CP312
Algorithm Design and Analysis I

LECTURE 3: MERGE SORT

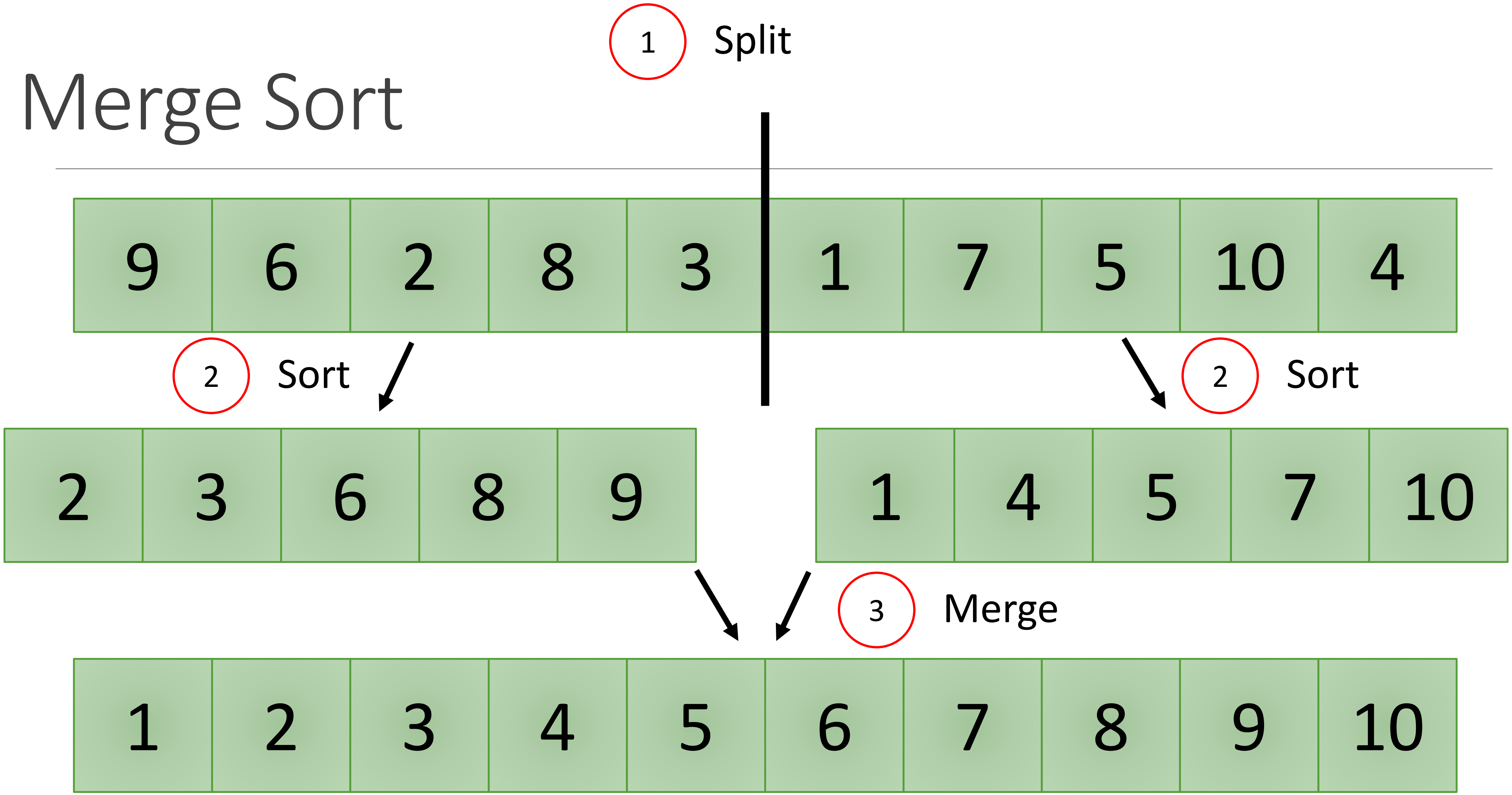
Algorithm Design

- There are various design techniques for building algorithms.

- We used the **incremental** approach to design insertion sort

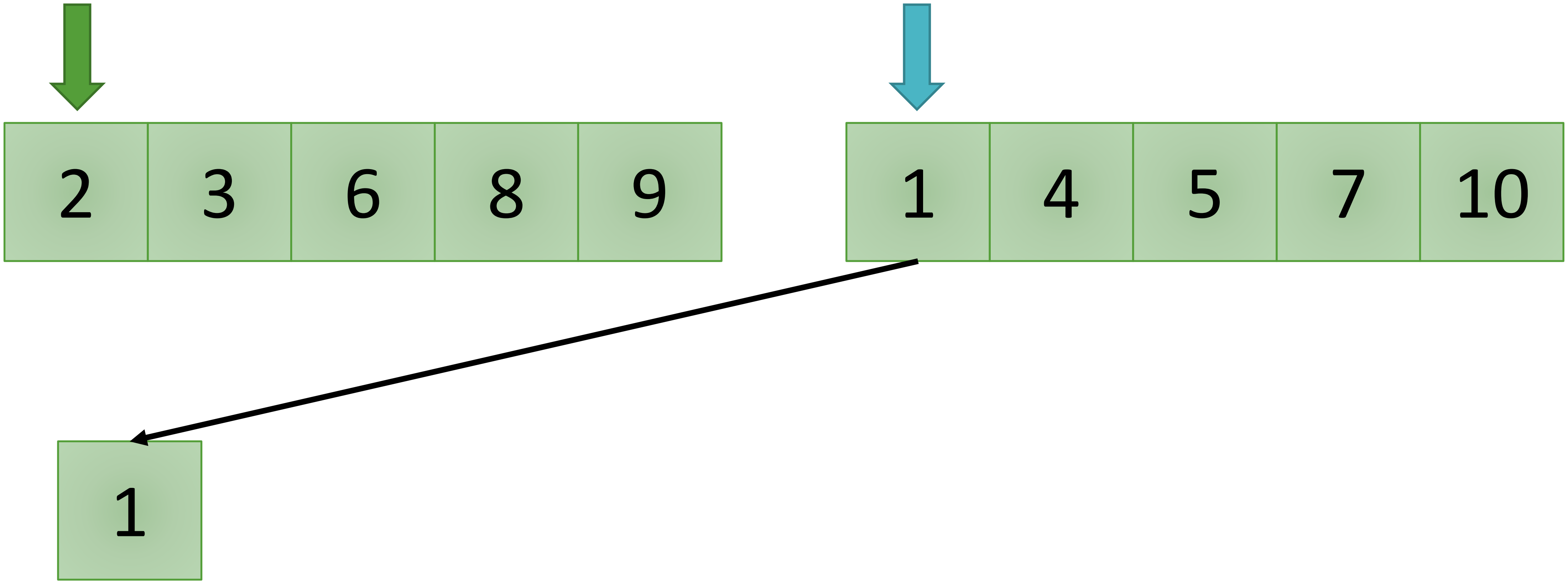
- Now we will see a different approach to design a different sorting algorithm.

Merge Sort

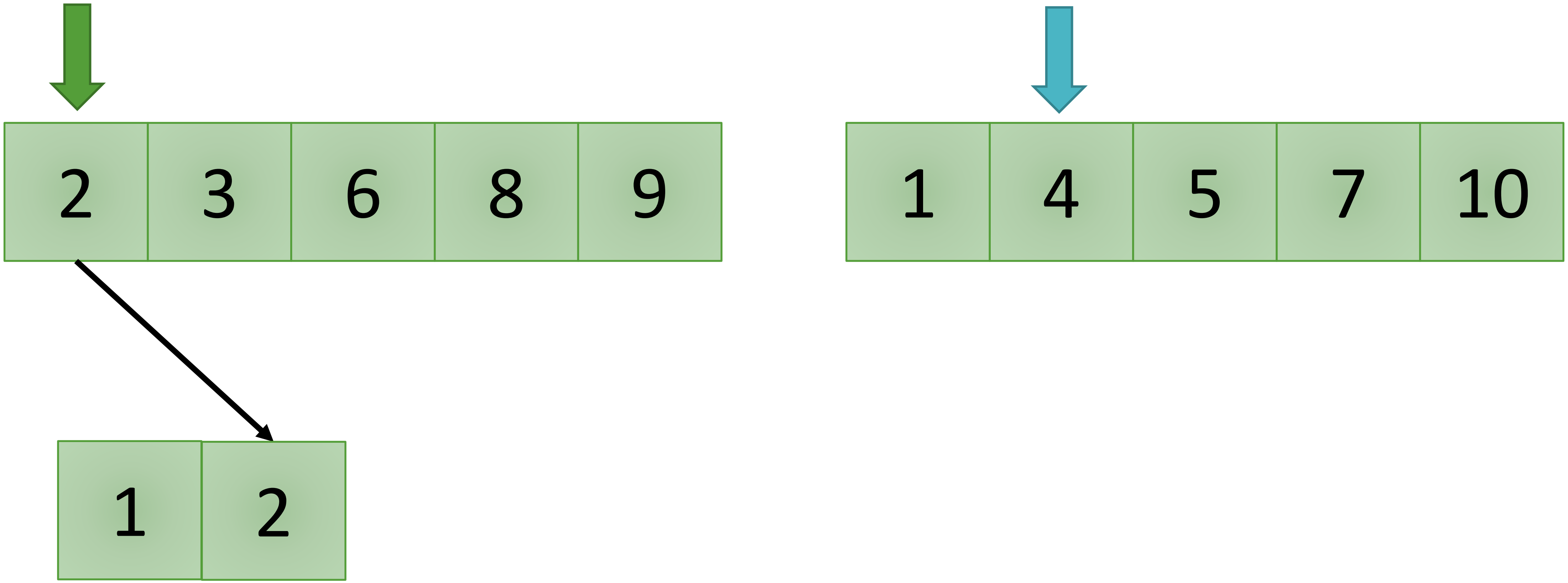


1 Split
9 6 2 8 3 1 7 5 10 4
2 Sort
2 Sort
2 3 6 8 9
1 4 5 7 10
3 Merge
1 2 3 4 5 6 7 8 9 10

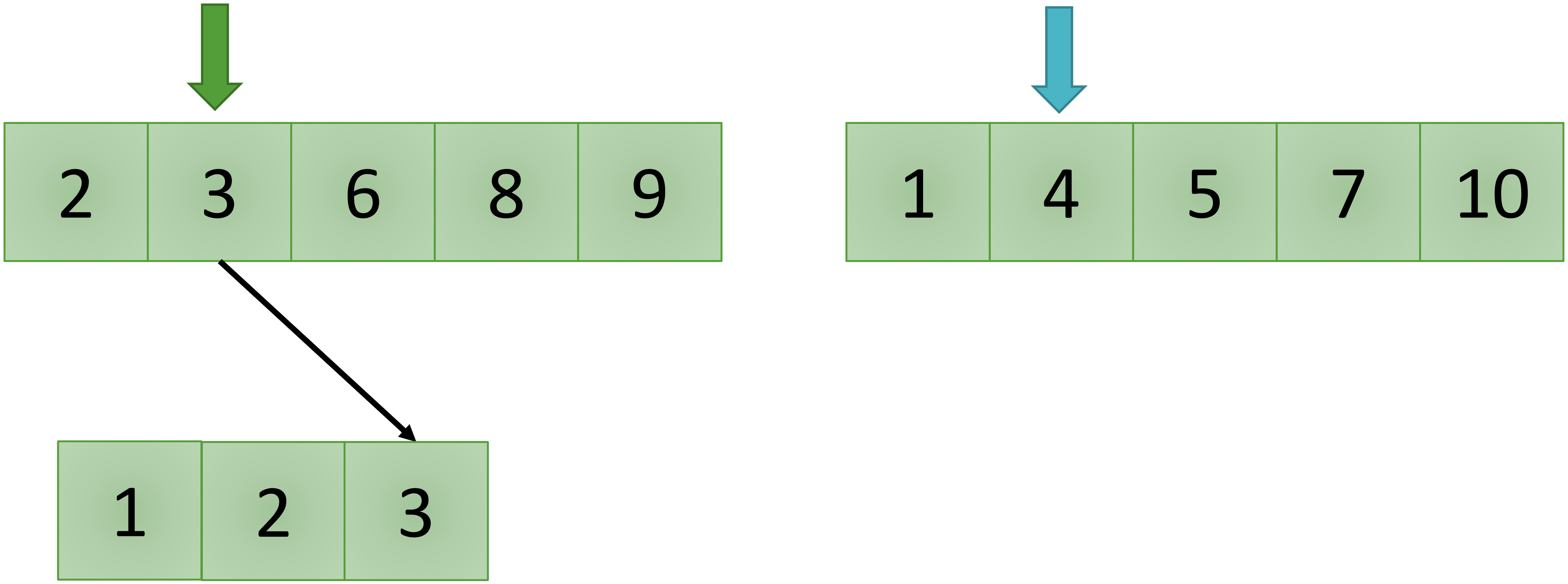
Merge Step



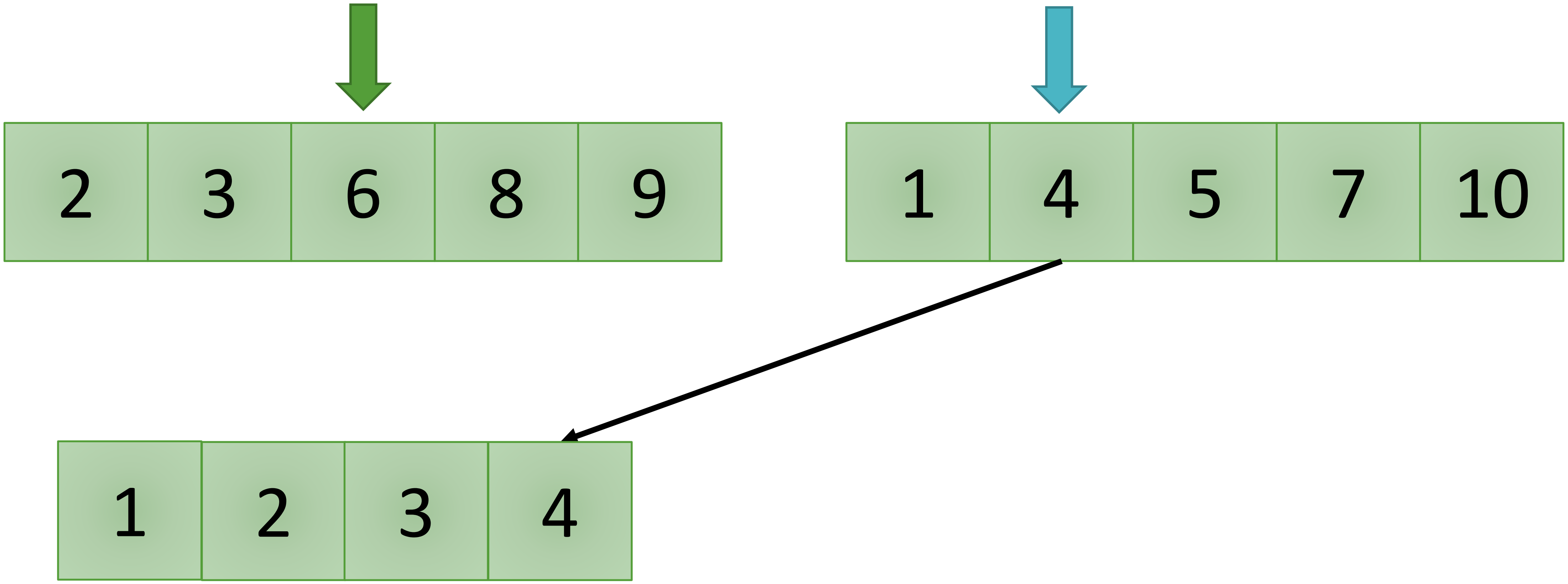
Merge Step



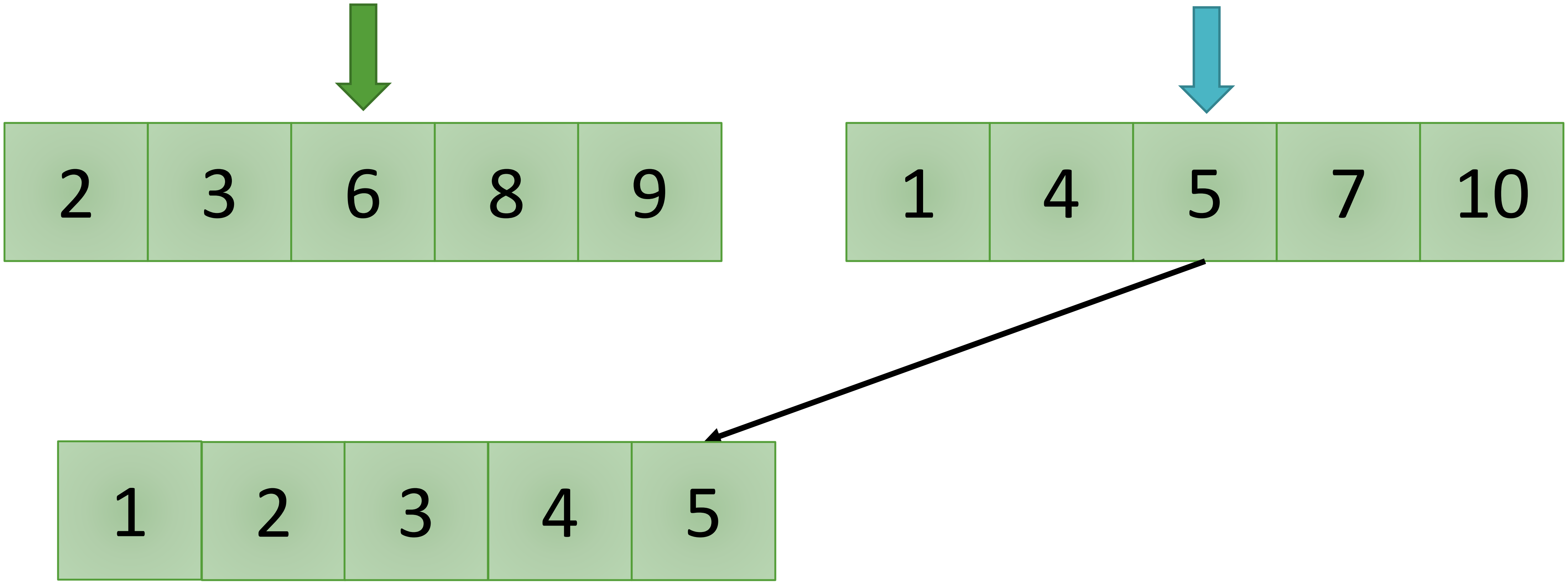
Merge Step



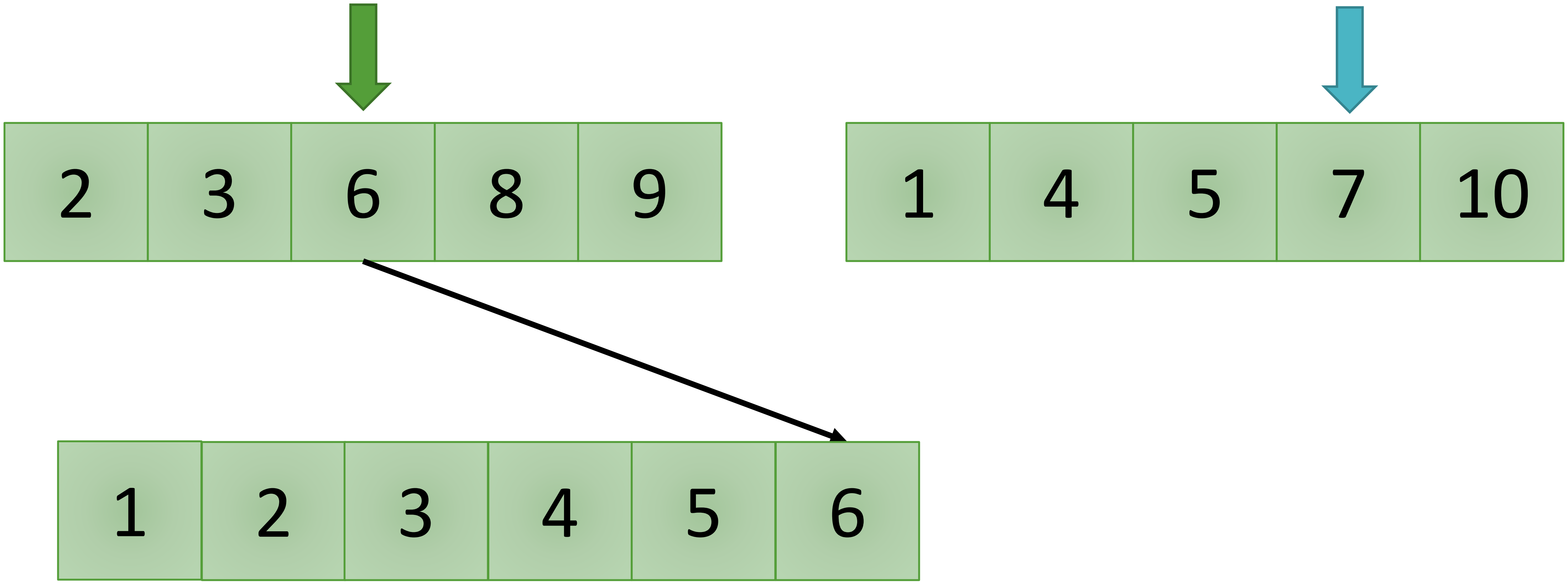
Merge Step



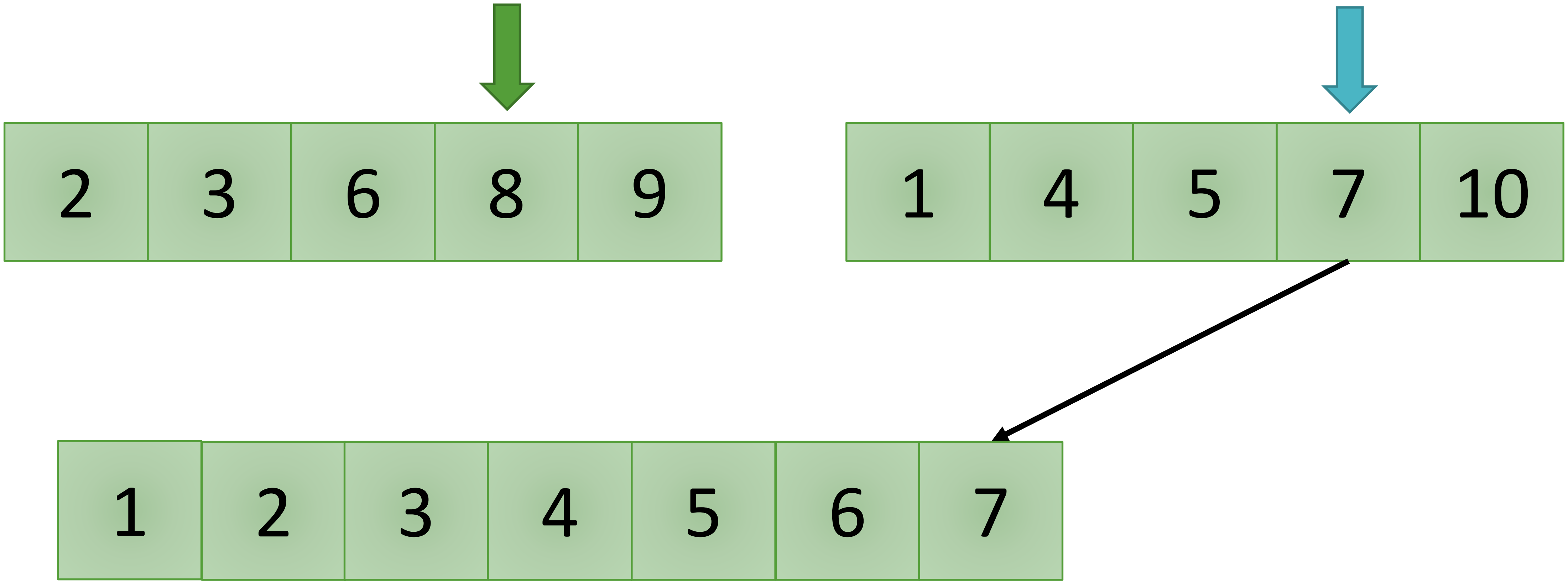
Merge Step



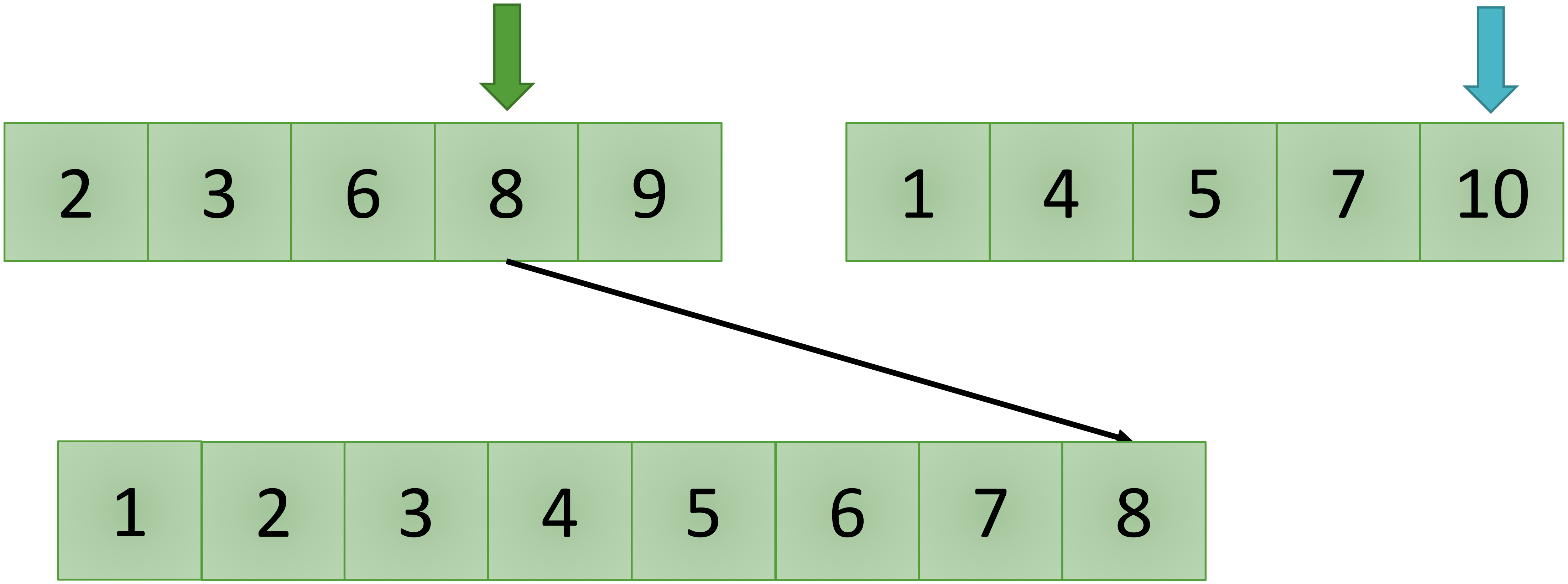
Merge Step



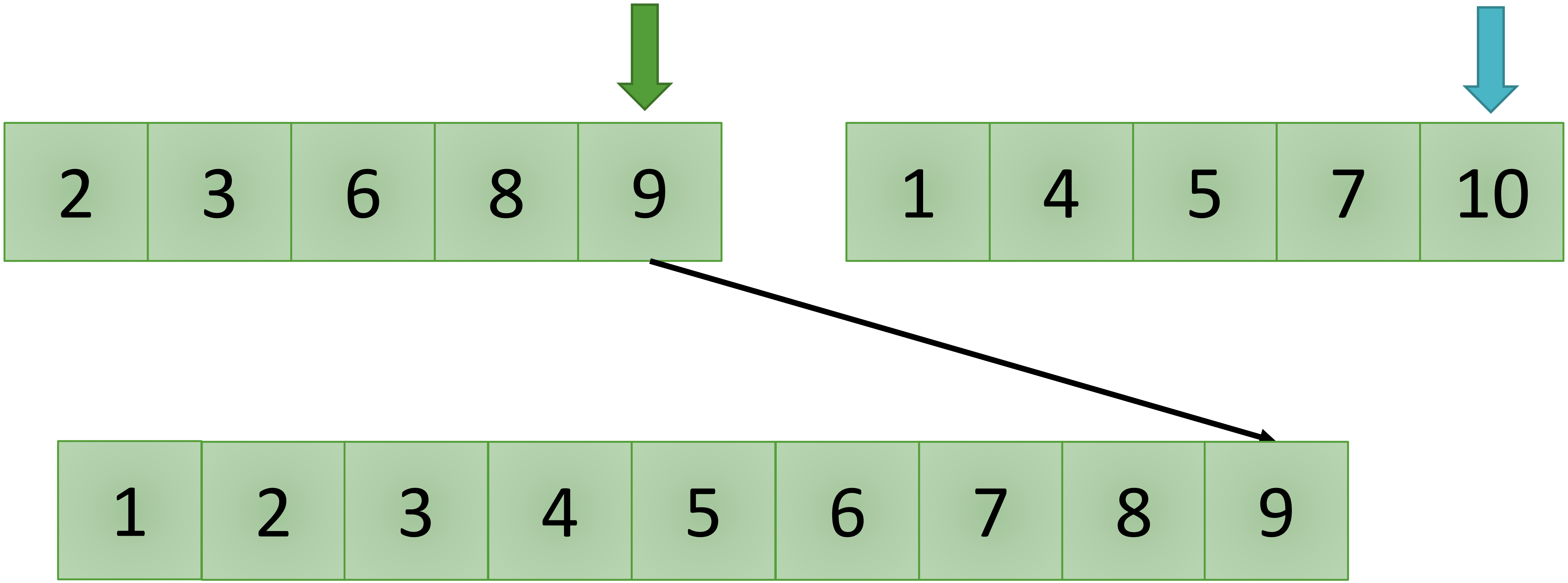
Merge Step



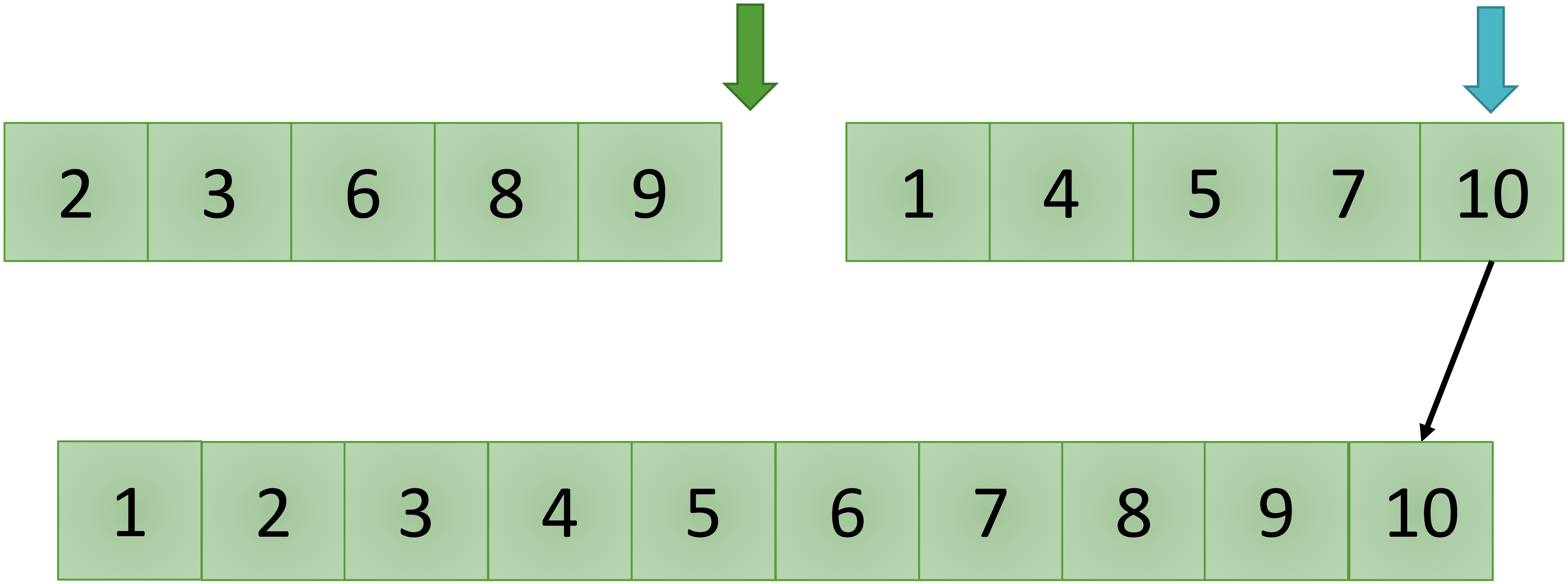
Merge Step



Merge Step



Merge Step



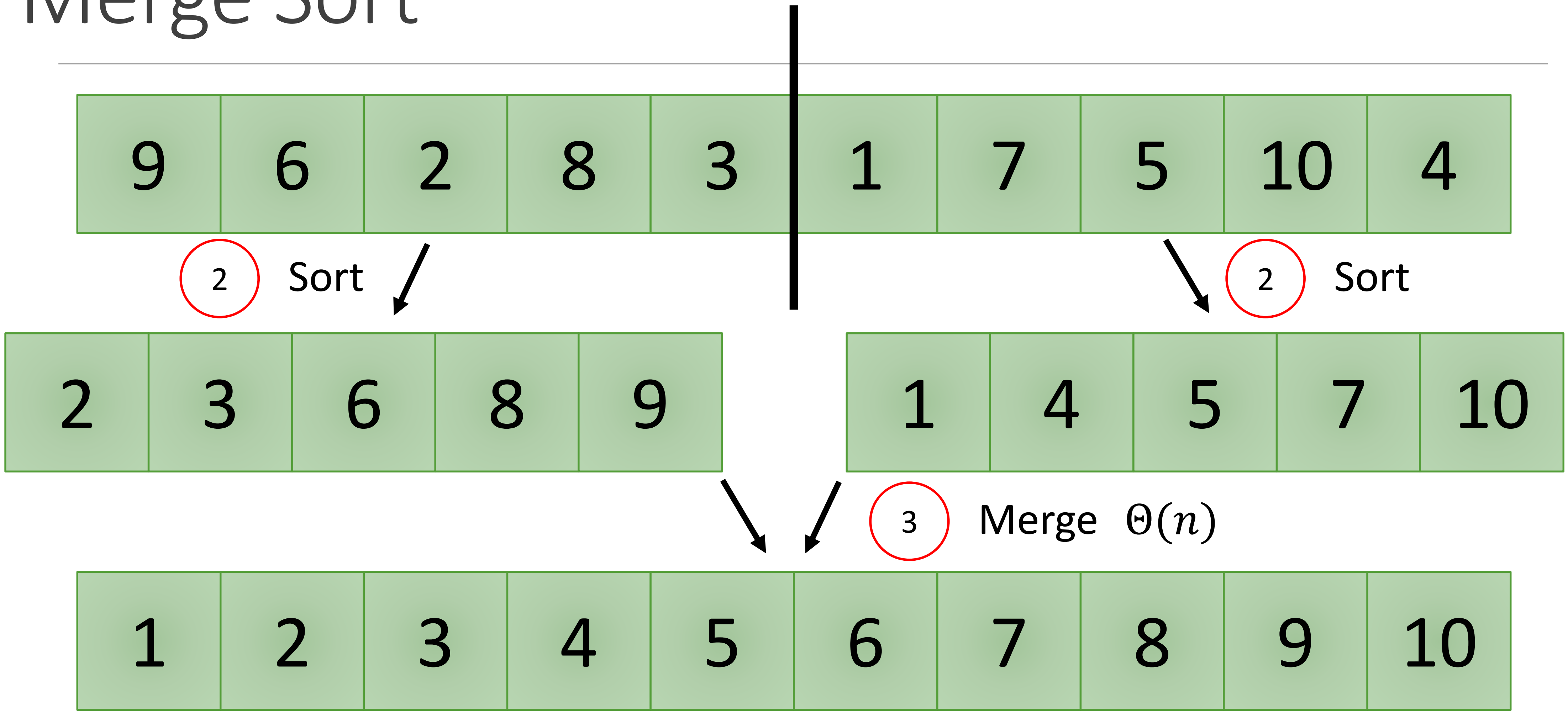
Merge Step Running Time

- We are merging 2 **sorted** lists each of size $n/2$

- In every step:
  - 1 comparison
  - 1 copy
  - 1 pointer increment

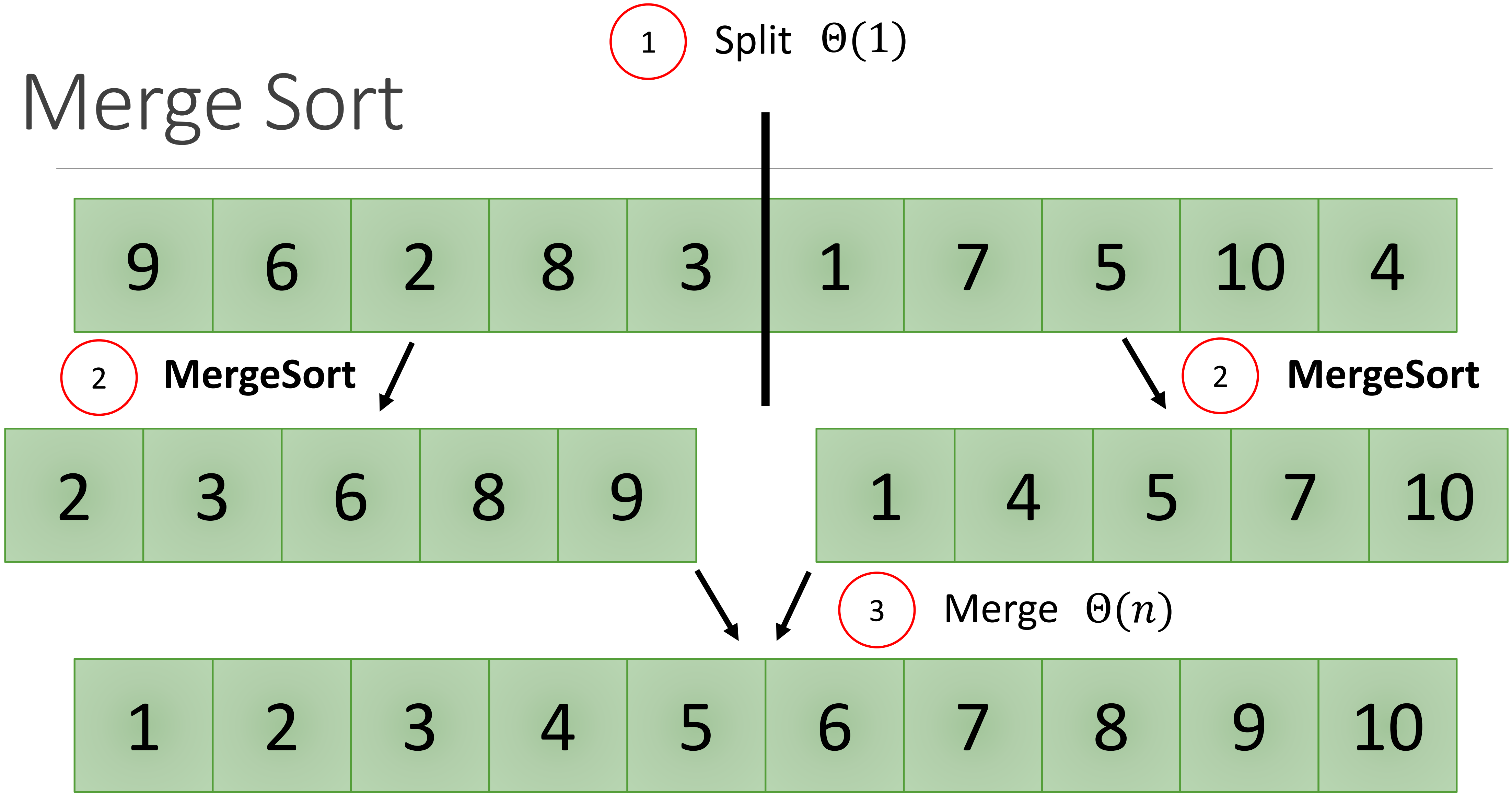
- How many steps do we perform in the **worst** case? n
- Merge Time: $3n = \Theta(n)$

Merge Sort



1 Split $\Theta(1)$
9 6 2 8 3 1 7 5 10 4
2 Sort
2 Sort
2 3 6 8 9
1 4 5 7 10
3 Merge $\Theta(n)$
1 2 3 4 5 6 7 8 9 10

Merge Sort


1 Split $\Theta(1)$
9 6 2 8 3 1 7 5 10 4
2 MergeSort
2 MergeSort
2 3 6 8 9
1 4 5 7 10
3 Merge $\Theta(n)$
1 2 3 4 5 6 7 8 9 10

Merge Sort

	p				q					r
A	9	6	2	8	3	1	7	5	10	4

$\text{MergeSort}(A, p, r)$:

If $p < r$

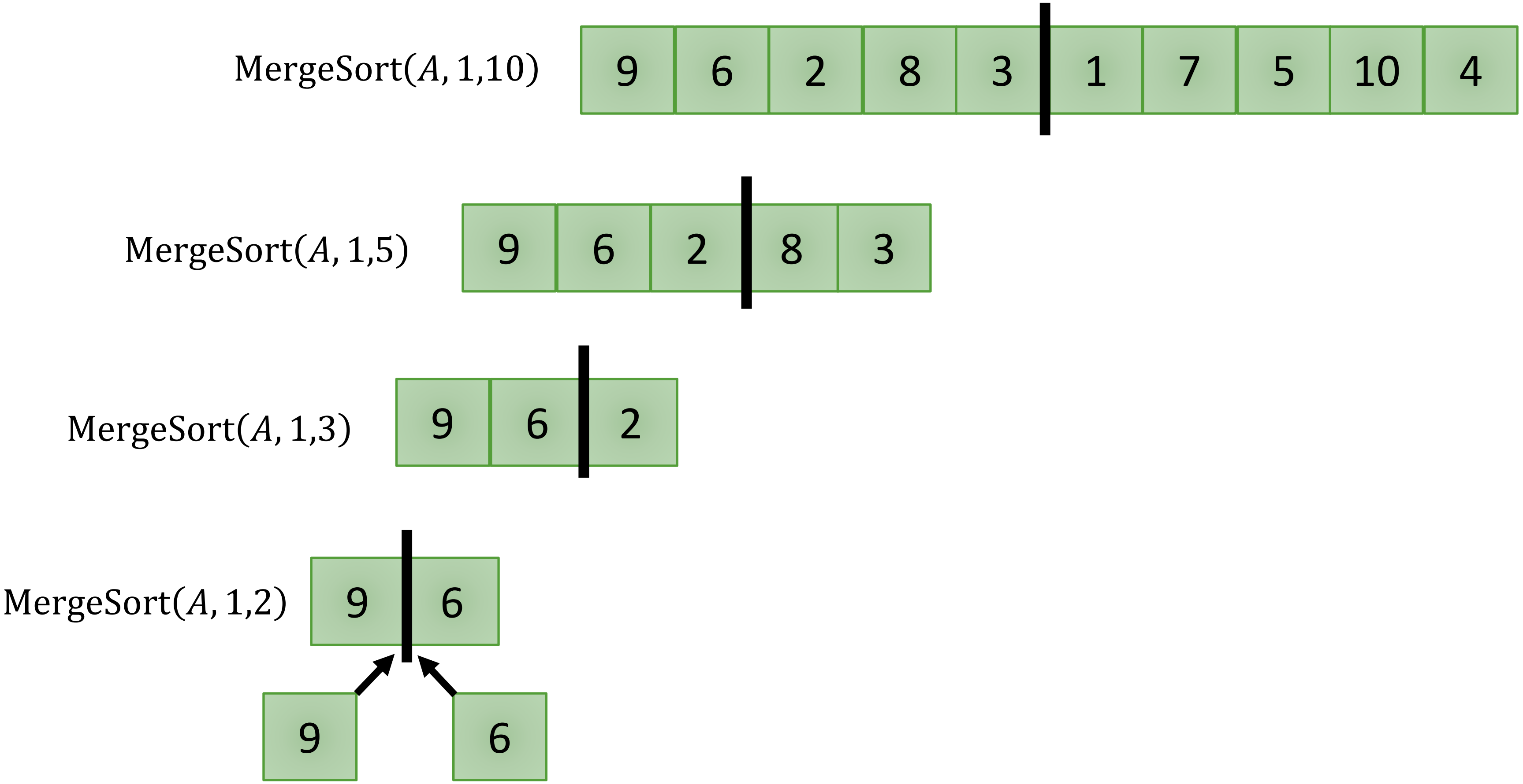
$q = \lfloor (p + r)/2 \rfloor$ // Split in half

$\text{MergeSort}(A, p, q)$ // Recursively merge-sort left half

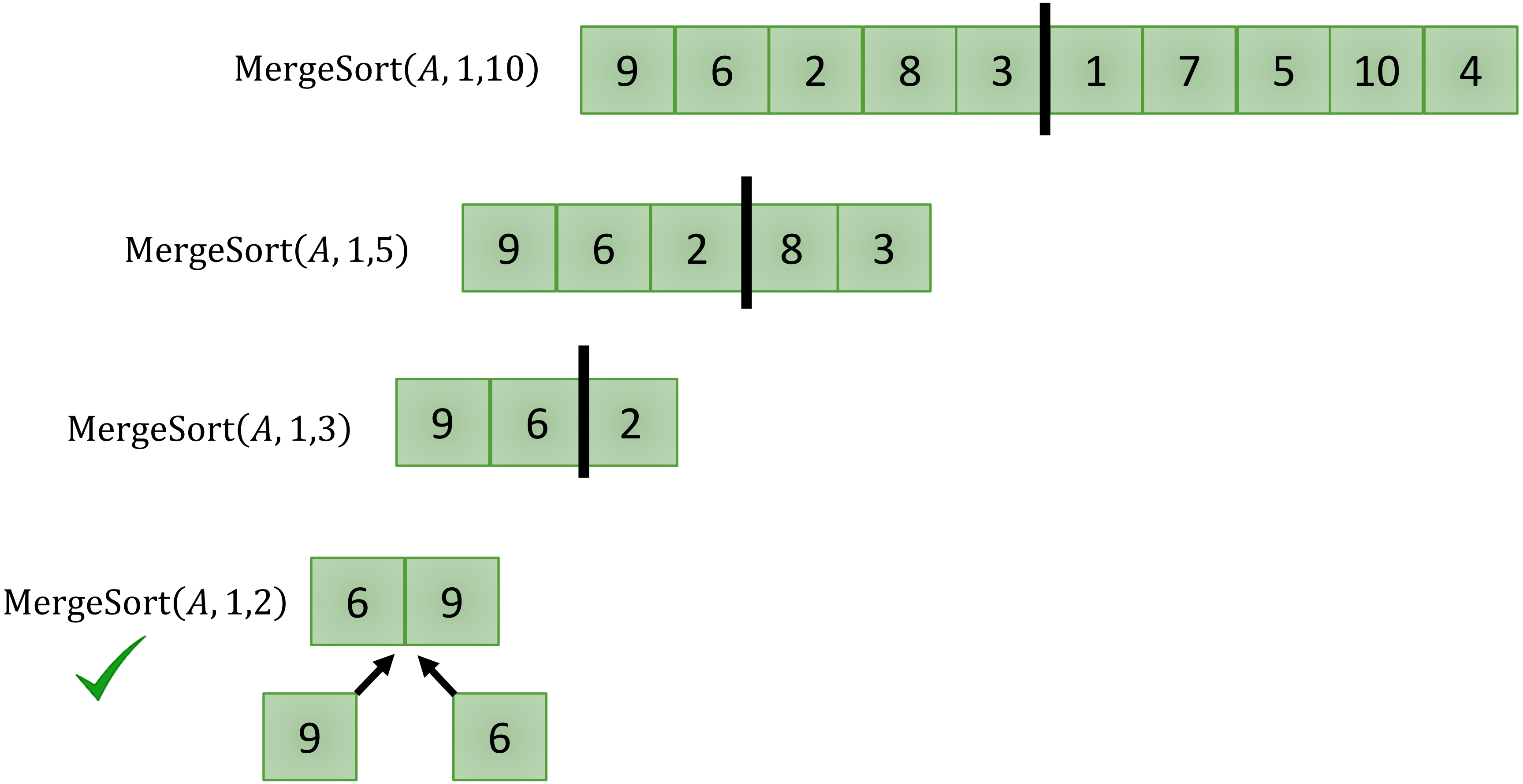
$\text{MergeSort}(A, q + 1, r)$ // Recursively merge-sort right half

$\text{Merge}(A, p, r)$ // Merge the two sorted lists

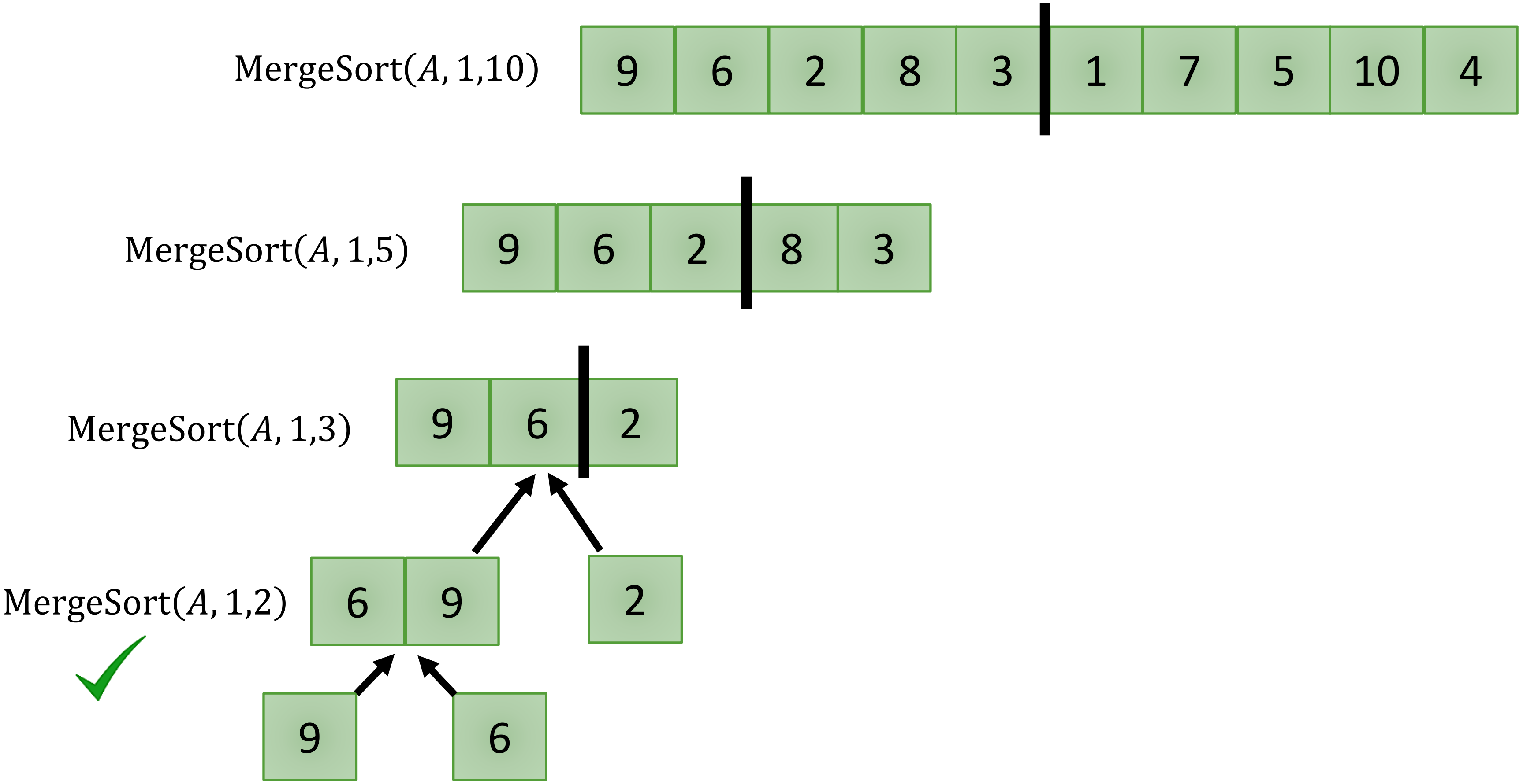
Merge Sort: Recursive Sorting



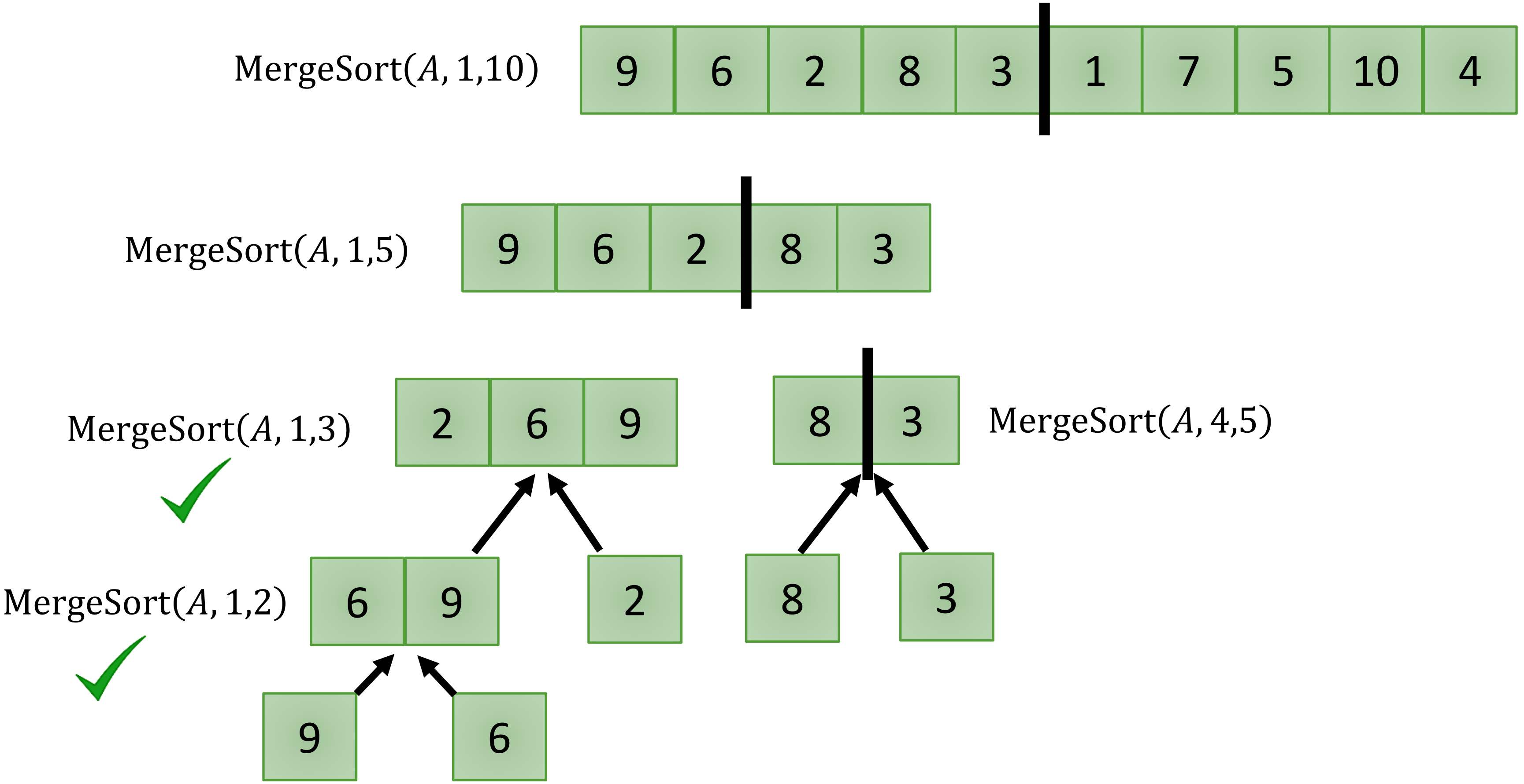
Merge Sort: Recursive Sorting



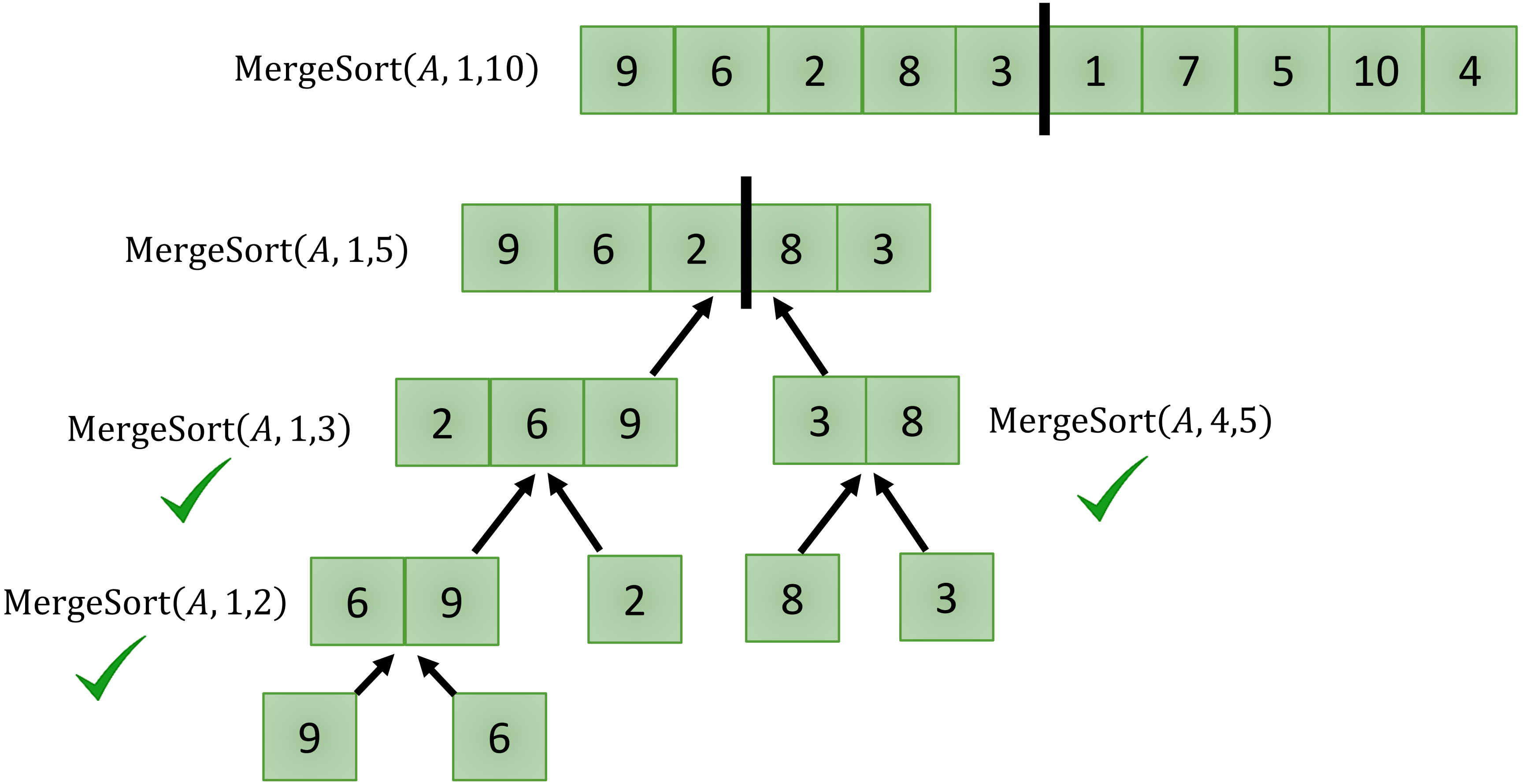
Merge Sort: Recursive Sorting



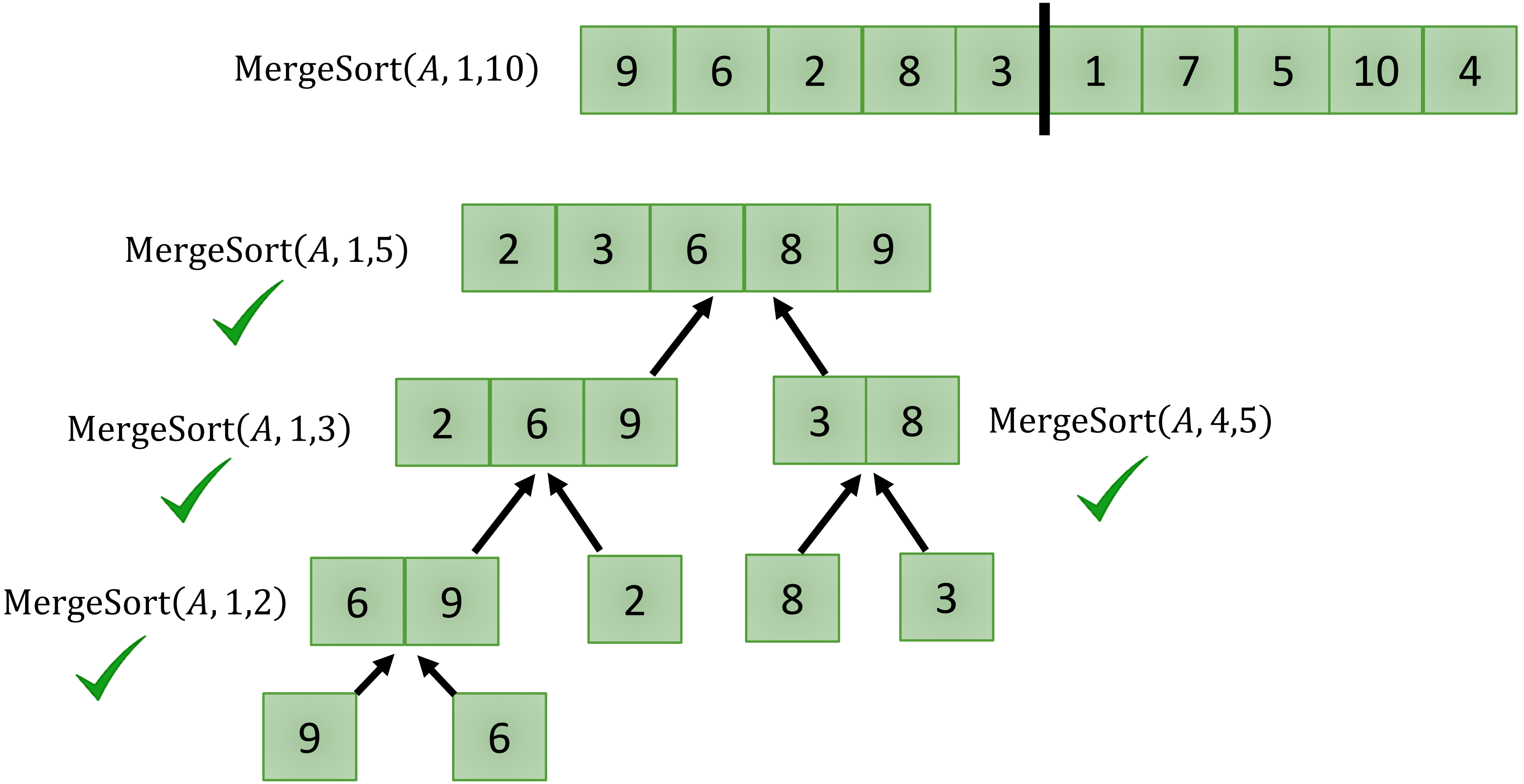
Merge Sort: Recursive Sorting



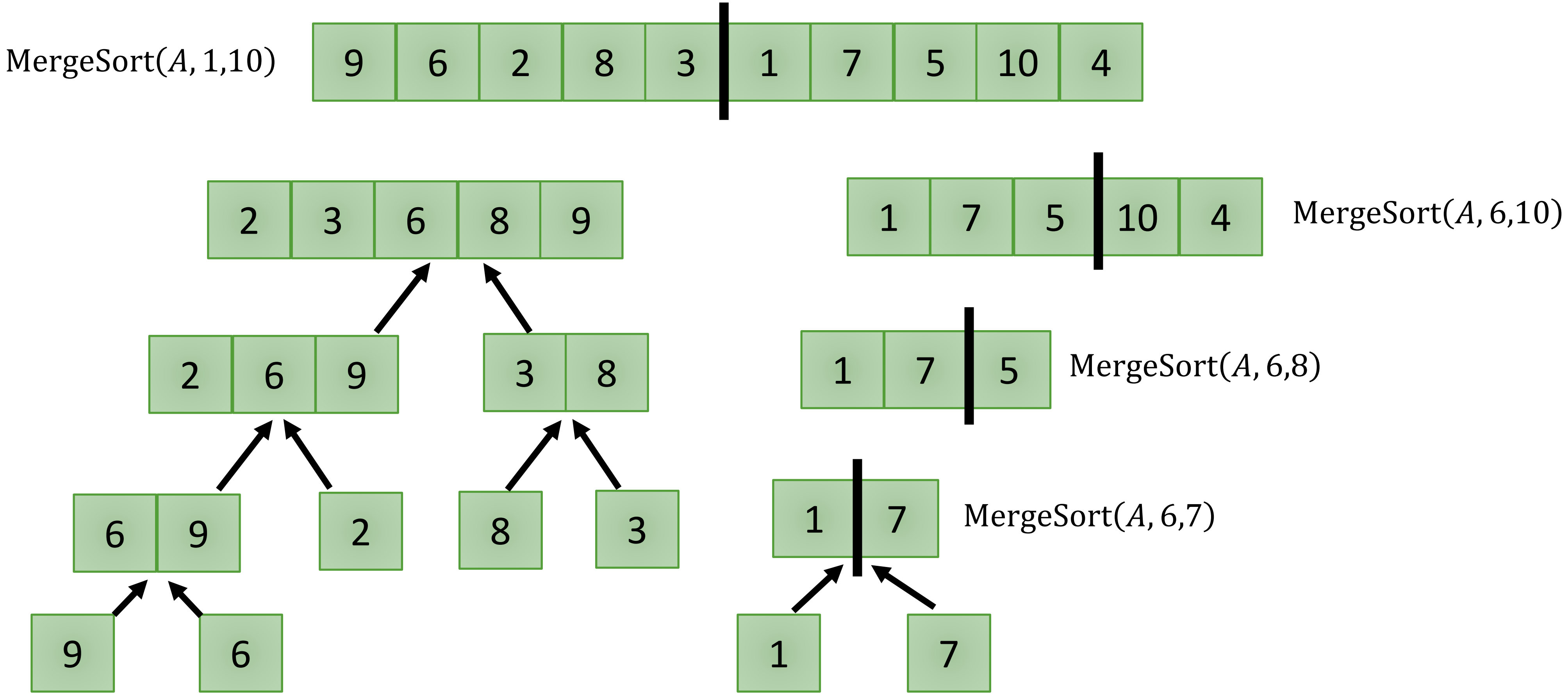
Merge Sort: Recursive Sorting



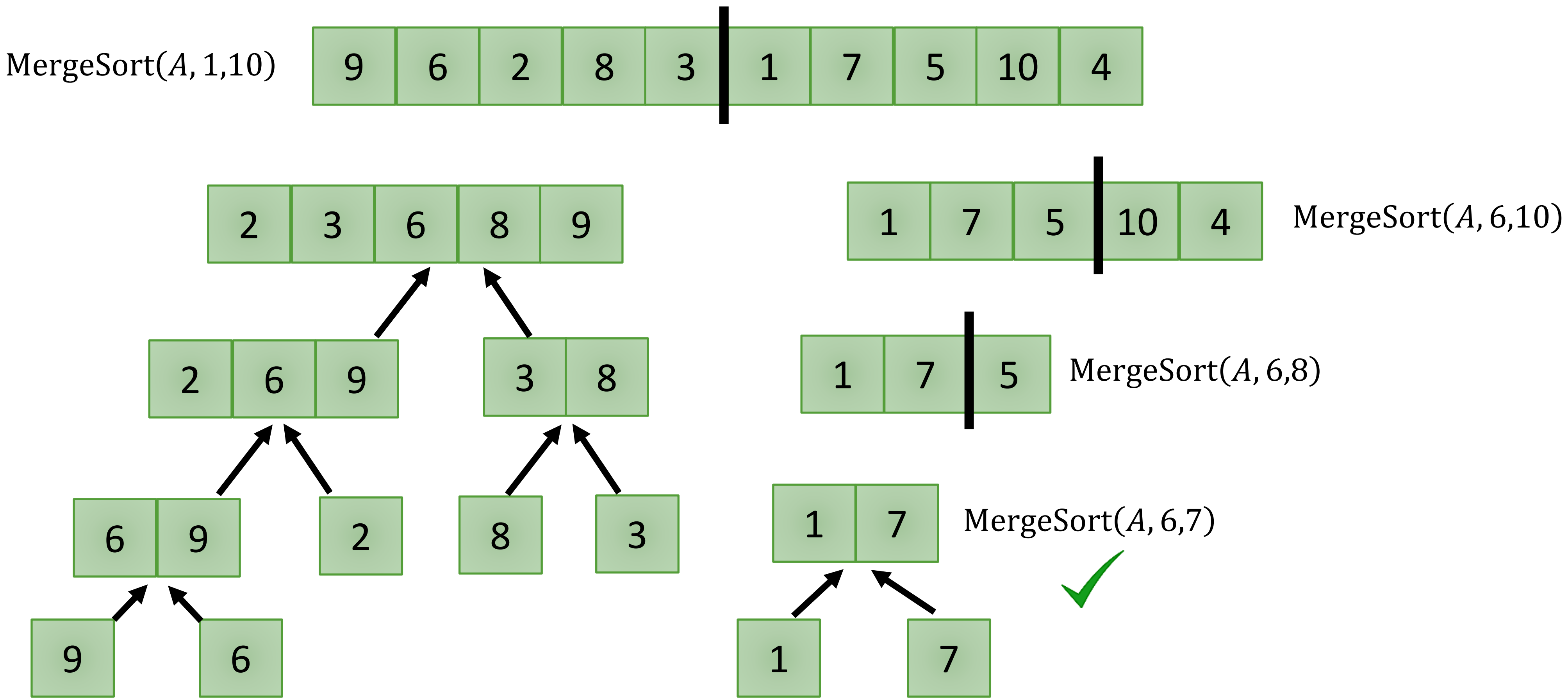
Merge Sort: Recursive Sorting



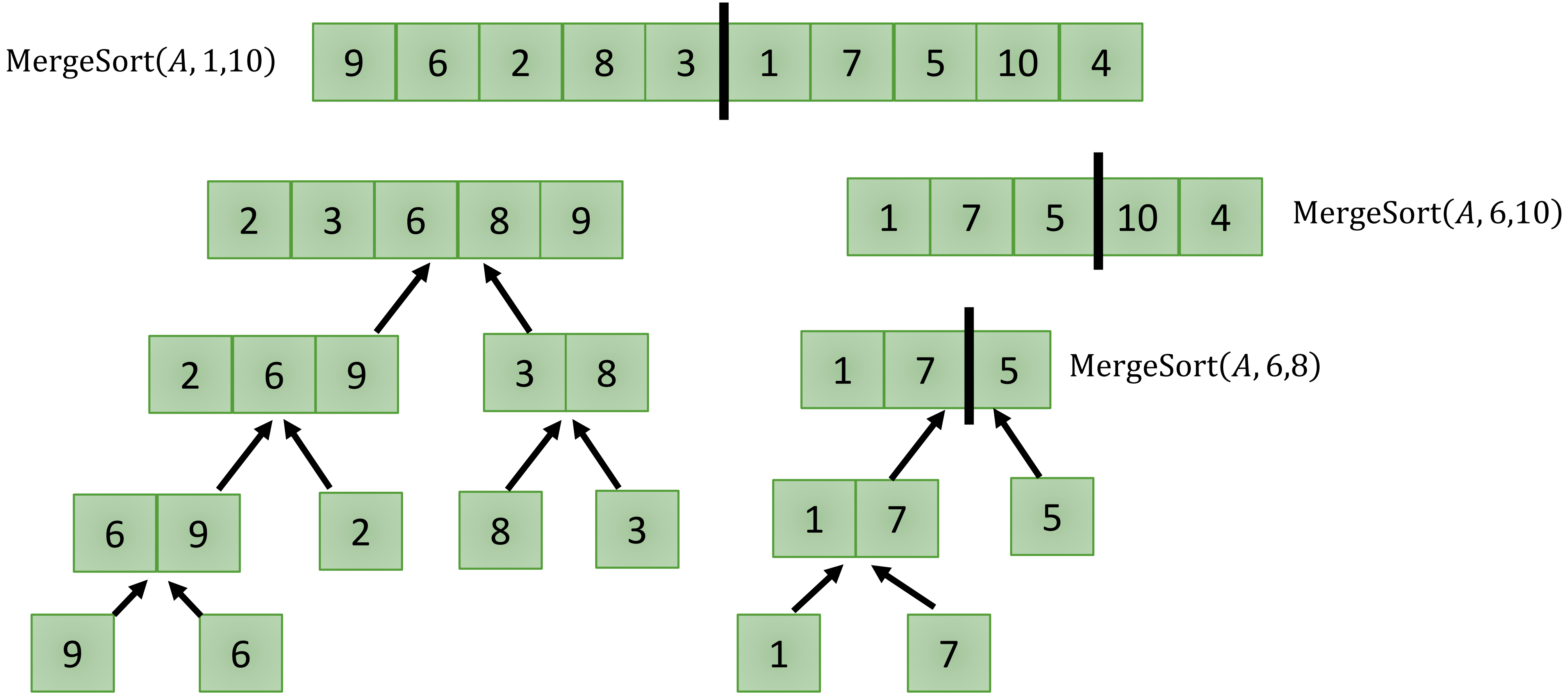
Merge Sort: Recursive Sorting



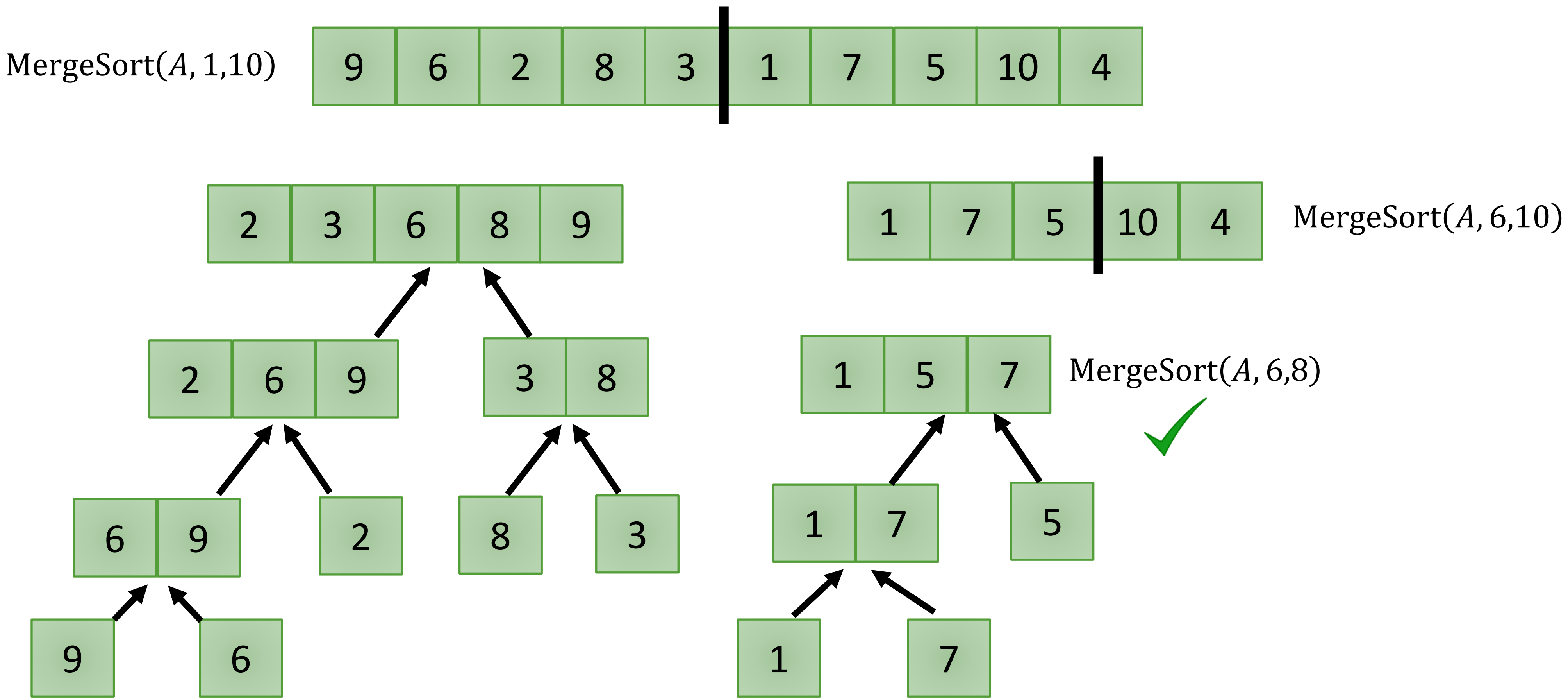
Merge Sort: Recursive Sorting



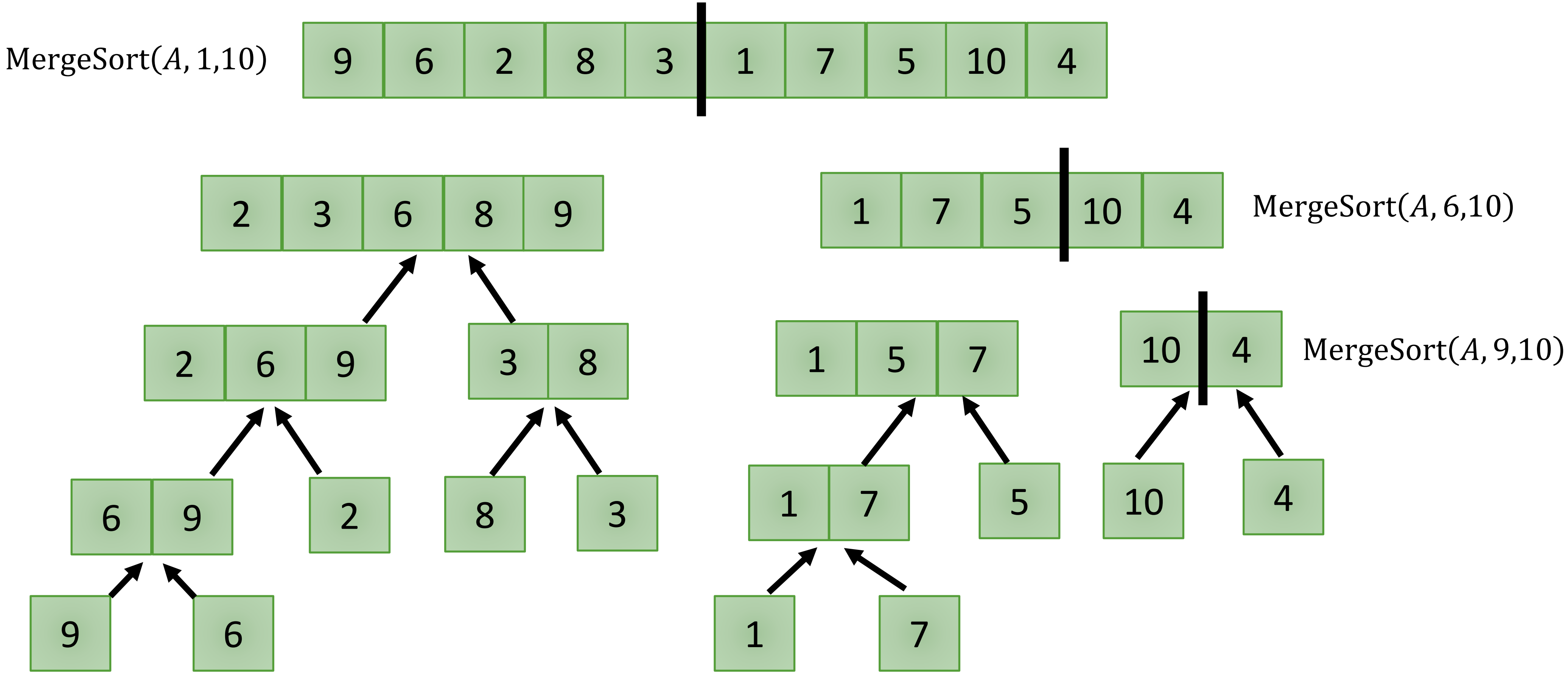
Merge Sort: Recursive Sorting



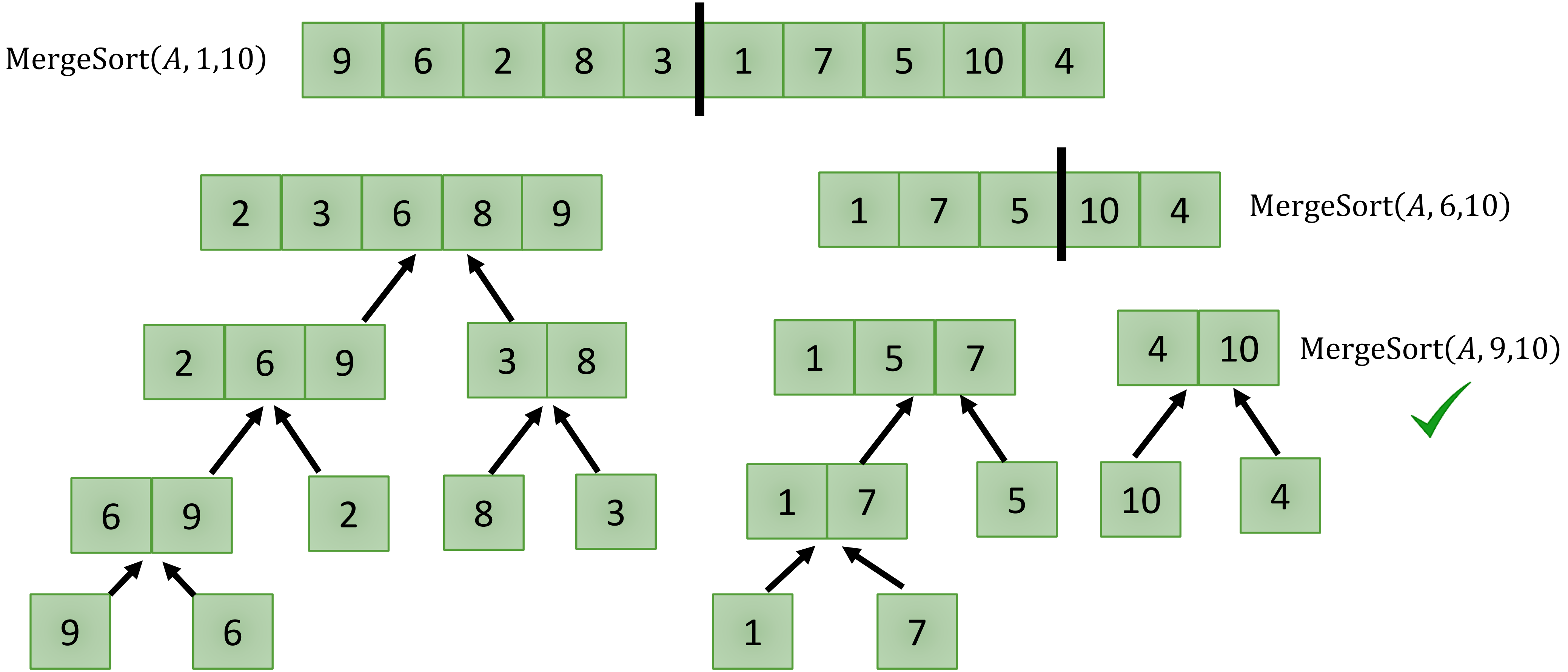
Merge Sort: Recursive Sorting



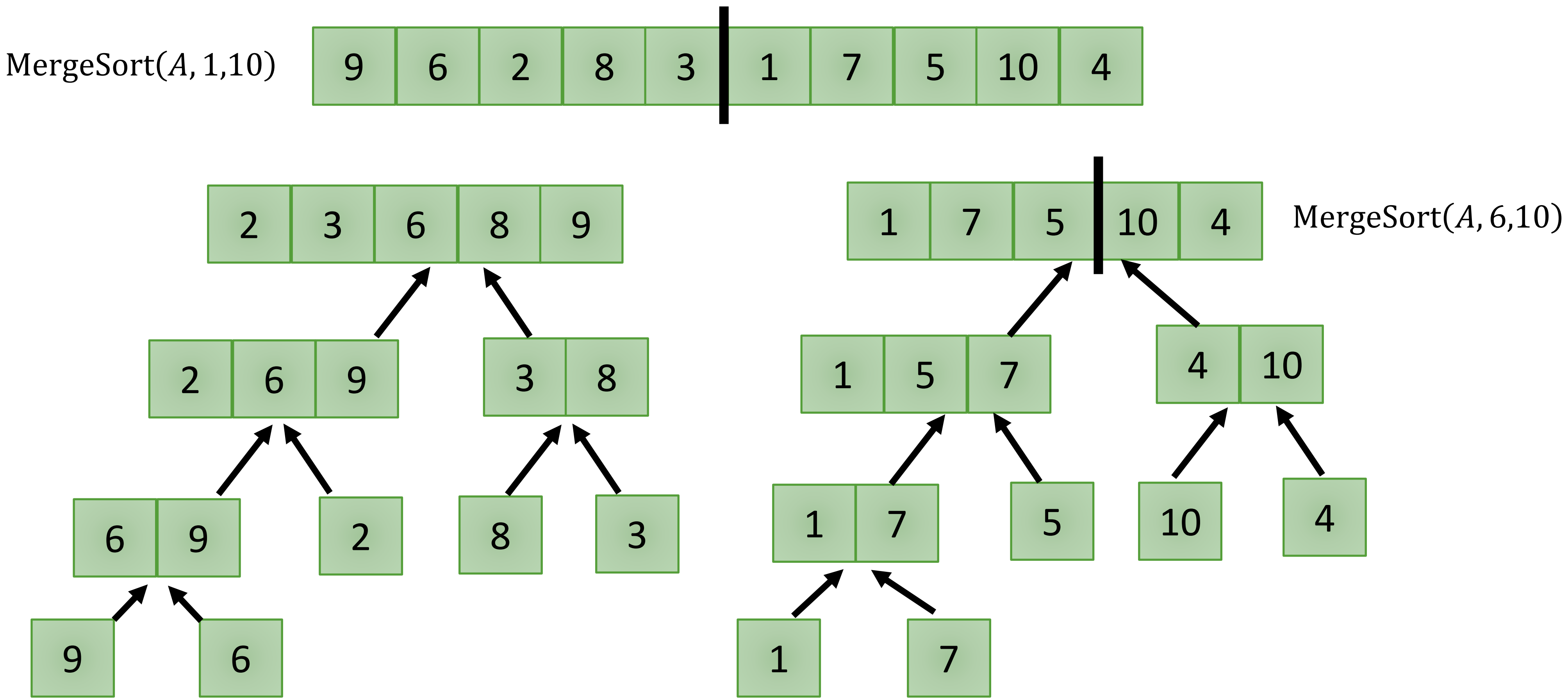
Merge Sort: Recursive Sorting



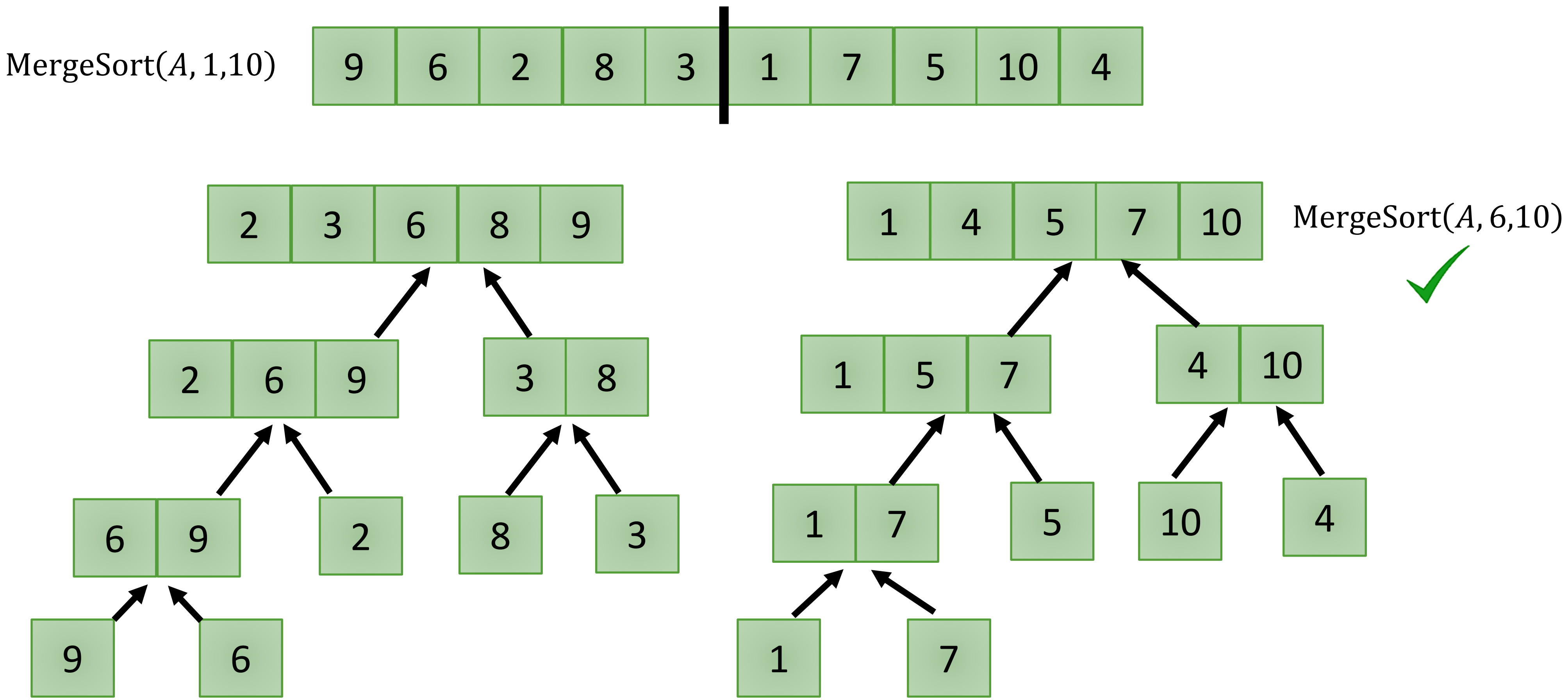
Merge Sort: Recursive Sorting



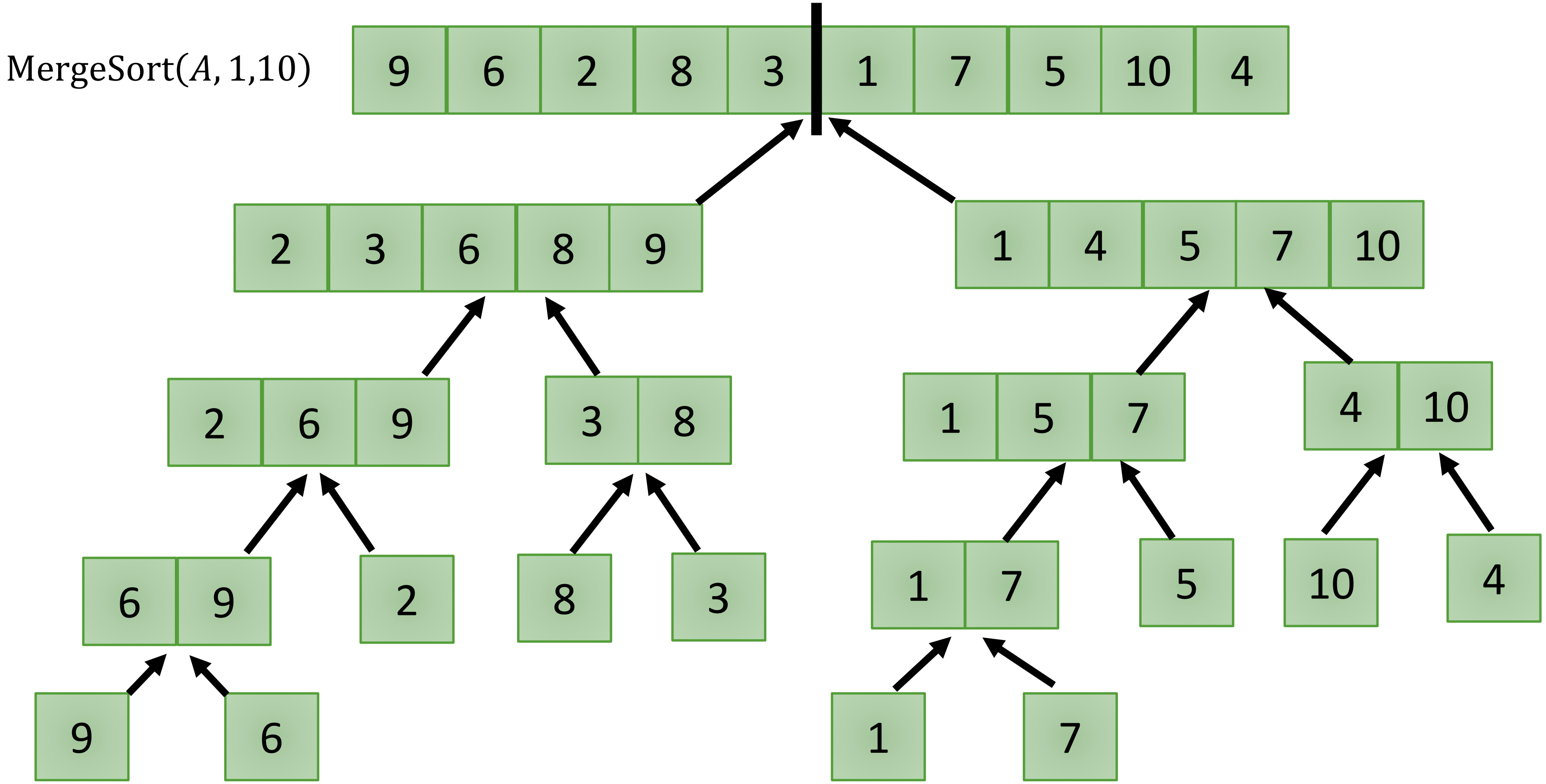
Merge Sort: Recursive Sorting



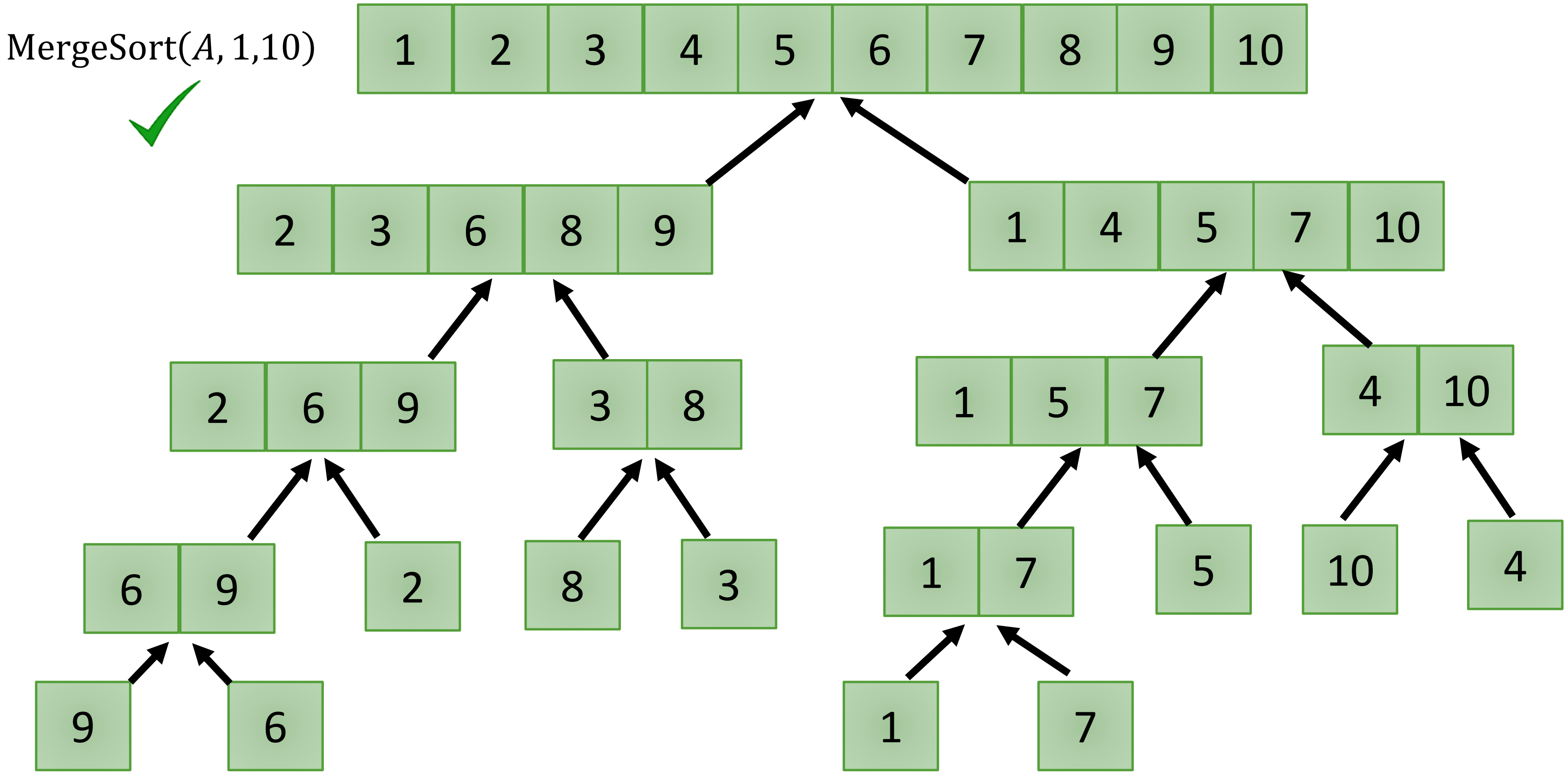
Merge Sort: Recursive Sorting



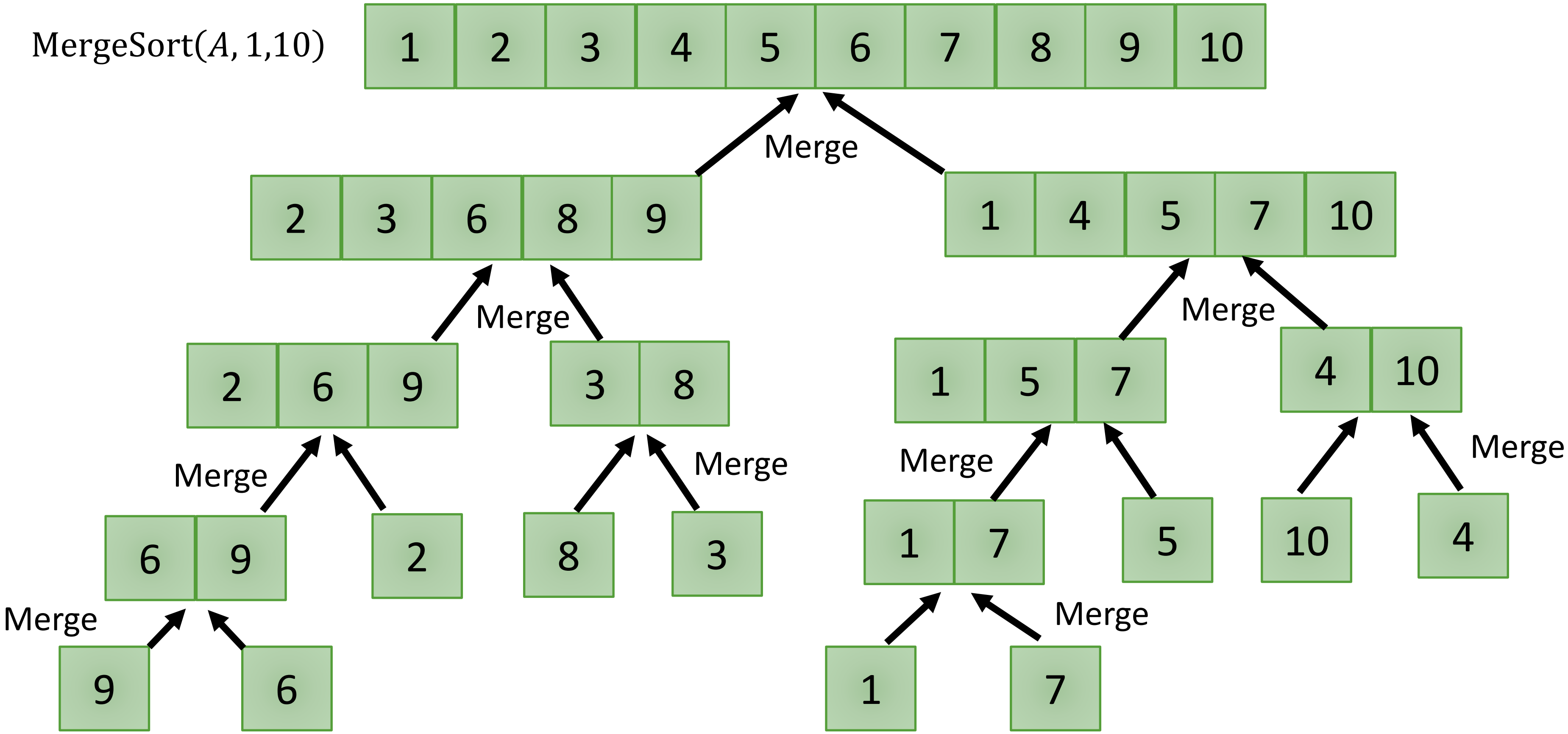
Merge Sort: Recursive Sorting



Merge Sort: Recursive Sorting



Merge Sort: Recursive Sorting



Merge Sort: Running-time Analysis

MergeSort(A, p, r):	$T(n)$
If $p < r$	
$\quad q = \lfloor (p+r)/2 \rfloor$	$\Theta(1)$
$\quad$ MergeSort(A, p, q)	$T(n/2)$
$\quad$ MergeSort$(A, q+1, r)$	$T(n/2)$
$\quad$ Merge(A, p, r)	$\Theta(n)$

$$\boxed{T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ 2T(n/2) + \Theta(n) & \text{if } n > 1 \end{cases}}$$

Recurrence

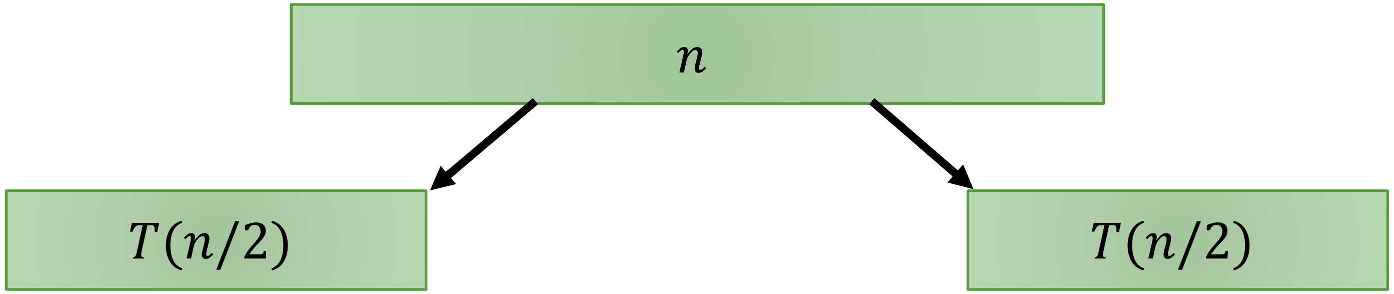
Recurrence Tree

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ 2T(n/2) + \Theta(n) & \text{if } n > 1 \end{cases}$$

$T(n)$

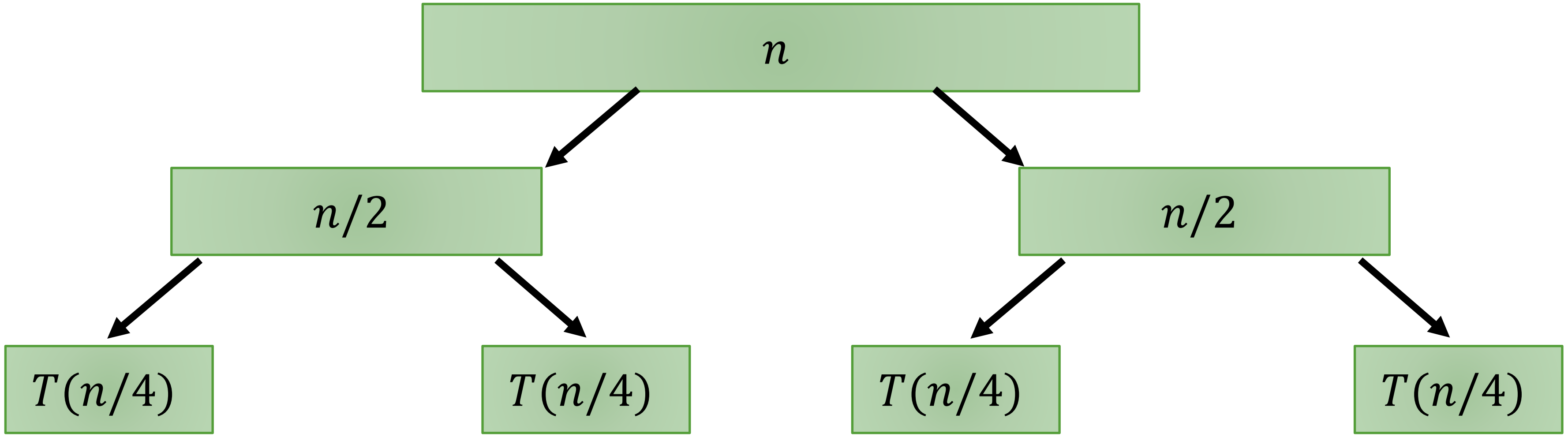
Recurrence Tree

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ 2T(n/2) + \Theta(n) & \text{if } n > 1 \end{cases}$$



Recurrence Tree

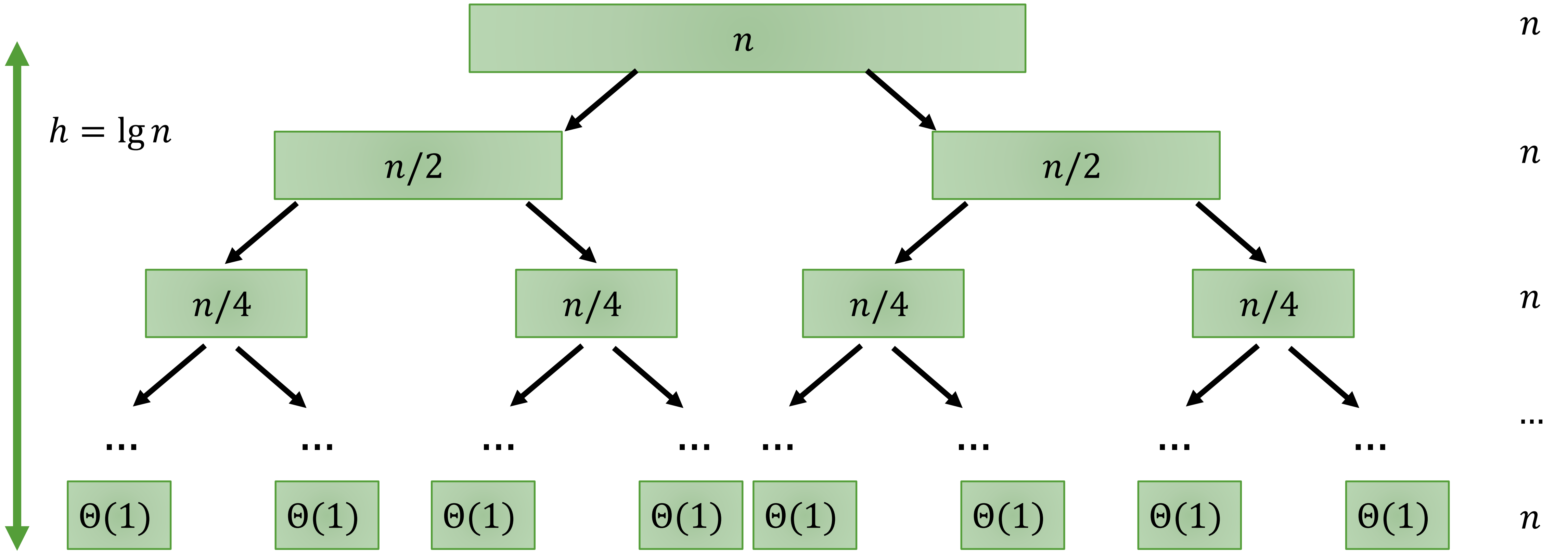
$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ 2T(n/2) + \Theta(n) & \text{if } n > 1 \end{cases}$$



Recurrence Tree

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ 2T(n/2) + \Theta(n) & \text{if } n > 1 \end{cases}$$

$T(n) = \Theta(n \lg n)$



Divide-and-Conquer

- Merge sort is based on the **divide-and-conquer** design paradigm

- **Divide** the problem into a number of subproblems
- **Conquer** the subproblems by solving them recursively
- **Combine** the solutions to the subproblems into a solution for the original problem

- Analyzing divide-and-conquer based algorithms involves solving **recurrences**.

Insertion Sort vs. Merge Sort

- Insertion sort **worst-case** running time = $\Theta(n^2)$
- Merge sort **worst-case** running time = $\Theta(n \lg n)$
- We say merge sort is **asymptotically faster** than insertion sort.

- In practice, merge sort beats insertion sort for values of roughly $n \geq 30$