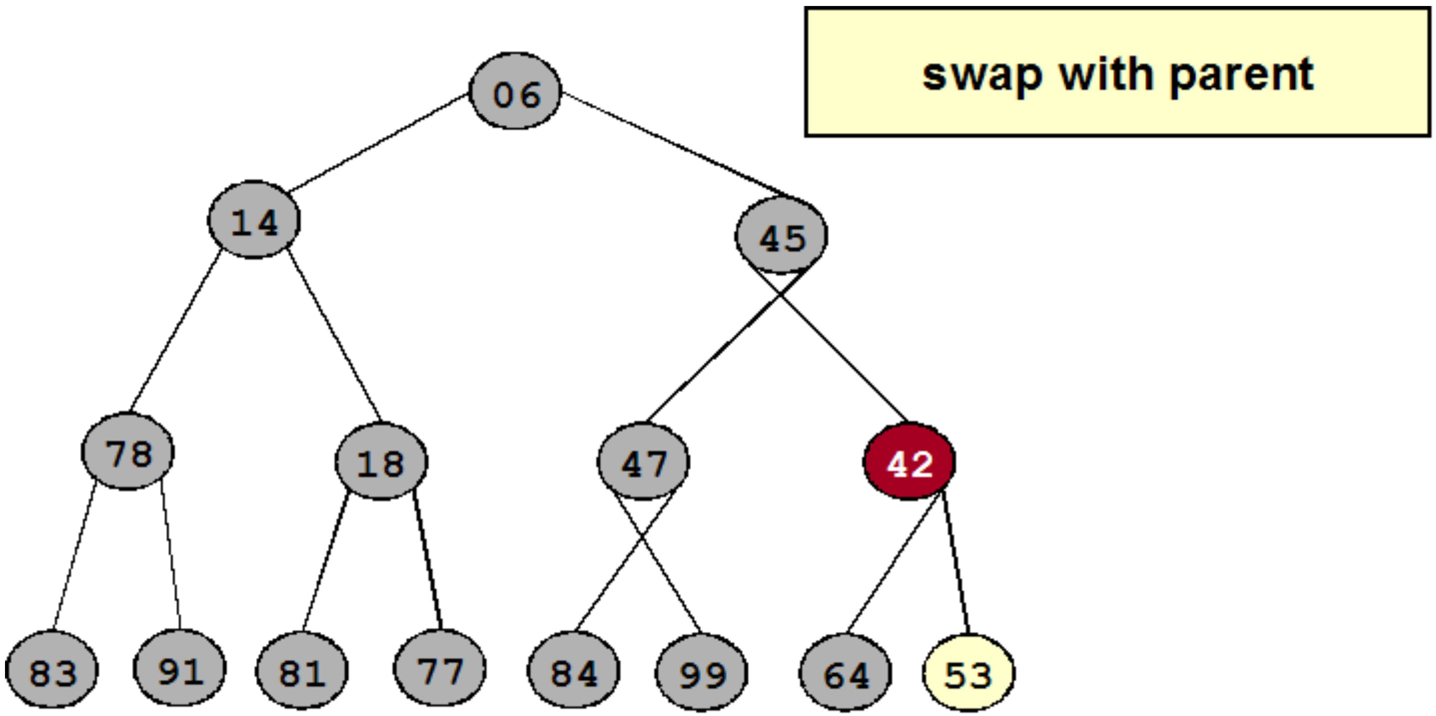
Binary Heaps: Insertion

- Insert element x into heap
  - Insert into next available slot
  - Bubble up until it is heap ordered



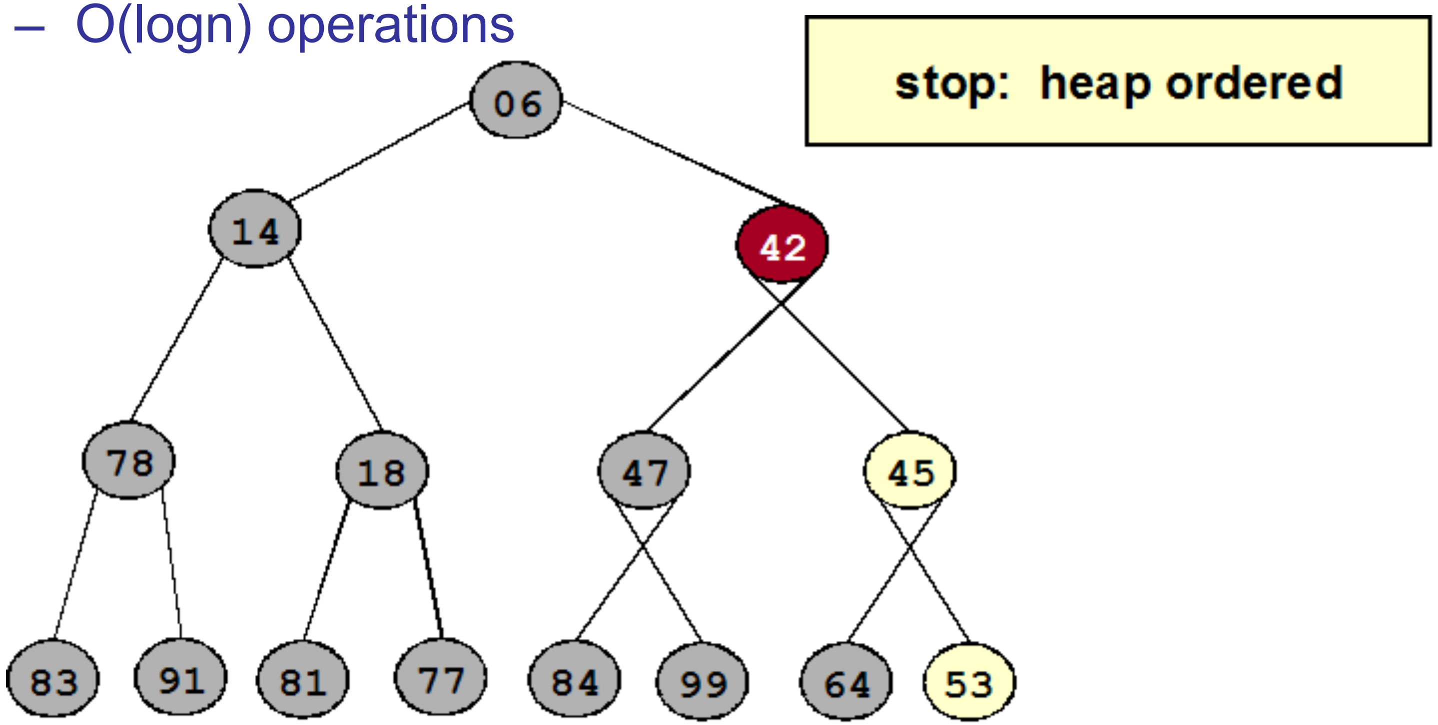
Binary Heaps: Insertion

- Insert element x into heap
  - Insert into next available slot
  - Bubble up until it is heap ordered



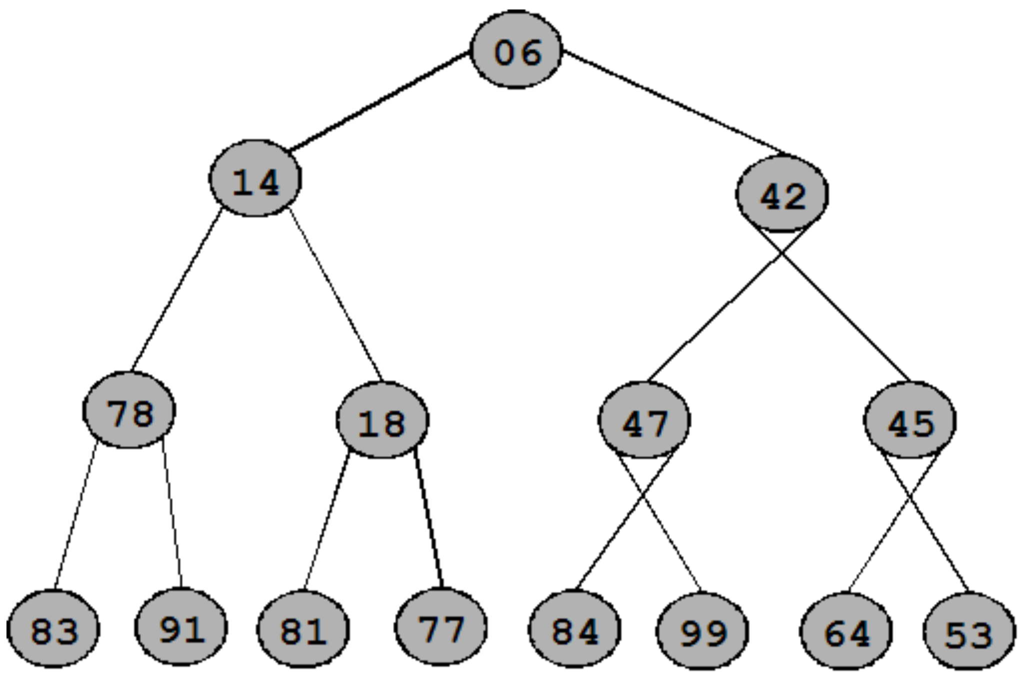
Binary Heaps: Insertion

- Insert element x into heap
  - Insert into next available slot
  - Bubble up until it is heap ordered
  - O(logn) operations



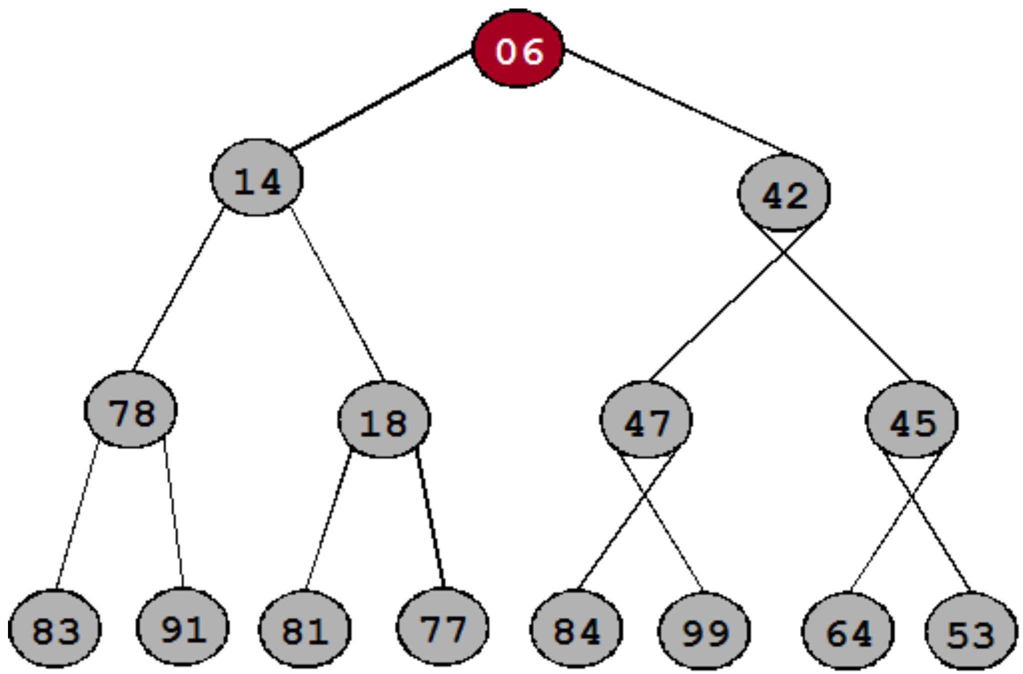
Binary Heaps: Decrease Key

- Decrease key of element x to k
  - Bubble up until it is heap ordered
  - O(logn) operations



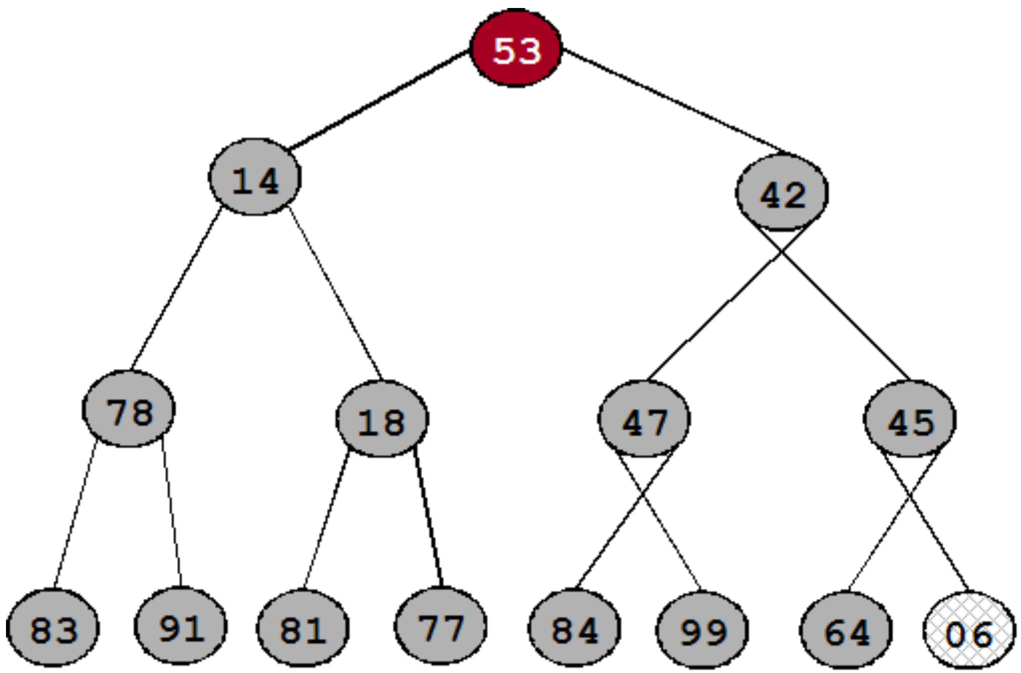
Binary Heaps: Delete Min

- Delete minimum element from heap
  - Exchange root with rightmost leaf
  - Bubble root down until it is heap ordered



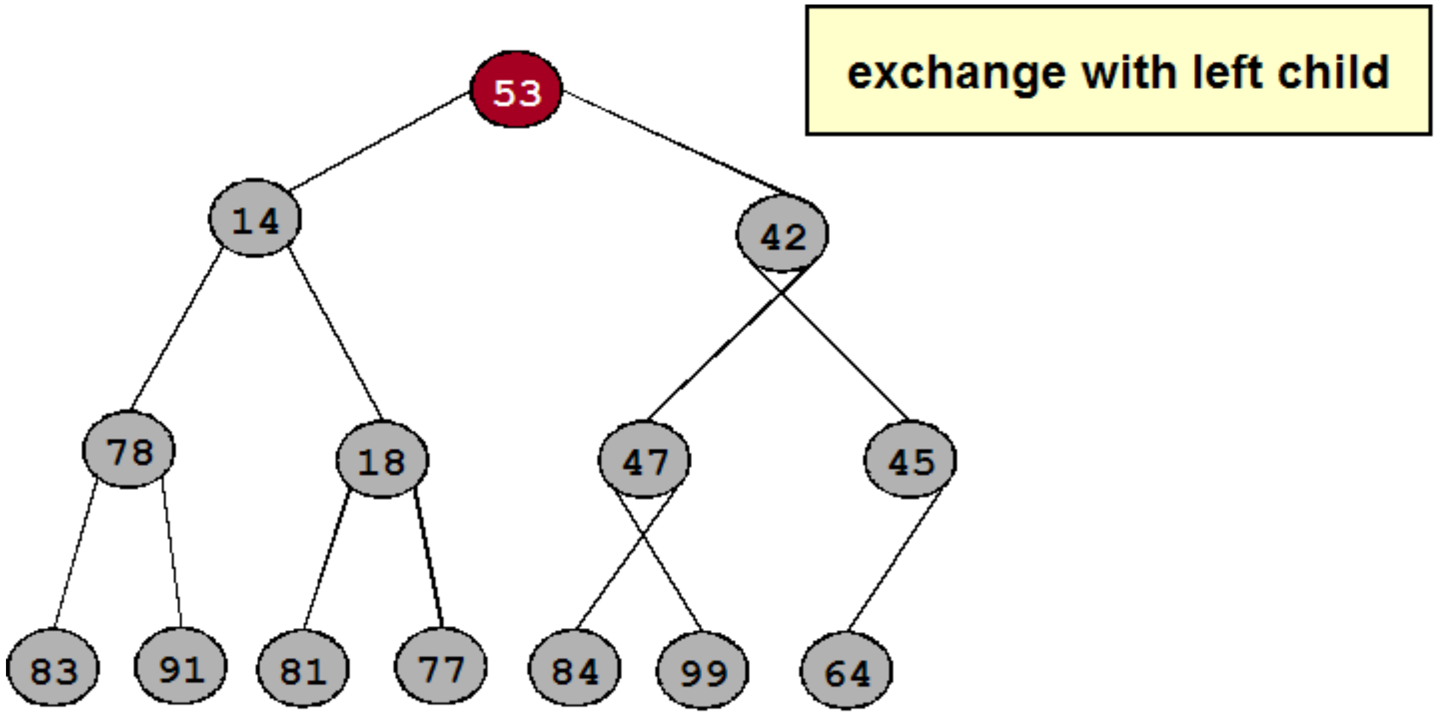
Binary Heaps: Delete Min

- Delete minimum element from heap
  - Exchange root with rightmost leaf
  - Bubble root down until it is heap ordered



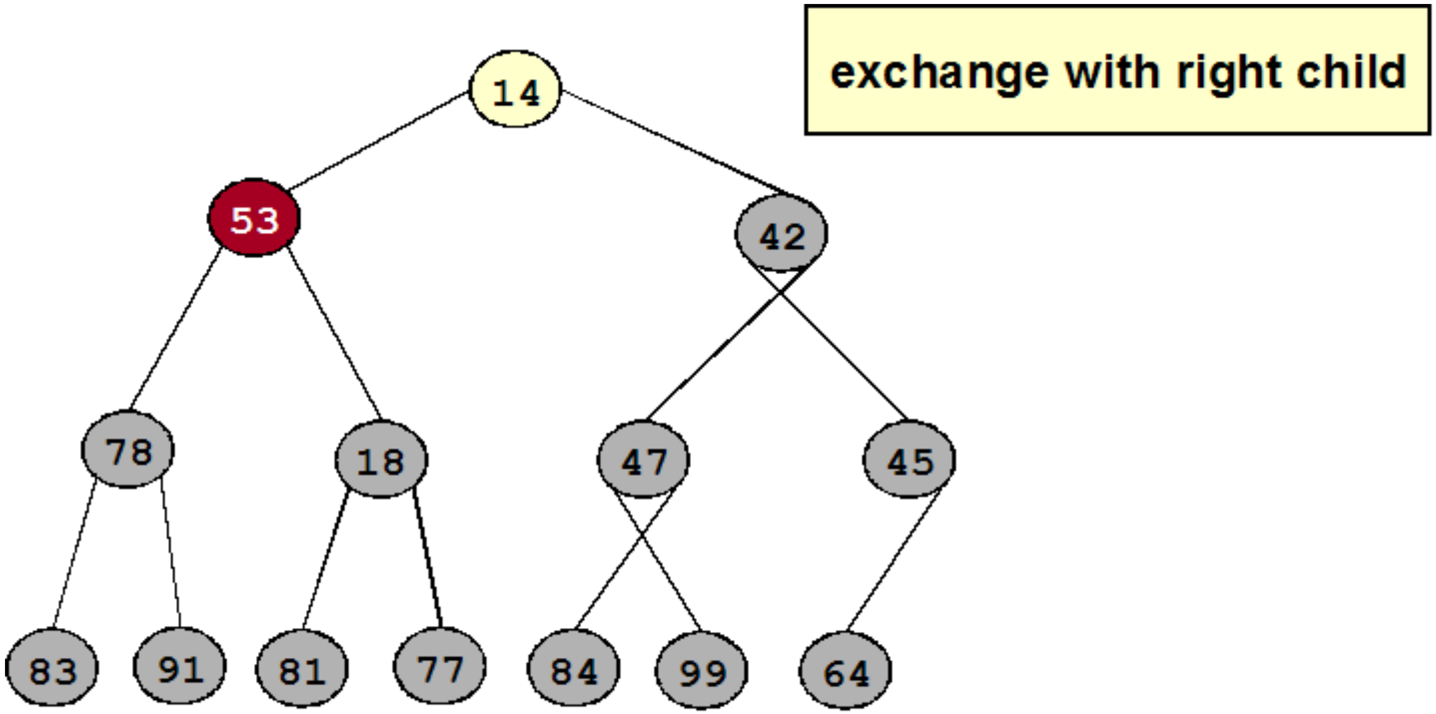
Binary Heaps: Delete Min

- Delete minimum element from heap
  - Exchange root with rightmost leaf
  - Bubble root down until it is heap ordered



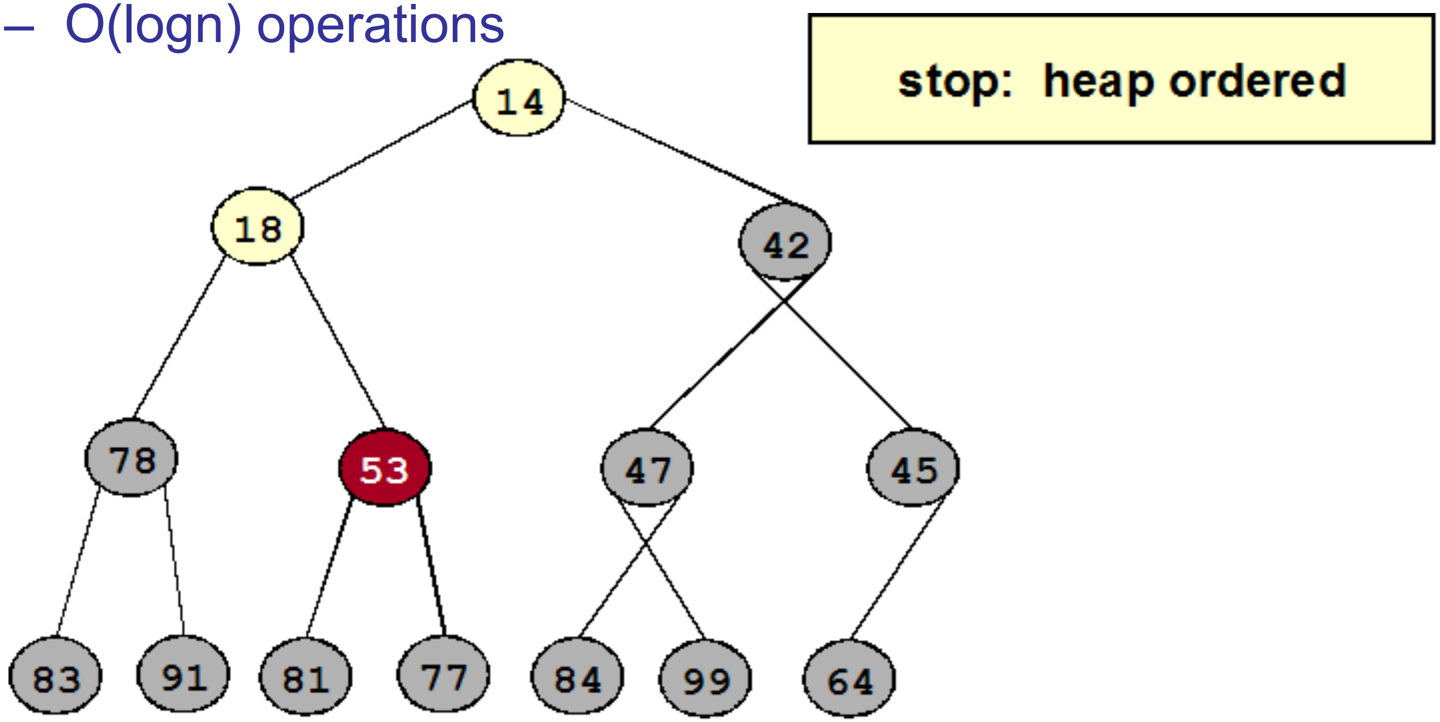
Binary Heaps: Delete Min

- Delete minimum element from heap
  - Exchange root with rightmost leaf
  - Bubble root down until it is heap ordered



Binary Heaps: Delete Min

- Delete minimum element from heap
  - Exchange root with rightmost leaf
  - Bubble root down until it is heap ordered
  - O(logn) operations

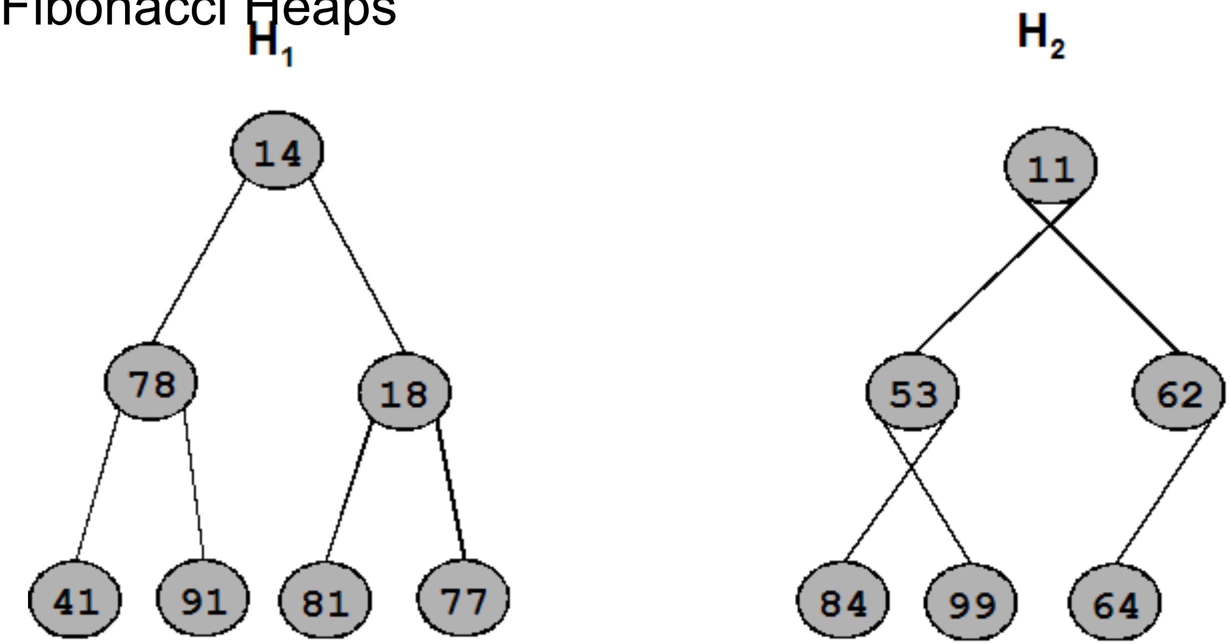


Binary Heaps: Heapsort

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

Binary Heaps: Union

- Combine two binary heaps H_1 and H_2 into single heap
- No easy solution
  - $\Omega(N)$ operations apparently required (Remember BUILD-MAX-HEAP's running time of O(N) for inserting N items.)
- Can support fast union with fancier heaps:
  - Binomial Heaps
  - Fibonacci Heaps



Mergeable Heap Operations

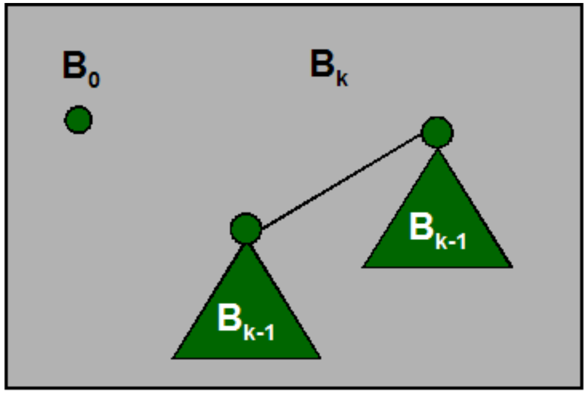
- MAKE-HEAP()
- INSERT(H,x)
- MINIMUM(H)
- EXTRACT-MIN(H)
- UNION(H1,H2)
- DECREASE-KEY(H,x,k)
- DELETE(H,x)
- H: heap, x: node in the heap, k: key value
- Inefficient search operation

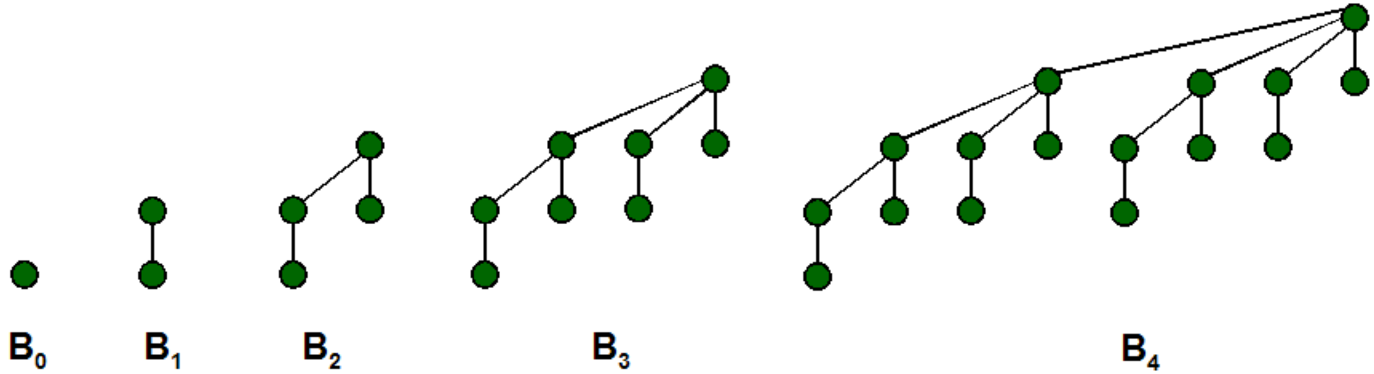
Priority Queues

		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

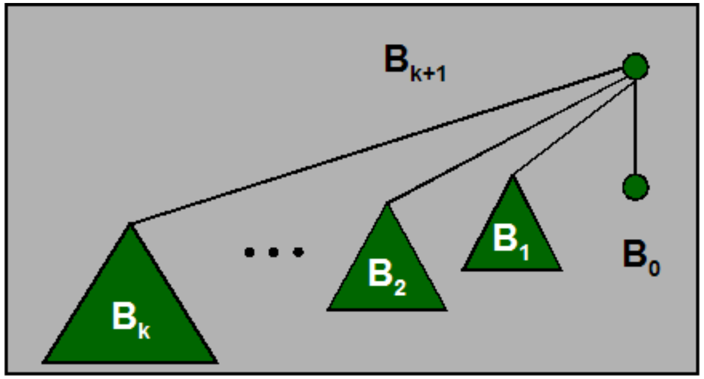
- Recursive definition



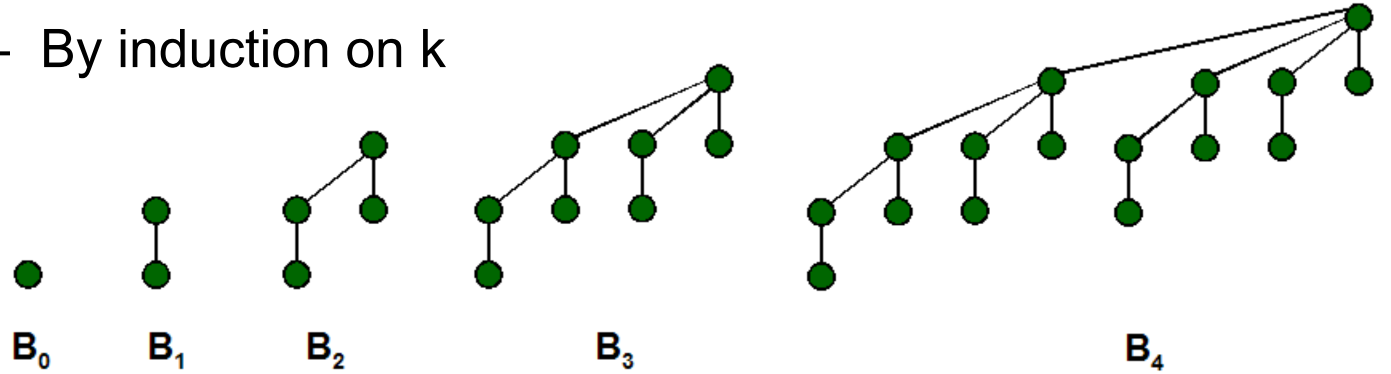


Binomial Tree

- 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}, … , B_0



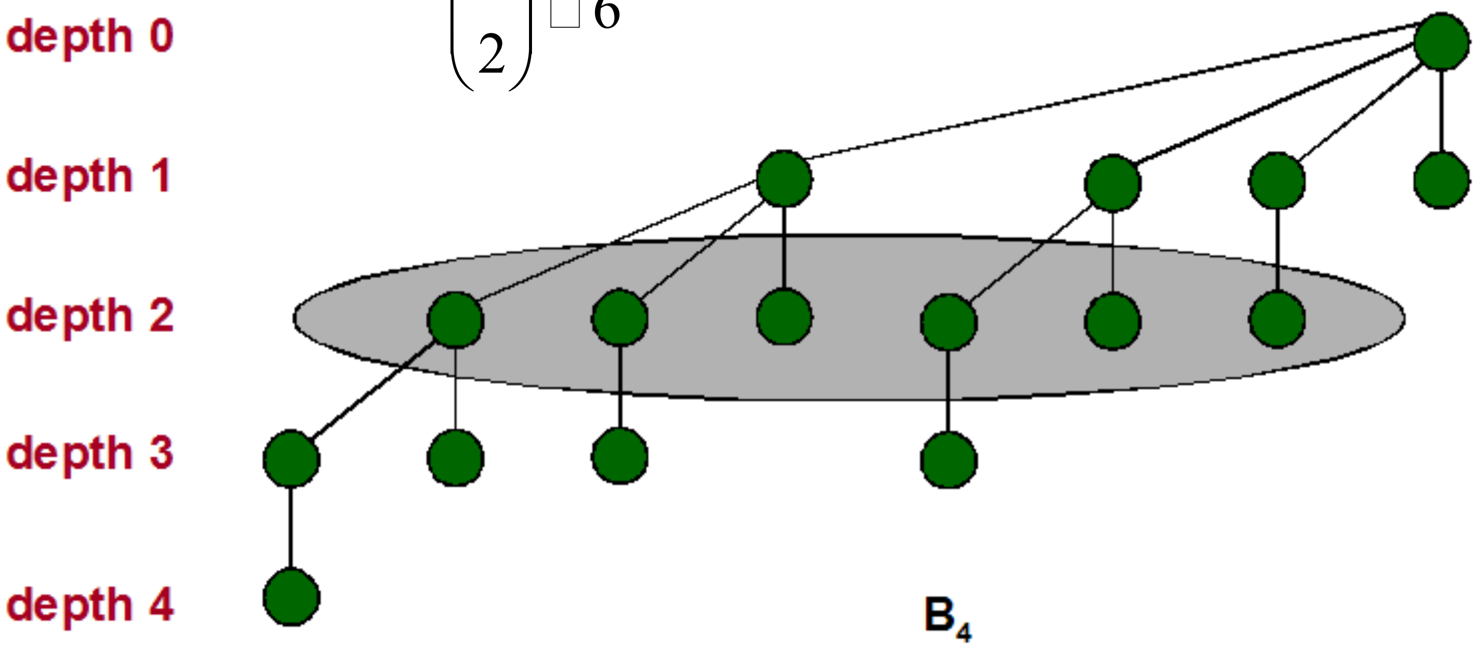
- Proof:
  - By induction on k



Binomial Tree

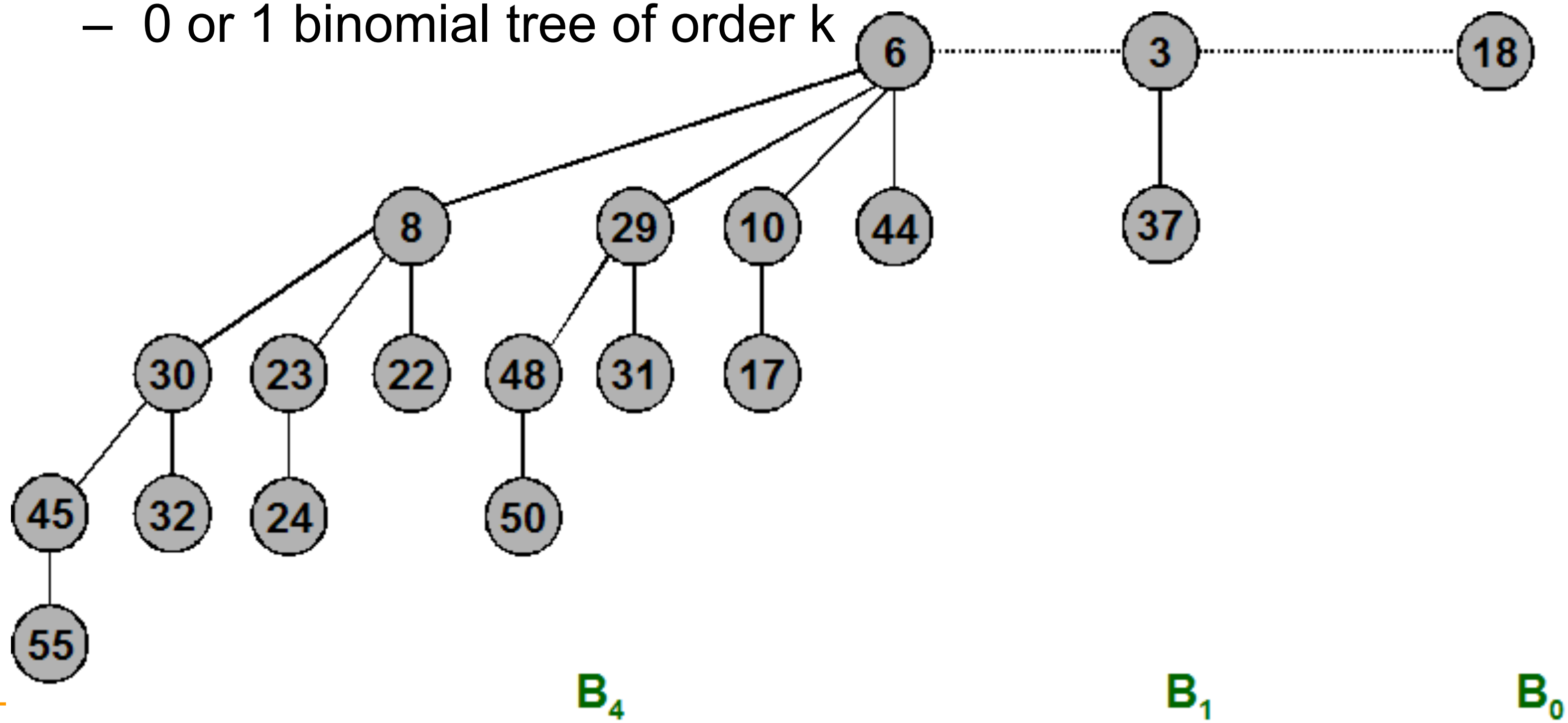
- A property useful for naming the data structure
- B_k has $\binom{k}{i}$ nodes at depth i

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



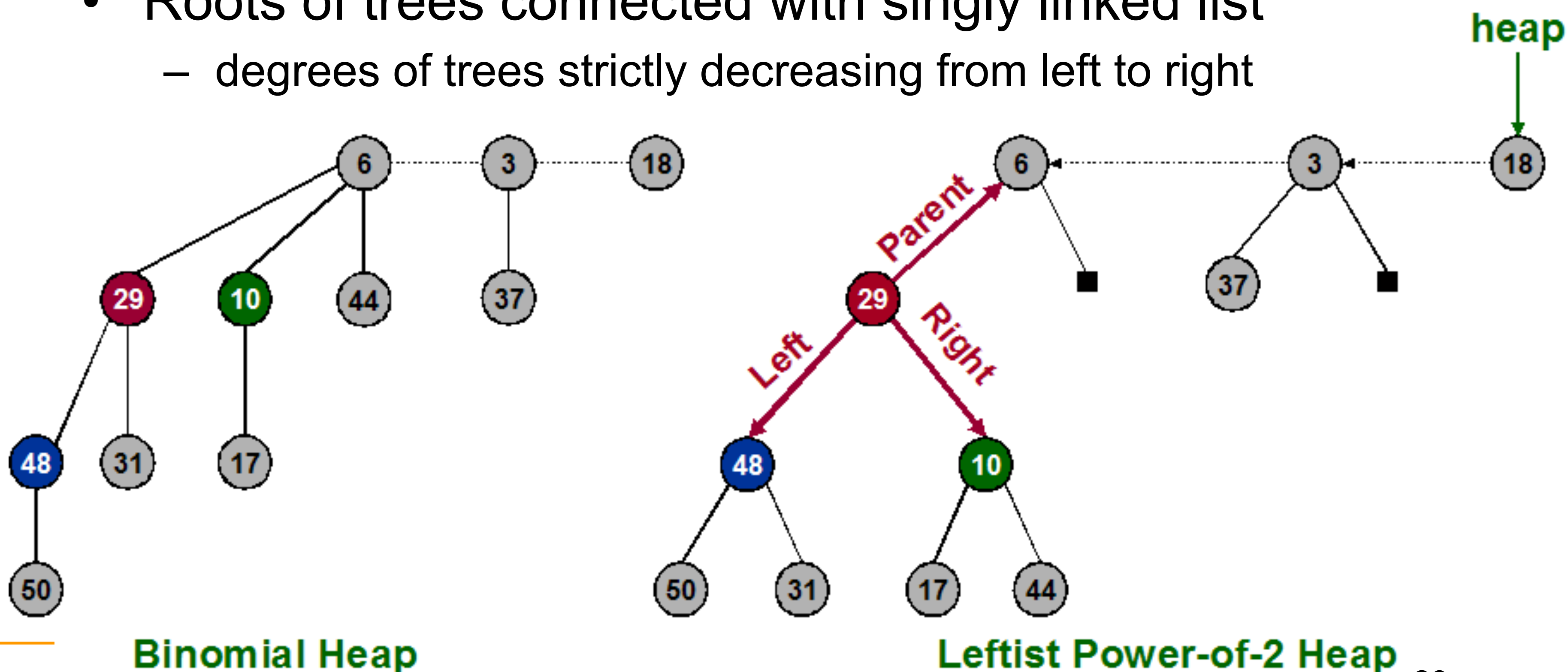
Binomial Heap

- 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



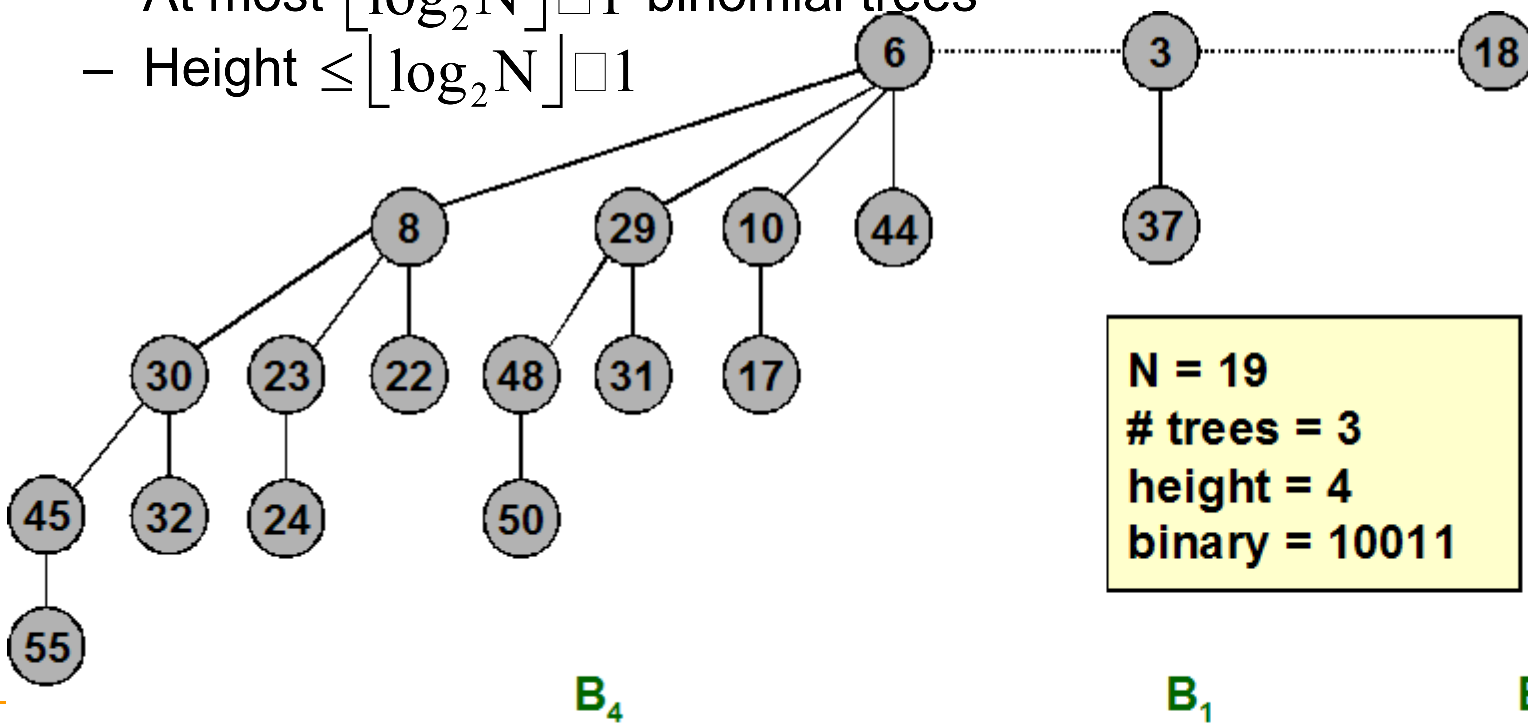
Binomial Heap: Implementation

- Represent trees using left-child, right sibling pointers [Ch. 10.4]
  - three links per node (parent, left, right)
- Roots of trees connected with singly linked list
  - degrees of trees strictly decreasing from left to right



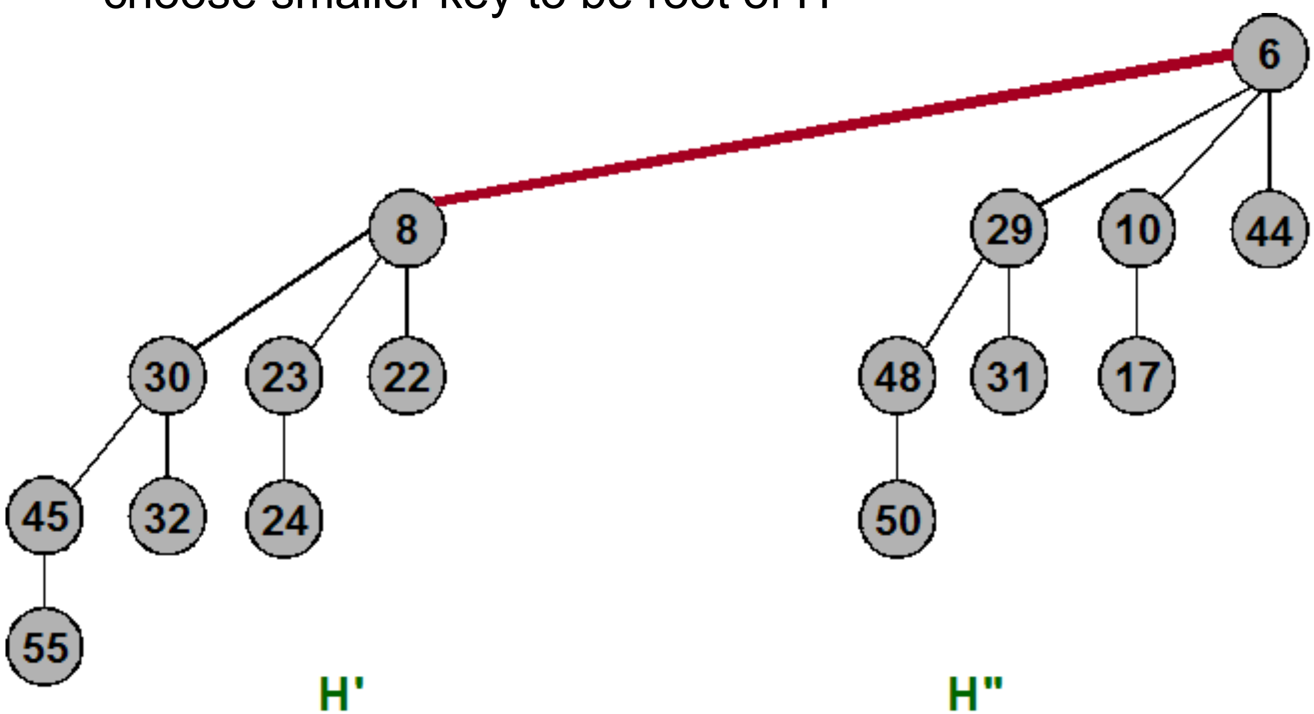
Binomial Heap: Properties

- 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 $\leq \lfloor \log_2 N \rfloor + 1$

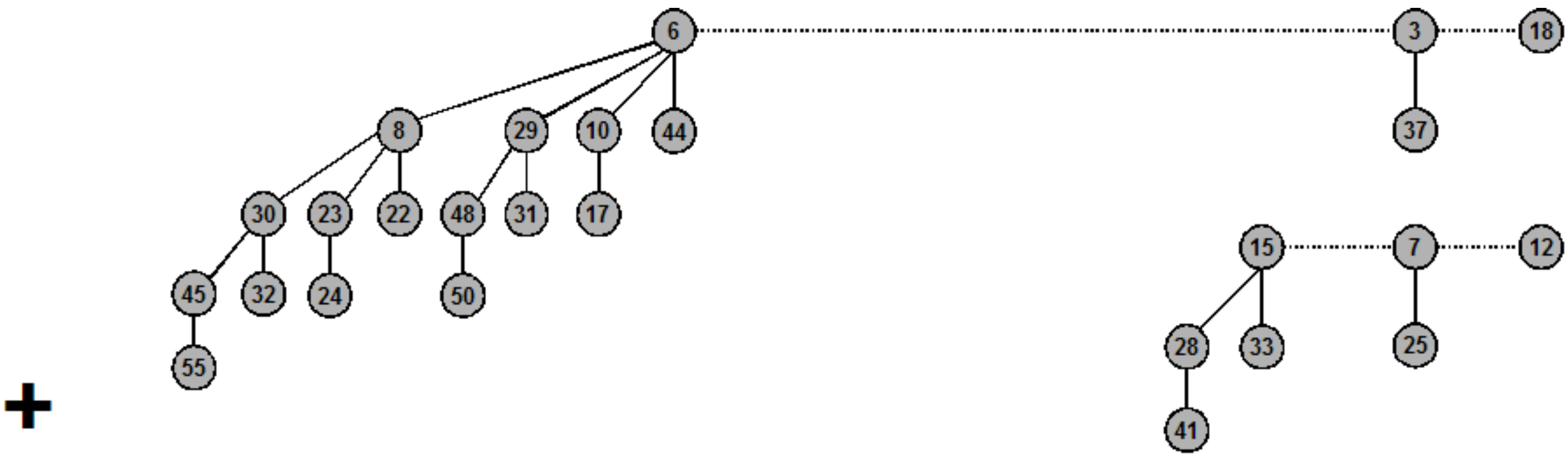


Binomial Heap: Union

- 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



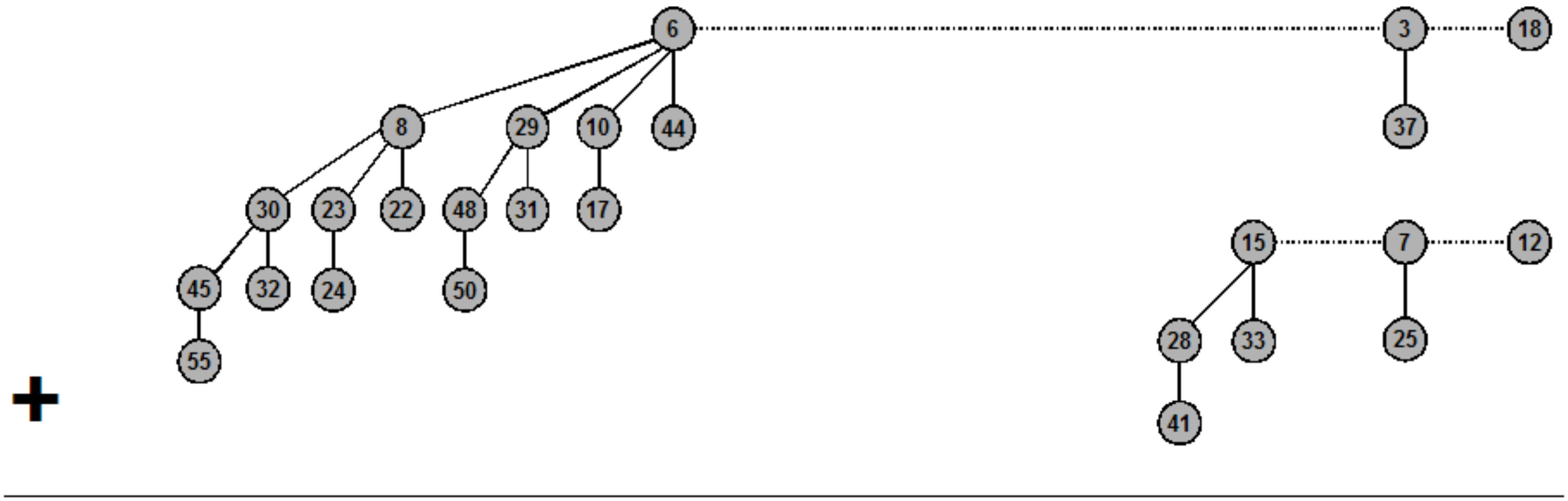
Binomial Heap: Union



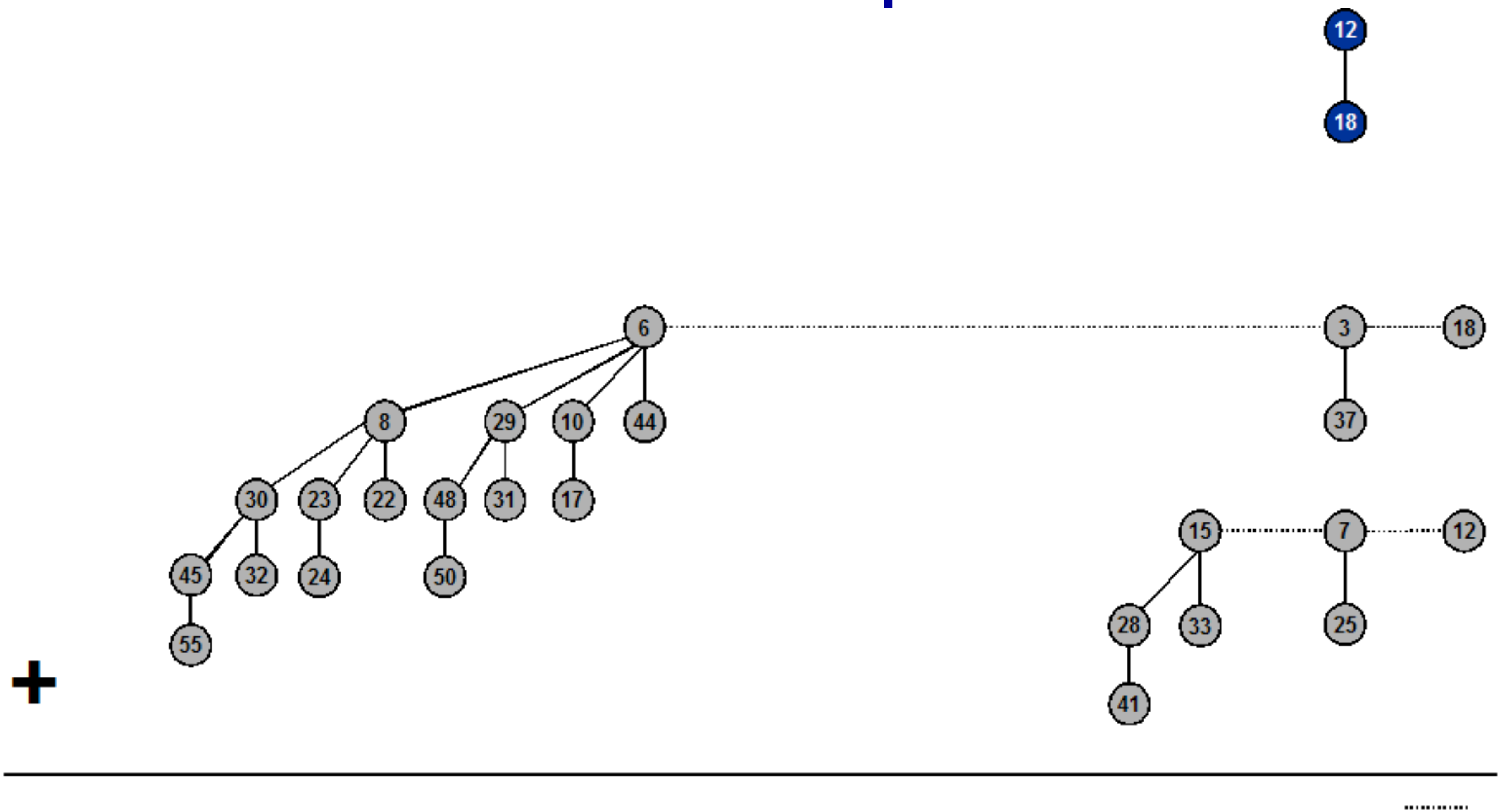
19 + 7 = 26

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

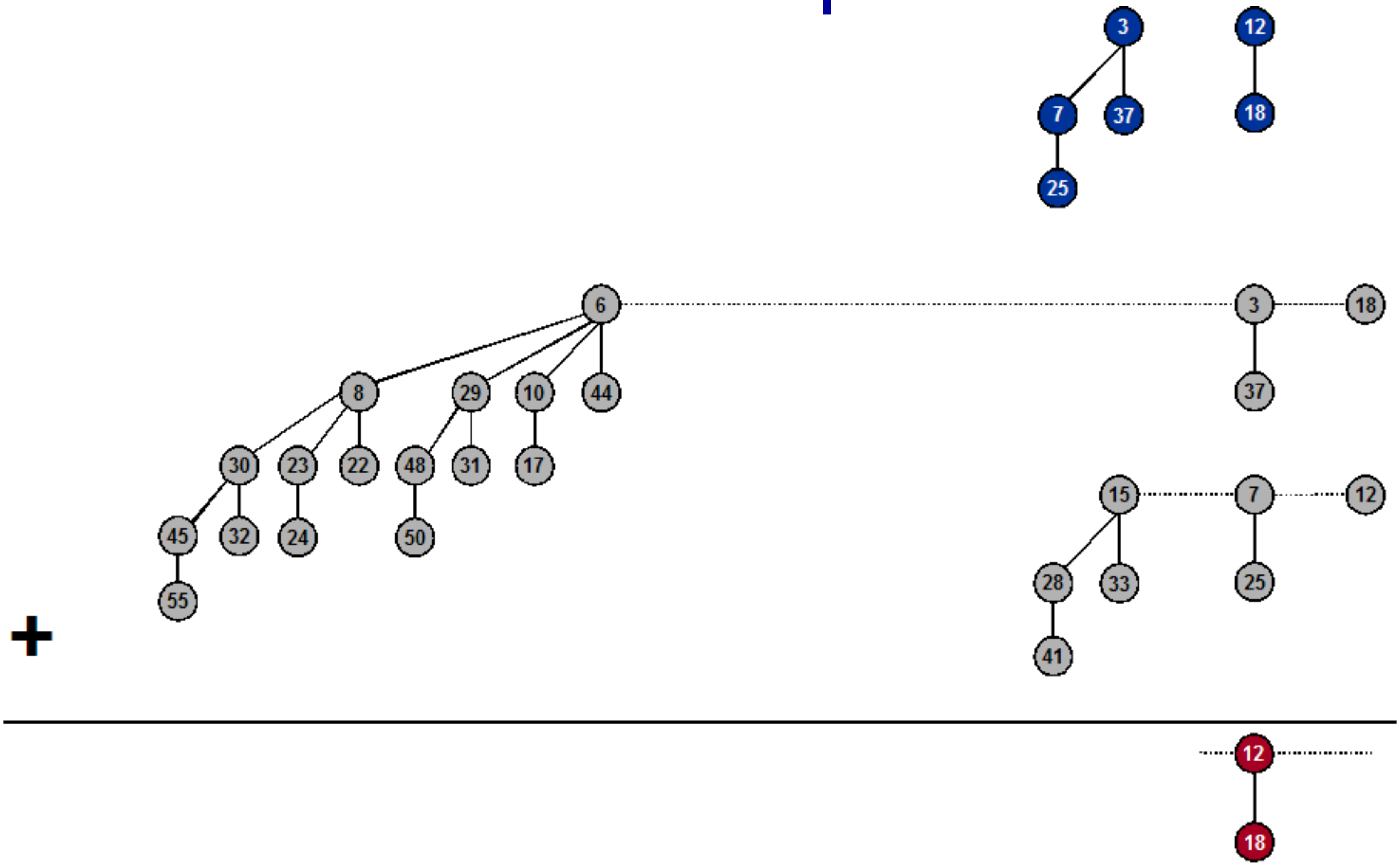
Binomial Heap: Union



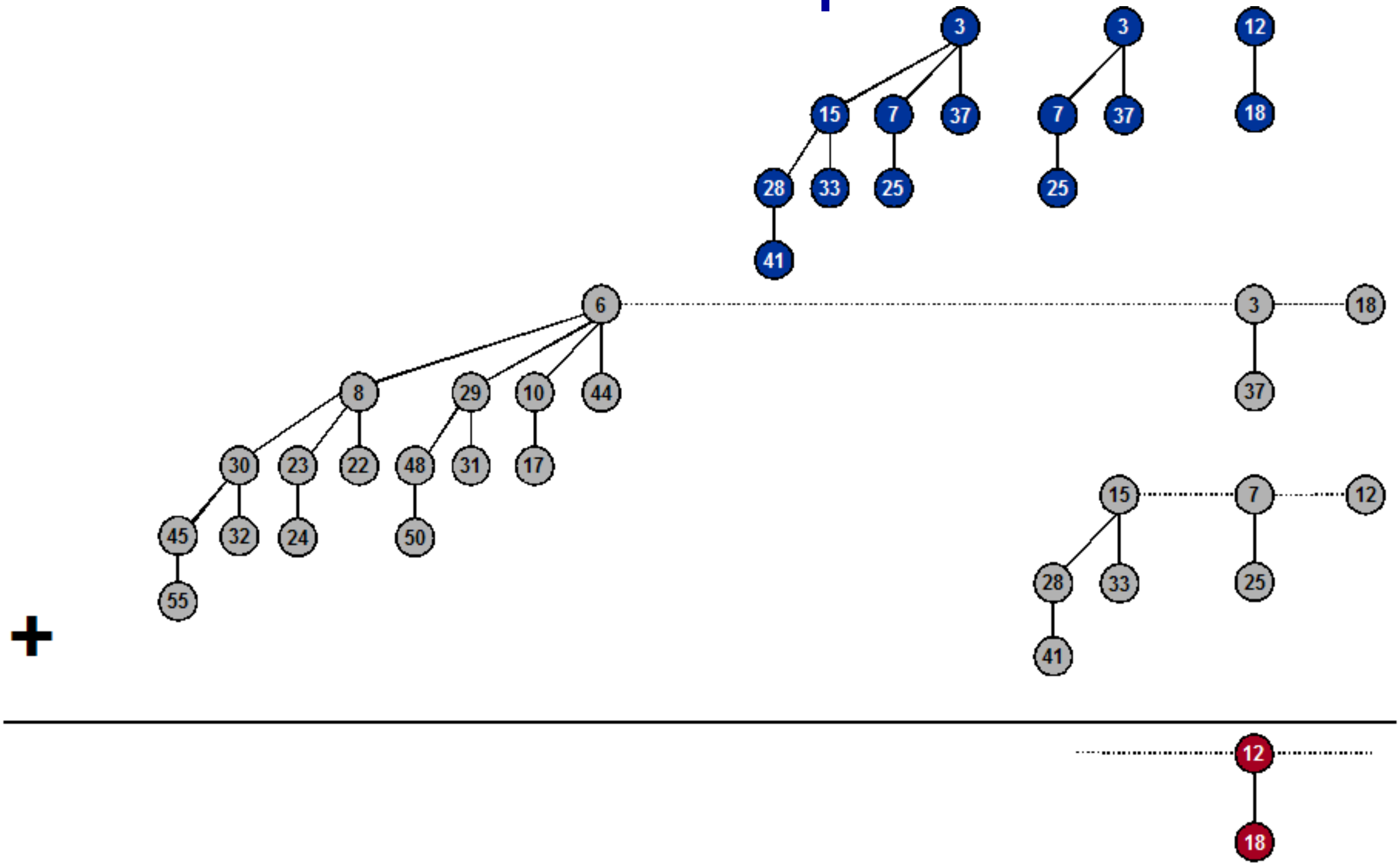
Binomial Heap: Union



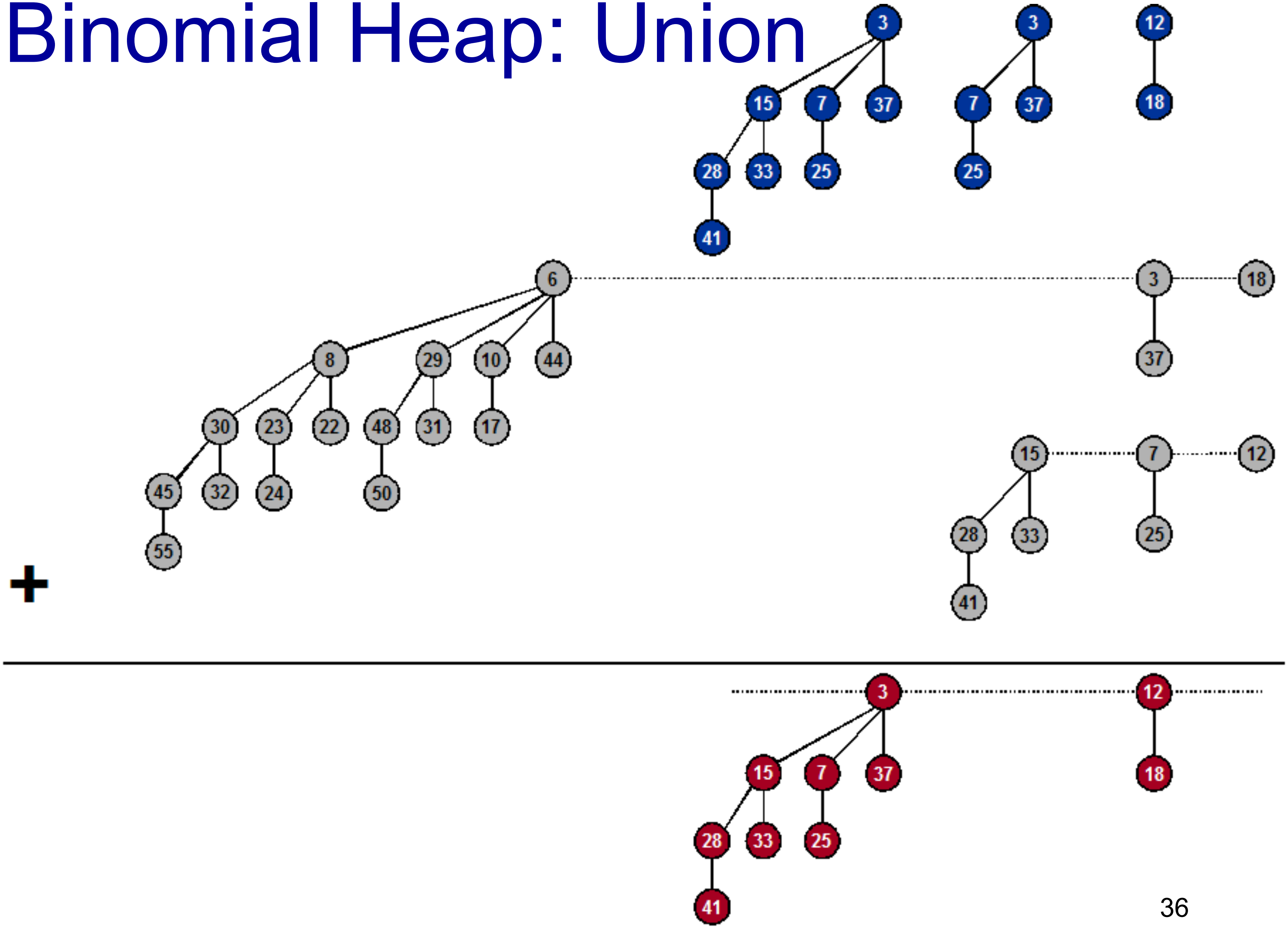
Binomial Heap: Union



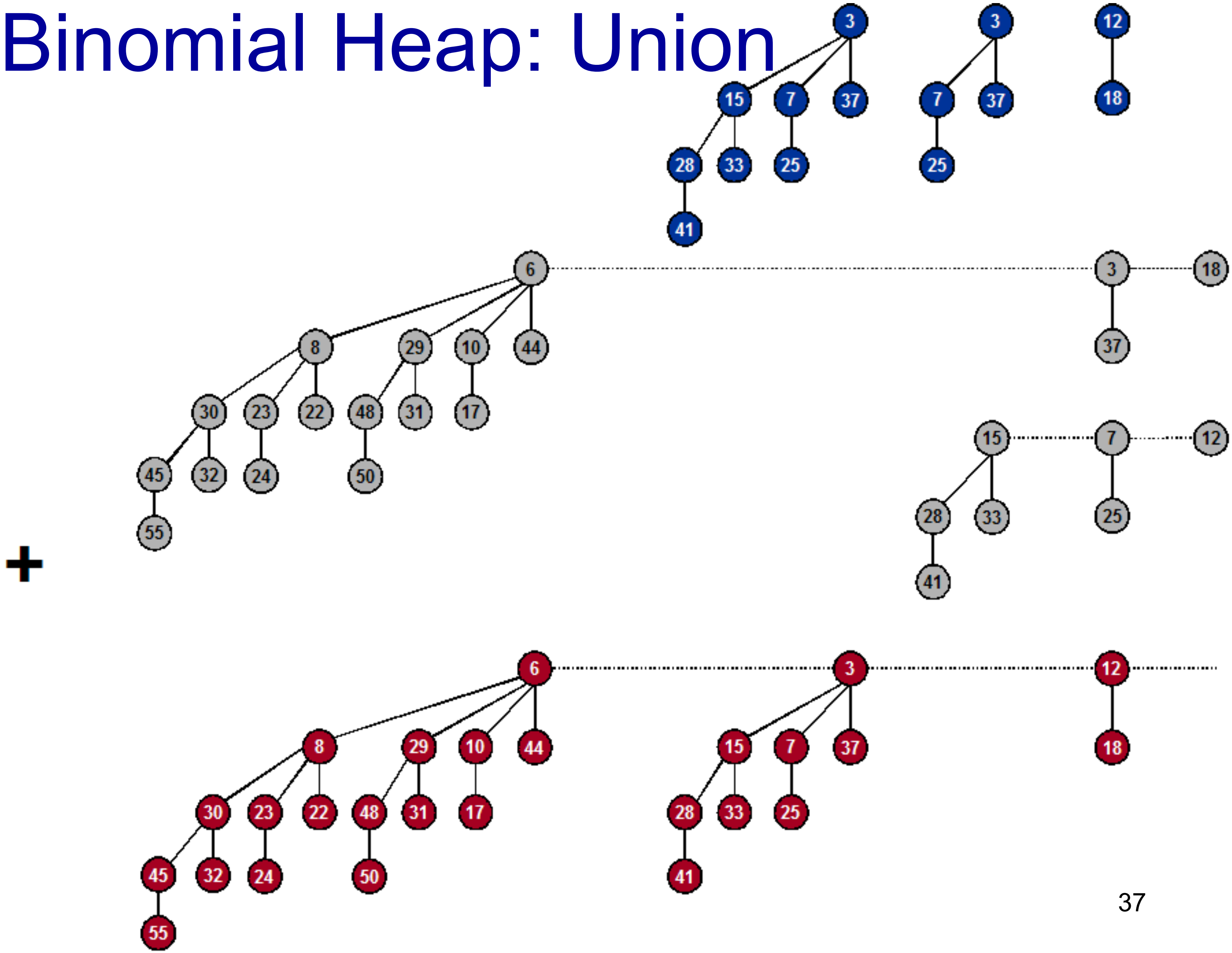
Binomial Heap: Union



Binomial Heap: Union



Binomial Heap: Union



Binary Heaps: Union

- Create heap H that is union of heaps H' and H''
  - Analogous to binary addition

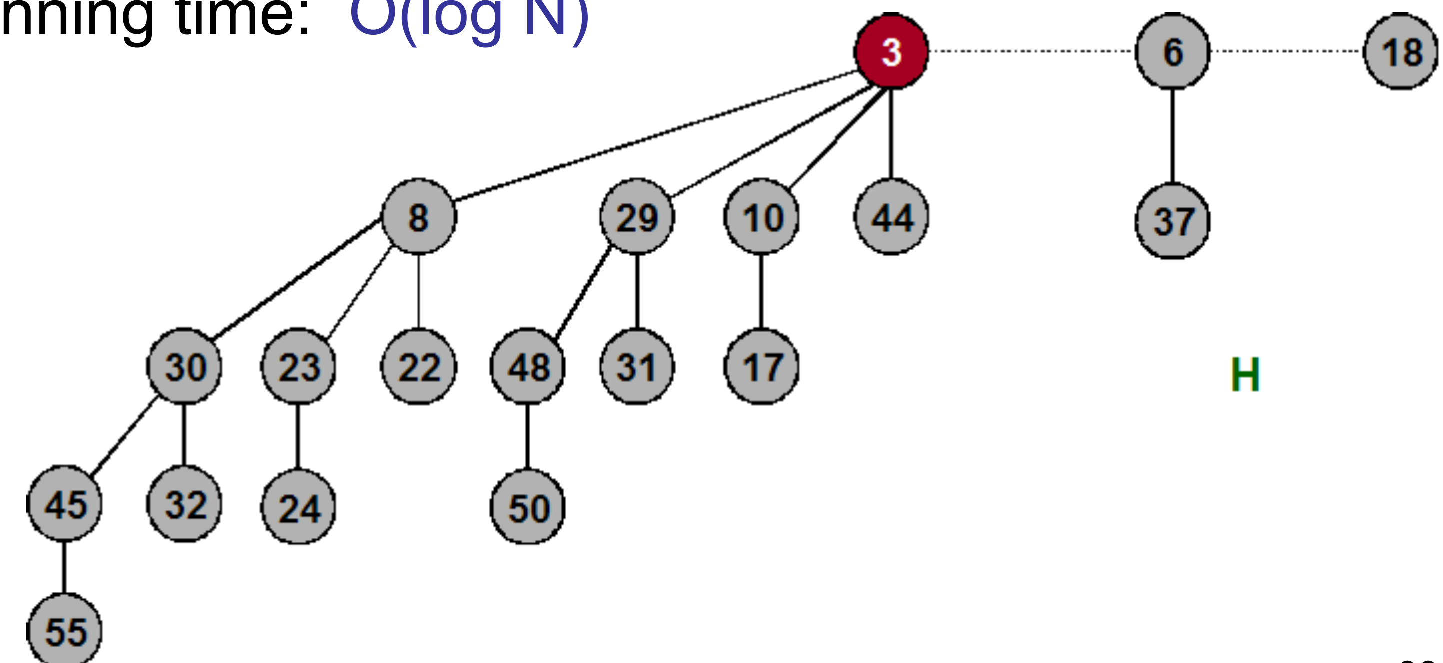
- Running time: O(log N)
  - Proportional to number of trees in root lists $\leq 2(\lfloor \log_2 N \rfloor + 1)$

19 + 7 = 26

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

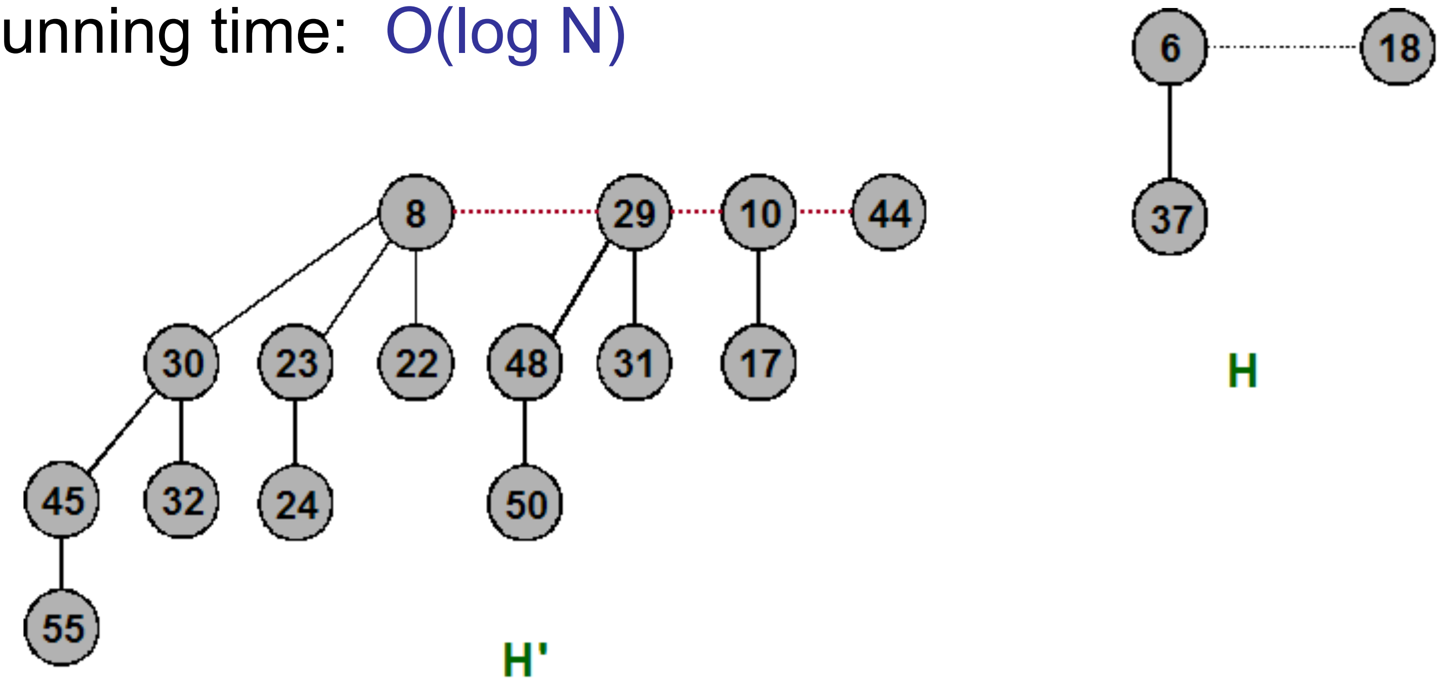
Binary Heaps: Delete Min

- 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)



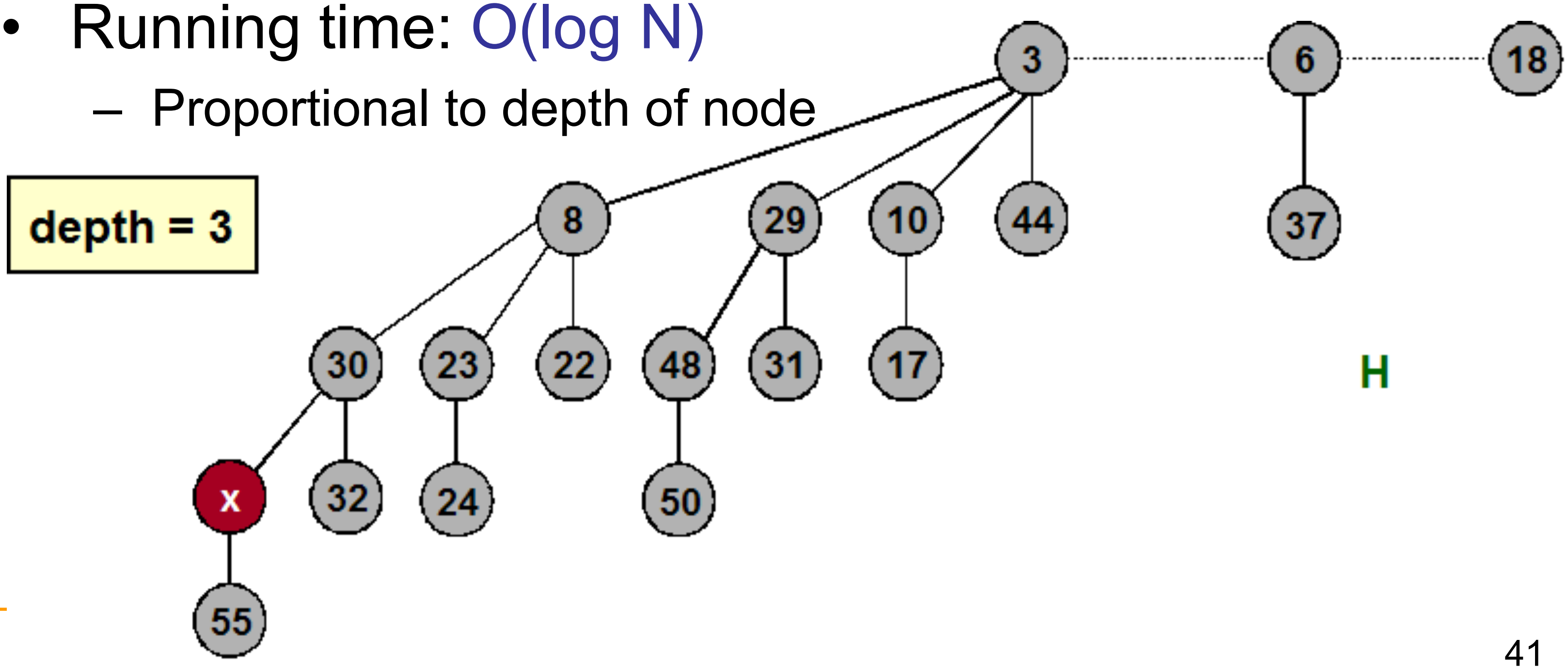
Binary Heaps: Delete Min

- 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)



Binary Heaps: 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



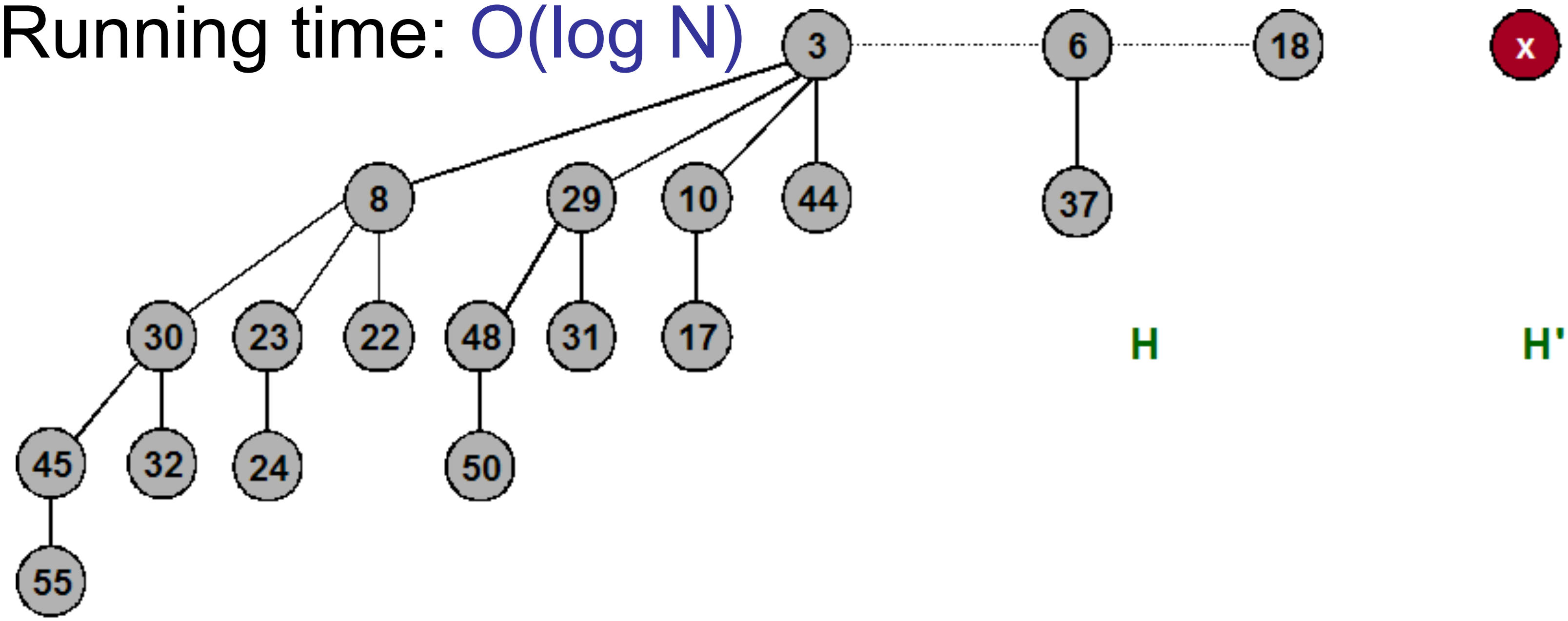
Binomial Heaps: Delete

- Delete node x in binomial heap H.
  - Decrease key of x to $-\infty$
  - Delete min
- Running time: O(log N)

Binomial Heaps: Insert

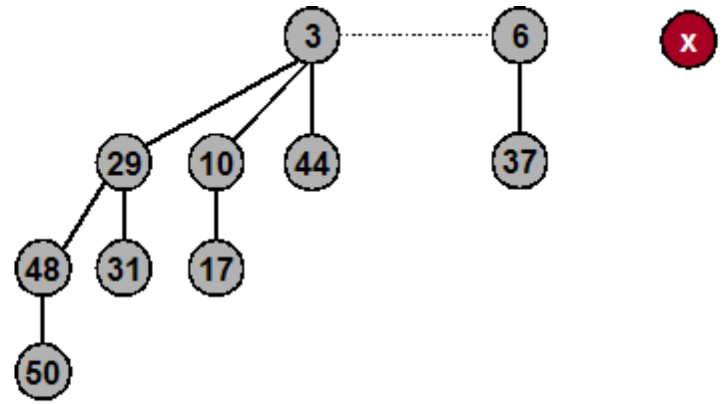
- Insert a new node x into binomial heap H
  - H' ← MakeHeap(x)
  - H ← Union(H', H)

- Running time: O(log N)



Binomial Heaps: 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 $\Omega(\log_2 \mathrm{N})$ time
  - If N = `11...111`, then $\log_2 \mathrm{N}$ steps

- But, inserting sequence of N items takes O(N) time!
  - (N/2)(1) + (N/4)(2) + (N/8)(3) + . . . ≤ 2N
  - Amortized analysis
  - Basis for getting most operations down to constant time

$$\sum_{i \square 1}^{N} \frac{i}{2^i} \square 2 - \frac{N}{2^N} - \frac{1}{2^{N-1}}$$

$$\leq 2$$

$$\leq N \sum_{i \square 0}^{\infty} \frac{1}{2^i} \square 2N$$

Amortized Analysis

- Worst-case analysis
  - Analyze running time as function of worst input of a given size
- Average case analysis
  - Analyze average running time over some distribution of inputs
  - Ex: quicksort
- Amortized analysis
  - Worst-case bound on sequence of operations
  - Ex: splay trees, union-find
- Competitive analysis
  - Make quantitative statements about online algorithms
  - Ex: paging, load balancing

Binomial Heap Operations

- **MAKE-HEAP()**
- **INSERT(H,x)**
- **MINIMUM(H)**
- **EXTRACT-MIN(H)**
- **UNION(H1,H2)**
- **DECREASE-KEY(H,x,k)**
- **DELETE(H,x)**

- **MAKE-HEAP() [Trivial, just create an object H, head[H]=nil]**
- **MINIMUM(H) [Trivial: Return the minimum among binomial tree roots in the binomial heap]**

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

Summary

- Binomial Trees and Binomial Heaps
- Operations on Binomial Heaps
  – Creating New Binomial Heap
  – Finding Minimum Key
  – Uniting Two Binomial Heaps
  – Inserting Node
  – Extracting Node with Minimum Key
  – Deleting Key
  – Decreasing Key