Algorithm Mergesort: $\Theta(n \log n)$ Complexity

Georgy Gimel'farb

COMPSCI 220 Algorithms and Data Structures

1. Mergesort: basic ideas and correctness

2. Merging sorted lists

3. Time complexity of mergesort

Mergesort: Worst-case Running time of $\Theta(n \log n)$

A recursive divide-and-conquer approach to data sorting introduced by Professor John von Neumann in **1945!**

- The best, worst, and average cases are similar.
- Particularly good for sorting data with slow access times, e.g., stored in external memory or linked lists.

Basic ideas behind the algorithm:

1. If the number of items is 0 or 1, return; otherwise:
    1. Separate the list into two lists of equal or nearly equal size.
    2. Recursively sort the first and the second halves separately.
2. Finally, merge the two sorted halves into one sorted list.

Almost all the work is performed in the merge steps.

Correctness of Mergesort

Lemma 2.8 (Textbook): Mergesort is correct.

Proof: by induction on the size n of the list.

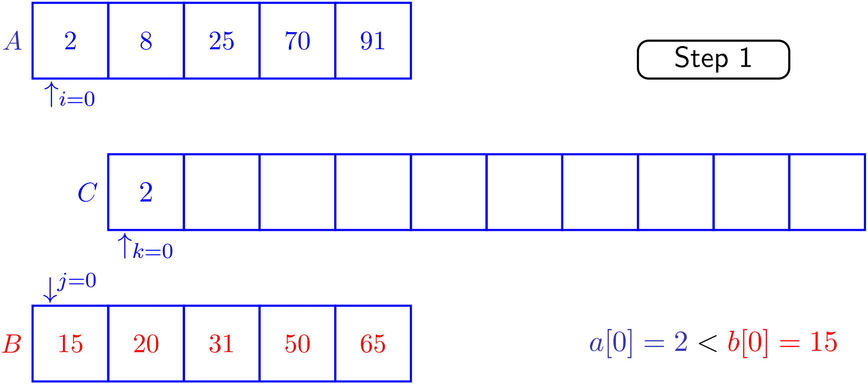
- **Basis:** If $n = 0$ or 1, mergesort is correct.
- **Inductive hypothesis:** Mergesort is correct for all $m < n$.
- **Inductive step:**
  - Mergesort calls itself recursively on two sublists.
  - Each of these sublists has size less than n and thus is correctly sorted by induction hypothesis.
  - Provided that the merge step is correct, the top level call of mergesort returns the correct answer.

- Linear time merge, $\Theta(n)$ yields complexity $\Theta(n \log n)$ for mergesort.
- The merge is at least linear in the total size of the two lists: in the worst case every element must be looked at for the correct ordering.

Linear Time, $\Theta(n)$, Merge of Sorted Arrays

```
if a[i] < b[j] then c[k++] = a[i++] else c[k++] = b[j++]
```

Example 2.10 (textbook)



A					
	2	8	25	70	91

C										
	2									

B					
	15	20	31	50	65

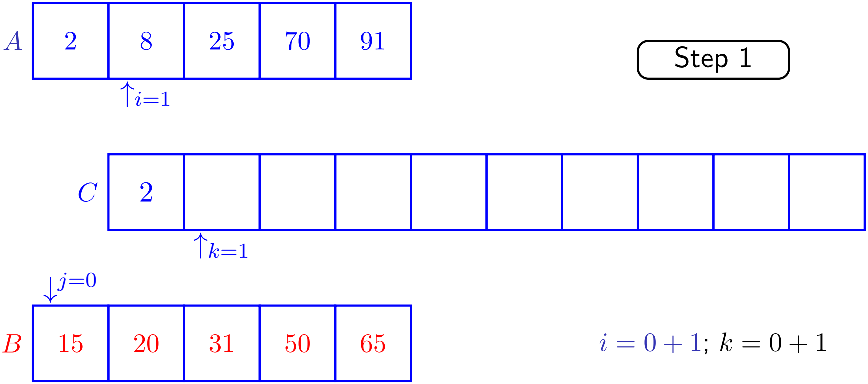
Step 1

$a[0] = 2 < b[0] = 15$

Linear Time, $\Theta(n)$, Merge of Sorted Arrays

```
if a[i] < b[j] then c[k++] = a[i++] else c[k++] = b[j++]
```

Example 2.10 (textbook)



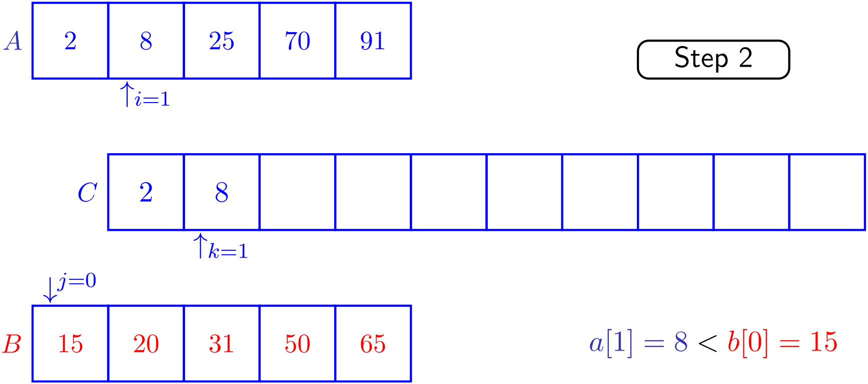
Step 1

$i = 0 + 1;\ k = 0 + 1$

Linear Time, $\Theta(n)$, Merge of Sorted Arrays

```
if a[i] < b[j] then c[k++] = a[i++] else c[k++] = b[j++]
```

Example 2.10 (textbook)



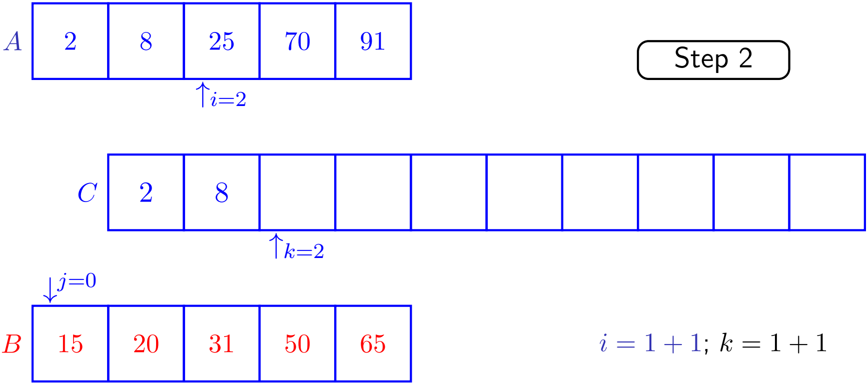
Step 2

$a[1] = 8 < b[0] = 15$

Linear Time, $\Theta(n)$, Merge of Sorted Arrays

```
if a[i] < b[j] then c[k++] = a[i++] else c[k++] = b[j++]
```

Example 2.10 (textbook)



A	2	8	25	70	91

$\uparrow i{=}2$

Step 2

C	2	8								

$\uparrow k{=}2$

$\downarrow j{=}0$

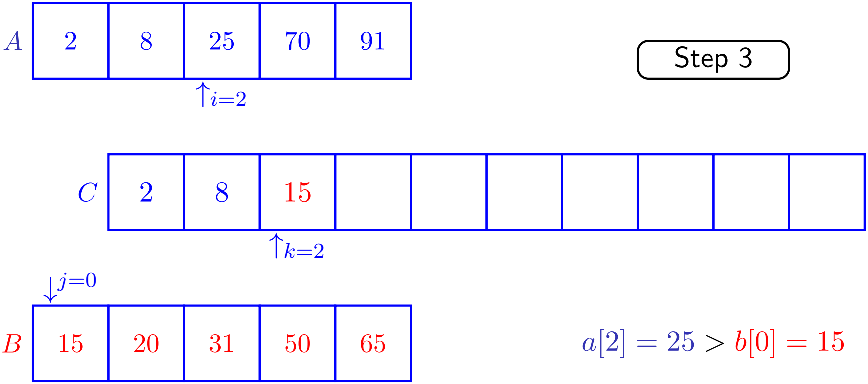
B	15	20	31	50	65

$i = 1 + 1$; $k = 1 + 1$

Linear Time, $\Theta(n)$, Merge of Sorted Arrays

```
if a[i] < b[j] then c[k++] = a[i++] else c[k++] = b[j++]
```

Example 2.10 (textbook)



A	2	8	25	70	91

$\uparrow i=2$

Step 3

C	2	8	15							

$\uparrow k=2$

$\downarrow j=0$

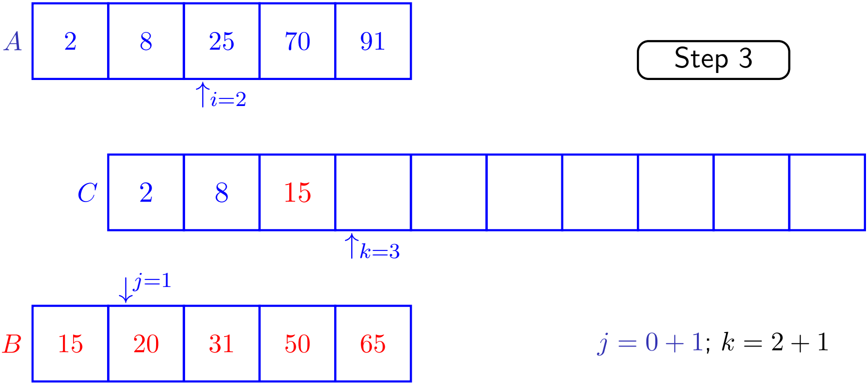
B	15	20	31	50	65

$a[2] = 25 > b[0] = 15$

Linear Time, $\Theta(n)$, Merge of Sorted Arrays

```
if a[i] < b[j] then c[k++] = a[i++] else c[k++] = b[j++]
```

Example 2.10 (textbook)



A

2	8	25	70	91

$\uparrow i=2$

C

2	8	15							

$\uparrow k=3$

$\downarrow j=1$

B

15	20	31	50	65

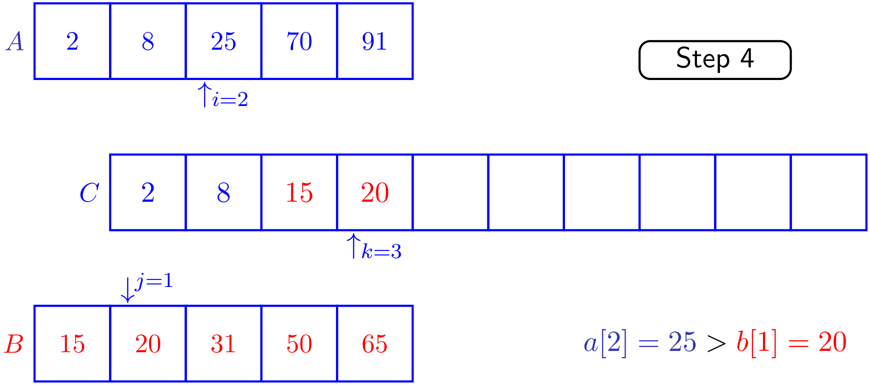
Step 3

$j = 0 + 1;\ k = 2 + 1$

Linear Time, $\Theta(n)$, Merge of Sorted Arrays

```
if a[i] < b[j] then c[k++] = a[i++] else c[k++] = b[j++]
```

Example 2.10 (textbook)



A	2	8	25	70	91
			↑$i{=}2$		

Step 4

C	2	8	15	20						
				↑$k{=}3$						

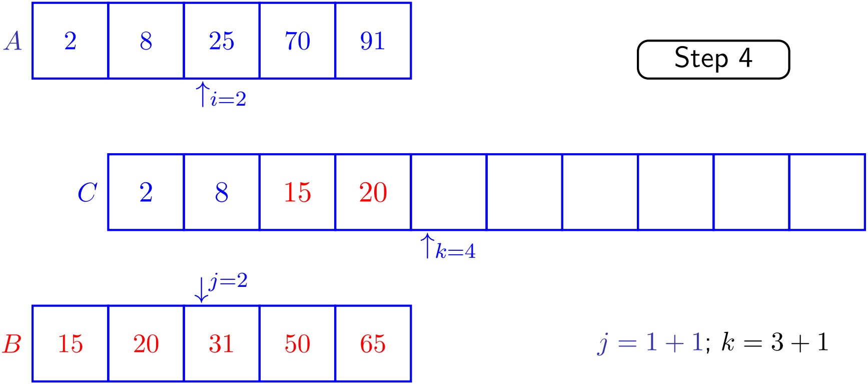
		↓$j{=}1$			
B	15	20	31	50	65

$a[2] = 25 > b[1] = 20$

Linear Time, $\Theta(n)$, Merge of Sorted Arrays

```
if a[i] < b[j] then c[k++] = a[i++] else c[k++] = b[j++]
```

Example 2.10 (textbook)



A

2	8	25	70	91

$\uparrow i=2$

Step 4

C

2	8	15	20						

$\uparrow k=4$

$\downarrow j=2$

B

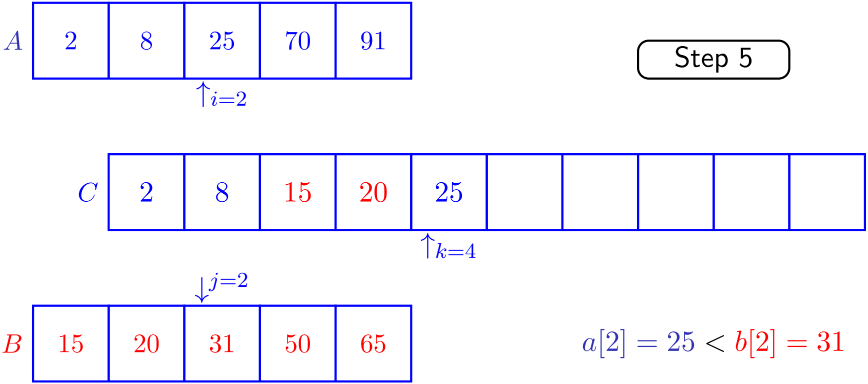
15	20	31	50	65

$j = 1 + 1;\ k = 3 + 1$

Linear Time, $\Theta(n)$, Merge of Sorted Arrays

```
if a[i] < b[j] then c[k++] = a[i++] else c[k++] = b[j++]
```

Example 2.10 (textbook)



A	2	8	25	70	91

$\uparrow i{=}2$

C	2	8	15	20	25					

$\uparrow k{=}4$

$\downarrow j{=}2$

B	15	20	31	50	65

Step 5

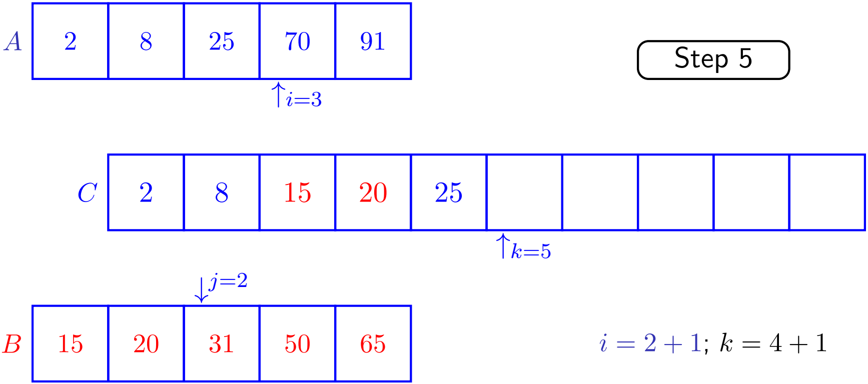
$$a[2] = 25 < b[2] = 31$$

Linear Time, $\Theta(n)$, Merge of Sorted Arrays

```
if a[i] < b[j] then c[k++] = a[i++] else c[k++] = b[j++]
```

Example 2.10 (textbook)

Step 5



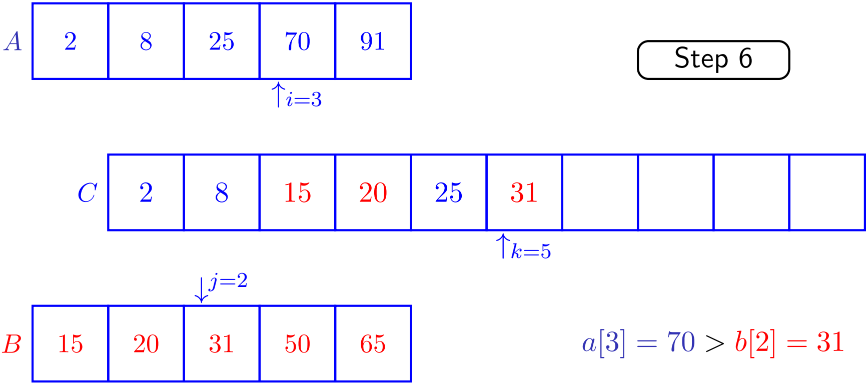
$i = 2 + 1;\ k = 4 + 1$

Linear Time, $\Theta(n)$, Merge of Sorted Arrays

```
if a[i] < b[j] then c[k++] = a[i++] else c[k++] = b[j++]
```

Example 2.10 (textbook)

Step 6



A	2	8	25	70	91

$\uparrow i{=}3$

C	2	8	15	20	25	31				

$\uparrow k{=}5$

$\downarrow j{=}2$

B	15	20	31	50	65

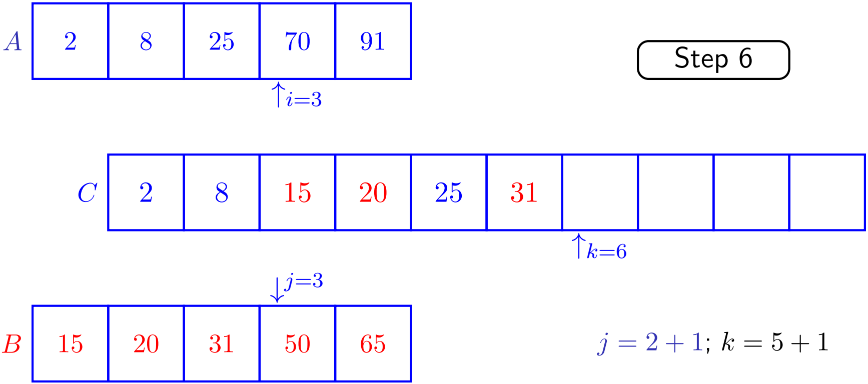
$a[3] = 70 > b[2] = 31$

Linear Time, $\Theta(n)$, Merge of Sorted Arrays

```
if a[i] < b[j] then c[k++] = a[i++] else c[k++] = b[j++]
```

Example 2.10 (textbook)

Step 6



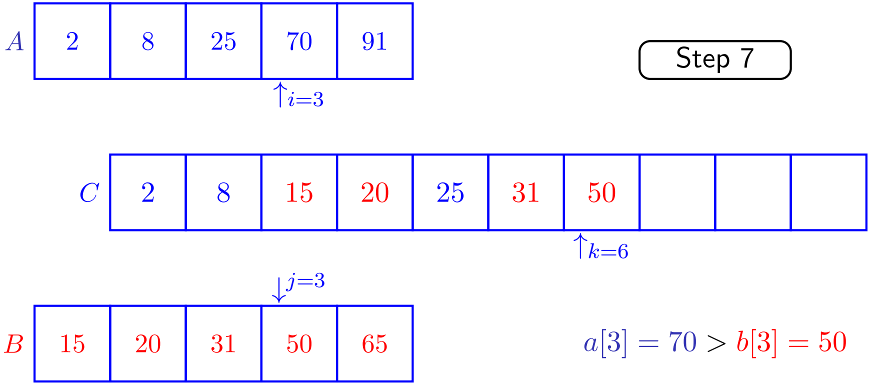
A	2	8	25	70	91					
C	2	8	15	20	25	31				
B	15	20	31	50	65					

$j = 2 + 1;\ k = 5 + 1$

Linear Time, $\Theta(n)$, Merge of Sorted Arrays

```
if a[i] < b[j] then c[k++] = a[i++] else c[k++] = b[j++]
```

Example 2.10 (textbook)



A	2	8	25	70	91

↑$i=3$

Step 7

C	2	8	15	20	25	31	50			

↑$k=6$

↓$j=3$

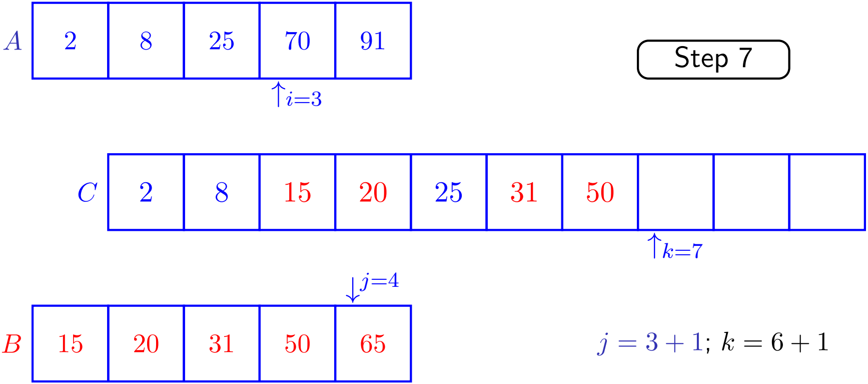
B	15	20	31	50	65

$$a[3] = 70 > b[3] = 50$$

Linear Time, $\Theta(n)$, Merge of Sorted Arrays

```
if a[i] < b[j] then c[k++] = a[i++] else c[k++] = b[j++]
```

Example 2.10 (textbook)



A	2	8	25	70	91

$\uparrow i{=}3$

Step 7

C	2	8	15	20	25	31	50			

$\uparrow k{=}7$

$\downarrow j{=}4$

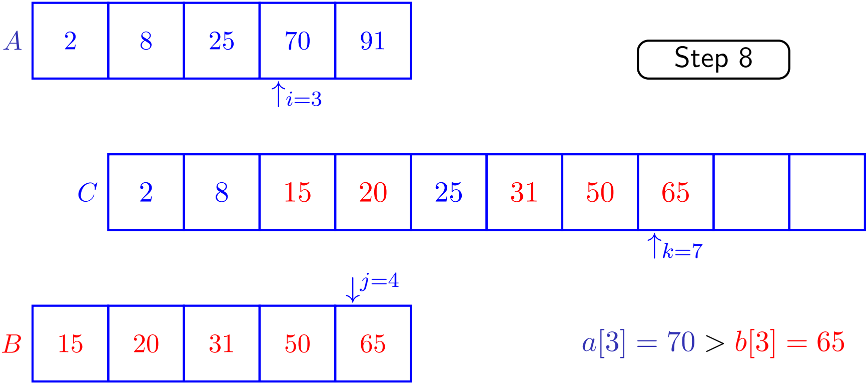
B	15	20	31	50	65

$j = 3 + 1;\ k = 6 + 1$

Linear Time, $\Theta(n)$, Merge of Sorted Arrays

```
if a[i] < b[j] then c[k++] = a[i++] else c[k++] = b[j++]
```

Example 2.10 (textbook)



A	2	8	25	70	91

$\uparrow i=3$

Step 8

C	2	8	15	20	25	31	50	65		

$\uparrow k=7$

$\downarrow j=4$

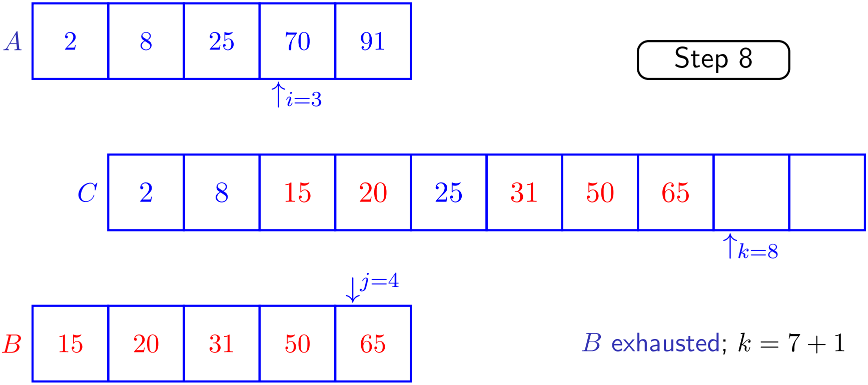
B	15	20	31	50	65

$a[3] = 70 > b[3] = 65$

Linear Time, $\Theta(n)$, Merge of Sorted Arrays

```
if a[i] < b[j] then c[k++] = a[i++] else c[k++] = b[j++]
```

Example 2.10 (textbook)



A	2	8	25	70	91

C	2	8	15	20	25	31	50	65		

B	15	20	31	50	65

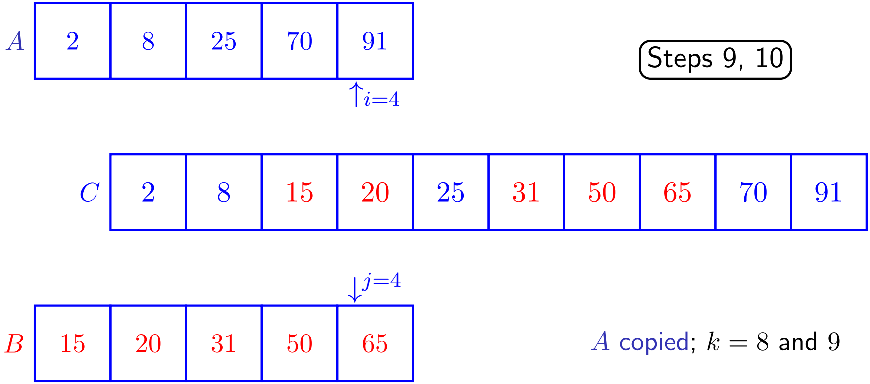
Step 8

B exhausted; $k = 7 + 1$

Linear Time, $\Theta(n)$, Merge of Sorted Arrays

```
if a[i] < b[j] then c[k++] = a[i++] else c[k++] = b[j++]
```

Example 2.10 (textbook)



A	2	8	25	70	91

$\uparrow i=4$

C	2	8	15	20	25	31	50	65	70	91

$\downarrow j=4$

B	15	20	31	50	65

Steps 9, 10

A copied; $k = 8$ and 9

Linear Time, $\Theta(n)$, Merge of Sorted Arrays

sorted subarrays $a[l..s-1]$ and $a[s..r]$ into $a[l..r]$

```
algorithm merge
Input: array $a[0..n-1]$; indices $l$, $r$; index $s$; array $t[0..n-1]$
begin   $i \leftarrow l; j \leftarrow s; k \leftarrow l$
    while $i \leq s-1$ and $j \leq r$ do
        if $a[i] \leq a[j]$ then $t[k] \leftarrow a[i]; k \leftarrow k+1; i \leftarrow i+1$
        else                   $t[k] \leftarrow a[j]; k \leftarrow k+1; j \leftarrow j+1$
        end if
    end while
    while $i \leq s-1$ do
        $t[k] \leftarrow a[i]; k \leftarrow k+1; i \leftarrow i+1$
    end while
    while $j \leq r$ do
        $t[k] \leftarrow a[j]; k \leftarrow k+1; j \leftarrow j+1$
    end while
    return $a \leftarrow t$
end
```

copy the rest of the 1^{st} half

copy the rest of the 2^{nd} half

Merging Sorted Lists: Linear Time Complexity

Theorem 2.9: Two input sorted lists $A = [a_1, \ldots, a_\nu]$ of size ν and $B = [b_1, \ldots, b_\mu]$ of size μ can be merged into an output sorted list $C = [c_1, \ldots, c_n]$ of size $n = \nu + \mu$ in linear time.

Proof. The number of comparisons needed is linear in n:

- Let pointers i, j, and k to current positions in A, B, and C, respectively, be initially at the first positions, $i = j = k = 1$.
- Each time the smaller of a_i and b_j is copied to c_k, and the pointers k and either i or j are incremented by 1:

$$(\,a_i > b_j\,)? \Rightarrow \begin{cases} a_i > b_j & \Rightarrow \quad c_k = b_j \quad j \leftarrow j+1; \quad k \leftarrow k+1 \\ a_i \leq b_j & \Rightarrow \quad c_k = a_i \quad i \leftarrow i+1; \quad k \leftarrow k+1 \end{cases}$$

- After A or B is exhausted, the rest of the other list is copied to C.
- Each comparison advances k so that the maximum number of comparisons is $n = \nu + \mu$, all other operations being linear, too.

Recursive `mergesort` for arrays

Easier than for linked lists: a constant time for splitting an array in the middle.

sorts the subarray $a[l..r]$

```
algorithm mergeSort
Input: array $a[0..n-1]$; array indices $l$, $r$; array $t[0..n-1]$
begin
    if $l < r$ then $m \leftarrow \left\lfloor \frac{l+r}{2} \right\rfloor$;
        mergeSort$(a, l, m, t)$;
        mergeSort$(a, m+1, r, t)$;
        merge$(a, l, m+1, r, t)$;
    end if
end
```

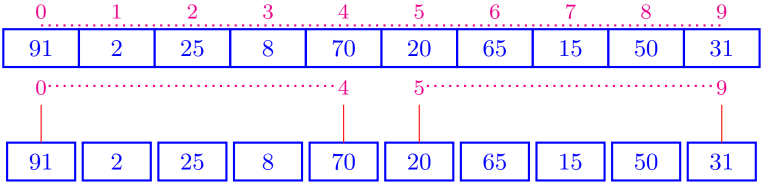
- The recursive version simply divides the list until it reaches lists of size 1, then merges these repeatedly.
- **Straight mergesort** eliminates the recursion by merging first lists of size 1 into lists of size 2, then lists of size 2 into lists of size 4, etc.

How Straight Mergesort Works

0	1	2	3	4	5	6	7	8	9
91	2	25	8	70	20	65	15	50	31

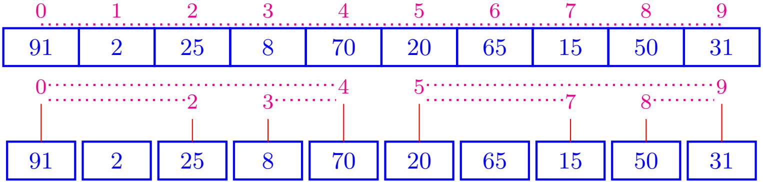
$2n$ or n comparisons for random or sorted/reverse data, respectively.

How Straight Mergesort Works



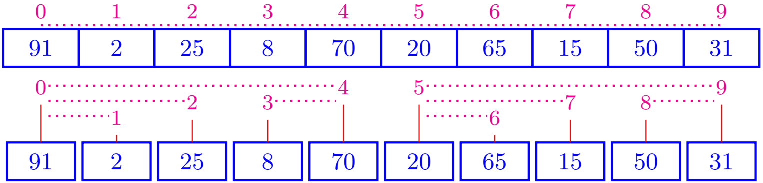
$2n$ or n comparisons for random or sorted/reverse data, respectively.

How Straight Mergesort Works



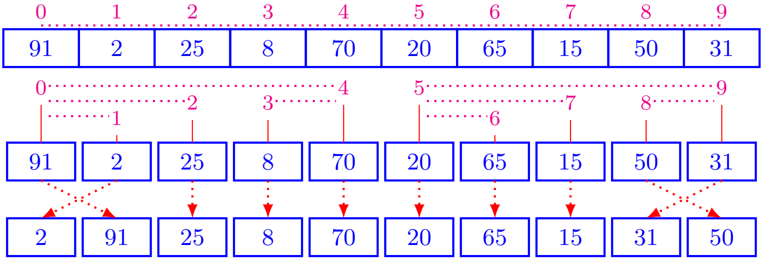
$2n$ or n comparisons for random or sorted/reverse data, respectively.

How Straight Mergesort Works



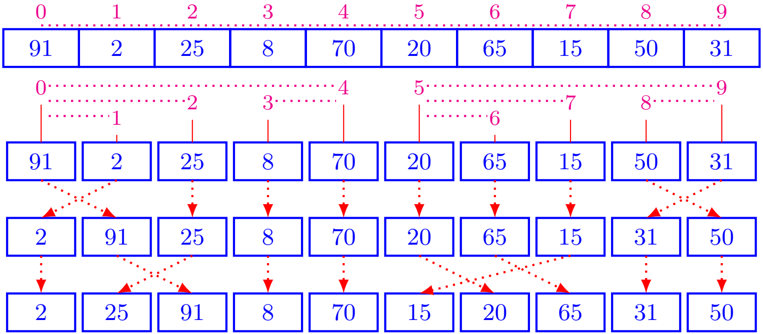
$2n$ or n comparisons for random or sorted/reverse data, respectively.

How Straight Mergesort Works



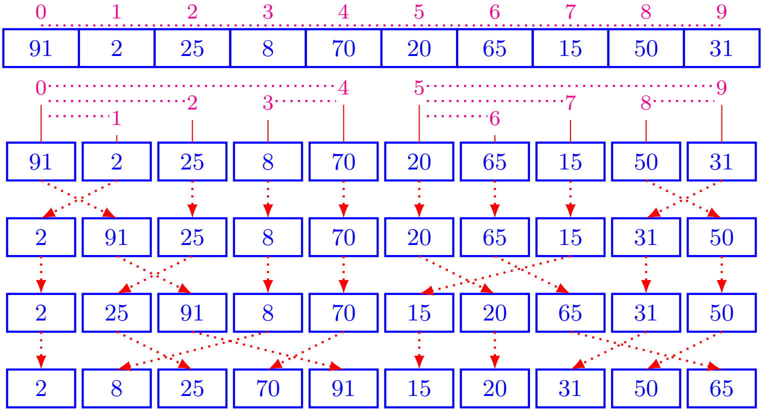
$2n$ or n comparisons for random or sorted/reverse data, respectively.

How Straight Mergesort Works



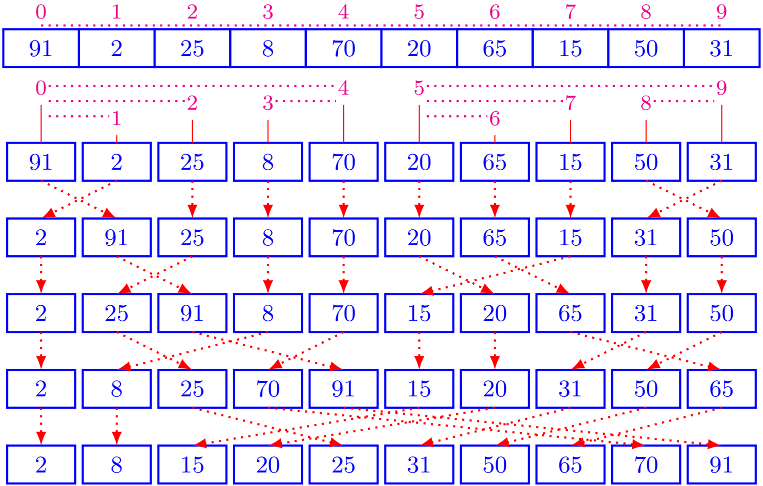
$2n$ or n comparisons for random or sorted/reverse data, respectively.

How Straight Mergesort Works



$2n$ or n comparisons for random or sorted/reverse data, respectively.

How Straight Mergesort Works



$2n$ or n comparisons for random or sorted/reverse data, respectively.

Analysis of Mergesort

Theorem 2.11: The running time of `mergesort` on an input list of size n is $\Theta(n \log n)$ in the best, worst, and average case.

Proof. The number of comparisons used by mergesort on an input of size n satisfies a recurrence of the form:

$$T(n) = T\left(\left\lceil \frac{n}{2} \right\rceil\right) + T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + a(n); \;\; 1 \leq a(n) \leq n - 1$$

It is straightforward to show that $T(n)$ is $\Theta(n \log n)$.

- The other elementary operations in the divide and combine steps depend on the implementation of the list, but in each case their number is $\Theta(n)$.
- Thus these operations satisfy a similar recurrence and do not affect the $\Theta(n \log n)$ answer.

Recurrence $T(n) = 2T\left(\frac{n}{2}\right) + \alpha n$; $T(1) = 0$

For $n = 2^m$, “telescoping” the recurrence $T(2^m) = 2T(2^{m-1}) + \alpha 2^m$ (see Lecture 06, Slides 19-20, and Textbook, Example 1.32):

$$
\begin{array}{lcll lcl}
T(2^m) & = & 2T(2^{m-1}) + \alpha \cdot 2^m & \rightarrow_{\times 2^0} & T(2^m) - 2T(2^{m-1}) & = & \alpha \cdot 2^m \\
T(2^{m-1}) & = & 2T(2^{m-2}) + \alpha \cdot 2^{m-1} & \rightarrow_{\times 2^1} & 2T(2^{m-1}) - 2^2T(2^{m-2}) & = & \alpha \cdot 2^m \\
T(2^{m-2}) & = & 2T(2^{m-3}) + \alpha \cdot 2^{m-1} & \rightarrow_{\times 2^2} & 2^2T(2^{m-2}) - 2^3T(2^{m-3}) & = & \alpha \cdot 2^m \\
\ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\
T(2^2) & = & 2T(2^1) + \alpha \cdot 2^2 & \rightarrow_{\times 2^{m-2}} & 2^{m-2}T(2^2) - 2^{m-1}T(2^1) & = & \alpha \cdot 2^m \\
T(2^1) & = & 2\,\underbrace{T(2^0)}_{T(1)=0} + \alpha \cdot 2^1 & \rightarrow_{\times 2^{m-1}} & 2^{m-1}T(2^1) - \underbrace{2^mT(2^0)}_{=0} & = & \alpha \cdot 2^m \\
\hline
\end{array}
$$

$$\boxed{T(2^m) = \alpha \cdot 2^m \cdot m}$$

So $T(n) \approx \alpha \cdot n \cdot \log_2 n$.

Analysis of Mergesort

+ The $\Theta(n \log n)$ best-, average-, and worst-case complexity because the merging is always linear.
  - Recall the basic recurrence:

$$T(n) = 2T\left(\frac{n}{2}\right) + cn \quad \Rightarrow \quad T(n) = cn \lg n$$

  and Theorem 2.11 (Slide 10).

– Extra $\Theta(n)$ temporary array for merging data.

– Extra copying to the temporary array and back.

- Algorithm `mergesort` is useful only for external sorting.
- For internal sorting: `quickSort` and `heapsort` are much better.