Binomial Queues

CSE 373

Data Structures

Lecture 12

Reading

* Reading
  * Section 6.8,

Merging heaps

- Binary Heap has limited (fast) functionality
  - › FindMin, DeleteMin and Insert only
  - › does not support fast merges of two heaps
- For some applications, the items arrive in prioritized clumps, rather than individually
- Is there somewhere in the heap design that we can give up a little performance so that we can gain faster merge capability?

Binomial Queues

- Binomial Queues are designed to be merged quickly with one another
- Using pointer-based design we can merge large numbers of nodes at once by simply pruning and grafting tree structures
- More overhead than Binary Heap, but the flexibility is needed for improved merging speed

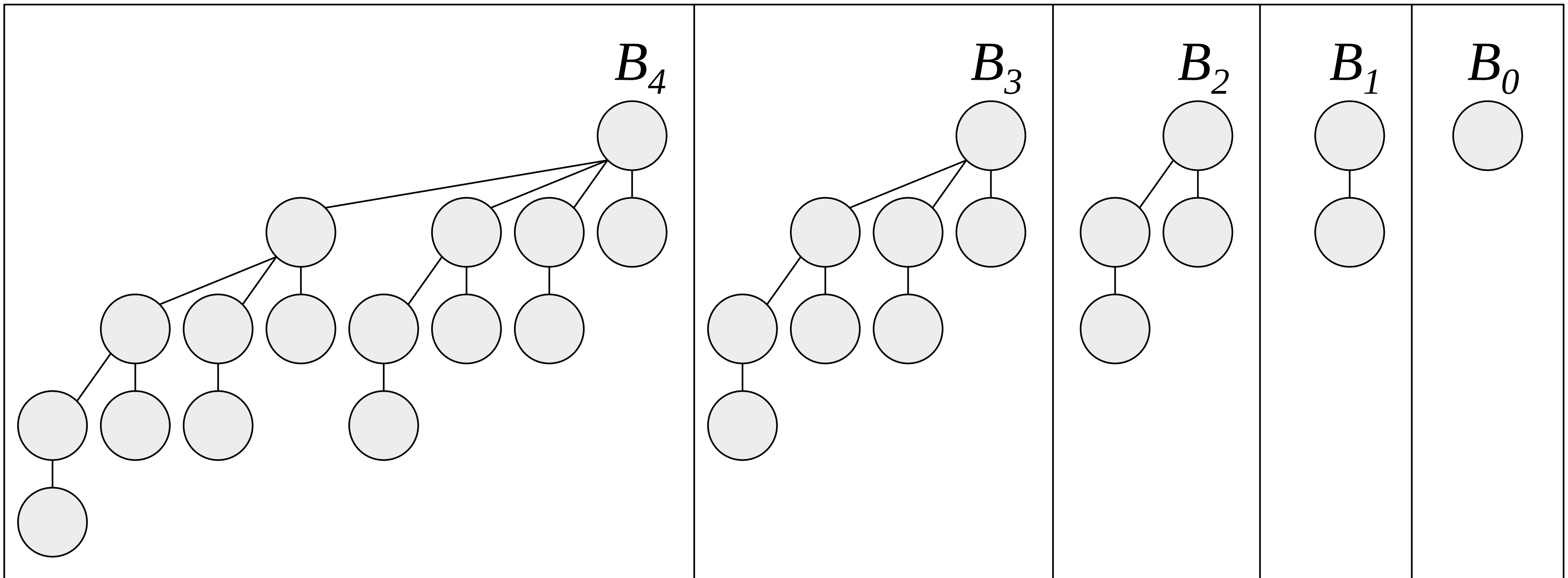
Worst Case Run Times

	Binary Heap	Binomial Queue
Insert	$\Theta(\log N)$	$\Theta(\log N)$
FindMin	$\Theta(1)$	$O(\log N)$
DeleteMin	$\Theta(\log N)$	$\Theta(\log N)$
Merge	$\Theta(N)$	$O(\log N)$

Binomial Queues

- Binomial queues give up $\Theta(1)$ FindMin performance in order to provide $O(\log N)$ merge performance
- A **binomial queue** is a collection (or *forest*) of heap-ordered trees
  - Not just one tree, but a collection of trees
  - each tree has a defined structure and capacity
  - each tree has the familiar heap-order property

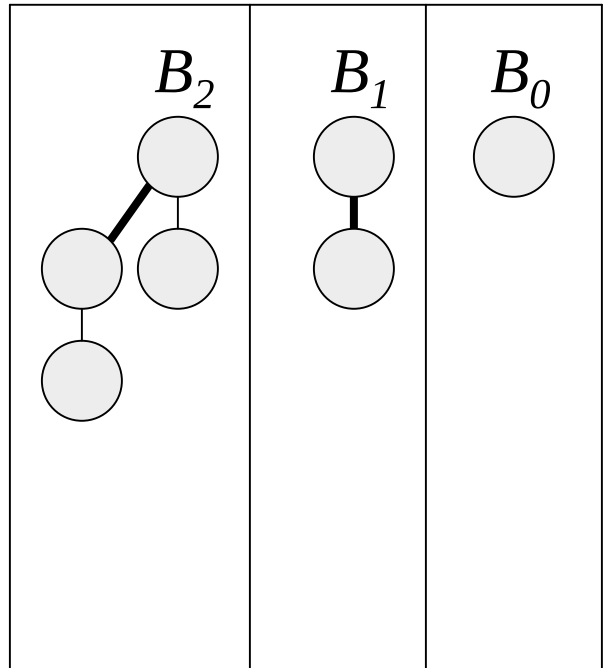
Binomial Queue with 5 Trees



depth	4	3	2	1	0
number of elements	$2^4 = 16$	$2^3 = 8$	$2^2 = 4$	$2^1 = 2$	$2^0 = 1$

Structure Property

- Each tree contains two copies of the previous tree
  - › the second copy is attached at the root of the first copy
- The number of nodes in a tree of depth *d* is exactly 2^d



depth	2	1	0
number of elements	$2^2 = 4$	$2^1 = 2$	$2^0 = 1$

Powers of 2 (one more time)

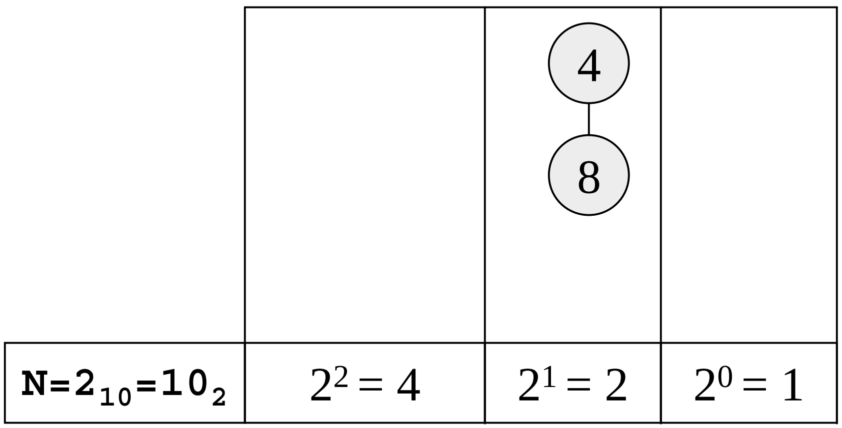
- Any number N can be represented in base 2: $\sum_{i=0}^{i=n-1} a_i 2^i$
  - A base 2 value identifies the powers of 2 that are to be included

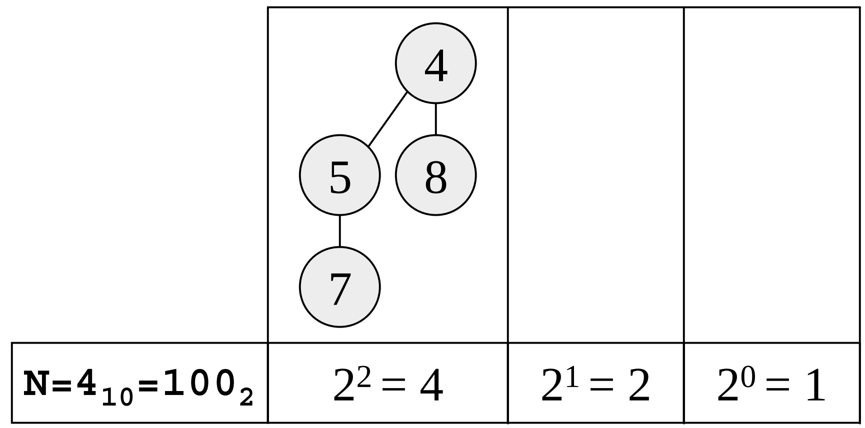
$2^3 = 8_{10}$	$2^2 = 4_{10}$	$2^1 = 2_{10}$	$2^0 = 1_{10}$	Hex_{16}	$Decimal_{10}$
		1	1	3	3
	1	0	0	4	4
	1	0	1	5	5

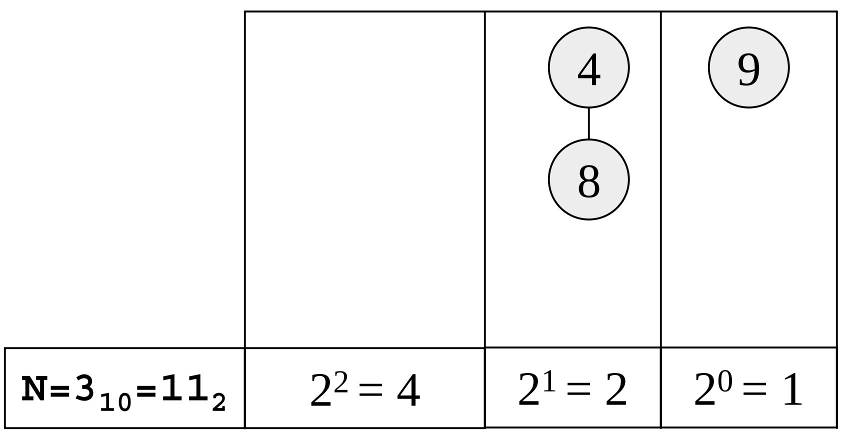
Numbers of nodes

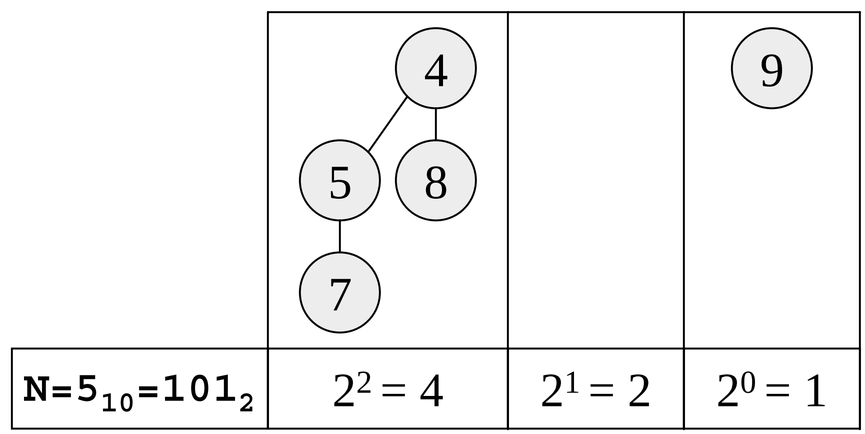
- Any number of entries in the binomial queue can be stored in a forest of binomial trees
- Each tree holds the number of nodes appropriate to its depth, i.e., 2^d nodes
- So the structure of a forest of binomial trees can be characterized with a single binary number
  - › $101_2 \rightarrow 1{\cdot}2^2 + 0{\cdot}2^1 + 1{\cdot}2^0 = 5$ nodes

Structure Examples









What is a merge?

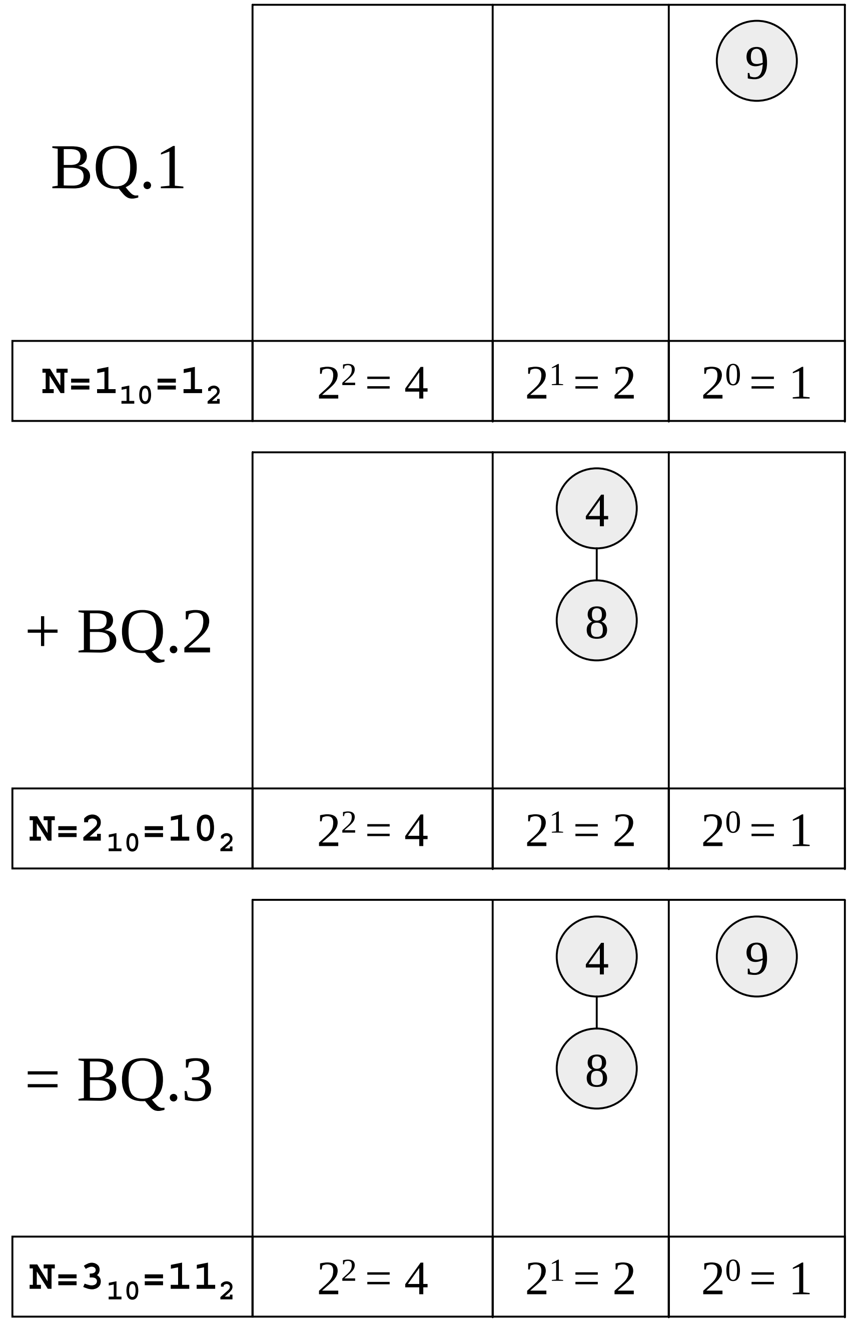
- There is a direct correlation between
  - the number of nodes in the tree
  - the representation of that number in base 2
  - and the actual structure of the tree
- When we merge two queues of sizes N_1 and N_2, the number of nodes in the new queue is the *sum of* N_1+N_2
- We can use that fact to help see how fast merges can be accomplished

Example 1.

Merge BQ.1 and BQ.2

Easy Case.

There are no comparisons and there is no restructuring.

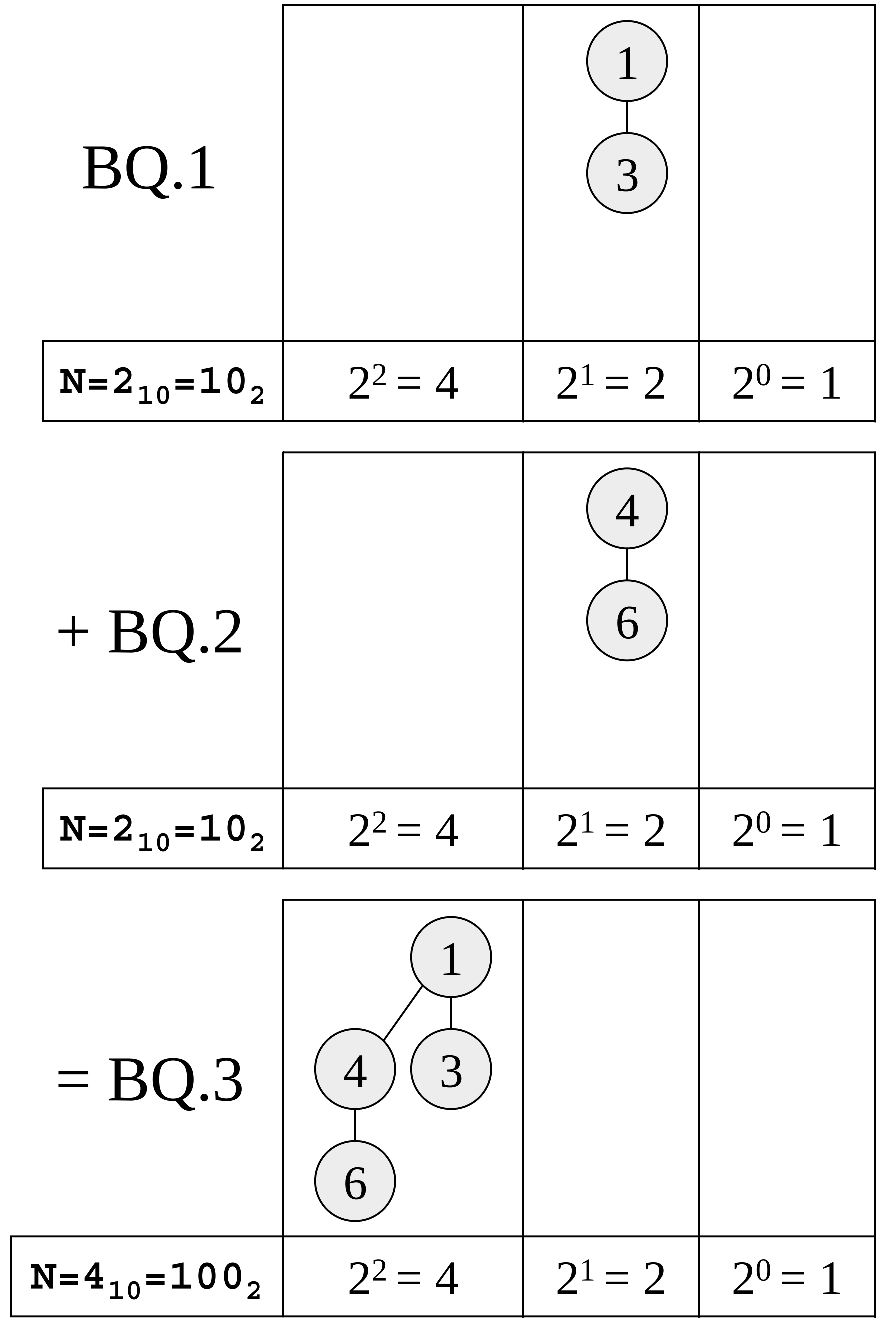


Example 2.

Merge BQ.1 and BQ.2

This is an add with a carry out.

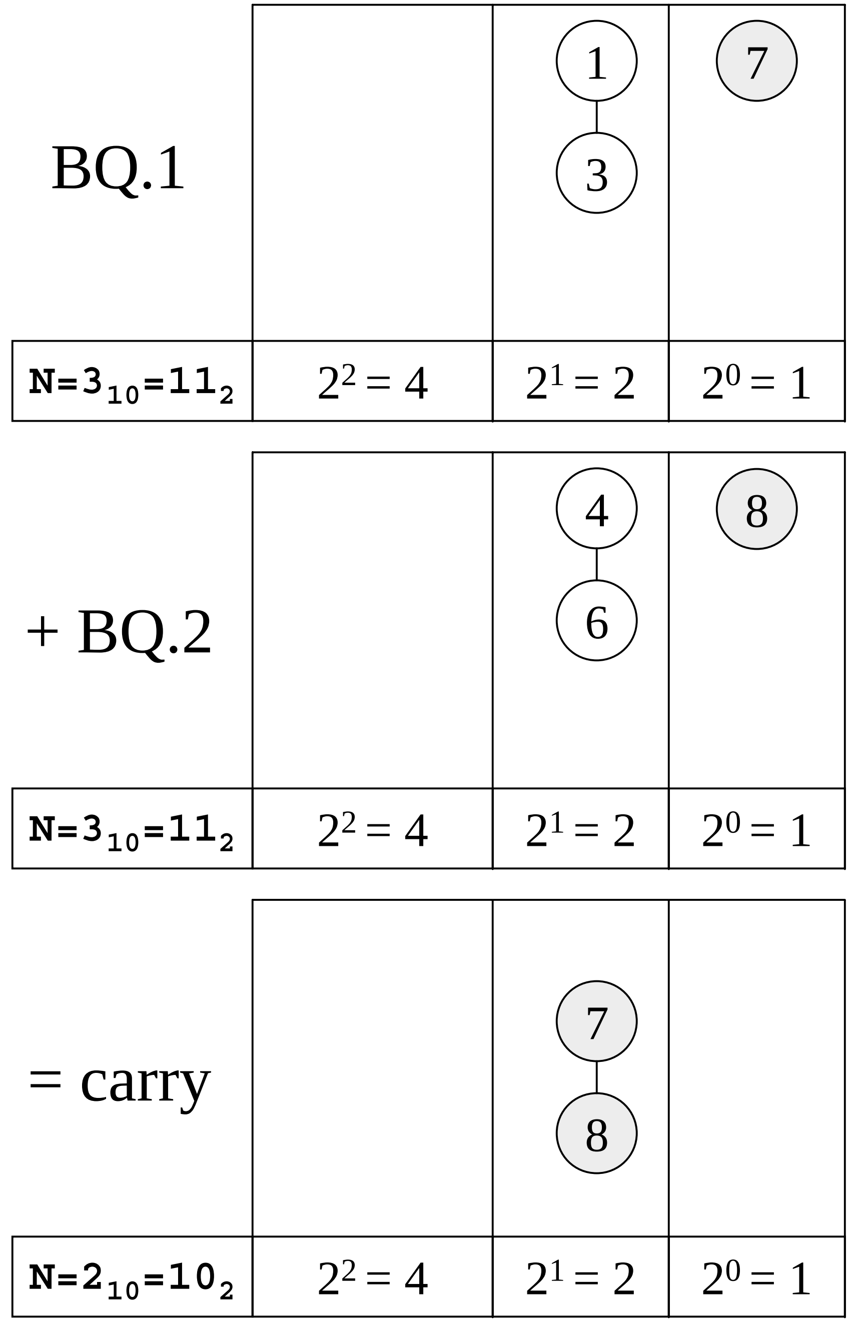
It is accomplished with one comparison and one pointer change: O(1)



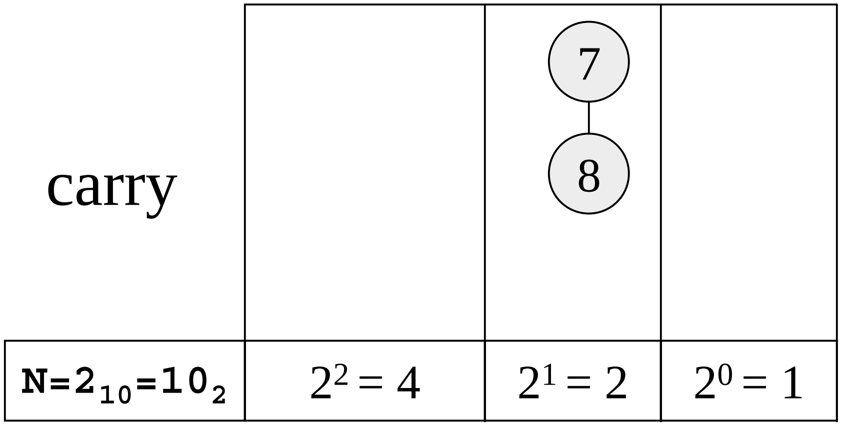
Example 3.

Merge BQ.1 and BQ.2

Part 1 - Form the carry.



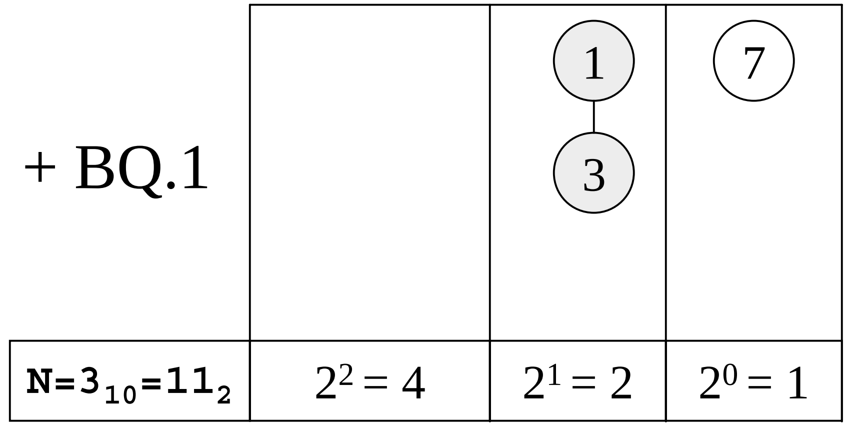
carry



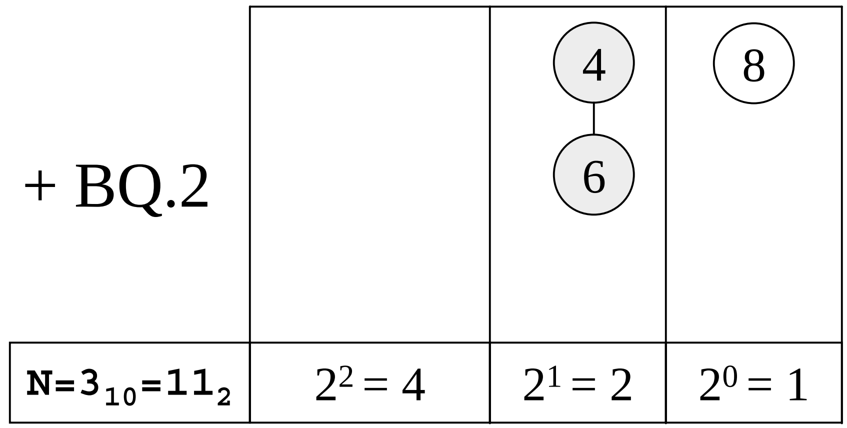
Example 3.

Part 2 - Add the existing values and the carry.

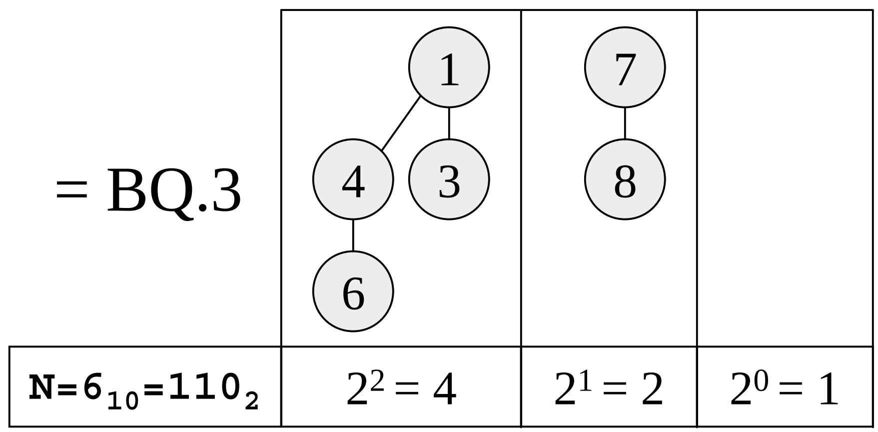
+ BQ.1



+ BQ.2



= BQ.3

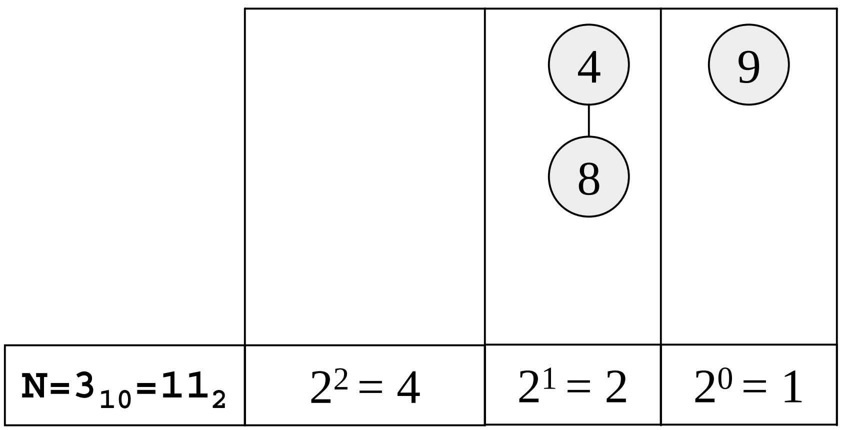


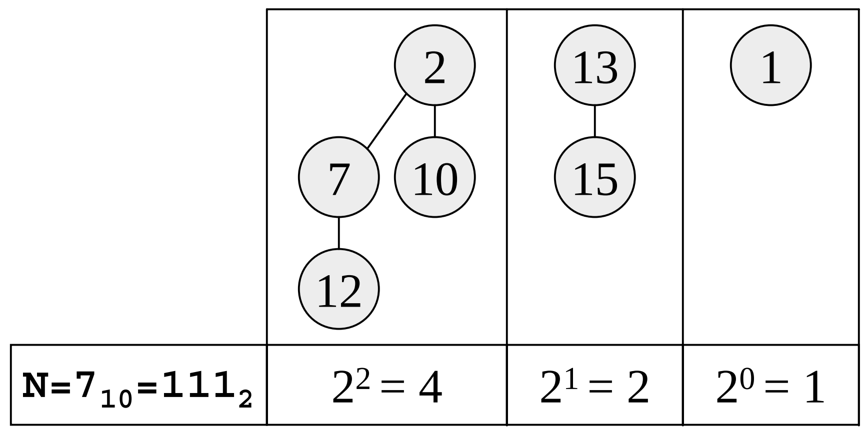
Merge Algorithm

- Just like binary addition algorithm
- Assume trees $X_0,\ldots,X_n$ and $Y_0,\ldots,Y_n$ are binomial queues
  - X_i and Y_i are of type B_i or null

```
C0 := null; //initial carry is null//
for i = 0 to n do
  combine Xi,Yi, and Ci to form Zi and new Ci+1
Zn+1 := Cn+1
```

Exercise





O(log N) time to Merge

- For N keys there are at most $\lceil \log_2 N \rceil$ trees in a binomial forest.
- Each merge operation only looks at the root of each tree.
- Total time to merge is O(log N).

Insert

- Create a single node queue B_0 with the new item and merge with existing queue
- O(log N) time

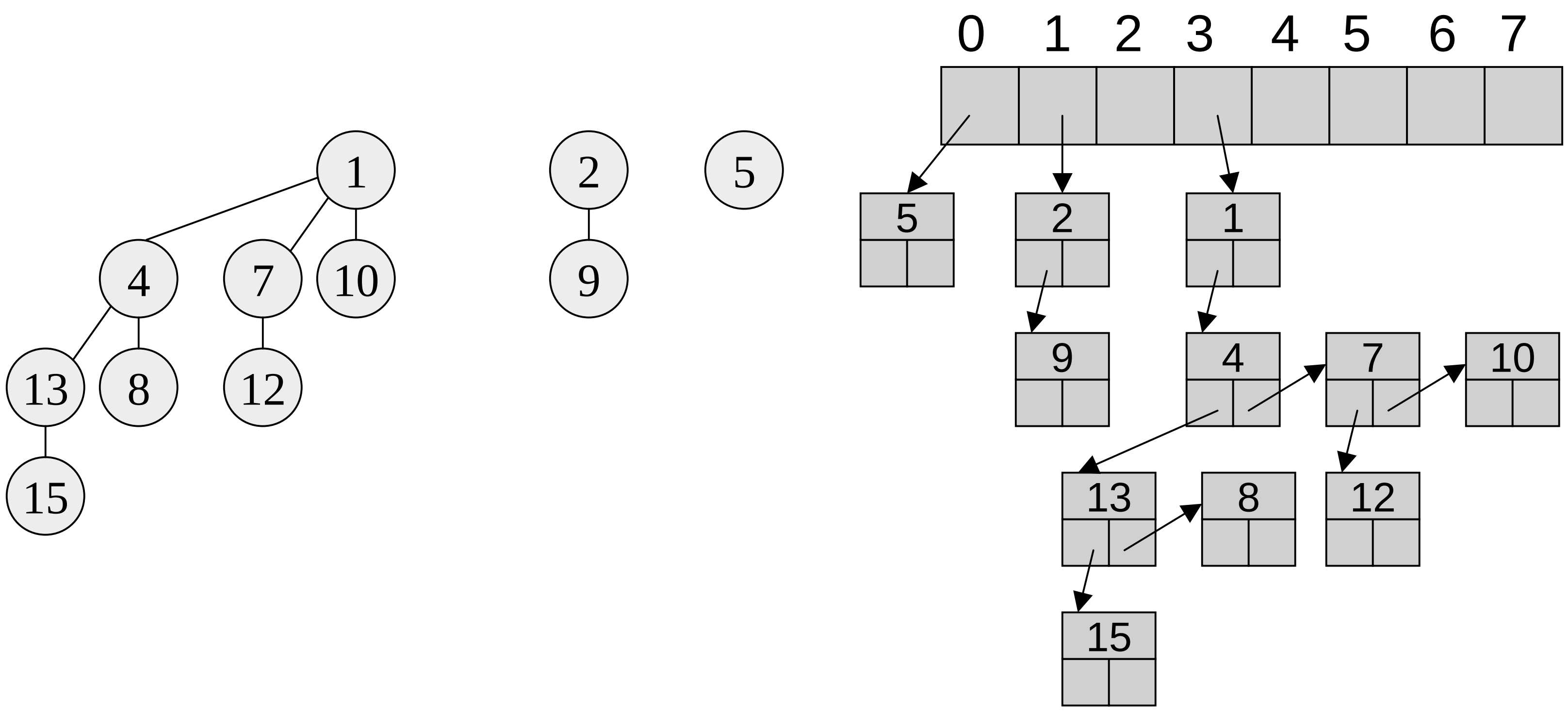
DeleteMin

1. Assume we have a binomial forest $X_0,\ldots,X_m$
2. Find tree X_k with the smallest root
3. Remove X_k from the queue
4. Remove root of X_k (return this value)
   - This yields a binomial forest Y_0, Y_1, …,Y_{k-1}.
5. Merge this new queue with remainder of the original (from step 3)

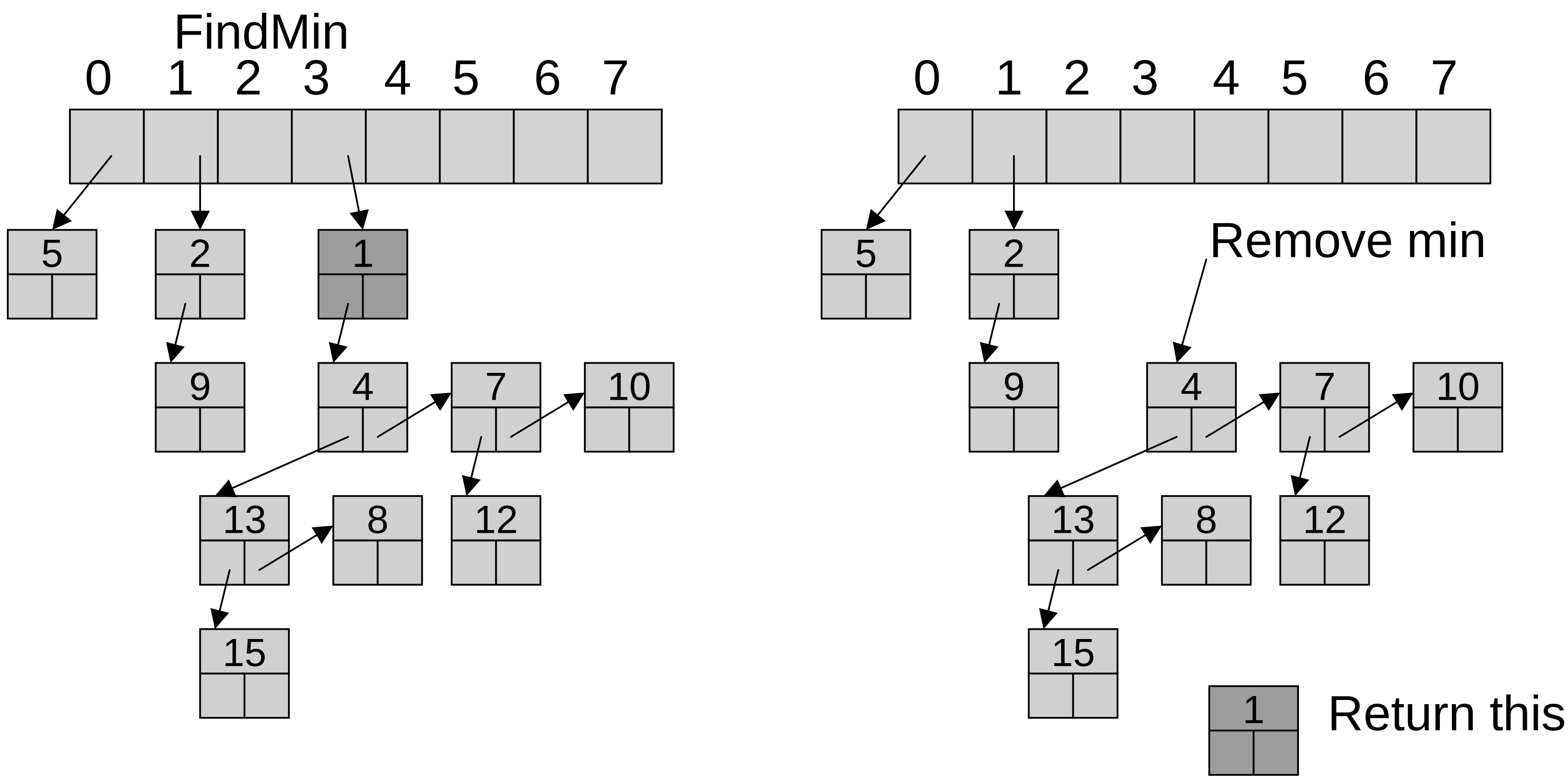
- Total time = O(log N)

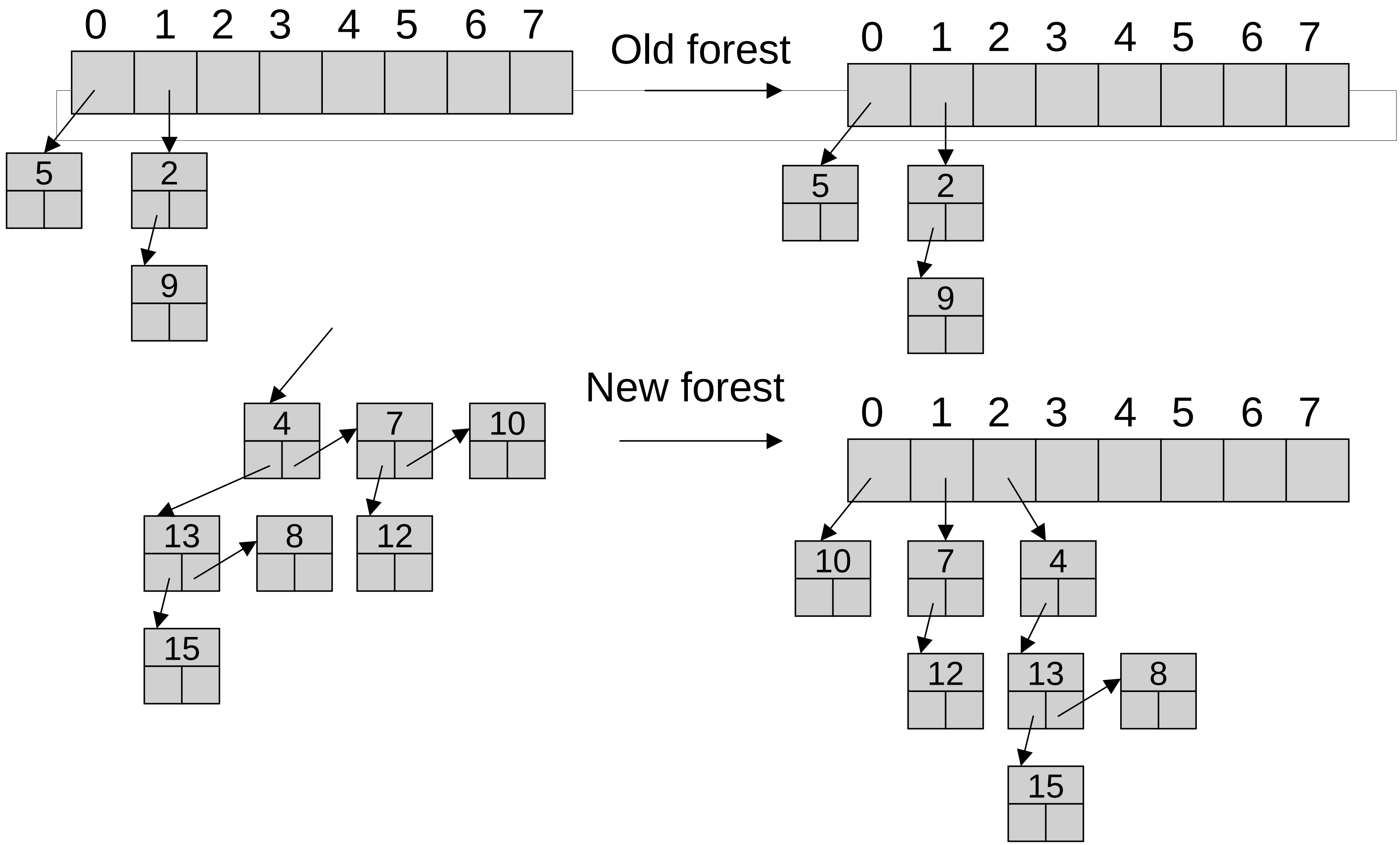
Implementation

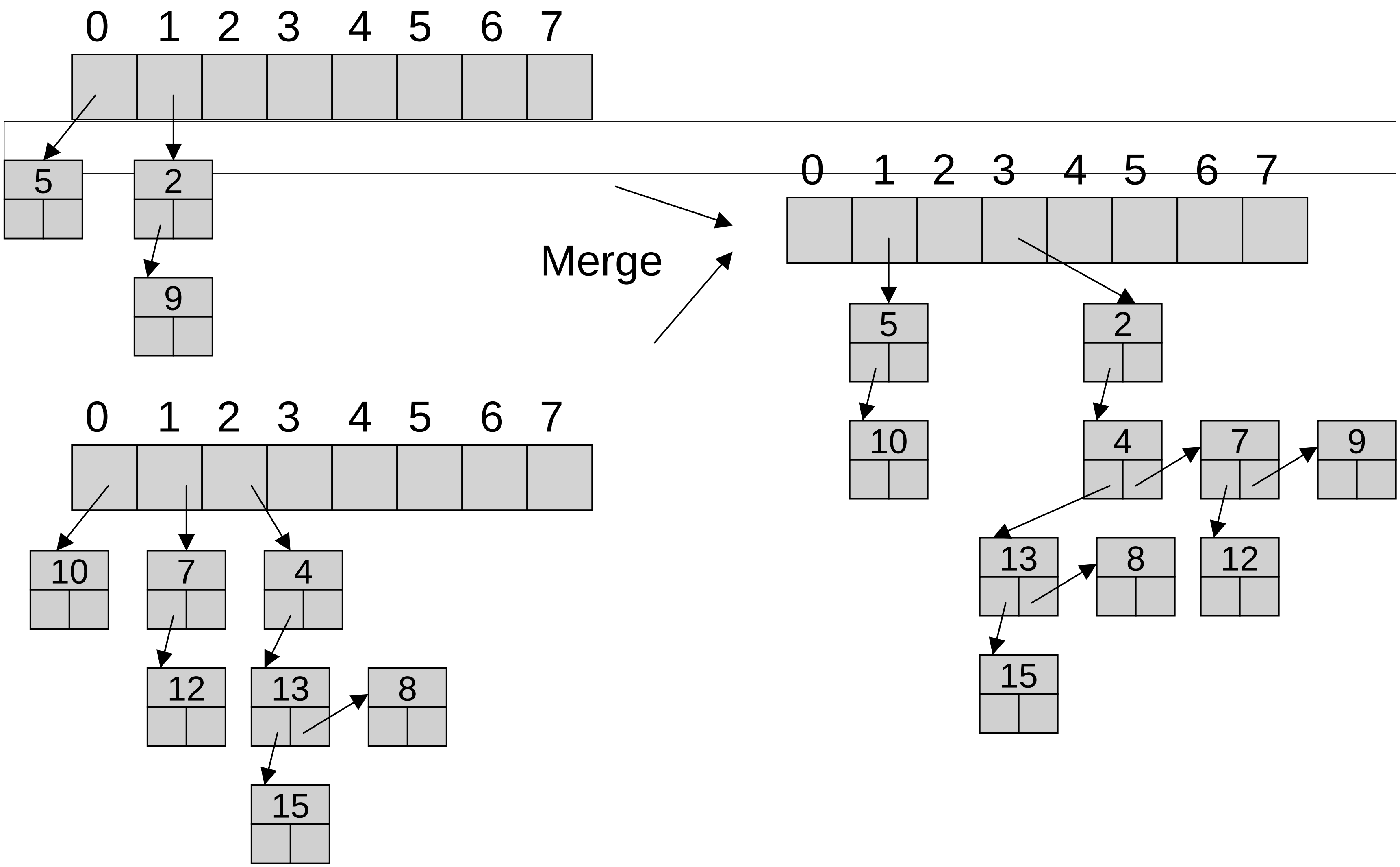
- Binomial forest as an array of multiway trees
  - › FirstChild, Sibling pointers



DeleteMin Example



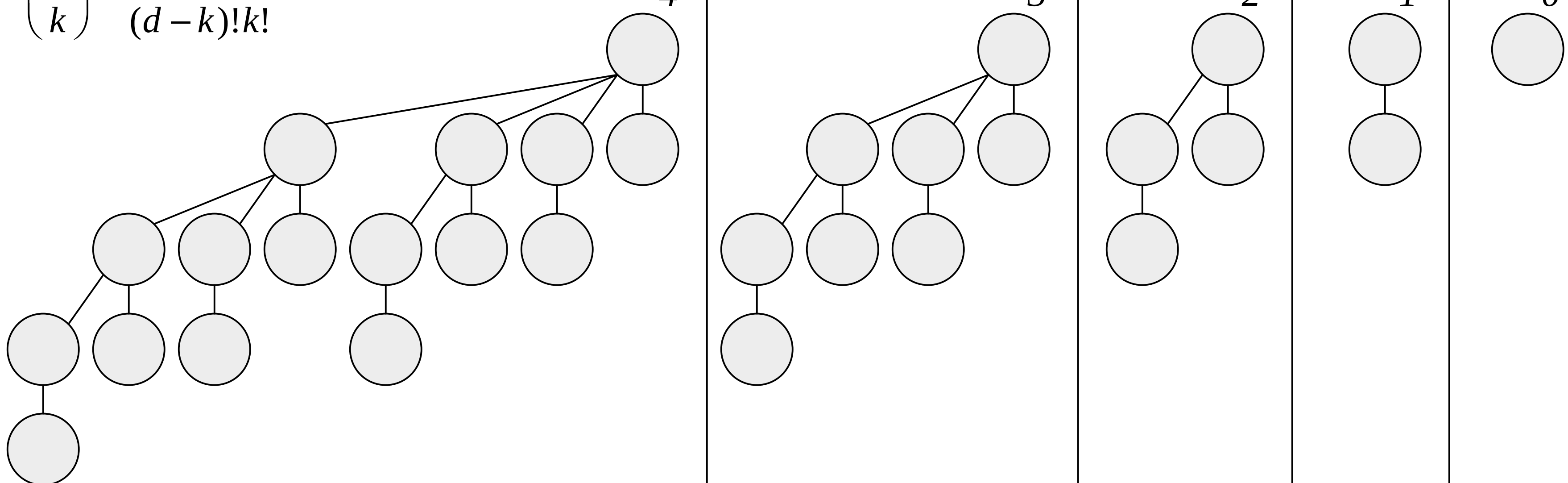

0 1 2 3 4 5 6 7
Old forest
0 1 2 3 4 5 6 7
5
2
9
5
2
9
4
7
10
13
8
12
15
New forest
0 1 2 3 4 5 6 7
10
7
4
12
13
8
15


0 1 2 3 4 5 6 7
5
2
9
0 1 2 3 4 5 6 7
10
7
4
12
13
8
15
Merge
0 1 2 3 4 5 6 7
5
2
10
4
7
9
13
8
12
15

Why Binomial?

$$\binom{d}{k} = \frac{d!}{(d-k)!k!}$$

B_4	B_3	B_2	B_1	B_0



tree depth d	4	3	2	1	0
nodes at depth k	1, 4, 6, 4, 1	1, 3, 3, 1	1, 2, 1	1, 1	1

Other Priority Queues

- Leftist Heaps
  - › O(log N) time for insert, deletemin, merge
  - › The idea is to have the left part of the heap be long and the right part short, and to perform most operations on the left part.
- Skew Heaps ("splaying leftist heaps")
  - › O(log N) amortized time for insert, deletemin, merge

Exercise Solution