Introduction

Computational Problems

- A computational problem is described by
  - Input
  - Output
  - The relationship between the input and the output

Multiplication

- Given two numbers, compute their product.
- Input: x, y
- Output: x×y
- Sample input: 3, 5
- Sample output: 15

Sorting

- Given a sequence of n numbers $\langle a_1,\ldots,a_n \rangle$, reorder it into $\langle b_1,\ldots,b_n \rangle$ where $b_1 \leq \ldots \leq b_n$.
- Input: $\langle a_1,\ldots,a_n \rangle$
- Output: $\langle b_1,\ldots,b_n \rangle$
- Sample Input: $\langle 0,3,5,7,1,2,1,2,1 \rangle$
- Sample Output: $\langle 0,1,1,1,2,2,3,5,7 \rangle$

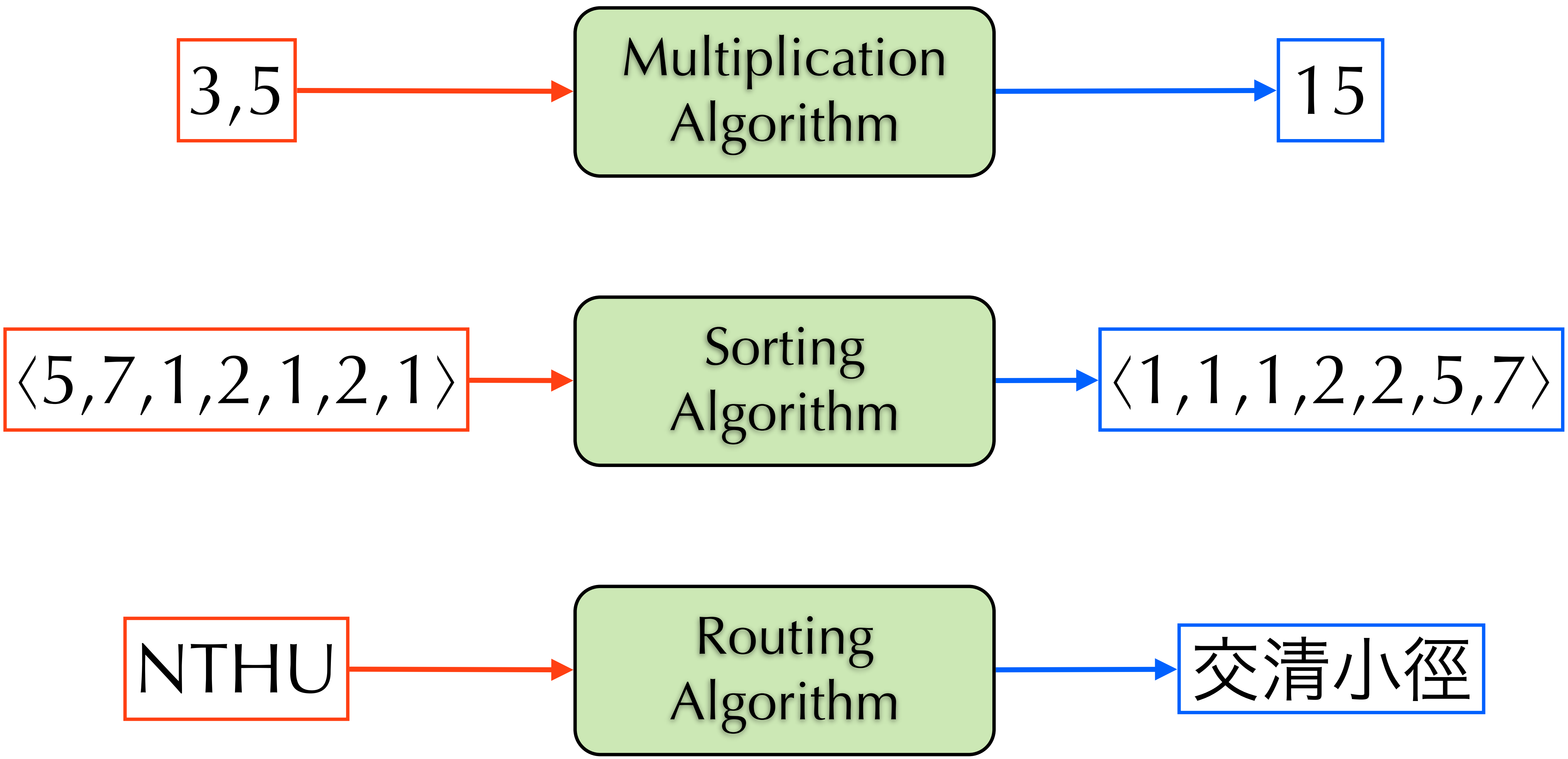
Routing

- Find out a route from NCTU to a given destination.
- Input: a location L
- Output: a path from NCTU to L
- Sample Input: NTHU
- Sample Output: 交清小徑

Algorithm

- A well-defined computation procedure
  - Takes input
  - Produces output
- A sequence of computational steps transforms the input to the output
- An algorithm is correct if, for every input, it halts with the correct output.

Algorithm



3,5
Multiplication Algorithm
15
⟨5,7,1,2,1,2,1⟩
Sorting Algorithm
⟨1,1,1,2,2,5,7⟩
NTHU
Routing Algorithm
交清小徑

Algorithm

- A natural question: Does every problem have a correct algorithm?
- A sad answer: No. The halting problem does not have any correct algorithm.
- The related topics are discussed in the formal language course.
- Good news: we only discuss the problems which have correct algorithms in this course.

Algorithm Design

- Two main approaches:
  - Incremental
    - Repeated addition
    - Insertion sort
  - Divide and conquer
    - Long multiplication
    - Merge sort

Repeated Addition

- I believe you know this algorithm.
- The following is a sample code in C:

```
for(prod=x,i=1; i<y; i++)
   prod=prod+x;
printf("%d",x);
```

Long Multiplication

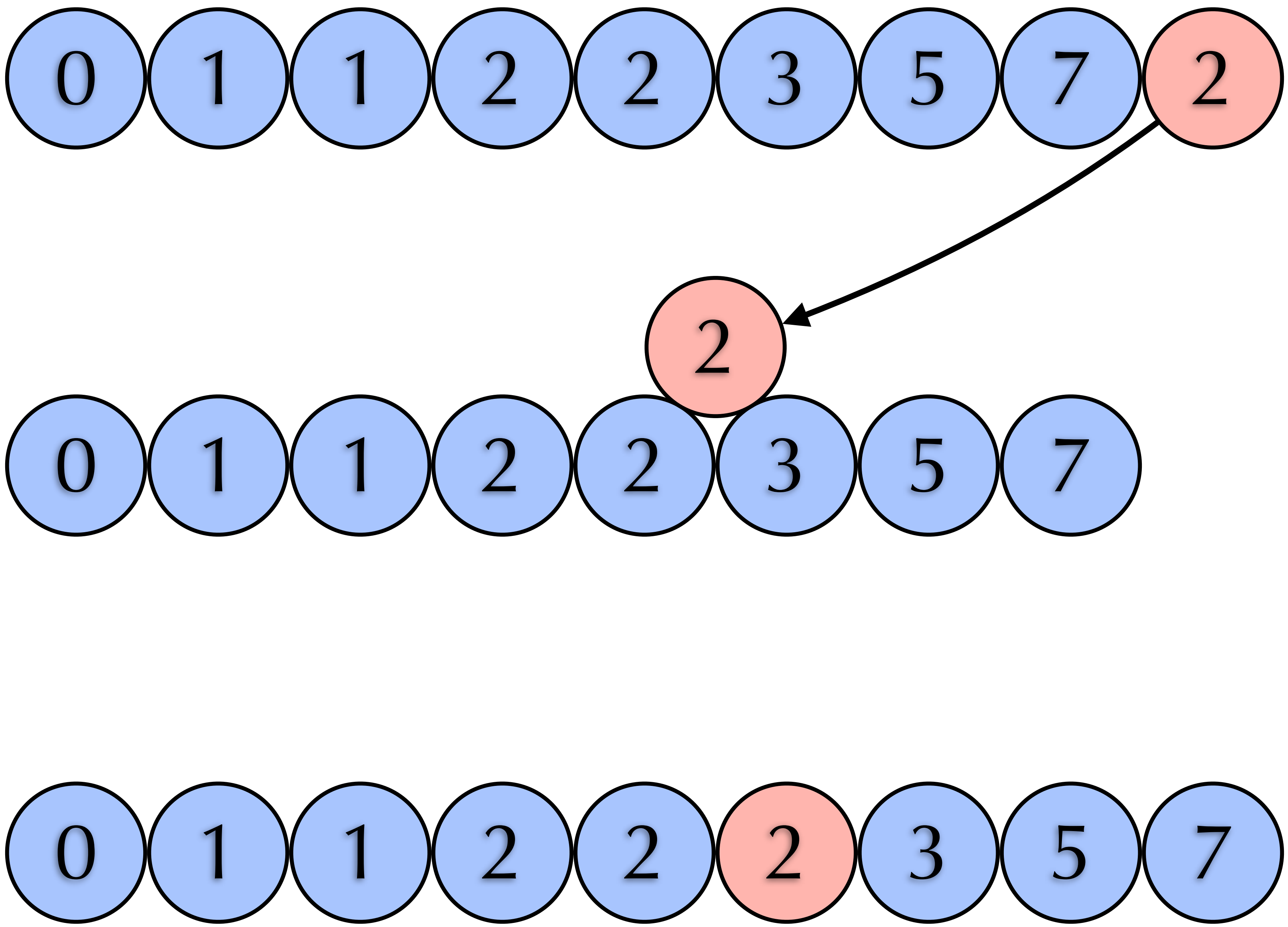
- I believe you know this algorithm, too.
- Example:

```
   234     234     234     234
  ×123    ×  3    ×  2    ×  1
  ----    ----    ----    ----
   702      12       8       4
  468       9       6       3
 234       6       4       2
 -----    ----    ----    ----
 28782    702     468     234
```

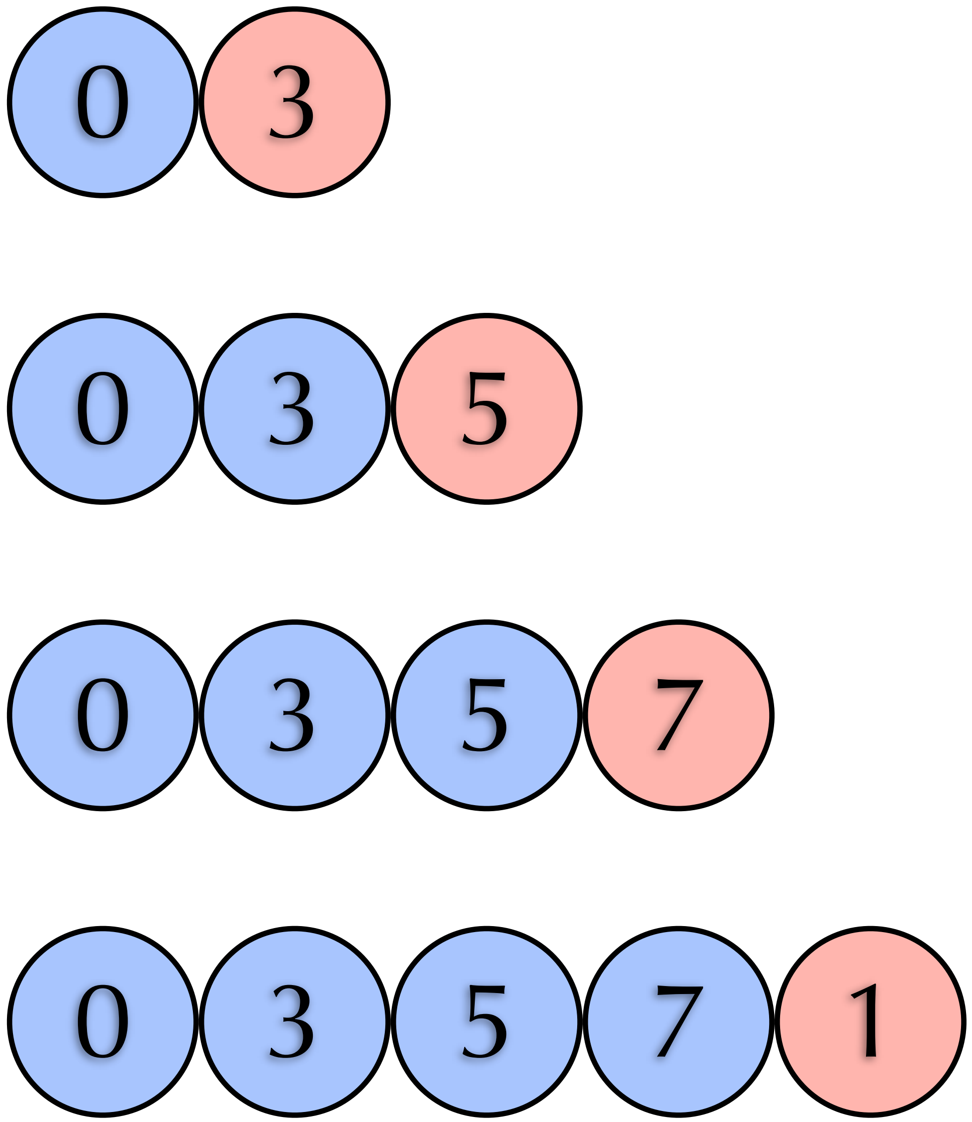
Insertion Sort

- Maintain a sorted list L.
- Repeat inserting a number into L until all numbers are inserted into L.
- Incremental: after each iteration, the length of L is increased by one.

Insert

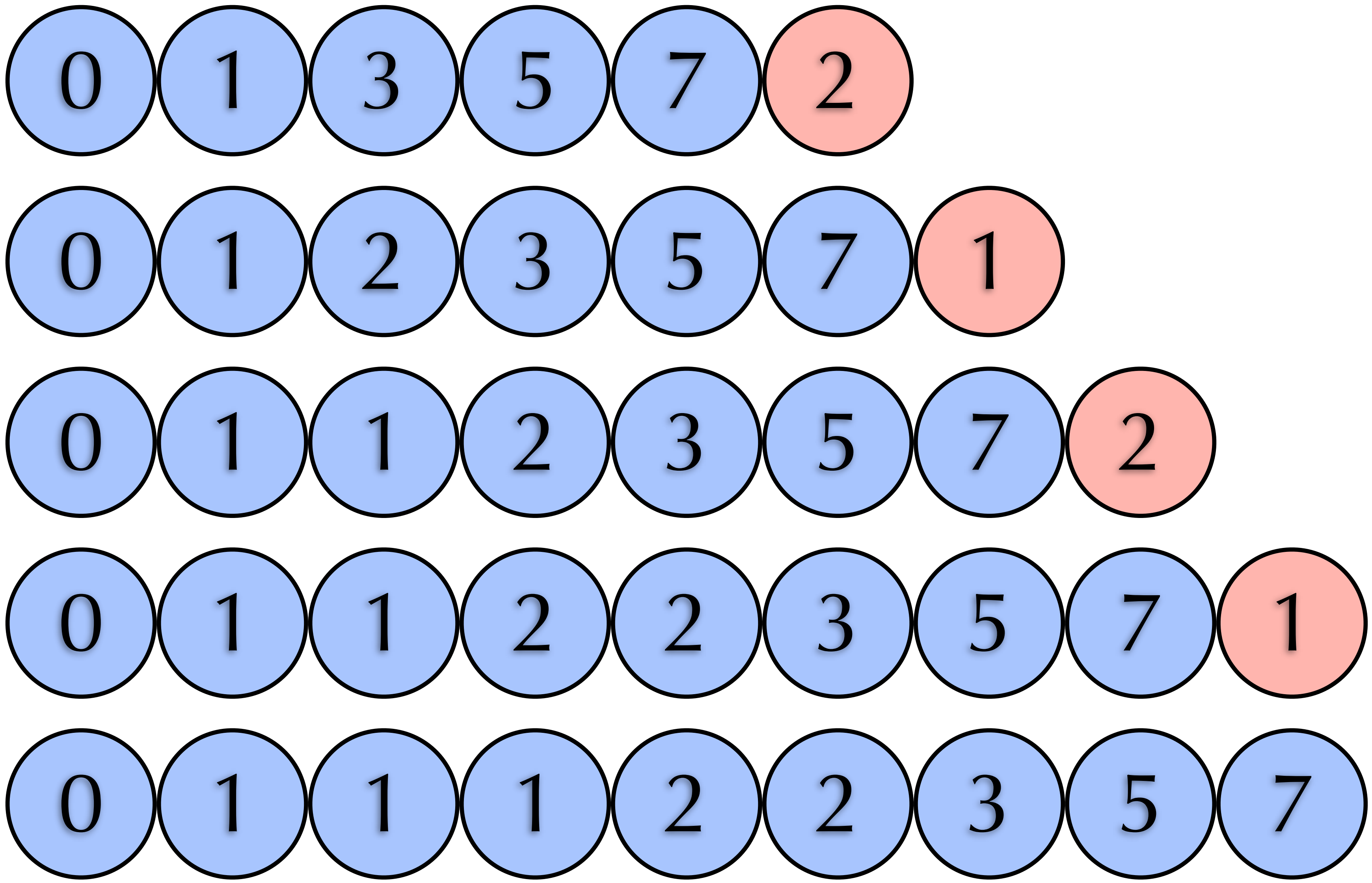


Insertion Sort



$\boxed{\langle 0,3,5,7,1,2,1,2,1 \rangle}$

Insertion Sort

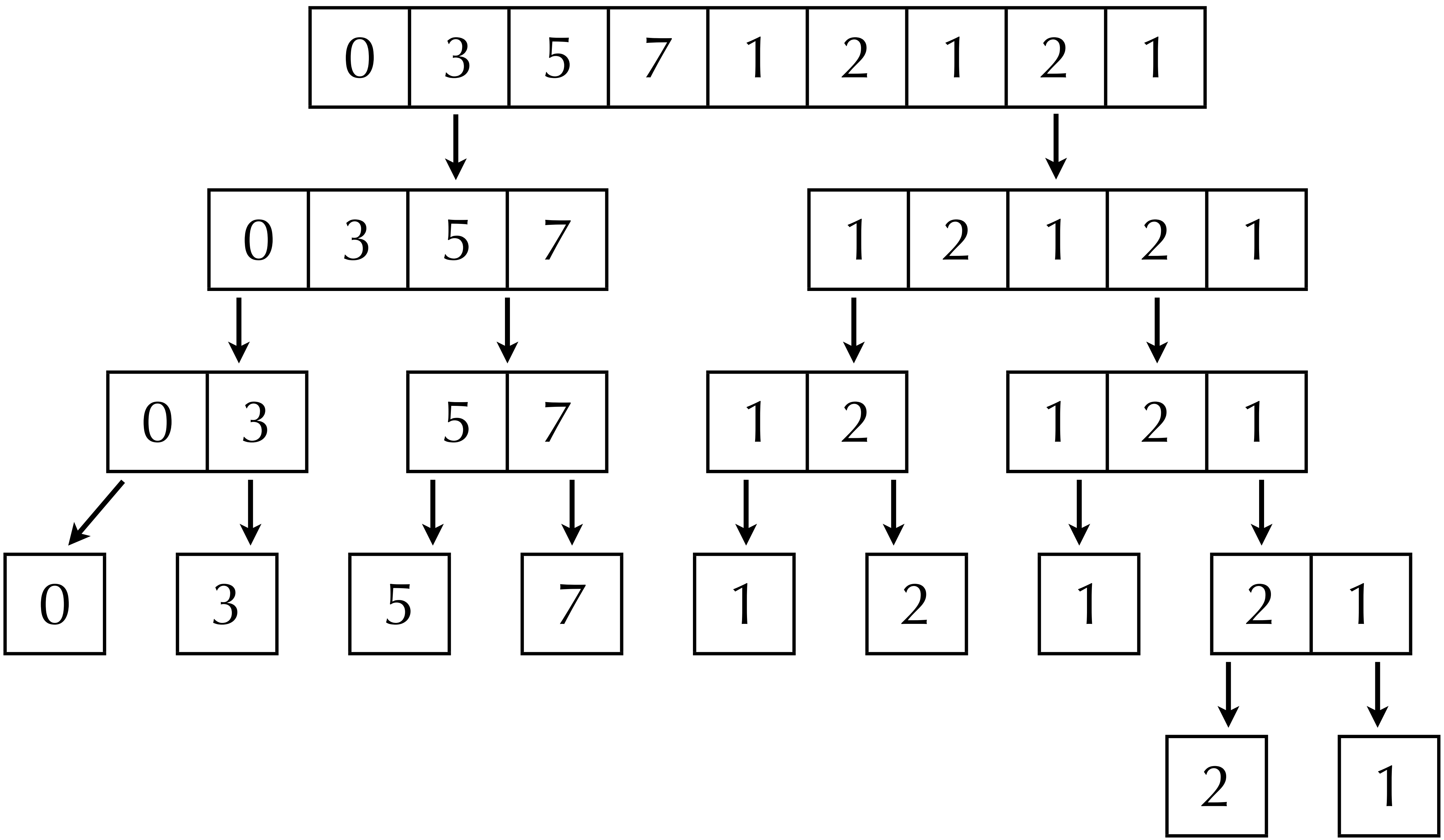


$\boxed{\langle 0,3,5,7,1,2,1,2,1 \rangle}$

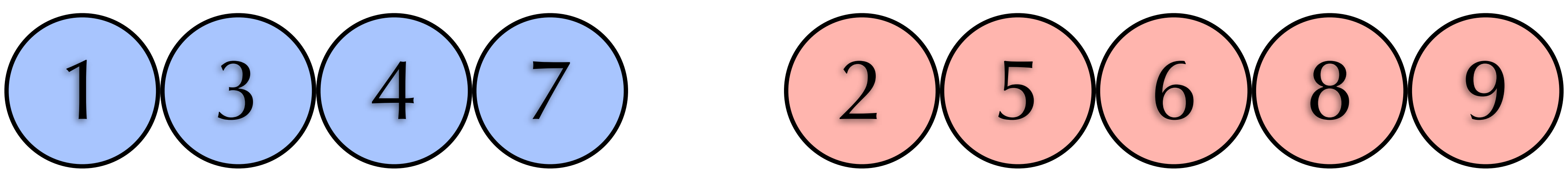
Merge Sort

- Input: $\langle a_1,\ldots,a_n\rangle$
- Termination: n=1. $\langle a_1\rangle$ is sorted.
- Divide: split $\langle a_1,\ldots,a_n\rangle$ into $\langle a_1,\ldots,a_{n/2}\rangle$ and $\langle a_{1+n/2},\ldots,a_n\rangle$.
- Conquer: Sort $\langle a_1,\ldots,a_{n/2}\rangle$ and $\langle a_{1+n/2},\ldots,a_n\rangle$
- Combine: Merge two sorted lists into a sorted list $\langle b_1,\ldots,b_n\rangle$

Merge Sort

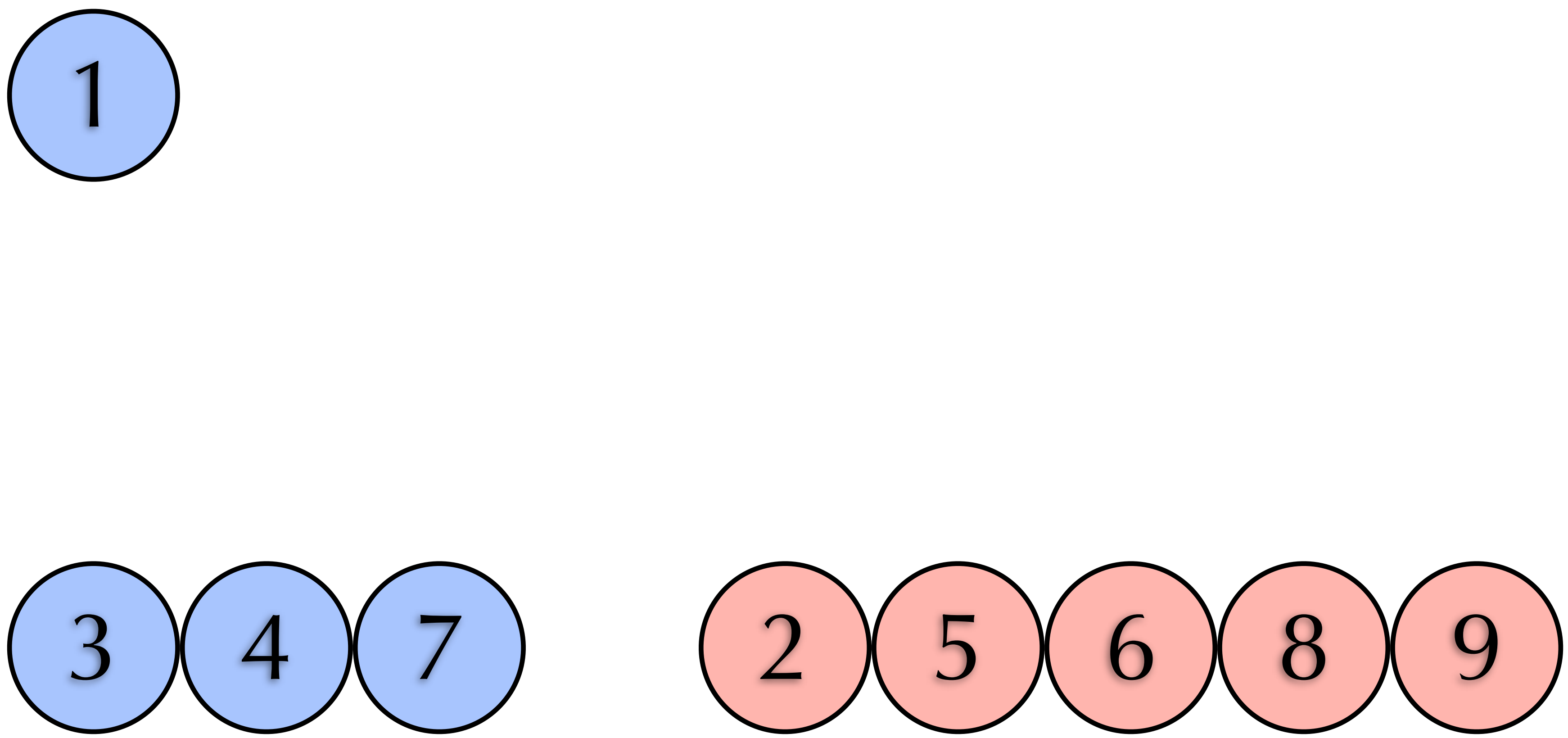


Merge



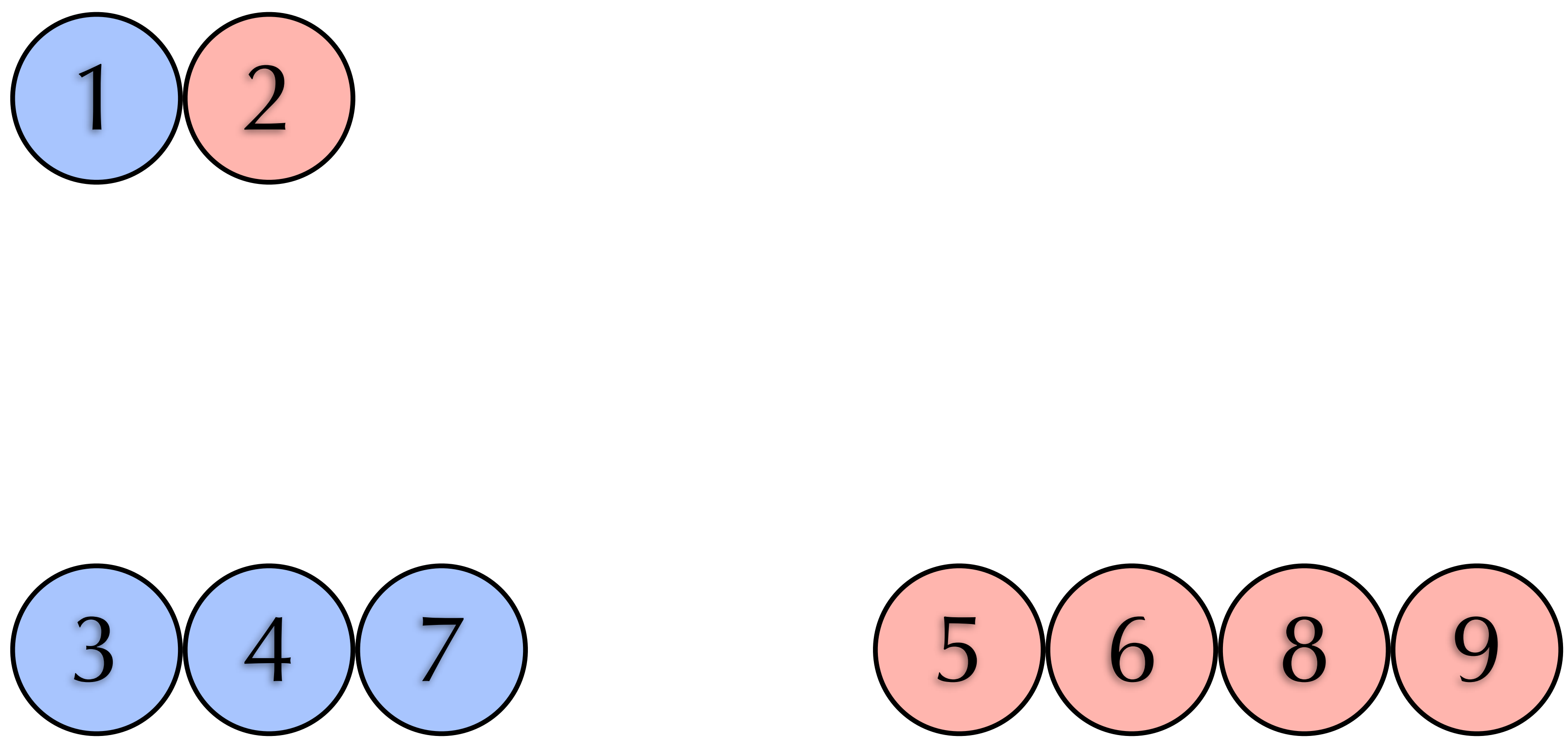
1<2

Merge



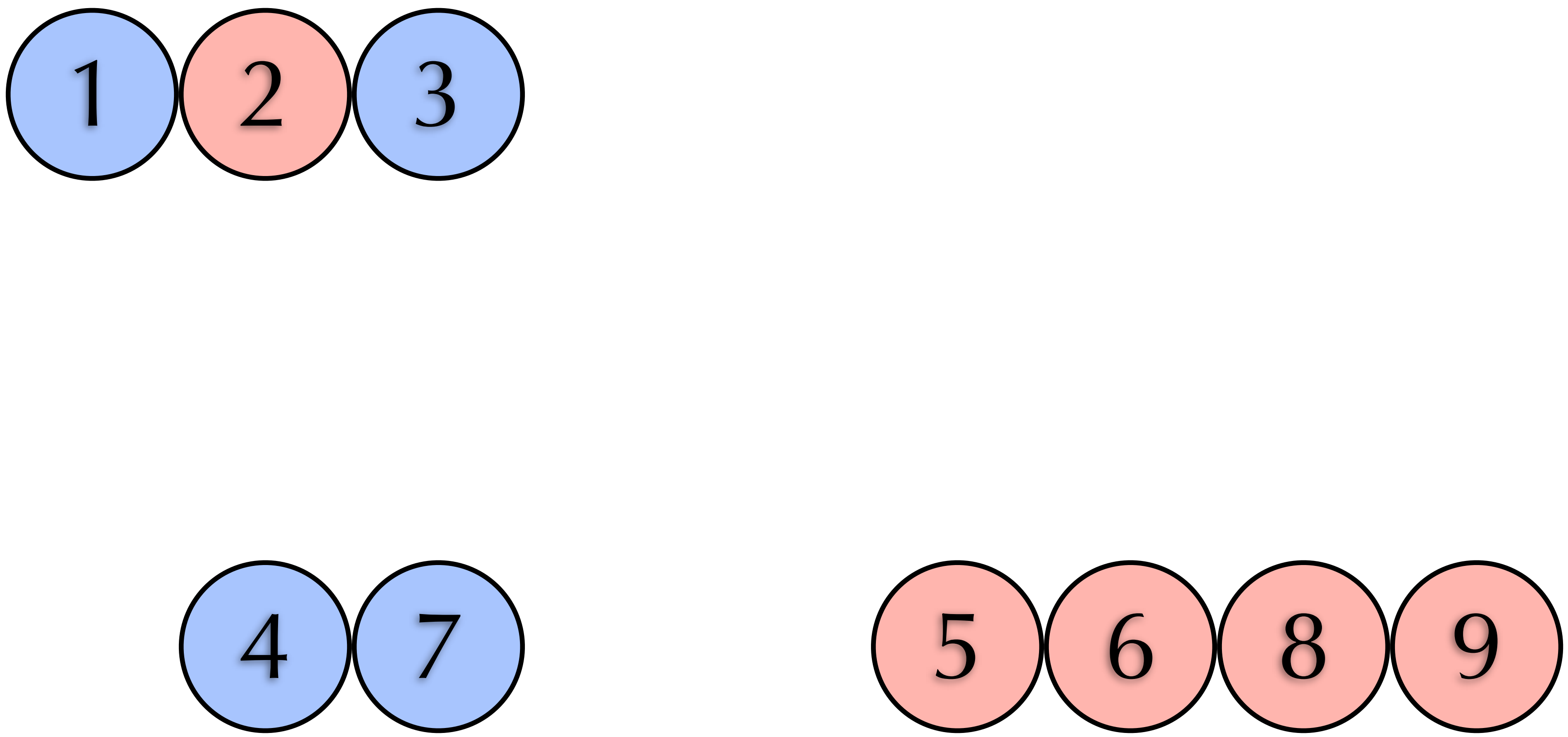
3>2

Merge



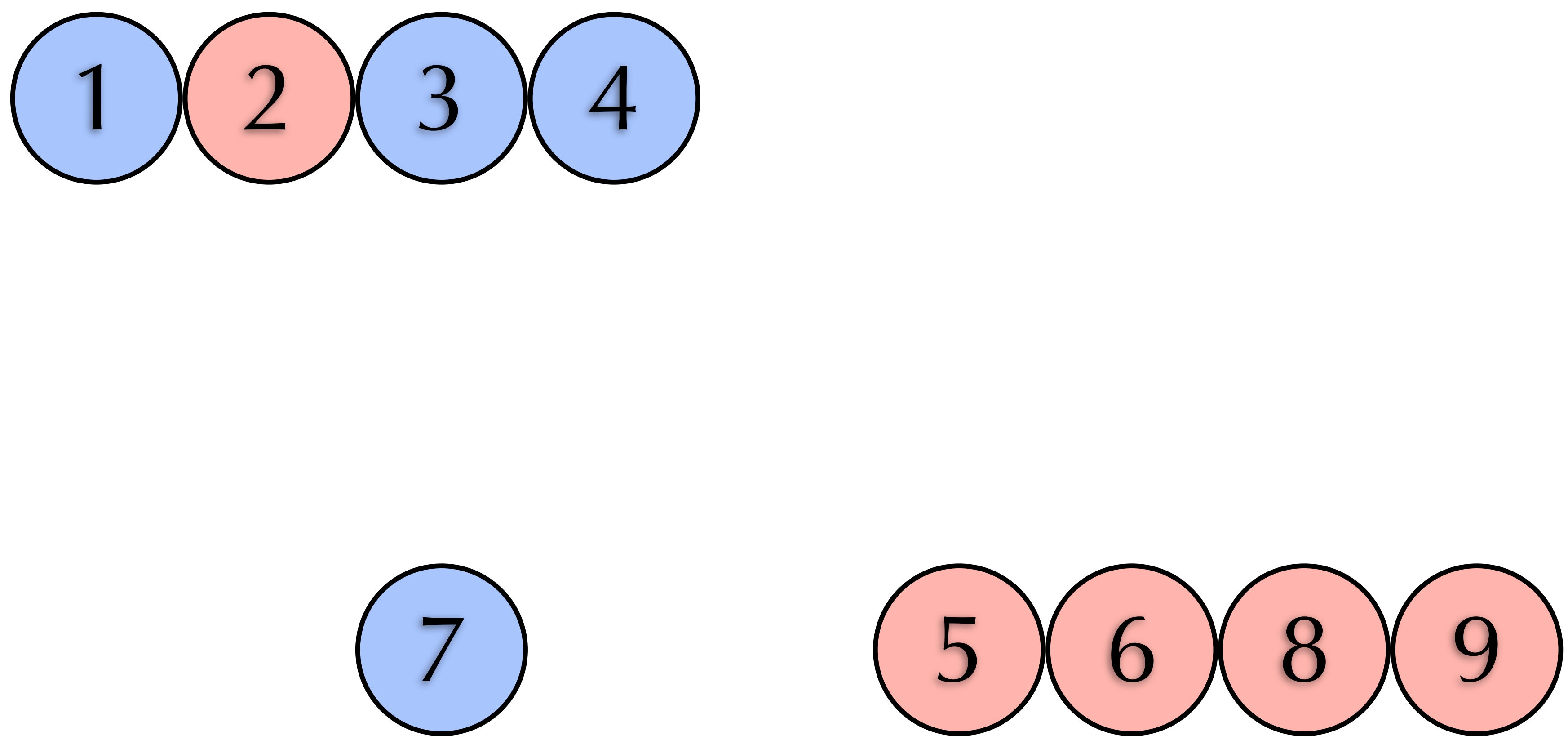
3<5

Merge



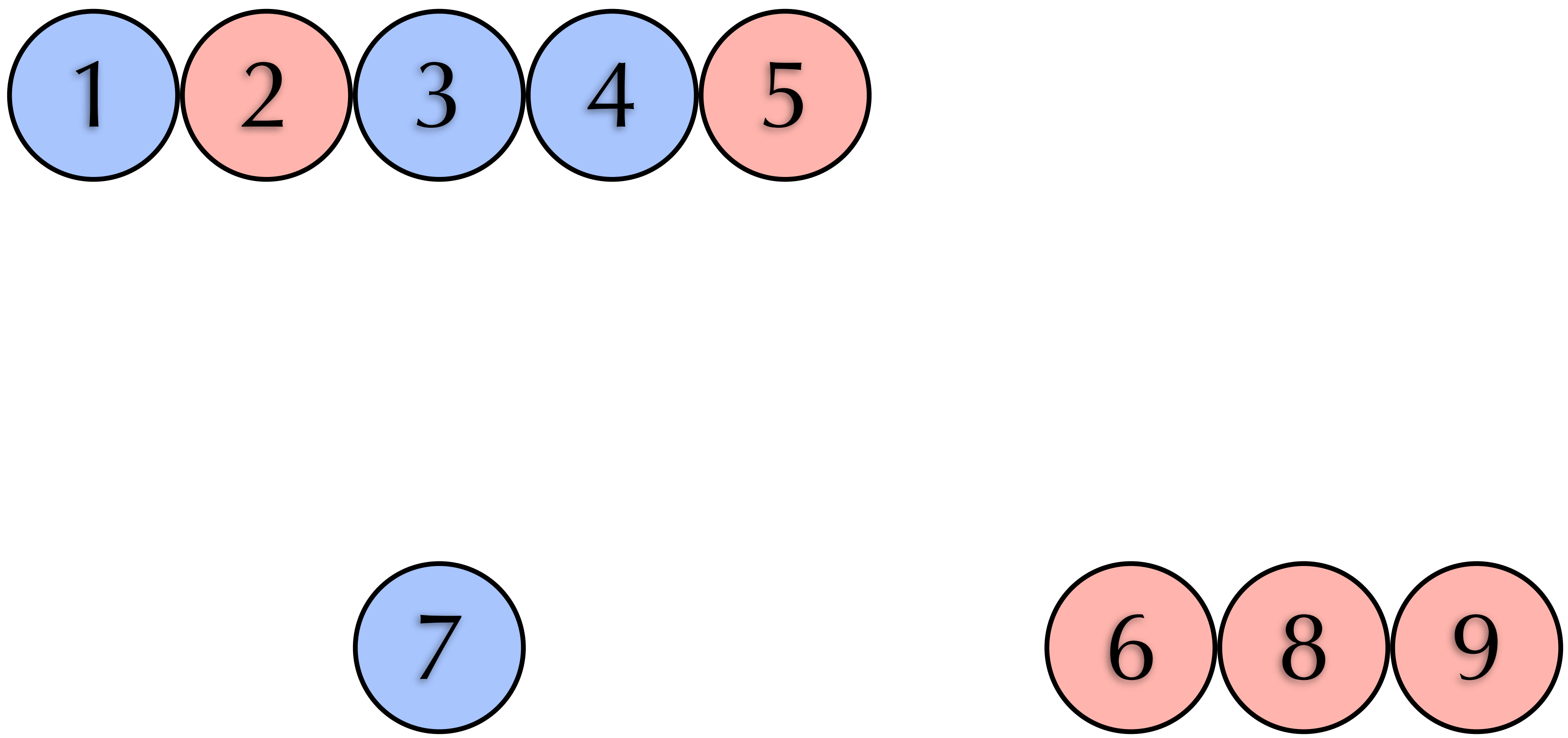
$4<5$

Merge



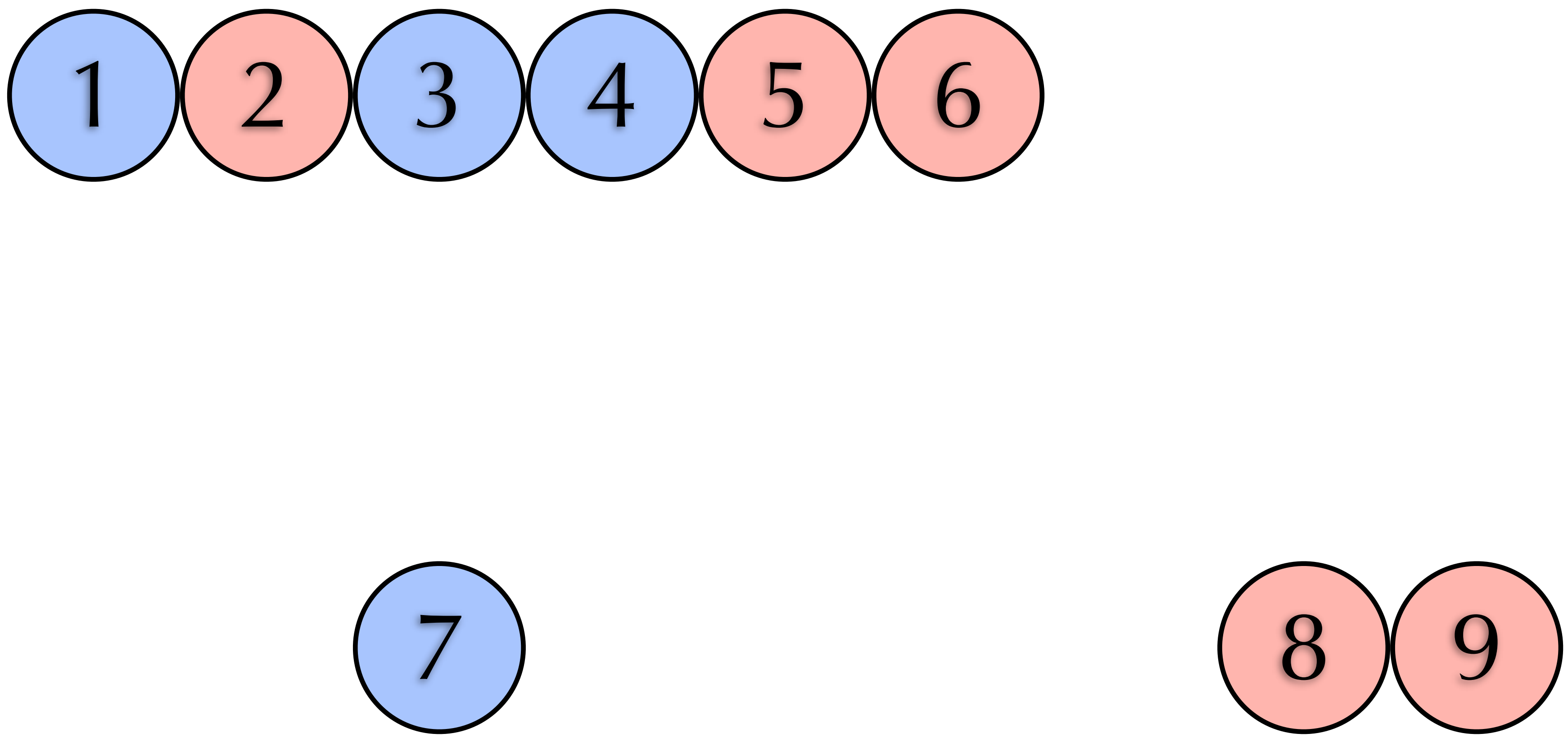
7>5

Merge



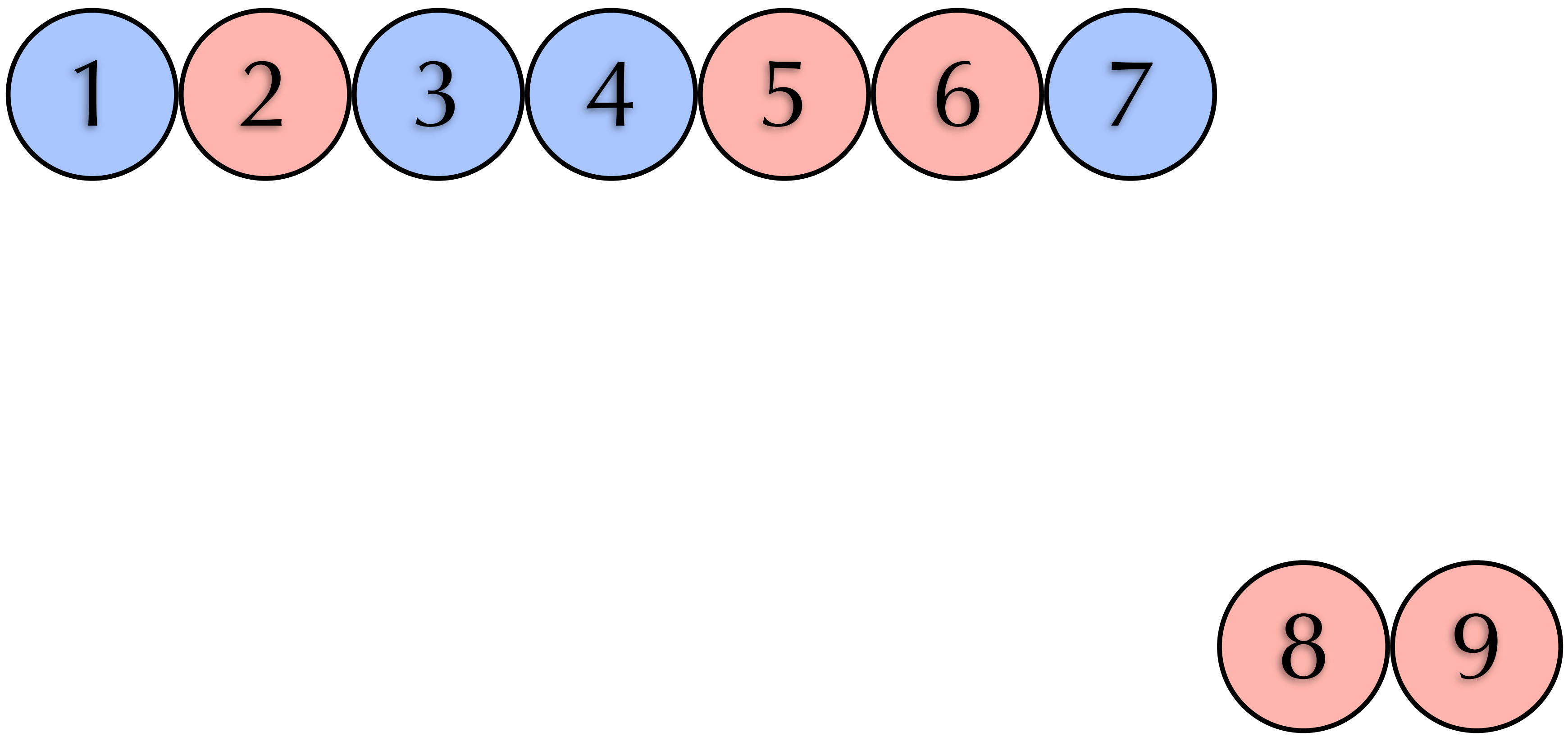
7>6

Merge



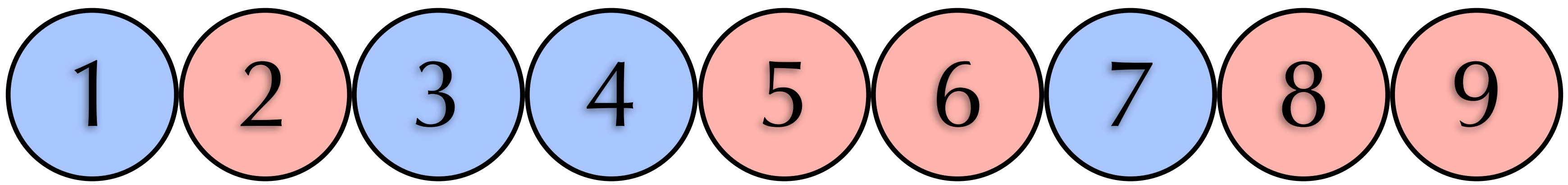
$7<8$

Merge

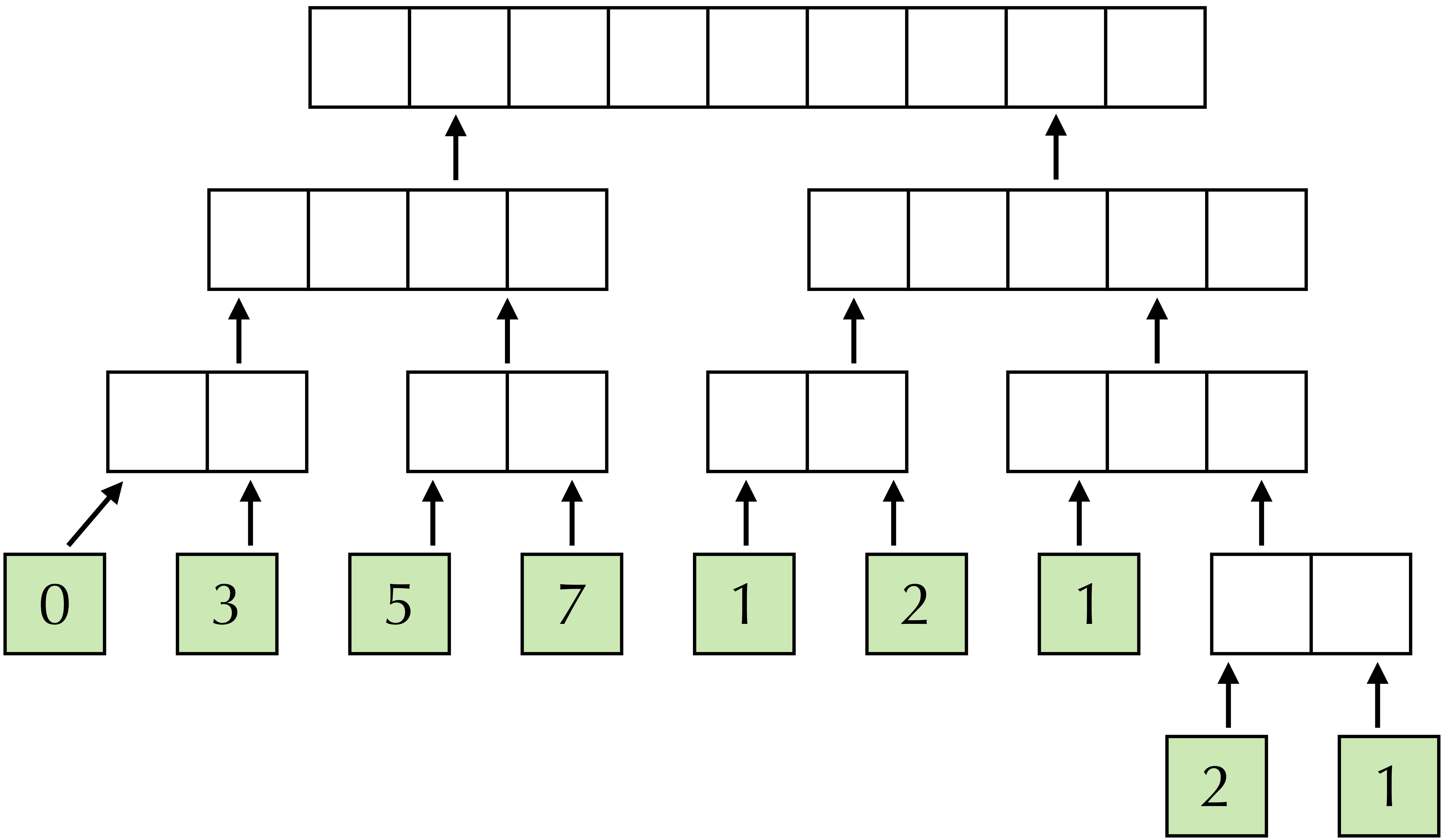


Left half is empty!

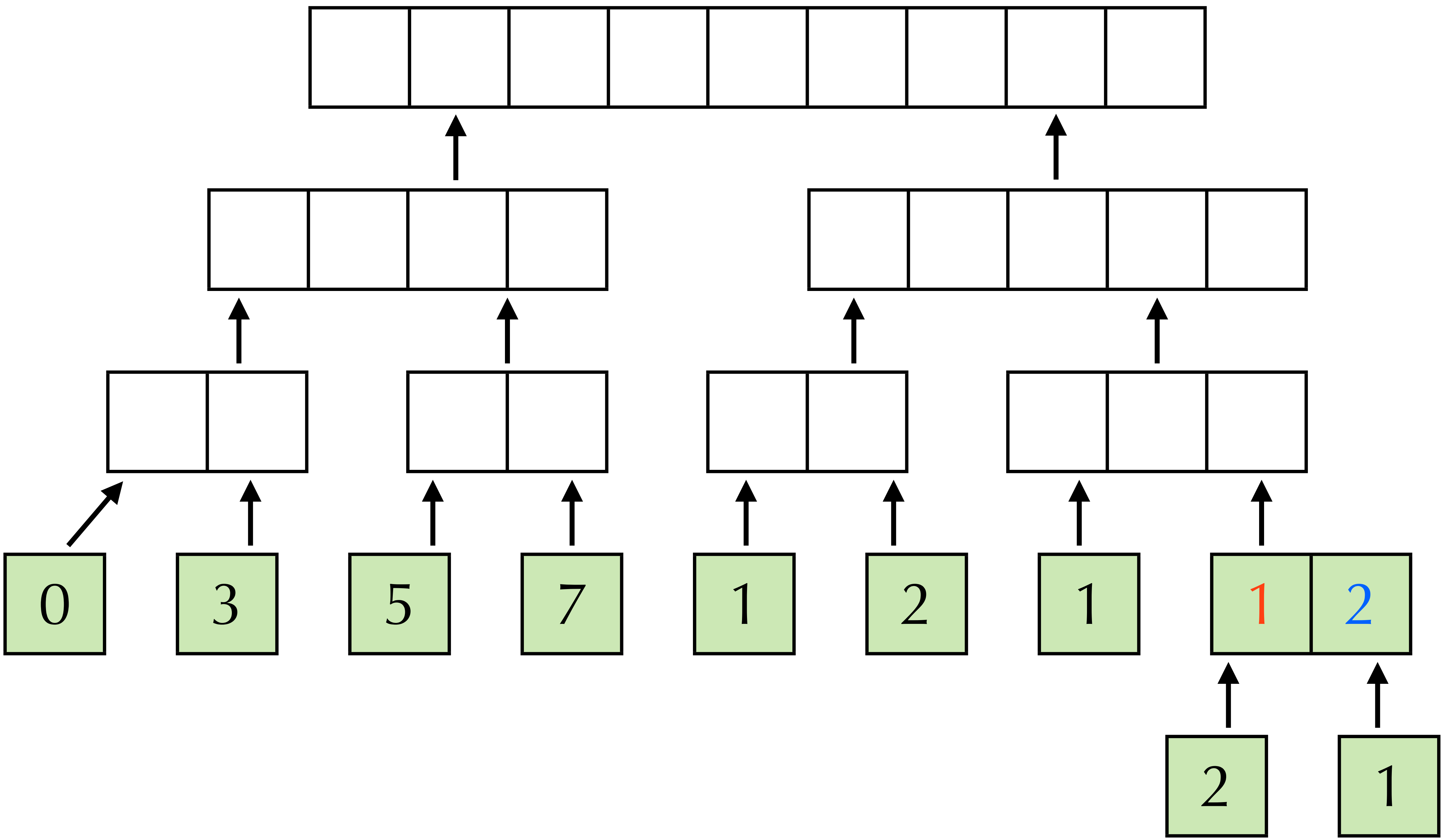
Merge



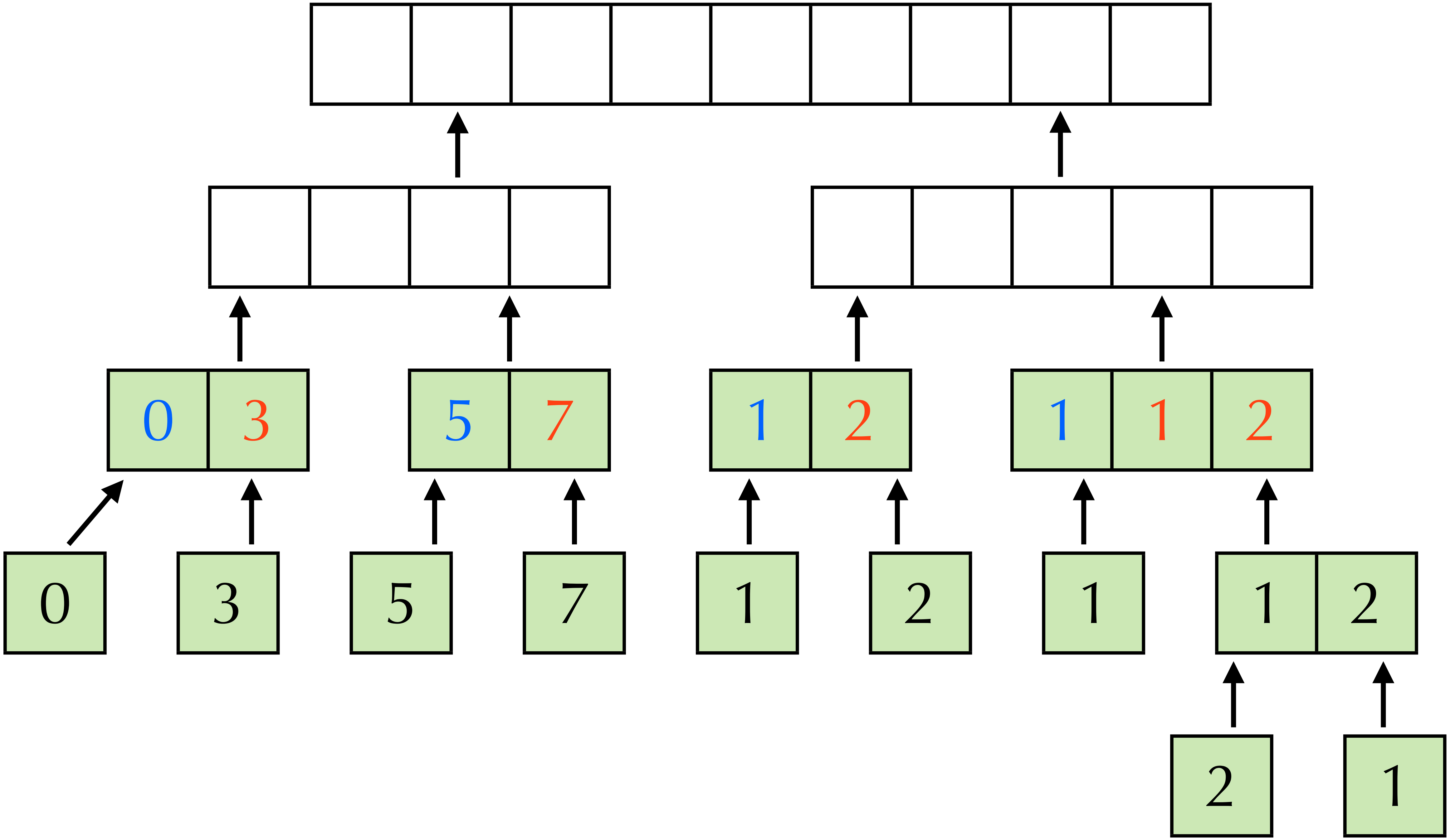
Merge Sort



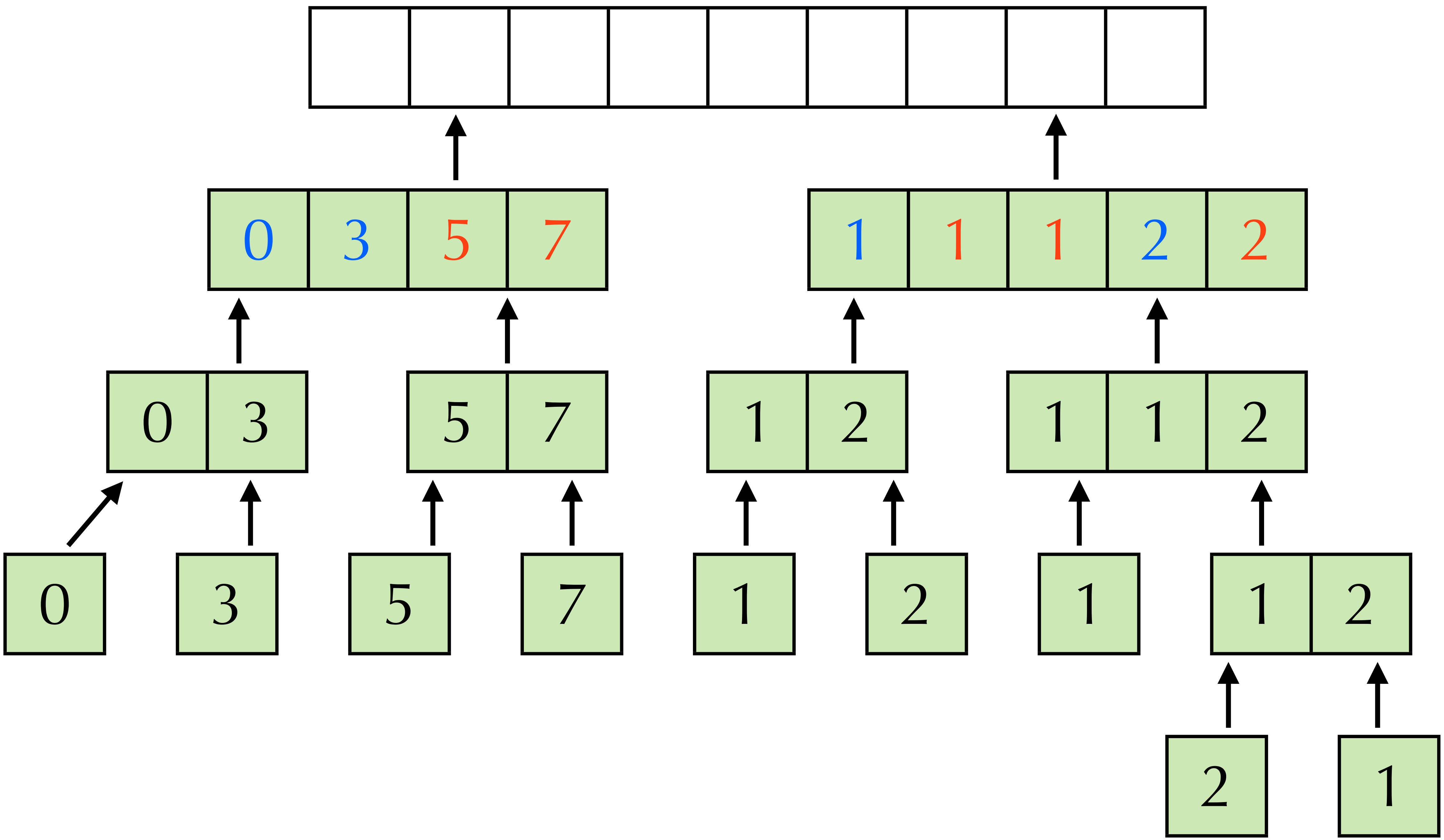
Merge Sort



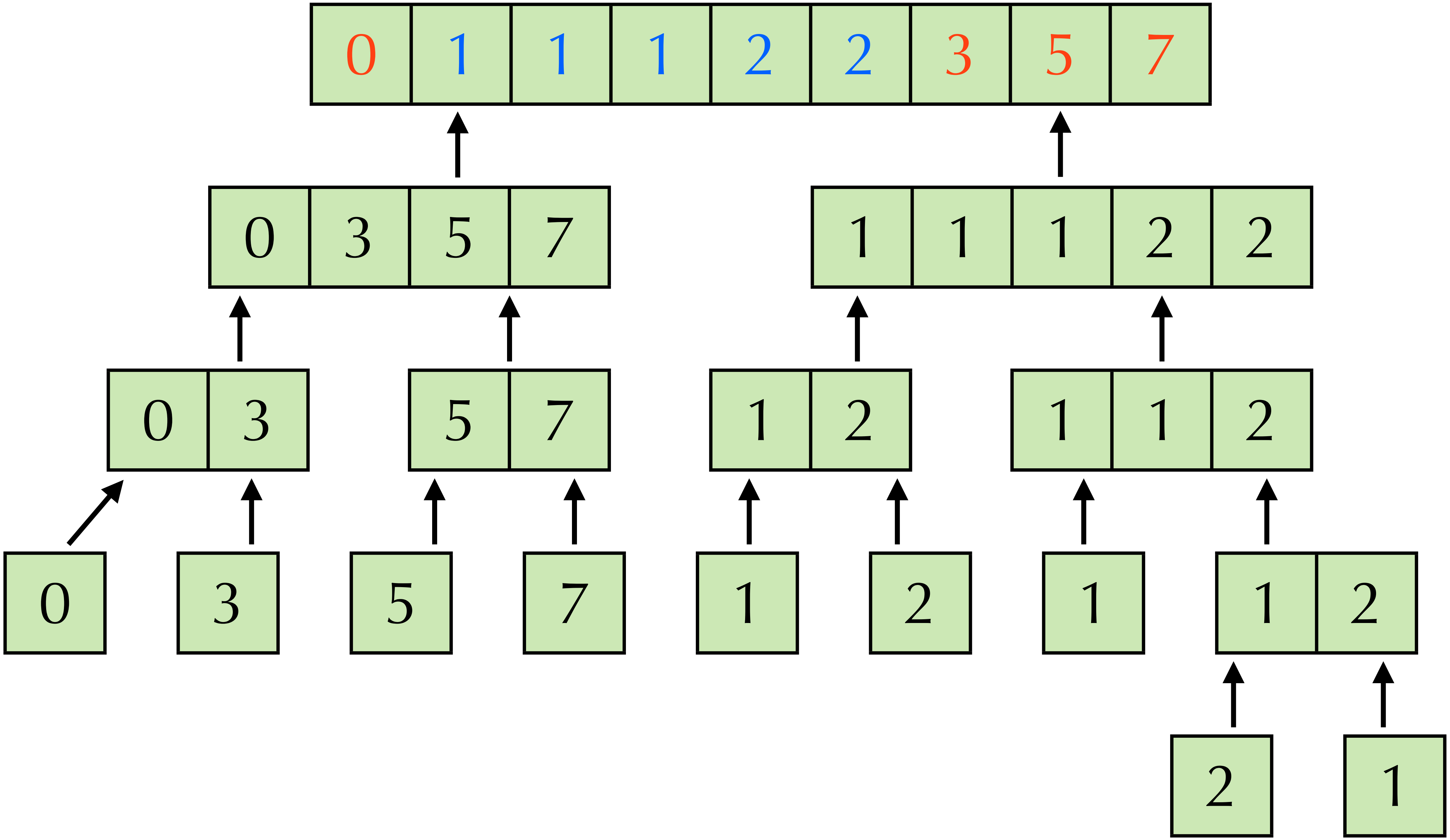
Merge Sort



Merge Sort



Merge Sort



Computational Complexity

- Complexity: the resources required to finish the execution of the algorithm.
  - Time complexity: number of steps
  - Space complexity: extra storage used
  - Kolmogorov complexity: description length
- In this course, we mainly discuss time complexity.

Asymptotic notation

- To precisely measure complexity is hard.
- For large enough n:
  - $T(n)=O(f(n))$: $T(n) \le c_1 f(n)$
  - $T(n)=\Omega(f(n))$: $T(n) \ge c_2 f(n)$
  - $T(n)=\Theta(f(n))$: $c_2 f(n) \le T(n) \le c_1 f(n)$
- $T(n)=o(f(n))$: $\lim_{n\to\infty} T(n)/f(n)=0$
- $T(n)=\omega(f(n))$: $\lim_{n\to\infty} f(n)/T(n)=0$

$\boxed{c_1 \text{ and } c_2 \text{ are constants}}$

Asymptotic notation

small-ω	big-Ω	Θ	big-O	small-o
$>$	$\geq$	$=$	$\leq$	$<$

Multiplication: Complexity

- Input: x, y
- Repeated addition:
  - $x-1$ or $y-1$ additions.
- Long multiplication:
  - $O(\log x \times \log y)$ additions and table lookups.

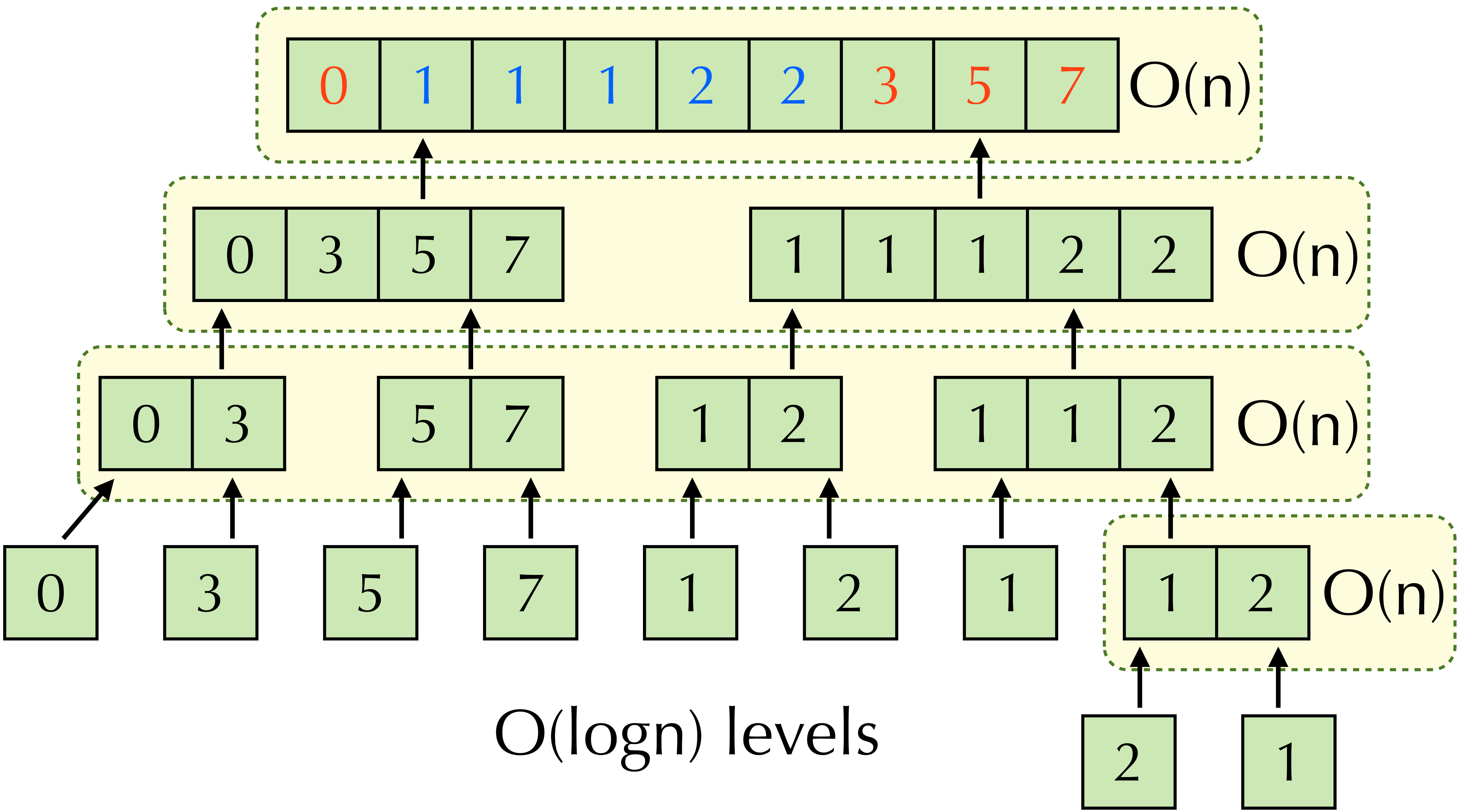
Insertion Sort: Complexity

- Input size: n
- Time complexity: T(n)
- Insert a_m into m−1 sorted numbers: O(m)
- $T(n)=T(n-1)+O(n)=O(1)+\ldots+O(n)=O(n^2)$

Merge Sort: Complexity

- Input size: n
- Time complexity: T(n)
- Divide: O(1)
- Conquer: 2T(n/2)
- Combine: O(n)
- $T(n)=2T(n/2)+O(n)=O(n\log n)$ (why?)

Merge Sort: Complexity



$O(n^2)$ versus $O(n\log n)$

- Suppose we can execute 10^8 instructions per second.
- When $n=10^7$: $n^2=10^{14}$ and $n\log n \approx 10^8$.
- We can sort 10^7 numbers in seconds by merge sort, but it might take a week to sort 10^7 numbers by insertion sort.

1 week = 7 days = 168 hours = 10080 minutes = 604800 seconds