Merge Sort

Shusen Wang

Merge Two Sorted Arrays

Merge two sorted arrays

$a =$

0	1	4	8	9

$b =$

2	3	5	6	7

$c =$

Merge two sorted arrays

$a =$ | 0 | 1 | 4 | 8 | 9 |

$b =$ | 2 | 3 | 5 | 6 | 7 |

Compare 0 and 2; choose the smaller.

$c =$ | | | | | | | | | | |

Merge two sorted arrays

$\boldsymbol{a} =$

0	1	4	8	9

$\boldsymbol{b} =$

2	3	5	6	7

Compare 0 and 2; choose the smaller.

$\boldsymbol{c} =$

Merge two sorted arrays

$a =$

0	1	4	8	9

$b =$

2	3	5	6	7

Compare 1 and 2; choose the smaller.

$c =$

0									

Merge two sorted arrays

$a =$

0	1	4	8	9

$b =$

2	3	5	6	7

Compare 1 and 2; choose the smaller.

$c =$

0									

Merge two sorted arrays

$a =$

0	1	4	8	9

$b =$

2	3	5	6	7

Compare 4 and 2; choose the smaller.

$c =$

0	1								

Merge two sorted arrays

$a =$

0	1	4	8	9

$b =$

2	3	5	6	7

Compare 4 and 2; choose the smaller.

$c =$

0	1								

Merge two sorted arrays

$a =$

0	1	4	8	9

$b =$

2	3	5	6	7

Compare 4 and 3; choose the smaller.

$c =$

0	1	2							

Merge two sorted arrays

$a =$ | 0 | 1 | 4 | 8 | 9 |

$b =$ | 2 | 3 | 5 | 6 | 7 |

Compare 4 and 3; choose the smaller.

$c =$ | 0 | 1 | 2 | | | | | | | |

Merge two sorted arrays

$a =$

0	1	4	8	9

$b =$

2	3	5	6	7

Compare 4 and 5; choose the smaller.

$c =$

0	1	2	3						

Merge two sorted arrays

$\boldsymbol{a} =$

0	1	4	8	9

$\boldsymbol{b} =$

2	3	5	6	7

Compare 4 and 5; choose the smaller.

$\boldsymbol{c} =$

0	1	2	3						

Merge two sorted arrays

$a =$

0	1	4	8	9

$b =$

2	3	5	6	7

Compare 8 and 5; choose the smaller.

$c =$

0	1	2	3	4					

Merge two sorted arrays

$a =$

0	1	4	8	9

$b =$

2	3	5	6	7

Compare 8 and 5; choose the smaller.

$c =$

0	1	2	3	4					

Merge two sorted arrays

$a =$

0	1	4	8	9

$b =$

2	3	5	6	7

Compare 8 and 6; choose the smaller.

$c =$

0	1	2	3	4	5				

Merge two sorted arrays

$a =$

0	1	4	8	9

$b =$

2	3	5	6	7

Compare 8 and 6; choose the smaller.

$c =$

0	1	2	3	4	5				

Merge two sorted arrays

$a =$

0	1	4	8	9

$b =$

2	3	5	6	7

Compare 8 and 7; choose the smaller.

$c =$

0	1	2	3	4	5	6			

Merge two sorted arrays

$a =$

0	1	4	8	9

$b =$

2	3	5	6	7

Compare 8 and 7; choose the smaller.

$c =$

0	1	2	3	4	5	6			

Merge two sorted arrays

$a =$

0	1	4	8	9

$b =$

2	3	5	6	7

This pointer is out of range.

$c =$

0	1	2	3	4	5	6	7		

Merge two sorted arrays

$a =$

0	1	4	8	9

$b =$

2	3	5	6	7

Copy the rest of a to c.

$c =$

0	1	2	3	4	5	6	7		

Merge two sorted arrays

$a =$

0	1	4	8	9

$b =$

2	3	5	6	7

Copy the rest of a to c.

$c =$

0	1	2	3	4	5	6	7	8	9

Time Complexity: $O(n)$

$a =$

0	1	4	8	9

$b =$

2	3	5	6	7

$c =$

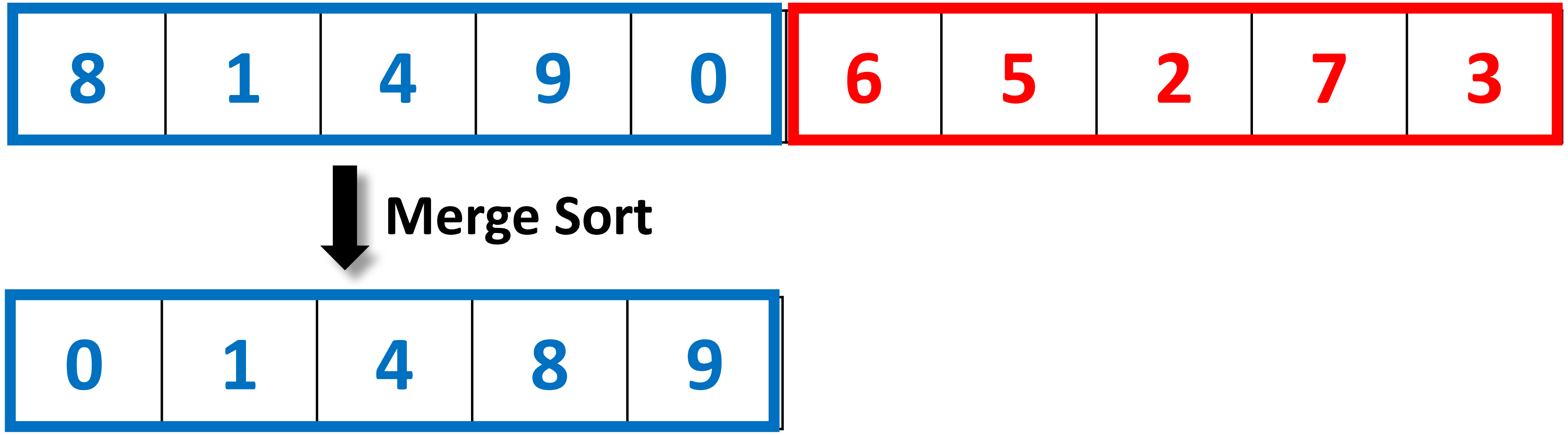
0	1	2	3	4	5	6	7	8	9

Merge Sort

Merge Sort via Recursion

8	1	4	9	0	6	5	2	7	3

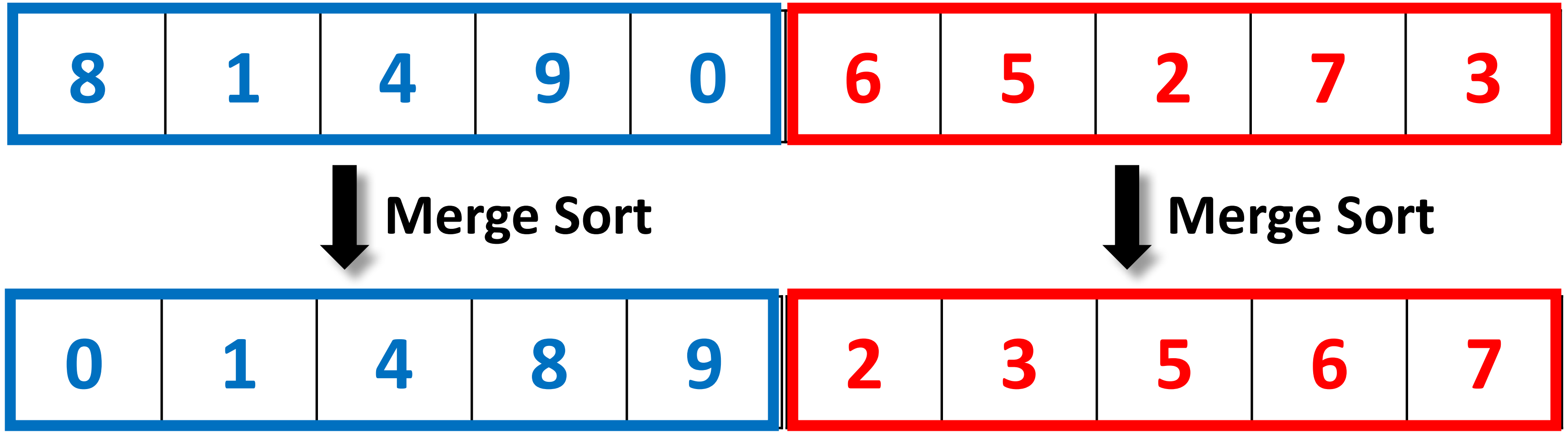
Merge Sort via Recursion



Procedure:

1. Apply merge sort to the left subarray.

Merge Sort via Recursion



Procedure:

1. Apply merge sort to the left subarray.
2. Apply merge sort to the right subarray.

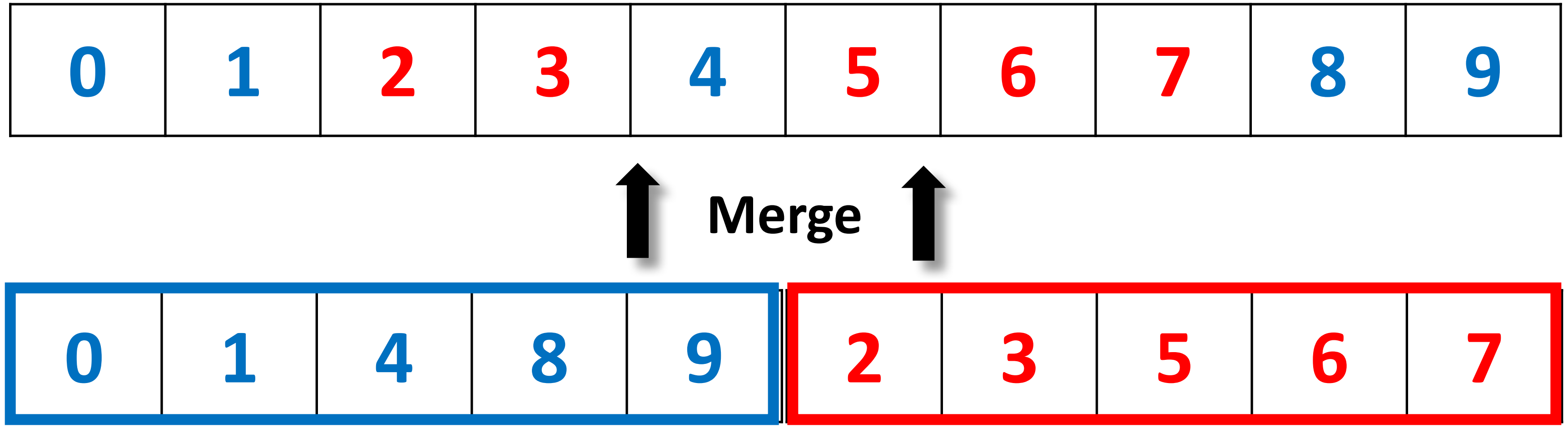
Merge Sort via Recursion

0	1	4	8	9	2	3	5	6	7

Procedure:

1. Apply merge sort to the left subarray.
2. Apply merge sort to the right subarray.

Merge Sort via Recursion



Procedure:

1. Apply merge sort to the left subarray.
2. Apply merge sort to the right subarray.
3. Merge the two halves.

Implementation

```
void mergesort(int arr[], int left, int right) {
    if (left >= right) return;
    // size of the subarray is at least 2:
    int mid = (left + right) / 2;
    mergesort(arr, left, mid);
    mergesort(arr, mid+1, right);
    merge(arr, left, mid, right);
}
```

Implementation

```
void mergesort(int arr[], int left, int right) {
    if (left >= right) return;
    // size of the subarray is at least 2:
    int mid = (left + right) / 2;
    mergesort(arr, left, mid);
    mergesort(arr, mid+1, right);
    merge(arr, left, mid, right);
}
```

Implementation

```
void mergesort(int arr[], int left, int right) {
    if (left >= right) return;
    // size of the subarray is at least 2:
    int mid = (left + right) / 2;
    mergesort(arr, left, mid);
    mergesort(arr, mid+1, right);
    merge(arr, left, mid, right);
}
```

Implementation

```
void mergesort(int arr[], int left, int right) {
    if (left >= right) return;
    // size of the subarray is at least 2:
    int mid = (left + right) / 2;
    mergesort(arr, left, mid);
    mergesort(arr, mid+1, right);
    merge(arr, left, mid, right);
}
```

Implementation

```
void mergesort(int arr[], int left, int right) {
    if (left >= right) return;
    // size of the subarray is at least 2:
    int mid = (left + right) / 2;
    mergesort(arr, left, mid);
    mergesort(arr, mid+1, right);
    merge(arr, left, mid, right);
}
```

Time Complexity

Time Complexity

- Merge sort is a divide-and-conquer algorithm.
- Let $T(n)$ be the time complexity of merge sort.
- Recurrence relation: $T(n) \;=\; 2 \cdot T(n/2) \;+\; cn.$

Time Complexity

- Merge sort is a divide-and-conquer algorithm.
- Let $T(n)$ be the time complexity of merge sort.
- Recurrence relation: $T(n) \;=\; 2 \cdot T(n/2) \;+\; cn.$

Partition the array into 2 sub-arrays.

Time Complexity

- Merge sort is a divide-and-conquer algorithm.
- Let $T(n)$ be the time complexity of merge sort.
- Recurrence relation: $T(n) \;=\; 2 \cdot T(n/2) \;+\; cn.$

Lengths of the two sub-arrays are both $n/2$.

Time Complexity

- Merge sort is a divide-and-conquer algorithm.
- Let $T(n)$ be the time complexity of merge sort.
- Recurrence relation: $T(n) \; = \; 2 \cdot T(n/2) \; + \; cn.$

cn is the time for merging two sorted arrays. (c is a constant.)

Time Complexity

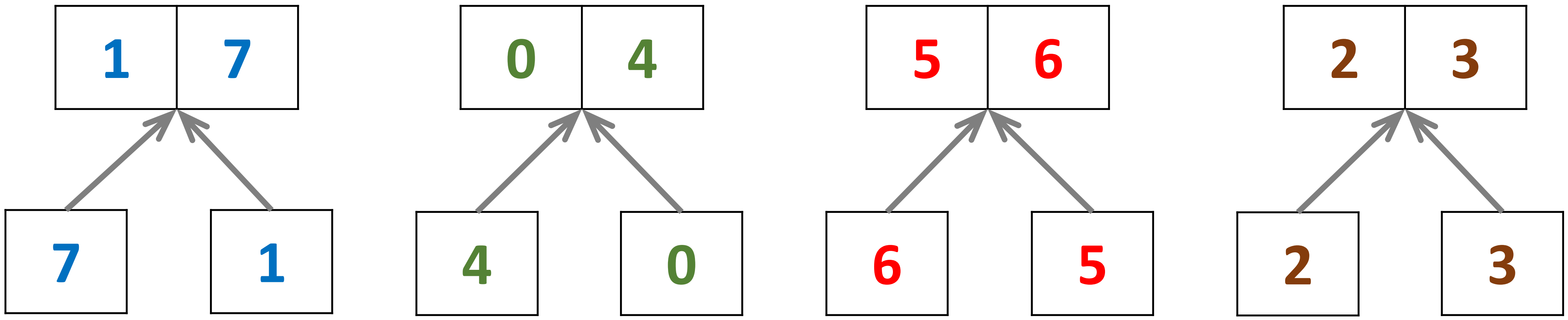
- Merge sort is a divide-and-conquer algorithm.
- Let $T(n)$ be the time complexity of merge sort.
- Recurrence relation: $T(n) \;=\; 2 \cdot T(n/2) \;+\; cn.$
- Using some math, we obtain

$$T(n) = O(n \log n).$$

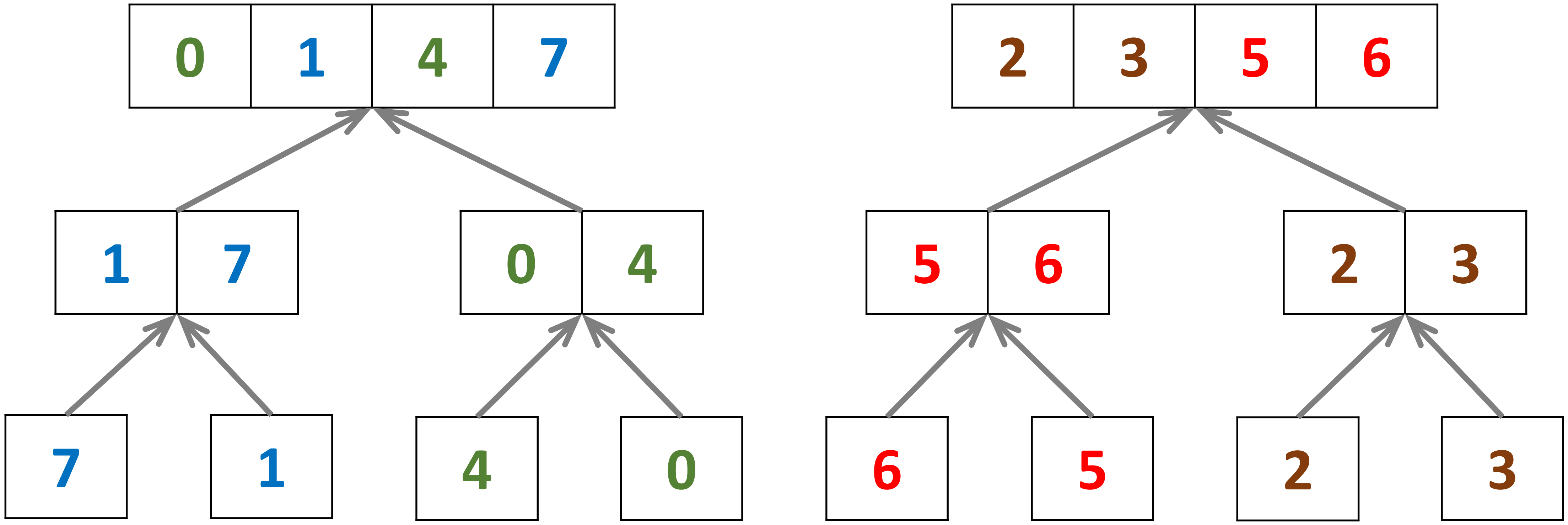
Bottom-Up Merge Sort

7	1	4	0	6	5	2	3

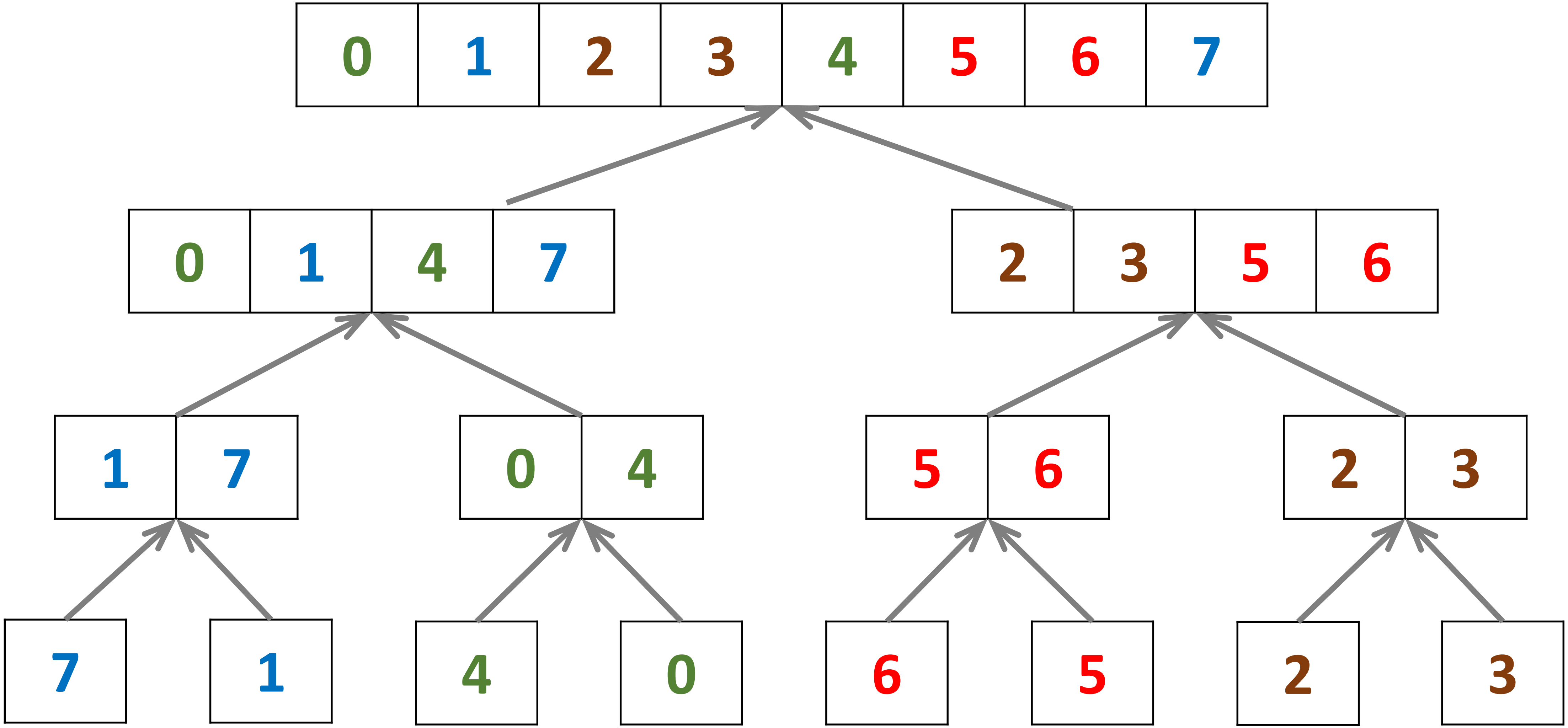
Bottom-Up Merge Sort



Bottom-Up Merge Sort



Bottom-Up Merge Sort



Thank You!

Proof of Time Complexity

Time Complexity

- $T(n)$: Time complexity of sorting size-n array.
- Sizes of Group 1 and Group 2 are both $\frac{n}{2}$.
- Time complexity:

$$T(n) \;=\; 2\,T(n/2) \;+\; n.$$

- ➔ $\frac{T(n)}{n} = \frac{T(n/2)}{n\,/\,2} + 1\,.$

Time Complexity

- $\frac{T(n)}{n} = \frac{T(n/2)}{n\,/\,2} + 1.$
- $\frac{T(n/2)}{n/2} = \frac{T(n/4)}{n/4} + 1.$
- $\frac{T(n/4)}{n/4} = \frac{T(n/8)}{n/8} + 1.$

$$\vdots$$

- $\frac{T(2)}{2} = \frac{T(1)}{1} + 1.$

$$\begin{aligned}\frac{T(n)}{n} &= \frac{T(n/2)}{n\,/\,2} + 1\\ &= \frac{T(n/4)}{n/4} + 2\\ &= \frac{T(n/8)}{n/8} + 3\\ &= \cdots\\ &= \frac{T(1)}{1} + \log_2 n\end{aligned}$$