

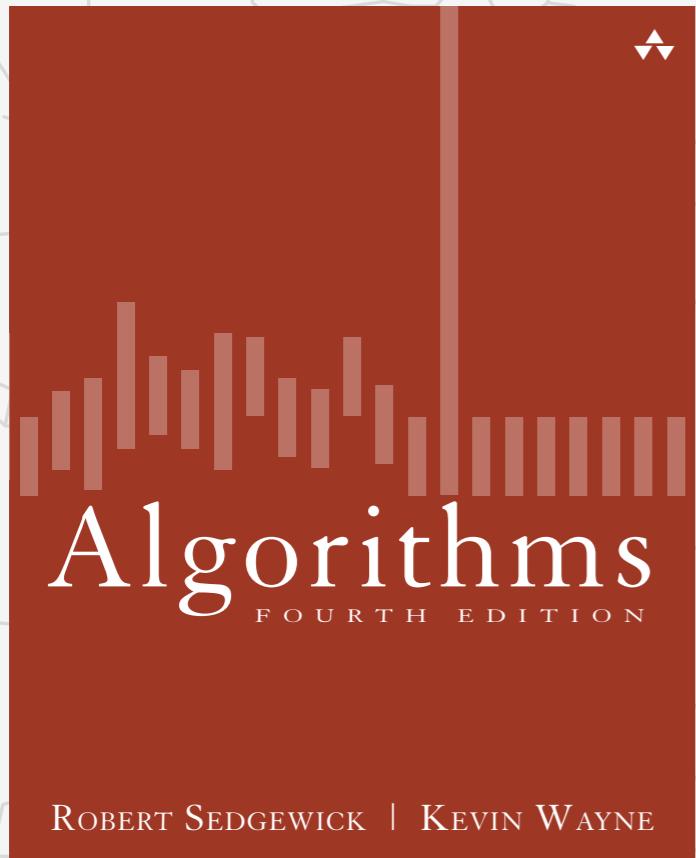
# SPRING 2023

# INFORMATION TECHNOLOGY RESEARCH

YI HAN

DEPARTMENT OF INFORMATION MANAGEMENT  
NATIONAL SUN YAT-SEN UNIVERSITY

Lecture slides are based on the supplemental materials of the textbook: <https://algs4.cs.princeton.edu>



<http://algs4.cs.princeton.edu>

## 2.2 MERGESORT

---

- ▶ *mergesort*
- ▶ *bottom-up mergesort*
- ▶ *comparators*
- ▶ *stability*

# Two classic sorting algorithms: mergesort and quicksort

---

Critical components in the world's computational infrastructure.

- Full scientific understanding of their properties has enabled us to develop them into practical system sorts.
- Quicksort honored as one of top 10 algorithms of 20<sup>th</sup> century in science and engineering.

Mergesort.



Quicksort.



# Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

## 2.2 MERGESORT

---

- ▶ *mergesort*
- ▶ *bottom-up mergesort*
- ▶ *comparators*
- ▶ *stability*

# Mergesort

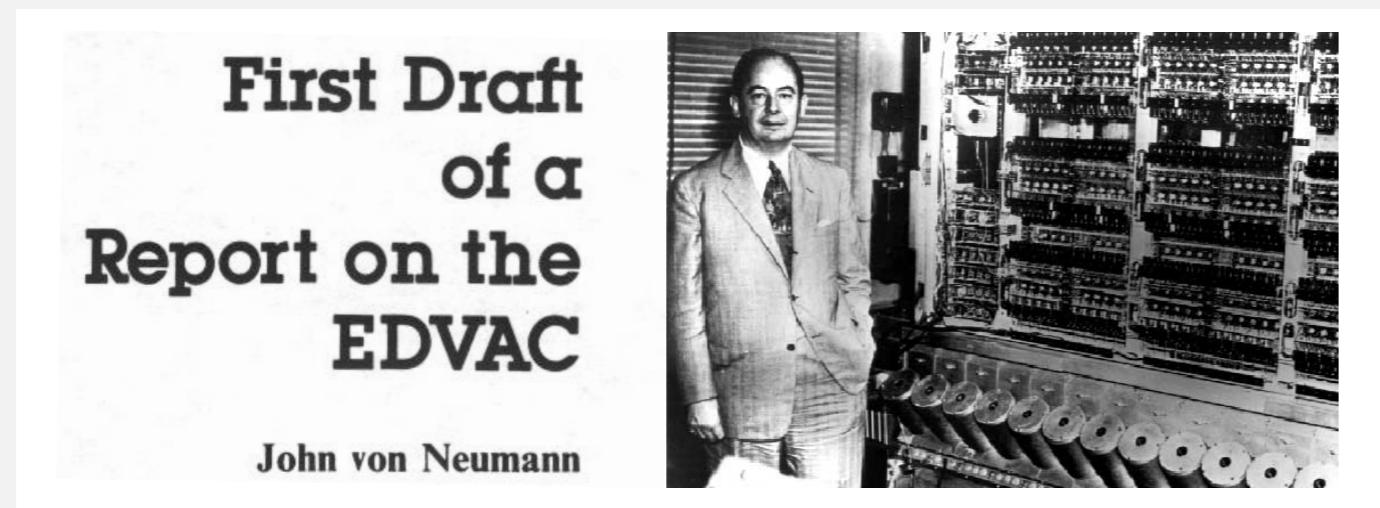
## Basic plan.

- Divide array into two halves.
- Recursively sort each half.
- Merge two halves.

<b>input</b>	M	E	R	G	E	S	O	R	T	E	X	A	M	P	L	E
<b>sort left half</b>	E	E	G	M	O	R	R	S	T	E	X	A	M	P	L	E
<b>sort right half</b>	E	E	G	M	O	R	R	S	A	E	E	L	M	P	T	X
<b>merge results</b>	A	E	E	E	E	G	L	M	M	O	P	R	R	S	T	X

Mergesort overview

Electronic Discrete Variable Automatic Computer (EDVAC): first program on it was mergesort.



Von Neumann architecture  
Processing unit  
Control unit  
Memory  
External mass storage  
I/O mechanisms

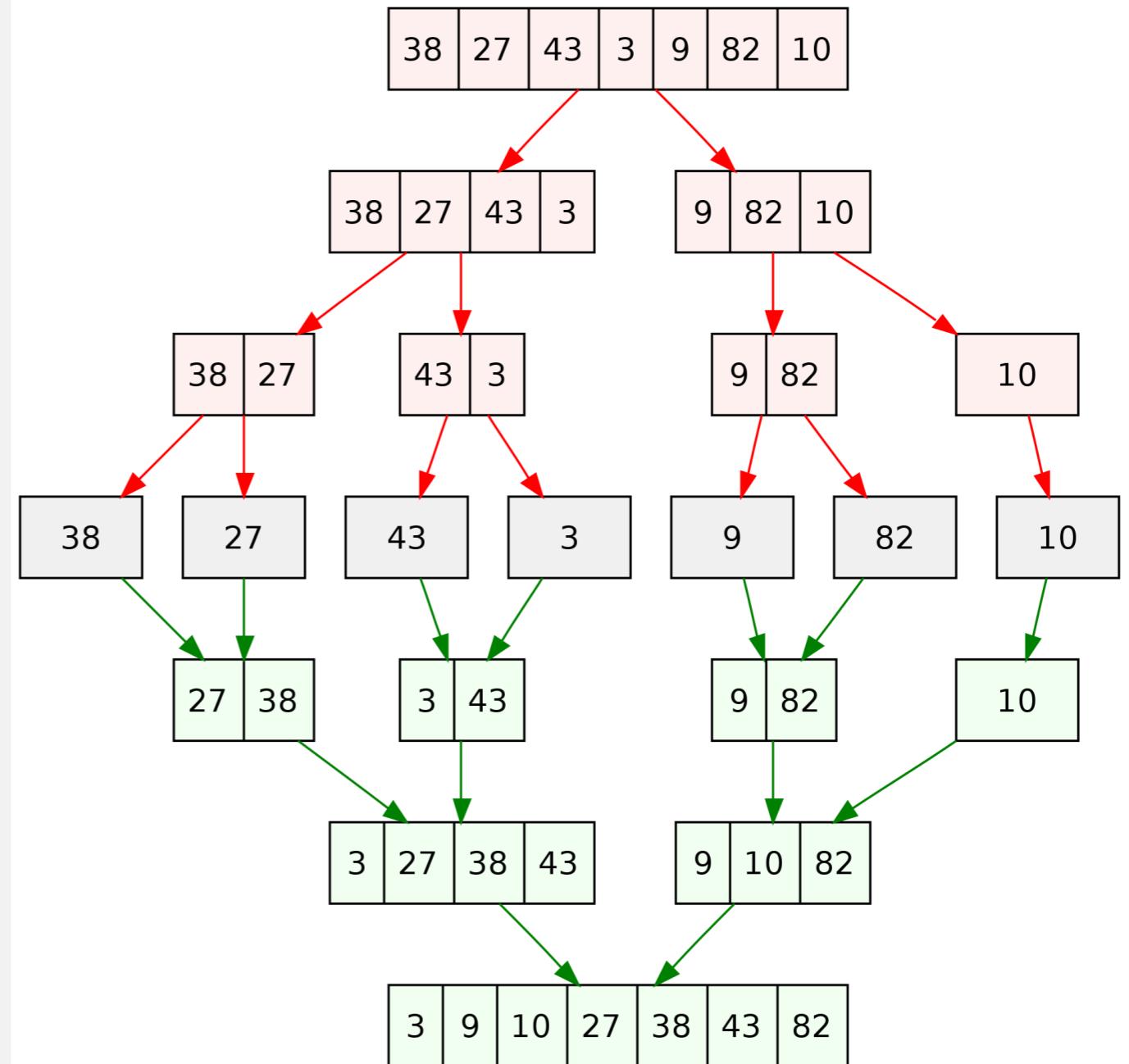
# Mergesort: Divide & merge

## Basic plan.

- Divide array into two halves.
- Recursively sort each half.
- Merge two halves.

## Base case of single element.

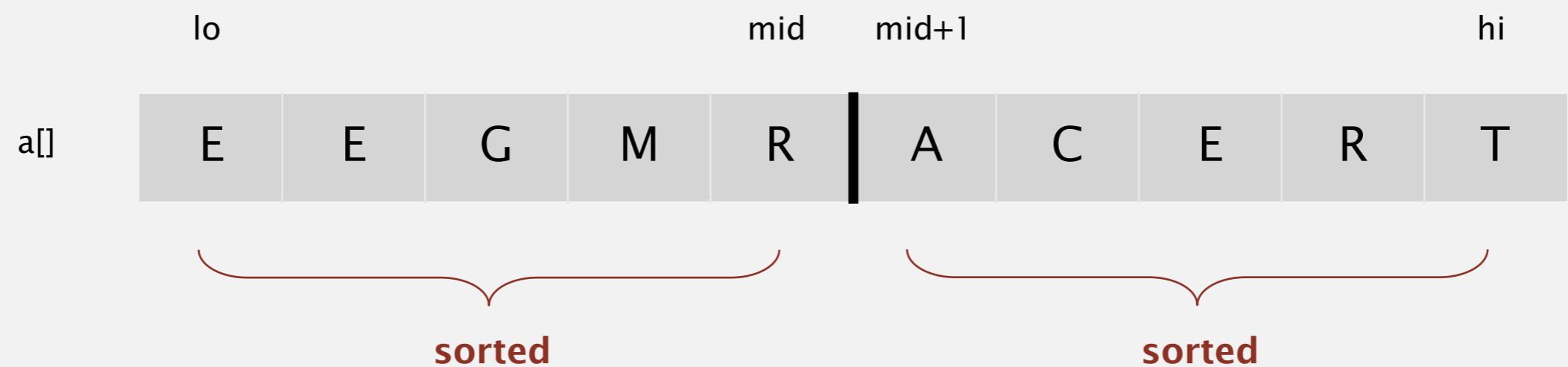
- Is well... sorted.



# Merging demo

---

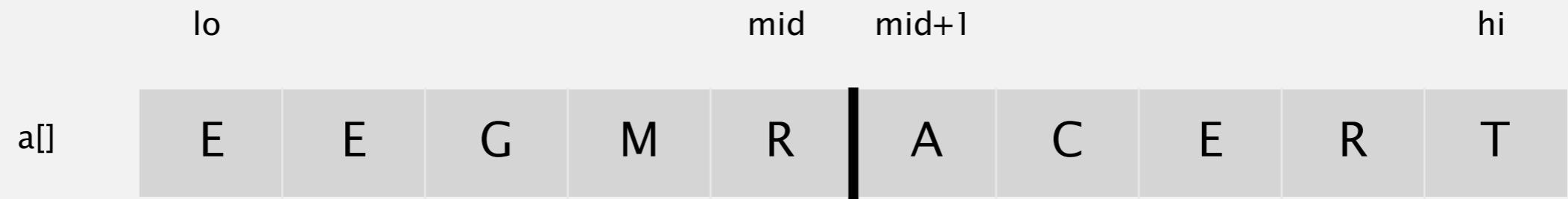
**Goal.** Given two sorted subarrays  $a[lo]$  to  $a[mid]$  and  $a[mid+1]$  to  $a[hi]$ , replace with sorted subarray  $a[lo]$  to  $a[hi]$ .



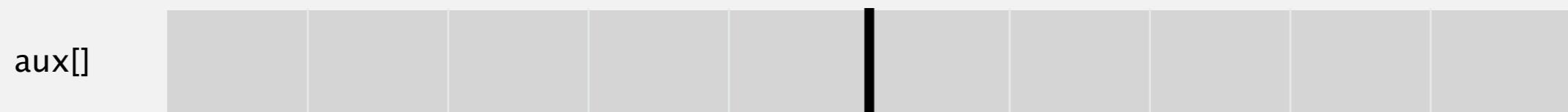
# Merging demo

---

**Goal.** Given two sorted subarrays  $a[lo]$  to  $a[mid]$  and  $a[mid+1]$  to  $a[hi]$ , replace with sorted subarray  $a[lo]$  to  $a[hi]$ .



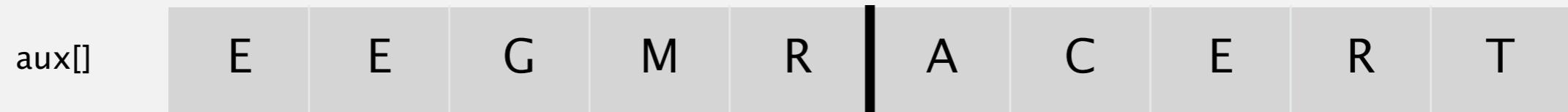
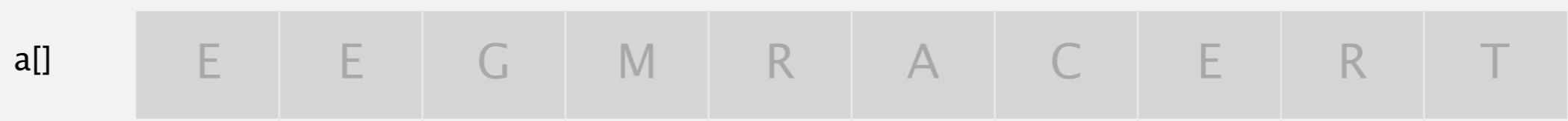
**copy to auxiliary array**



# Merging demo

---

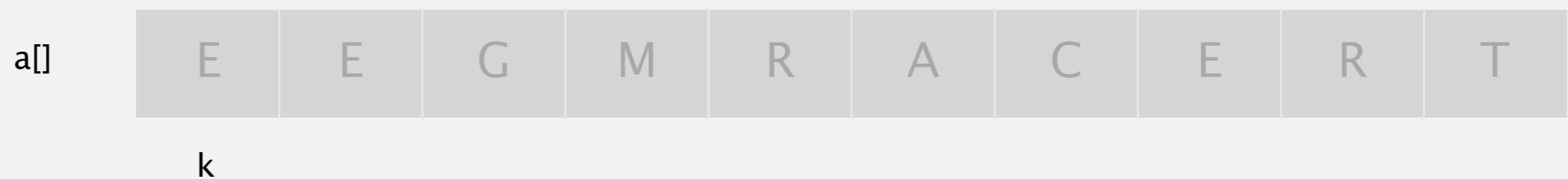
**Goal.** Given two sorted subarrays  $a[lo]$  to  $a[mid]$  and  $a[mid+1]$  to  $a[hi]$ , replace with sorted subarray  $a[lo]$  to  $a[hi]$ .



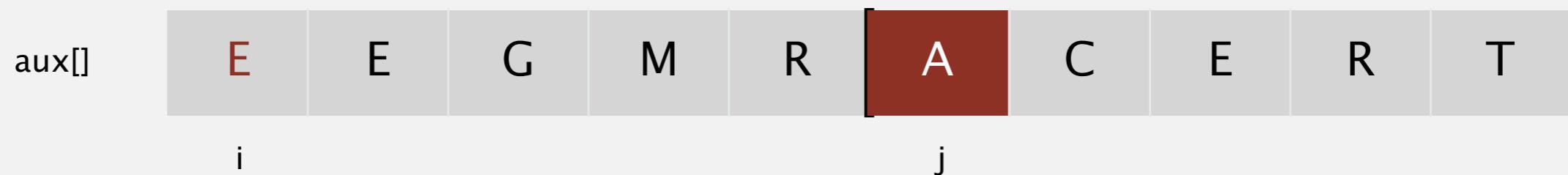
# Merging demo

---

**Goal.** Given two sorted subarrays  $a[lo]$  to  $a[mid]$  and  $a[mid+1]$  to  $a[hi]$ , replace with sorted subarray  $a[lo]$  to  $a[hi]$ .



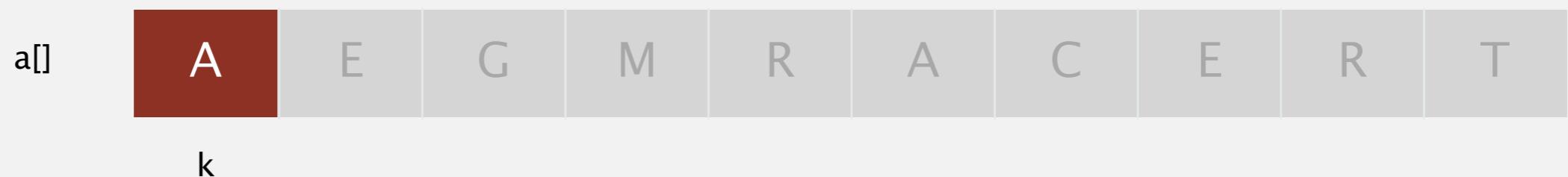
**compare minimum in each subarray**



# Merging demo

---

**Goal.** Given two sorted subarrays  $a[lo]$  to  $a[mid]$  and  $a[mid+1]$  to  $a[hi]$ , replace with sorted subarray  $a[lo]$  to  $a[hi]$ .



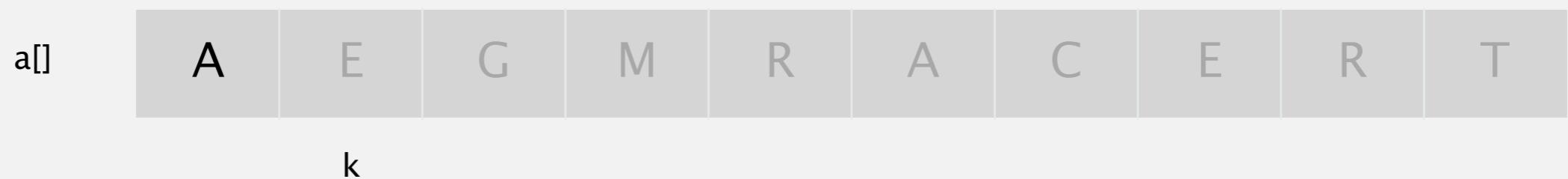
compare minimum in each subarray



# Merging demo

---

**Goal.** Given two sorted subarrays  $a[lo]$  to  $a[mid]$  and  $a[mid+1]$  to  $a[hi]$ , replace with sorted subarray  $a[lo]$  to  $a[hi]$ .



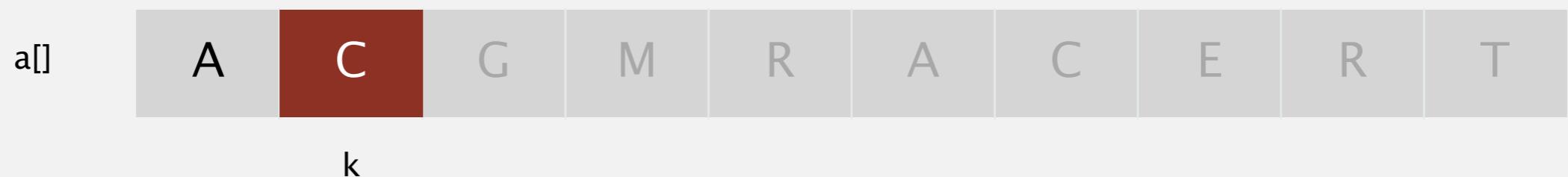
**compare minimum in each subarray**



# Merging demo

---

**Goal.** Given two sorted subarrays  $a[lo]$  to  $a[mid]$  and  $a[mid+1]$  to  $a[hi]$ , replace with sorted subarray  $a[lo]$  to  $a[hi]$ .



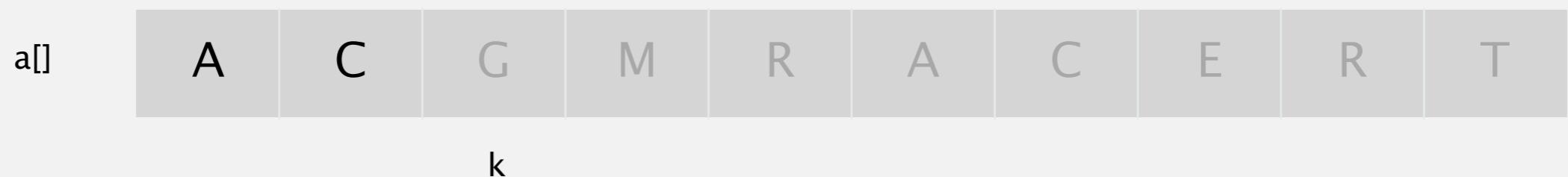
compare minimum in each subarray



# Merging demo

---

**Goal.** Given two sorted subarrays  $a[lo]$  to  $a[mid]$  and  $a[mid+1]$  to  $a[hi]$ , replace with sorted subarray  $a[lo]$  to  $a[hi]$ .



compare minimum in each subarray



# Merging demo

**Goal.** Given two sorted subarrays  $a[lo]$  to  $a[mid]$  and  $a[mid+1]$  to  $a[hi]$ , replace with sorted subarray  $a[lo]$  to  $a[hi]$ .

The diagram shows a character array  $a[ ]$  with 10 slots. The characters are: A, C, E, M, R, A, C, E, R, T. The slot containing 'E' is highlighted in red, and the slot index  $k$  is indicated below it.

**compare minimum in each subarray**

The diagram illustrates a sequence matching problem. On the left, the array `aux[]` contains the characters E, E, G, M, R, A, C, E, R, T. Below the array, the index `i` points to the character E at index 1, and the index `j` points to the character C at index 6. A vertical black bar is positioned between the character R at index 5 and the character A at index 6, indicating a mismatch or a boundary between two segments of the string. The characters A, C, E, R, and T are shown in gray, while E and G are in red.

# Merging demo

**Goal.** Given two sorted subarrays  $a[lo]$  to  $a[mid]$  and  $a[mid+1]$  to  $a[hi]$ , replace with sorted subarray  $a[lo]$  to  $a[hi]$ .

The diagram illustrates a step in a string matching algorithm. A horizontal row of boxes represents the string  $a[]$ : the first box contains 'A', the second 'C', the third 'E', the fourth 'M', the fifth 'R', the sixth 'A', the seventh 'C', the eighth 'E', the ninth 'R', and the tenth 'T'. Below this row, the letter 'k' is centered, indicating the current position being compared. The characters at positions  $k=3$  ('E') and  $k=4$  ('M') do not match ('E' ≠ 'M'), so the algorithm proceeds to the next character.

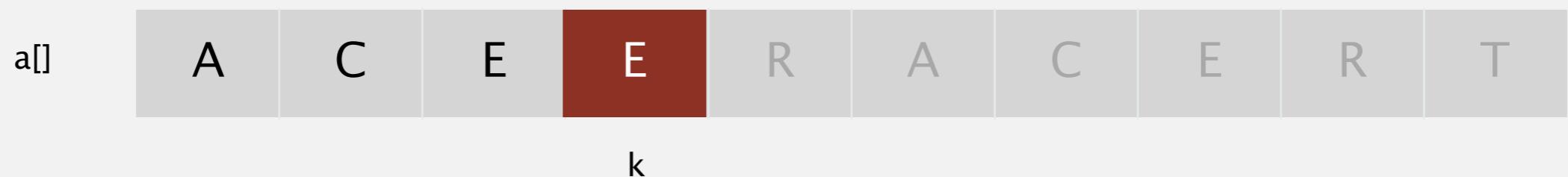
**compare minimum in each subarray**

The diagram shows a sequence of tokens from `aux[]` to `T`. The tokens are represented by letters in boxes. The token at index `i` is `E`, which is highlighted in red. The token at index `j` is `A`. The other tokens are `G`, `M`, `R`, `C`, `E`, `R`, and `T`.

# Merging demo

---

**Goal.** Given two sorted subarrays  $a[lo]$  to  $a[mid]$  and  $a[mid+1]$  to  $a[hi]$ , replace with sorted subarray  $a[lo]$  to  $a[hi]$ .



compare minimum in each subarray



# Merging demo

**Goal.** Given two sorted subarrays  $a[lo]$  to  $a[mid]$  and  $a[mid+1]$  to  $a[hi]$ , replace with sorted subarray  $a[lo]$  to  $a[hi]$ .

a[ ]	A	C	E	E	R	A	C	E	R	T
						k				

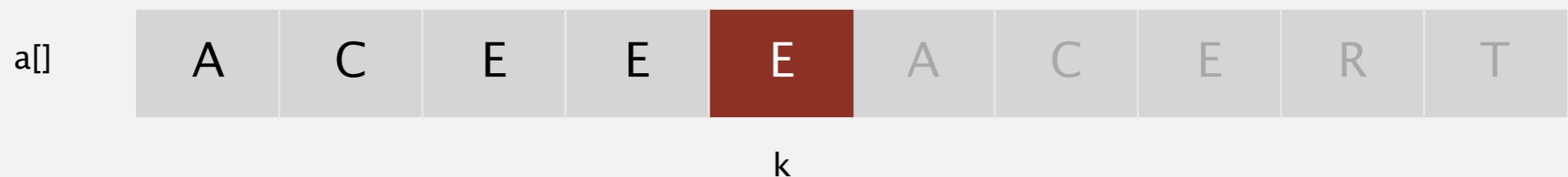
**compare minimum in each subarray**

A horizontal array of 11 cells, each containing a character. The characters are: aux[], E, E, G, M, R, A, C, E, R, T. The cell containing 'G' is highlighted in red. The cell containing 'E' at index 8 is highlighted in dark red. Index  $i$  is marked below the cell containing 'M'. Index  $j$  is marked below the cell containing 'C'.

# Merging demo

---

**Goal.** Given two sorted subarrays  $a[lo]$  to  $a[mid]$  and  $a[mid+1]$  to  $a[hi]$ , replace with sorted subarray  $a[lo]$  to  $a[hi]$ .



**compare minimum in each subarray**



# Merging demo

**Goal.** Given two sorted subarrays  $a[lo]$  to  $a[mid]$  and  $a[mid+1]$  to  $a[hi]$ , replace with sorted subarray  $a[lo]$  to  $a[hi]$ .

a[] A C E E E A C E R T k

**compare minimum in each subarray**

The diagram shows two horizontal rows of characters. The top row is labeled "aux[]" and the bottom row is labeled with indices "i" and "j". The characters are as follows:

E	E	G	M	R	A	C	E	R	T
i					j				

The character 'G' in the top row is highlighted with a dark red background, indicating it is the current character being compared or processed.

# Merging demo

**Goal.** Given two sorted subarrays  $a[lo]$  to  $a[mid]$  and  $a[mid+1]$  to  $a[hi]$ , replace with sorted subarray  $a[lo]$  to  $a[hi]$ .

The diagram shows a horizontal array of characters labeled  $a[ ]$  at the top left. The array consists of ten cells, each containing a single uppercase letter: A, C, E, E, E, G, C, E, R, T. The cell containing 'G' is filled with a dark red color, while all other cells are light gray. Below the array, the letter 'k' is centered, indicating the index of the character 'G'.

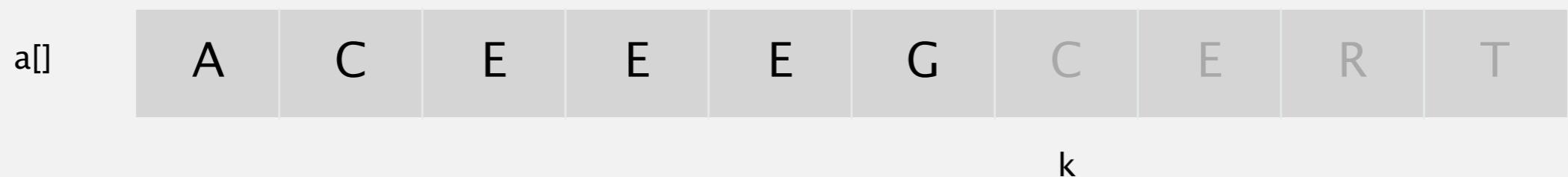
**compare minimum in each subarray**

A diagram of a suffix tree node for the string "GEMAR". The node contains the characters "A", "C", "E", "R", and "T". A vertical black bar is positioned between the character "R" and the character "A". Below the node, the index "i" is centered under the character "G", and the index "j" is centered under the character "T".

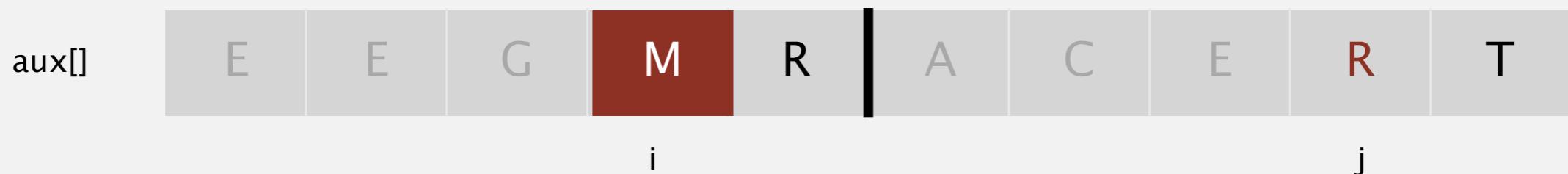
# Merging demo

---

**Goal.** Given two sorted subarrays  $a[lo]$  to  $a[mid]$  and  $a[mid+1]$  to  $a[hi]$ , replace with sorted subarray  $a[lo]$  to  $a[hi]$ .



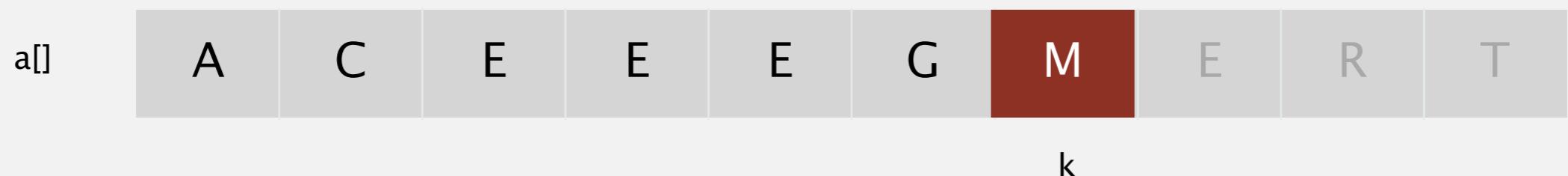
compare minimum in each subarray



# Merging demo

---

**Goal.** Given two sorted subarrays  $a[lo]$  to  $a[mid]$  and  $a[mid+1]$  to  $a[hi]$ , replace with sorted subarray  $a[lo]$  to  $a[hi]$ .



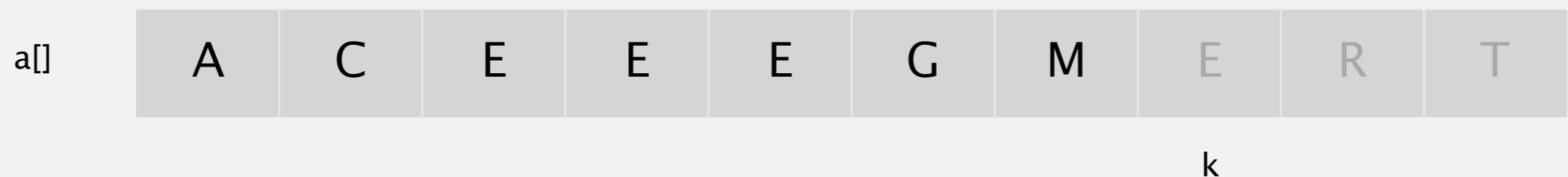
compare minimum in each subarray



# Merging demo

---

**Goal.** Given two sorted subarrays  $a[lo]$  to  $a[mid]$  and  $a[mid+1]$  to  $a[hi]$ , replace with sorted subarray  $a[lo]$  to  $a[hi]$ .



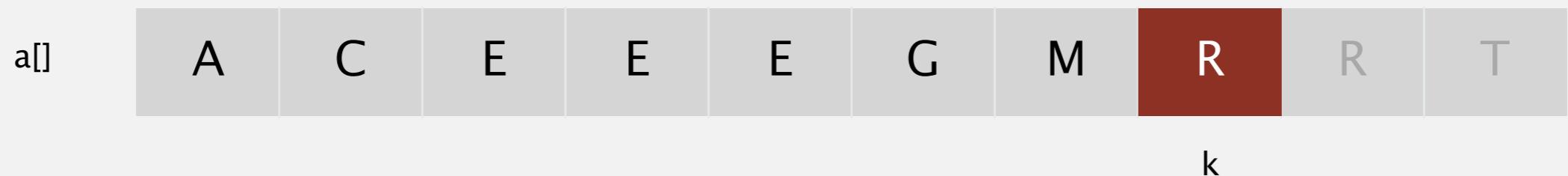
compare minimum in each subarray



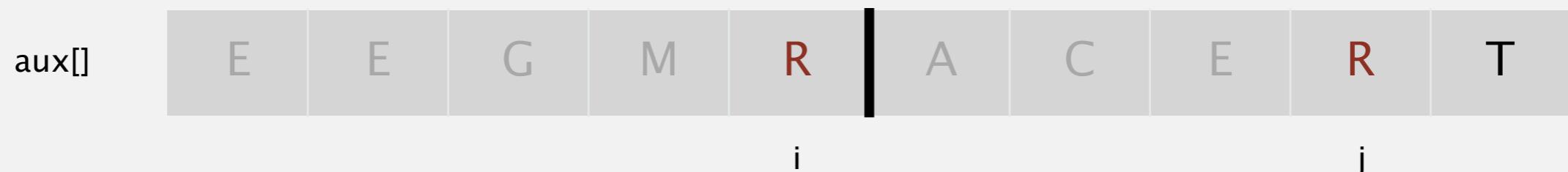
# Merging demo

---

**Goal.** Given two sorted subarrays  $a[lo]$  to  $a[mid]$  and  $a[mid+1]$  to  $a[hi]$ , replace with sorted subarray  $a[lo]$  to  $a[hi]$ .



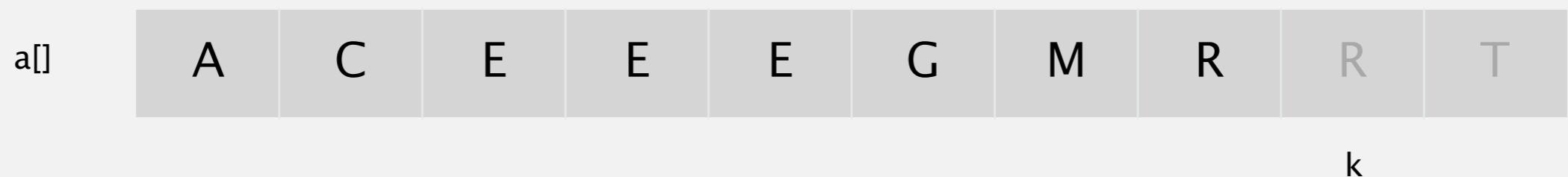
compare minimum in each subarray



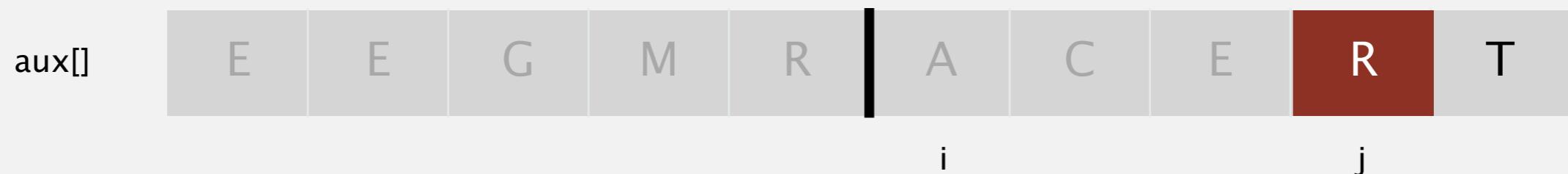
# Merging demo

---

**Goal.** Given two sorted subarrays  $a[lo]$  to  $a[mid]$  and  $a[mid+1]$  to  $a[hi]$ , replace with sorted subarray  $a[lo]$  to  $a[hi]$ .



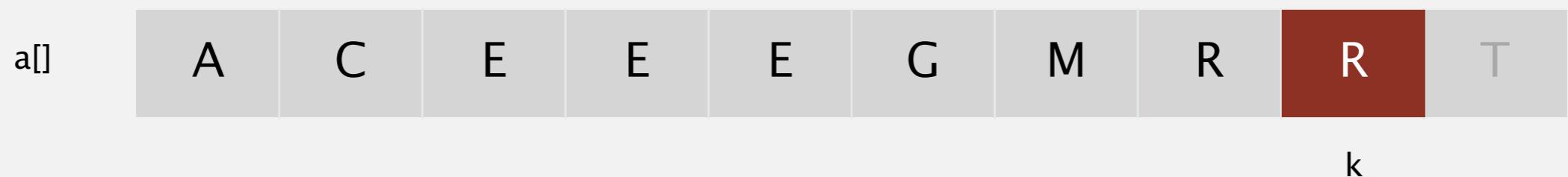
**one subarray exhausted, take from other**



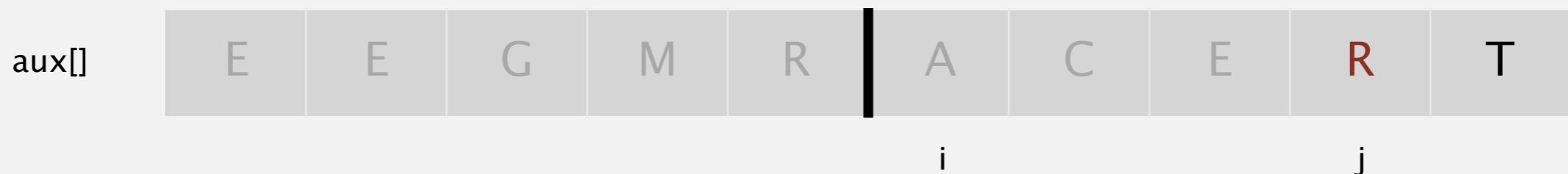
# Merging demo

---

**Goal.** Given two sorted subarrays  $a[lo]$  to  $a[mid]$  and  $a[mid+1]$  to  $a[hi]$ , replace with sorted subarray  $a[lo]$  to  $a[hi]$ .

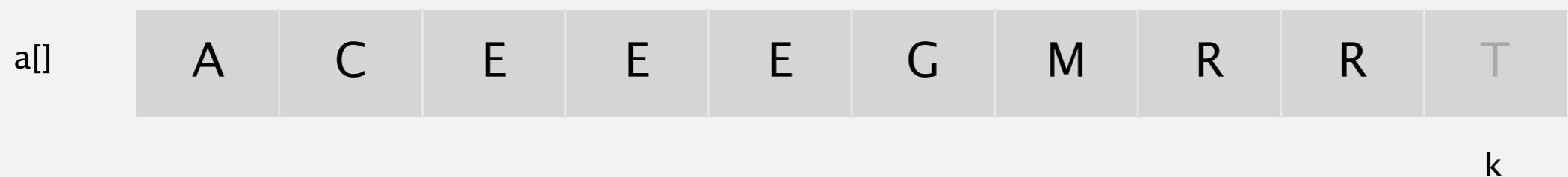


**one subarray exhausted, take from other**

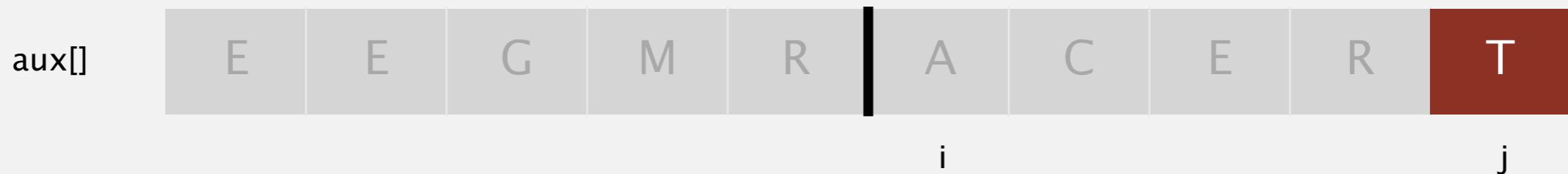


# Merging demo

**Goal.** Given two sorted subarrays  $a[lo]$  to  $a[mid]$  and  $a[mid+1]$  to  $a[hi]$ , replace with sorted subarray  $a[lo]$  to  $a[hi]$ .



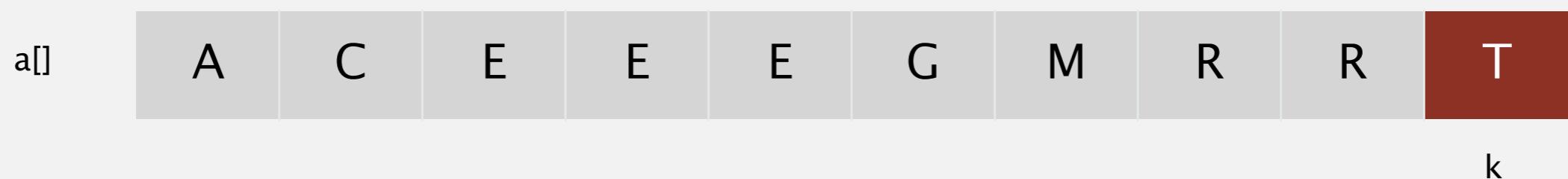
**one subarray exhausted, take from other**



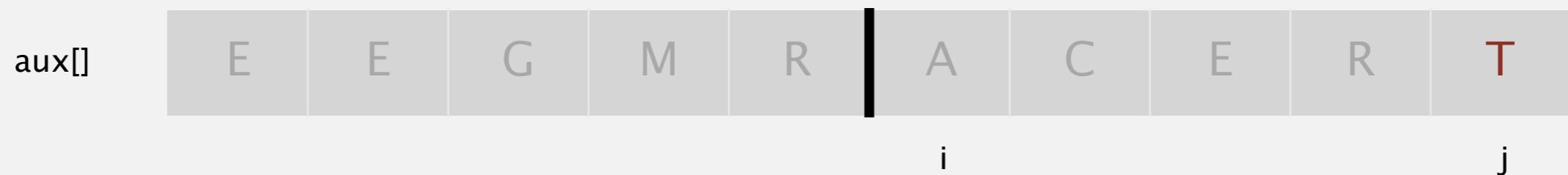
# Merging demo

---

**Goal.** Given two sorted subarrays  $a[lo]$  to  $a[mid]$  and  $a[mid+1]$  to  $a[hi]$ , replace with sorted subarray  $a[lo]$  to  $a[hi]$ .



one subarray exhausted, take from other



# Merging demo

**Goal.** Given two sorted subarrays  $a[lo]$  to  $a[mid]$  and  $a[mid+1]$  to  $a[hi]$ , replace with sorted subarray  $a[lo]$  to  $a[hi]$ .

a[ ]	A	C	E	E	E	G	M	R	R	T
------	---	---	---	---	---	---	---	---	---	---

**both subarrays exhausted, done**

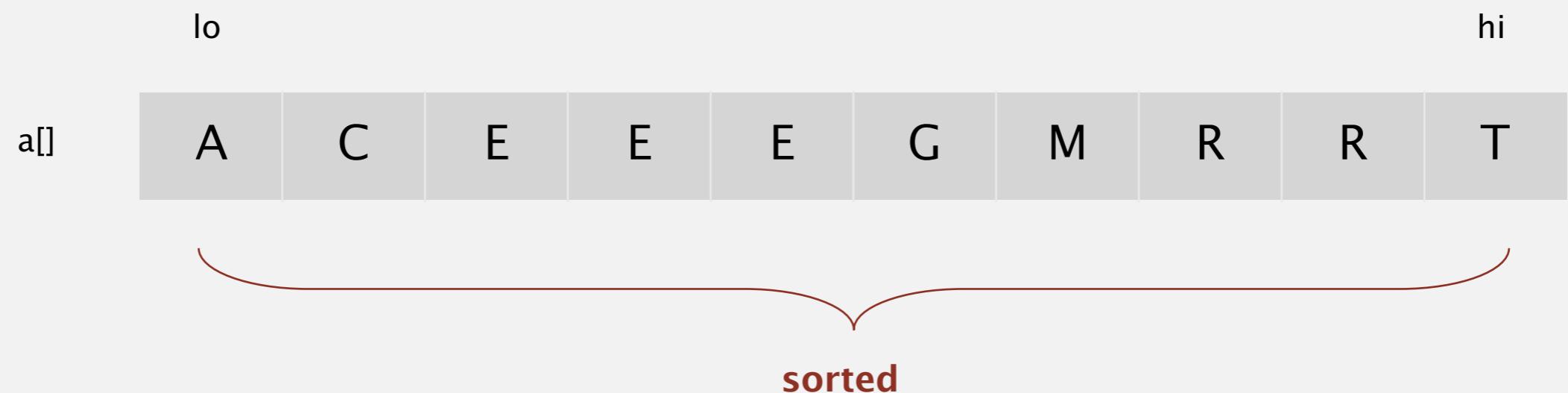
aux[] E E G M R | A C E R T

i j

# Merging demo

---

**Goal.** Given two sorted subarrays  $a[lo]$  to  $a[mid]$  and  $a[mid+1]$  to  $a[hi]$ , replace with sorted subarray  $a[lo]$  to  $a[hi]$ .



# Merging: Java implementation

```
private static void merge(Comparable[] a, Comparable[] aux, int lo, int mid, int hi)
{
    for (int k = lo; k <= hi; k++)
        aux[k] = a[k]; copy

    int i = lo, j = mid+1;
    for (int k = lo; k <= hi; k++)
    {
        if (i > mid) merge
            a[k] = aux[j++];
    }
}
```

What to write here? Pen and Paper, 4 minutes. Use less for comparison.



# Merging: Java implementation

```
private static void merge(Comparable[] a, Comparable[] aux, int lo, int mid, int hi)
{
    for (int k = lo; k <= hi; k++)
        aux[k] = a[k]; copy

    int i = lo, j = mid+1;
    for (int k = lo; k <= hi; k++)
    {
        if      (i > mid)          a[k] = aux[j++]; merge
        else if (j > hi)           a[k] = aux[i++];
        else if (less(aux[j], aux[i])) a[k] = aux[j++];
        else                          a[k] = aux[i++];
    }
} merge
```



# Assertions

---

**Assertion.** Statement to test assumptions about your program.

- Helps detect logic bugs.
- Documents code.

**Java assert statement.** Throws exception unless boolean condition is true.

```
assert isSorted(a, lo, hi);
```

**Can enable or disable at runtime.** ⇒ No cost in production code.

```
% java -ea MyProgram    // enable assertions  
% java -da MyProgram    // disable assertions (default)
```

**Best practices.** Use assertions to check internal invariants;  
assume assertions will be disabled in production code.

# Merging: Java implementation

```
private static void merge(Comparable[] a, Comparable[] aux, int lo, int mid, int hi)
{
    assert isSorted(a, lo, mid);      // precondition: a[lo..mid] sorted
    assert isSorted(a, mid+1, hi);   // precondition: a[mid+1..hi] sorted

    for (int k = lo; k <= hi; k++)                                copy
        aux[k] = a[k];

    int i = lo, j = mid+1;
    for (int k = lo; k <= hi;)                                     merge
    {
        if          (i > mid)           a[k] = aux[j++];
        else if (j > hi)             a[k] = aux[i++];
        else if (less(aux[j], aux[i])) a[k] = aux[j++];
        else                          a[k] = aux[i++];
    }

    assert isSorted(a, lo, hi);      // postcondition: a[lo..hi] sorted
}
```



# Mergesort: Java implementation

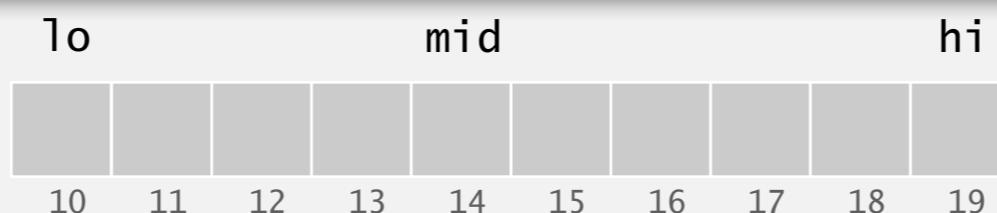
```
public class Merge
{
    private static void merge(Comparable[] a, Comparable[] aux, int lo, int mid,
    int hi)
    { /* as before */ }

    private static void sort(Comparable[] a, Comparable[] aux, int lo, int hi)
    {
        if (hi <= lo) return;
        int mid = lo + (hi - lo) / 2;

        What to write here?
        Pen and Paper, 3 minutes. Hint: recursive.

    }
}

public static void sort(Comparable[] a)
{
    Comparable[] aux = new Comparable[a.length];
    sort(a, aux, 0, a.length - 1);
}
```

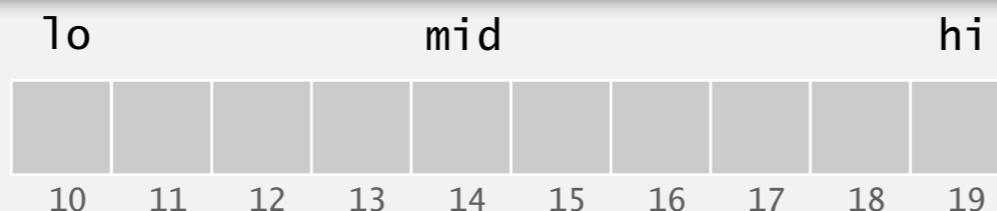


# Mergesort: Java implementation

```
public class Merge
{
    private static void merge(Comparable[] a, Comparable[] aux, int lo, int mid,
    int hi)
    { /* as before */ }

    private static void sort(Comparable[] a, Comparable[] aux, int lo, int hi)
    {
        if (hi <= lo) return;
        int mid = lo + (hi - lo) / 2;
        sort(a, aux, lo, mid);
        sort(a, aux, mid+1, hi);
        merge(a, aux, lo, mid, hi);
    }

    public static void sort(Comparable[] a)
    {
        Comparable[] aux = new Comparable[a.length];
        sort(a, aux, 0, a.length - 1);
    }
}
```



# Mergesort: trace

---

a[]

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
to	M	E	R	G	E	S	0	R	T	E	X	A	M	P	L	E	
hi	E	M	R	G	E	S	0	R	T	E	X	A	M	P	L	E	
merge(a, aux, 0, 0, 1)	E	M	G	R	E	S	0	R	T	E	X	A	M	P	L	E	
merge(a, aux, 2, 2, 3)	E	G	M	R	E	S	0	R	T	E	X	A	M	P	L	E	
merge(a, aux, 0, 1, 3)	E	G	M	R	E	S	0	R	T	E	X	A	M	P	L	E	
merge(a, aux, 4, 4, 5)	E	G	M	R	E	S	0	R	T	E	X	A	M	P	L	E	
merge(a, aux, 6, 6, 7)	E	G	M	R	E	S	0	R	T	E	X	A	M	P	L	E	
merge(a, aux, 4, 5, 7)	E	G	M	R	E	O	R	S	T	E	X	A	M	P	L	E	
merge(a, aux, 0, 3, 7)	E	E	G	M	O	R	R	S	T	E	X	A	M	P	L	E	
merge(a, aux, 8, 8, 9)	E	E	G	M	O	R	R	S	E	T	X	A	M	P	L	E	
merge(a, aux, 10, 10, 11)	E	E	G	M	O	R	R	S	E	T	A	X	M	P	L	E	
merge(a, aux, 8, 9, 11)	E	E	G	M	O	R	R	S	A	E	T	X	M	P	L	E	
merge(a, aux, 12, 12, 13)	E	E	G	M	O	R	R	S	A	E	T	X	M	P	L	E	
merge(a, aux, 14, 14, 15)	E	E	G	M	O	R	R	S	A	E	T	X	M	P	E	L	
merge(a, aux, 12, 13, 15)	E	E	G	M	O	R	R	S	A	E	T	X	E	L	M	P	
merge(a, aux, 8, 11, 15)	E	E	G	M	O	R	R	S	A	E	E	L	M	P	T	X	
merge(a, aux, 0, 7, 15)	A	E	E	E	E	E	G	L	M	M	O	P	R	R	S	T	X

result after recursive call

# Mergesort: animation

## 50 random items

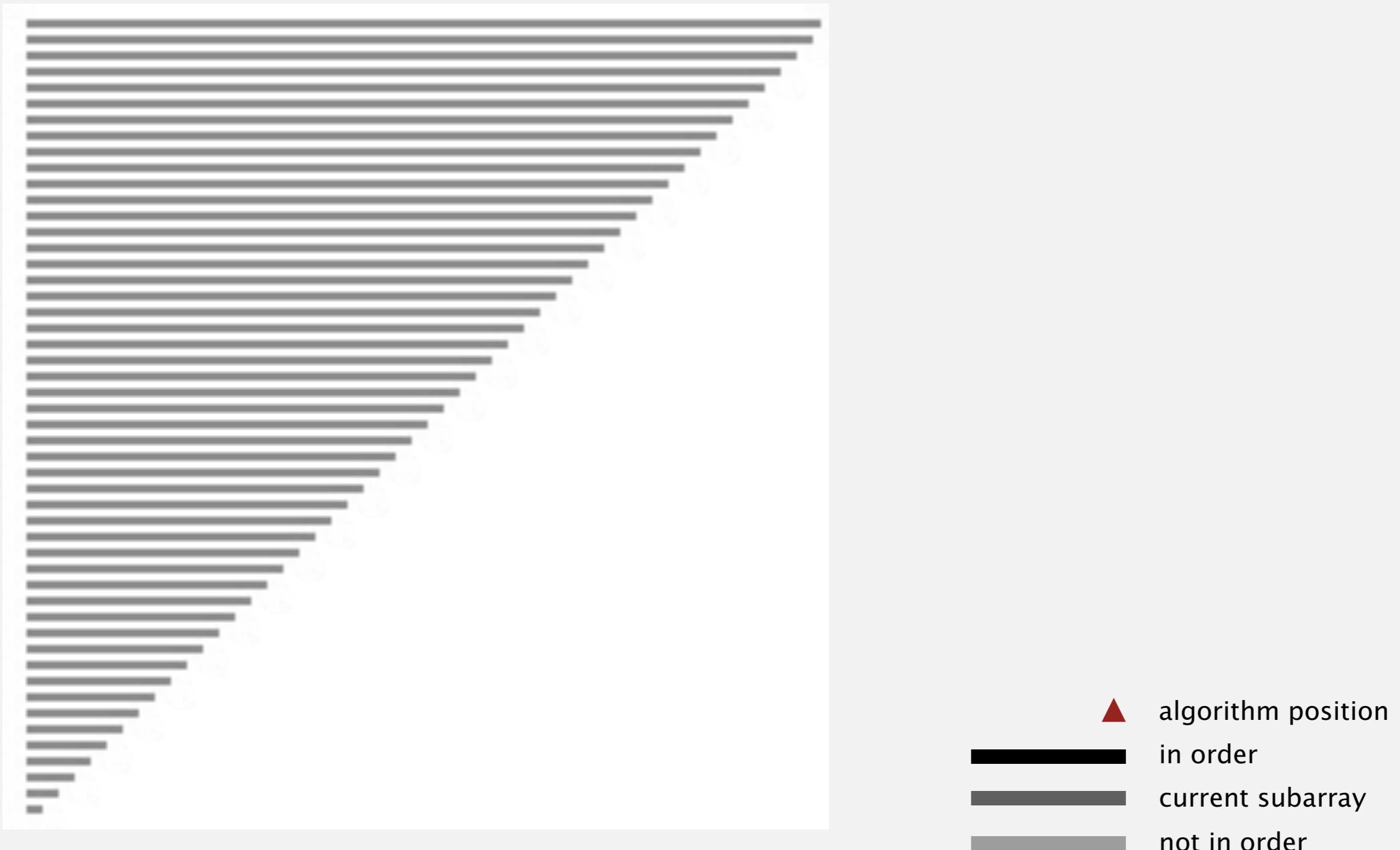


<http://www.sorting-algorithms.com/merge-sort>

# Mergesort: animation

---

50 reverse-sorted items



<http://www.sorting-algorithms.com/merge-sort>

# Mergesort: empirical analysis

---

## Running time estimates:

- Laptop executes  $10^8$  compares/second.
- Supercomputer executes  $10^{12}$  compares/second.

	insertion sort ( $N^2$ )			mergesort ( $N \log N$ )		
computer	thousand	million	billion	thousand	million	billion
home	instant	2.8 hours	317 years	instant	1 second	18 min
super	instant	1 second	1 week	instant	instant	instant

Bottom line. Good algorithms are better than supercomputers.

## Mergesort: number of compares

---

**Proposition.** Mergesort uses  $\leq N \lg N$  compares to sort an array of length  $N$ .

**Pf sketch.** The number of compares  $T(N)$  to mergesort an array of length  $N$  satisfies the recurrence:

$$T(N) \leq T(\lceil N/2 \rceil) + T(\lfloor N/2 \rfloor) + N \quad \text{for } N > 1, \text{ with } T(1) = 0.$$

↑  
left half      ↑      right half      ↑  
merge

We solve the recurrence when  $N$  is a power of 2:



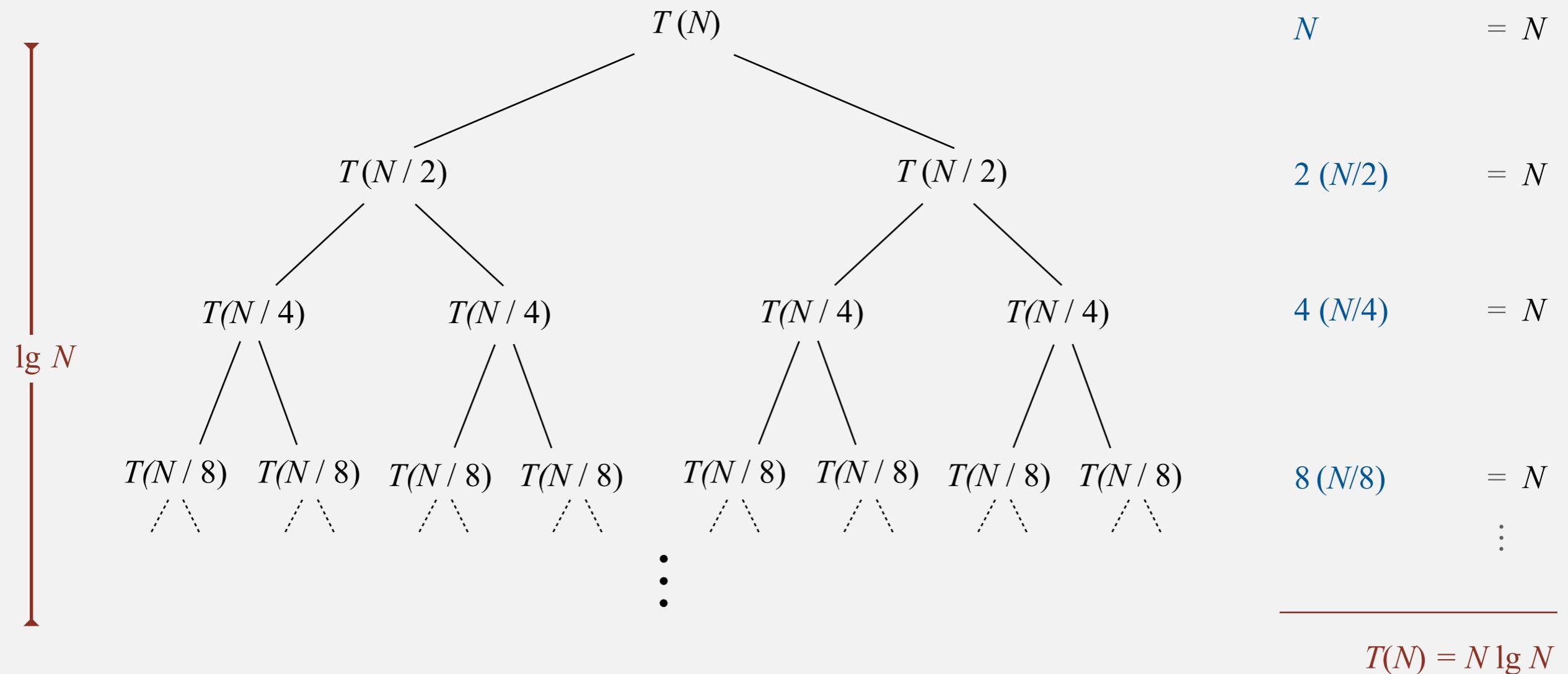
result holds for all  $N$   
(analysis cleaner in this case)

$$T(N) = 2 T(N/2) + N, \text{ for } N > 1, \text{ with } T(1) = 0.$$

# Divide-and-conquer recurrence: proof by picture

**Proposition.** If  $T(N)$  satisfies  $T(N) = 2 T(N/2) + N$  for  $N > 1$ , with  $T(1) = 0$ , then  $T(N) = N \lg N$ .

Pf 1. [assuming  $N$  is a power of 2]



# Mergesort: practical improvements

---

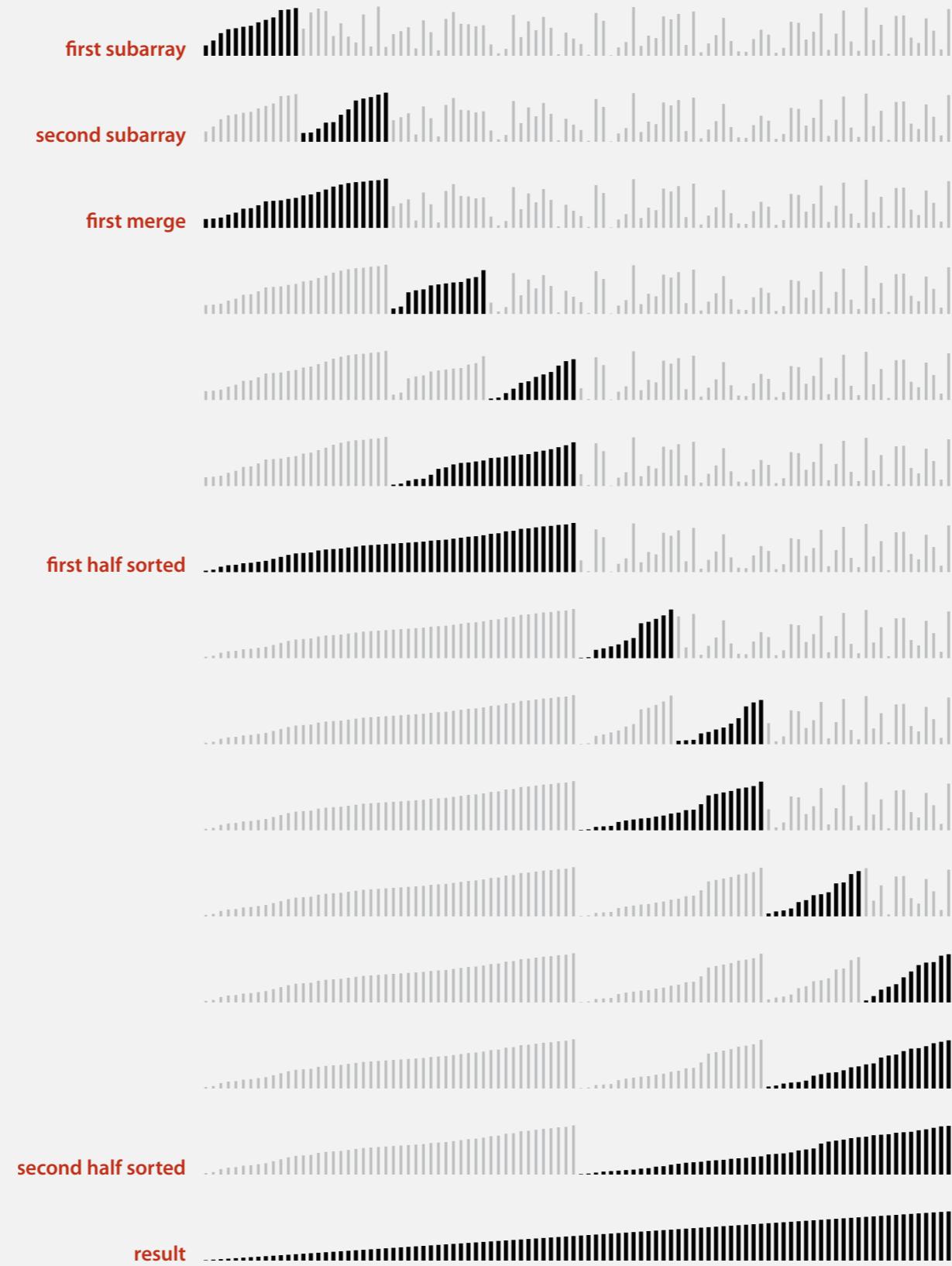
## Use insertion sort for small subarrays.

- Mergesort has too much overhead for tiny subarrays.
- Cutoff to insertion sort for  $\approx 10$  items.

```
private static void sort(Comparable[] a, Comparable[] aux, int lo, int hi)
{
    if (hi <= lo + CUTOFF - 1)
    {
        Insertion.sort(a, lo, hi);
        return;
    }
    int mid = lo + (hi - lo) / 2;
    sort (a, aux, lo, mid);
    sort (a, aux, mid+1, hi);
    merge(a, aux, lo, mid, hi);
}
```

# Mergesort with cutoff to insertion sort: visualization

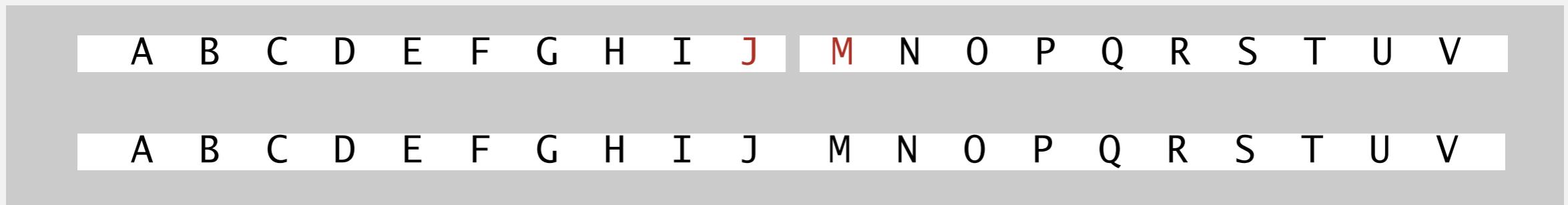
---



# Mergesort: practical improvements

Stop if already sorted.

- Is largest item in first half  $\leq$  smallest item in second half?
- Helps for partially-ordered arrays.



```
private static void sort(Comparable[] a, Comparable[] aux, int lo, int hi)
{
    if (hi <= lo) return;
    int mid = lo + (hi - lo) / 2;
    sort(a, aux, lo, mid);
    sort(a, aux, mid+1, hi);
```

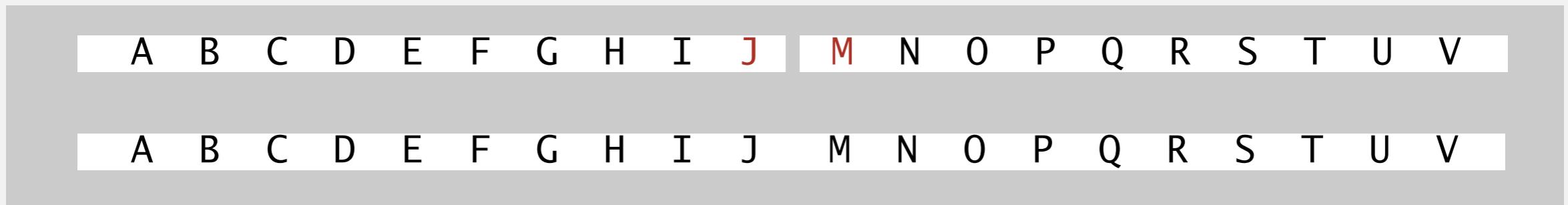
What to write here? Pen and Paper, 2 minutes.

```
    merge(a, aux, lo, mid, hi);
}
```

# Mergesort: practical improvements

Stop if already sorted.

- Is largest item in first half  $\leq$  smallest item in second half?
- Helps for partially-ordered arrays.



```
private static void sort(Comparable[] a, Comparable[] aux, int lo, int hi)
{
    if (hi <= lo) return;
    int mid = lo + (hi - lo) / 2;
    sort(a, aux, lo, mid);
    sort(a, aux, mid+1, hi);

    if (!less(a[mid+1], a[mid])) return;

    merge(a, aux, lo, mid, hi);
}
```

# Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

## 2.2 MERGESORT

---

- ▶ *mergesort*
- ▶ *bottom-up mergesort*
- ▶ *comparators*
- ▶ *stability*

# Bottom-up mergesort

---

## Basic plan.

- Pass through array, merging subarrays of size 1.
- Repeat for subarrays of size 2, 4, 8, ....

	a[i]																
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
<b>sz = 1</b>	M	E	R	G	E	S	O	R	T	E	X	A	M	P	L	E	
merge(a, aux, 0, 0, 1)	E	M	R	G	E	S	O	R	T	E	X	A	M	P	L	E	
merge(a, aux, 2, 2, 3)	E	M	G	R	E	S	O	R	T	E	X	A	M	P	L	E	
merge(a, aux, 4, 4, 5)	E	M	G	R	E	S	O	R	T	E	X	A	M	P	L	E	
merge(a, aux, 6, 6, 7)	E	M	G	R	E	S	O	R	T	E	X	A	M	P	L	E	
merge(a, aux, 8, 8, 9)	E	M	G	R	E	S	O	R	E	T	X	A	M	P	L	E	
merge(a, aux, 10, 10, 11)	E	M	G	R	E	S	O	R	E	T	A	X	M	P	L	E	
merge(a, aux, 12, 12, 13)	E	M	G	R	E	S	O	R	E	T	A	X	M	P	L	E	
merge(a, aux, 14, 14, 15)	E	M	G	R	E	S	O	R	E	T	A	X	M	P	E	L	
<b>sz = 2</b>	E	G	M	R	E	S	O	R	E	T	A	X	M	P	E	L	
merge(a, aux, 0, 1, 3)	E	G	M	R	E	O	R	S	E	T	A	X	M	P	E	L	
merge(a, aux, 4, 5, 7)	E	G	M	R	E	O	R	S	A	E	T	X	M	P	E	L	
merge(a, aux, 8, 9, 11)	E	G	M	R	E	O	R	S	A	E	T	X	M	P	E	L	
merge(a, aux, 12, 13, 15)	E	G	M	R	E	O	R	S	A	E	T	X	E	L	M	P	
<b>sz = 4</b>	E	E	G	M	O	R	R	S	A	E	T	X	E	L	M	P	
merge(a, aux, 0, 3, 7)	E	E	G	M	O	R	R	S	A	E	E	L	M	P	T	X	
merge(a, aux, 8, 11, 15)	A	E	E	E	E	E	G	L	M	M	O	P	R	R	S	T	X
<b>sz = 8</b>																	
merge(a, aux, 0, 7, 15)																	

## Bottom-up mergesort: Java implementation

```
public class MergeBU
{
    private static void merge(Comparable[] a, Comparable[] aux, int
lo, int mid, int hi)
    { /* as before */ }

    public static void sort(Comparable[] a)
    {
        int N = a.length;
        Comparable[] aux = new Comparable[N];
        for (int sz = 1; sz < N; sz = sz+sz)
            for (int lo = 0; merge(a, aux,
                }                                What to write here? Pen and Paper, 4 minutes.
        }
    }
}
```

Bottom line. Simple and non-recursive version of mergesort.

## Bottom-up mergesort: Java implementation

---

```
public class MergeBU
{
    private static void merge(Comparable[] a, Comparable[] aux, int
lo, int mid, int hi)
    { /* as before */ }

    public static void sort(Comparable[] a)
    {
        int N = a.length;
        Comparable[] aux = new Comparable[N];
        for (int sz = 1; sz < N; sz = sz+sz)
            for (int lo = 0; lo < N-sz; lo += sz+sz)
                merge(a, aux,
    }
}
```

Bottom line. Simple and non-recursive version of mergesort.

## Bottom-up mergesort: Java implementation

---

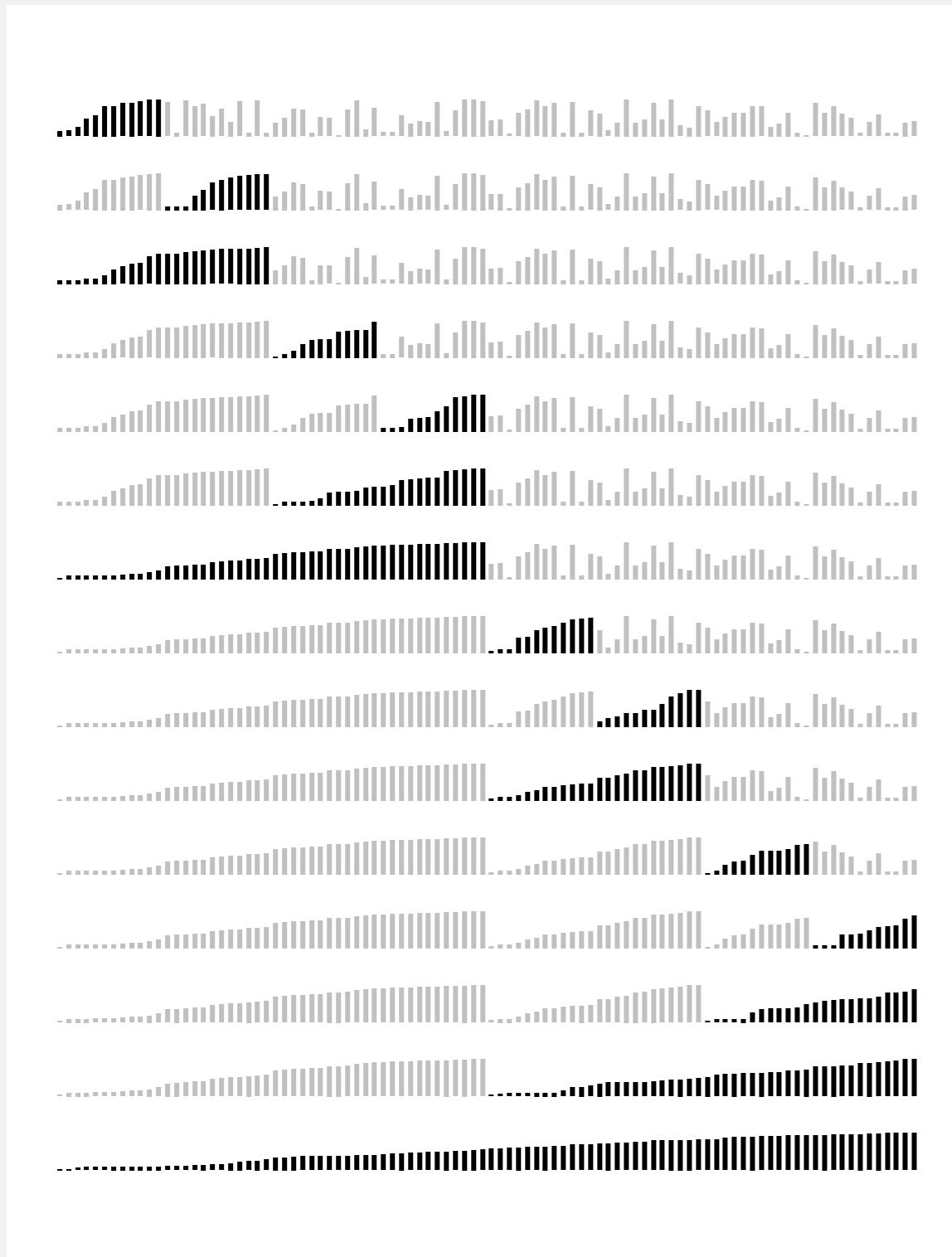
```
public class MergeBU
{
    private static void merge(Comparable[] a, Comparable[] aux, int
lo, int mid, int hi)
    { /* as before */ }

    public static void sort(Comparable[] a)
    {
        int N = a.length;
        Comparable[] aux = new Comparable[N];
        for (int sz = 1; sz < N; sz = sz+sz)
            for (int lo = 0; lo < N-sz; lo += sz+sz)
                merge(a, aux, lo, lo+sz-1, Math.min(lo+sz+sz-1, N-1));
    }
}
```

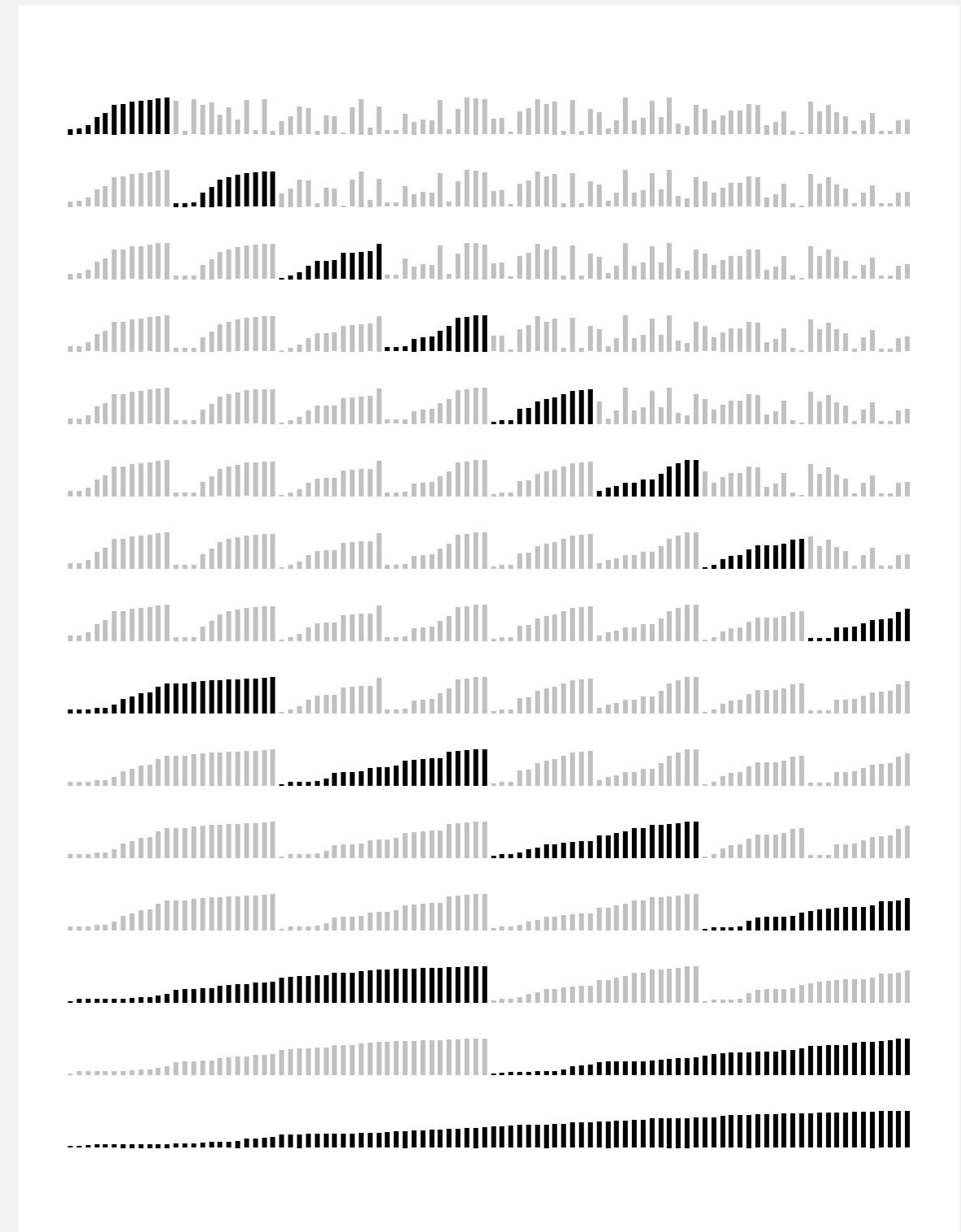
Bottom line. Simple and non-recursive version of mergesort.

# Mergesort: visualizations

---



**top-down mergesort (cutoff = 12)**



**bottom-up mergesort (cutoff = 12)**

# Natural mergesort

---

Idea. Exploit pre-existing order by identifying naturally-occurring runs.

input

1	5	10	16	3	4	23	9	13	2	7	8	12	14
---	---	----	----	---	---	----	---	----	---	---	---	----	----

first run

1	5	10	16	3	4	23	9	13	2	7	8	12	14
---	---	----	----	---	---	----	---	----	---	---	---	----	----

second run

1	5	10	16	3	4	23	9	13	2	7	8	12	14
---	---	----	----	---	---	----	---	----	---	---	---	----	----

merge two runs

1	3	4	5	10	16	23	9	13	2	7	8	12	14
---	---	---	---	----	----	----	---	----	---	---	---	----	----

Tradeoff. Fewer passes vs. extra compares per pass to identify runs.

# Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

## 2.2 MERGESORT

---

- ▶ *mergesort*
- ▶ *bottom-up mergesort*
- ▶ *comparators*
- ▶ *stability*

# Sort countries by gold medals

---

NOC	Gold	Silver	Bronze	Total
United States (USA)	46	29	29	104
China (CHN)§	38	28	22	88
Great Britain (GBR)*	29	17	19	65
Russia (RUS)§	24	25	32	81
South Korea (KOR)	13	8	7	28
Germany (GER)	11	19	14	44
France (FRA)	11	11	12	34
Italy (ITA)	8	9	11	28
Hungary (HUN)§	8	4	6	18
Australia (AUS)	7	16	12	35

# Sort countries by total medals

---

NOC	Gold	Silver	Bronze	Total
United States (USA)	46	29	29	104
China (CHN)§	38	28	22	88
Russia (RUS)§	24	25	32	81
Great Britain (GBR)*	29	17	19	65
Germany (GER)	11	19	14	44
Japan (JPN)	7	14	17	38
Australia (AUS)	7	16	12	35
France (FRA)	11	11	12	34
South Korea (KOR)	13	8	7	28
Italy (ITA)	8	9	11	28

# Sort music library by artist

The image shows a music library interface with a list of songs and an album preview for Bruce Springsteen's 'Born In The U.S.A.'

**Album Preview:** The top half of the screen displays the album cover for 'Born In The U.S.A.' by Bruce Springsteen. The cover features a man in a dark jacket and jeans standing in front of a red and white striped wall. The title 'BORN IN THE U.S.A./BRUCE SPRINGSTEEN' is at the top, and 'Born In The U.S.A.' and 'Bruce Springsteen' are in the center.

**Song List:** Below the album preview is a table listing 38 songs. The columns are labeled: #, Name, Artist, Time, and Album. The table includes the following data:

#	Name	Artist	Time	Album
12	<input checked="" type="checkbox"/> Let It Be	The Beatles	4:03	Let It Be
13	<input checked="" type="checkbox"/> Take My Breath Away	BERLIN	4:13	Top Gun - Soundtrack
14	<input checked="" type="checkbox"/> Circle Of Friends	Better Than Ezra	3:27	Empire Records
15	<input checked="" type="checkbox"/> Dancing With Myself	Billy Idol	4:43	Don't Stop
16	<input checked="" type="checkbox"/> Rebel Yell	Billy Idol	4:49	Rebel Yell
17	<input checked="" type="checkbox"/> Piano Man	Billy Joel	5:36	Greatest Hits Vol. 1
18	<input checked="" type="checkbox"/> Pressure	Billy Joel	3:16	Greatest Hits, Vol. II (1978 - 1985) (Disc 2)
19	<input checked="" type="checkbox"/> The Longest Time	Billy Joel	3:36	Greatest Hits, Vol. II (1978 - 1985) (Disc 2)
20	<input checked="" type="checkbox"/> Atomic	Blondie	3:50	Atomic: The Very Best Of Blondie
21	<input checked="" type="checkbox"/> Sunday Girl	Blondie	3:15	Atomic: The Very Best Of Blondie
22	<input checked="" type="checkbox"/> Call Me	Blondie	3:33	Atomic: The Very Best Of Blondie
23	<input checked="" type="checkbox"/> Dreaming	Blondie	3:06	Atomic: The Very Best Of Blondie
24	<input checked="" type="checkbox"/> Hurricane	Bob Dylan	8:32	Desire
25	<input checked="" type="checkbox"/> The Times They Are A-Changin'	Bob Dylan	3:17	Greatest Hits
26	<input checked="" type="checkbox"/> Livin' On A Prayer	Bon Jovi	4:11	Cross Road
27	<input checked="" type="checkbox"/> Beds Of Roses	Bon Jovi	6:35	Cross Road
28	<input checked="" type="checkbox"/> Runaway	Bon Jovi	3:53	Cross Road
29	<input checked="" type="checkbox"/> Rasputin (Extended Mix)	Boney M	5:50	Greatest Hits
30	<input checked="" type="checkbox"/> Have You Ever Seen The Rain	Bonnie Tyler	4:10	Faster Than The Speed Of Night
31	<input checked="" type="checkbox"/> Total Eclipse Of The Heart	Bonnie Tyler	7:02	Faster Than The Speed Of Night
32	<input checked="" type="checkbox"/> Straight From The Heart	Bonnie Tyler	3:41	Faster Than The Speed Of Night
33	<input checked="" type="checkbox"/> Holding Out For A Hero	Bonnie Tyler	5:49	Meat Loaf And Friends
34	<input checked="" type="checkbox"/> Dancing In The Dark	Bruce Springsteen	4:05	Born In The U.S.A.
35	<input checked="" type="checkbox"/> Thunder Road	Bruce Springsteen	4:51	Born To Run
36	<input checked="" type="checkbox"/> Born To Run	Bruce Springsteen	4:30	Born To Run
37	<input checked="" type="checkbox"/> Jungleland	Bruce Springsteen	9:34	Born To Run
38	<input checked="" type="checkbox"/> Turn! Turn! Turn! (To Everything)	The Byrds	3:57	Forrest Gump The Soundtrack (Disc 2)

# Sort music library by song name

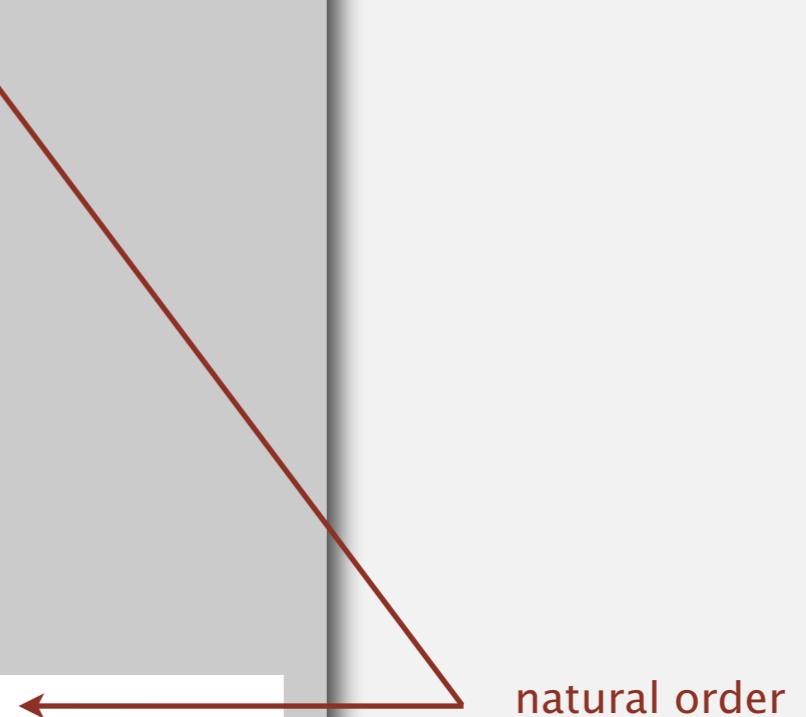
	Name	Artist	Time	Album
1	Alive	Pearl Jam	5:41	Ten
2	All Over The World	Pixies	5:27	Bossanova
3	All Through The Night	Cyndi Lauper	4:30	She's So Unusual
4	Allison Road	Gin Blossoms	3:19	New Miserable Experience
5	Ama, Ama, Ama Y Ensancha El ...	Extremoduro	2:34	Deltoya (1992)
6	And We Danced	Hooters	3:50	Nervous Night
7	As I Lay Me Down	Sophie B. Hawkins	4:09	Whaler
8	Atomic	Blondie	3:50	Atomic: The Very Best Of Blondie
9	Automatic Lover	Jay-Jay Johanson	4:19	Antenna
10	Baba O'Riley	The Who	5:01	Who's Better, Who's Best
11	Beautiful Life	Ace Of Base	3:40	The Bridge
12	<input checked="" type="checkbox"/> Beds Of Roses	Bon Jovi	6:35	Cross Road
13	Black	Pearl Jam	5:44	Ten
14	Bleed American	Jimmy Eat World	3:04	Bleed American
15	Borderline	Madonna	4:00	The Immaculate Collection
16	Born To Run	Bruce Springsteen	4:30	Born To Run
17	Both Sides Of The Story	Phil Collins	6:43	Both Sides
18	Bouncing Around The Room	Phish	4:09	A Live One (Disc 1)
19	Boys Don't Cry	The Cure	2:35	Staring At The Sea: The Singles 1979–1985
20	Brat	Green Day	1:43	Insomniac
21	Breakdown	Deerheart	3:40	Deerheart
22	Bring Me To Life (Kevin Roen Mix)	Evanescence Vs. Pa...	9:48	
23	Californication	Red Hot Chili Pepp...	1:40	
24	Call Me	Blondie	3:33	Atomic: The Very Best Of Blondie
25	Can't Get You Out Of My Head	Kylie Minogue	3:50	Fever
26	Celebration	Kool & The Gang	3:45	Time Life Music Sounds Of The Seventies – C
27	Chaiwala Chaiwala	Culbhawukde Singh	5:11	Bombay Dreams

# Comparable interface: review

Comparable interface: sort using a type's **natural order**.

```
public class Date implements Comparable<Date>
{
    private final int month, day, year;

    public Date(int m, int d, int y)
    {
        month = m;
        day   = d;
        year  = y;
    }
    ...
    public int compareTo(Date that)
    {
        if (this.year < that.year) return -1;
        if (this.year > that.year) return +1;
        if (this.month < that.month) return -1;
        if (this.month > that.month) return +1;
        if (this.day   < that.day)  return -1;
        if (this.day   > that.day)  return +1;
        return 0;
    }
}
```



natural order

# Comparator interface

Comparator interface: sort using an alternate order.

```
public interface Comparator<Key>
```

```
    int compare(Key v, Key w)
```

*compare keys v and w*

Required property. Must be a total order.

string order	example
<b>natural order</b>	Now is the time
<b>case insensitive</b>	is Now the time
<b>Spanish language</b>	café cafetero cuarto churro nube ñoño
<b>British phone book</b>	McKinley Mackintosh



pre-1994 order for  
digraphs ch and ll and rr

# Comparator interface: system sort

To use with Java system sort:

- Create Comparator object.
- Pass as second argument to `Arrays.sort()`.

```
String[] a;           uses natural order
...
Arrays.sort(a);      uses alternate order defined by
...
Arrays.sort(a, String.CASE_INSENSITIVE_ORDER);
...
Arrays.sort(a, Collator.getInstance(new Locale("es")));
...
Arrays.sort(a, new BritishPhoneBookOrder());
...
```

The diagram consists of several lines of Java code within a light gray box. Red arrows point from specific parts of the code to text descriptions to its right. One arrow points from the first line 'String[] a;' to the text 'uses natural order'. Another arrow points from the line 'Arrays.sort(a, String.CASE\_INSENSITIVE\_ORDER);' to the text 'uses alternate order defined by Comparator<String> object'. A third arrow points from the line 'Arrays.sort(a, new BritishPhoneBookOrder());' to the same text 'uses alternate order defined by Comparator<String> object'.

**Bottom line.** Decouples the definition of the data type from the definition of what it means to compare two objects of that type.

# Comparator interface: using with our sorting libraries

---

To support comparators in our sort implementations:

- Use Object instead of Comparable.
- Pass Comparator to sort() and less() and use it in less().

**insertion sort using a Comparator**

```
public static void sort(Object[] a, Comparator comparator)
{
    int N = a.length;
    for (int i = 0; i < N; i++)
        for (int j = i; j > 0 && less(comparator, a[j], a[j-1]); j--)
            exch(a, j, j-1);
}

private static boolean less(Comparator c, Object v, Object w)
{ return c.compare(v, w) < 0; }

private static void exch(Object[] a, int i, int j)
{ Object swap = a[i]; a[i] = a[j]; a[j] = swap; }
```

# Comparator interface: implementing

---

## To implement a comparator:

- Define a (nested) class that implements the Comparator interface.
- Implement the compare() method.

```
public class Student
{
    private final String name;
    private final int section;
    ...

    public static class ByName implements Comparator<Student>
    {
        public int compare(Student v, Student w)
        { return v.name.compareTo(w.name); }
    }

    public static class BySection implements Comparator<Student>
    {
        public int compare(Student v, Student w)
        { return v.section - w.section; }
    }
}
```

# Comparator interface: implementing

---

To implement a comparator:

- Define a (nested) class that implements the Comparator interface.
- Implement the compare() method.

`Arrays.sort(a, new Student.ByName());`

Andrews	3	A	664-480-0023	097 Little
Battle	4	C	874-088-1212	121 Whitman
Chen	3	A	991-878-4944	308 Blair
Fox	3	A	884-232-5341	11 Dickinson
Furia	1	A	766-093-9873	101 Brown
Gazsi	4	B	766-093-9873	101 Brown
Kanaga	3	B	898-122-9643	22 Brown
Rohde	2	A	232-343-5555	343 Forbes

`Arrays.sort(a, new Student.BySection());`

Furia	1	A	766-093-9873	101 Brown
Rohde	2	A	232-343-5555	343 Forbes
Andrews	3	A	664-480-0023	097 Little
Chen	3	A	991-878-4944	308 Blair
Fox	3	A	884-232-5341	11 Dickinson
Kanaga	3	B	898-122-9643	22 Brown
Battle	4	C	874-088-1212	121 Whitman
Gazsi	4	B	766-093-9873	101 Brown

# Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

## 2.2 MERGESORT

---

- ▶ *mergesort*
- ▶ *bottom-up mergesort*
- ▶ *comparators*
- ▶ ***stability***

# Stability

---

A typical application. First, sort by name; **then** sort by section.

`Selection.sort(a, new Student.ByName());`

Andrews	3	A	664-480-0023	097 Little
Battle	4	C	874-088-1212	121 Whitman
Chen	3	A	991-878-4944	308 Blair
Fox	3	A	884-232-5341	11 Dickinson
Furia	1	A	766-093-9873	101 Brown
Gazsi	4	B	766-093-9873	101 Brown
Kanaga	3	B	898-122-9643	22 Brown
Rohde	2	A	232-343-5555	343 Forbes

`Selection.sort(a, new Student.BySection());`

Furia	1	A	766-093-9873	101 Brown
Rohde	2	A	232-343-5555	343 Forbes
Chen	3	A	991-878-4944	308 Blair
Fox	3	A	884-232-5341	11 Dickinson
Andrews	3	A	664-480-0023	097 Little
Kanaga	3	B	898-122-9643	22 Brown
Gazsi	4	B	766-093-9873	101 Brown
Battle	4	C	874-088-1212	121 Whitman

@#%&@! Students in section 3 no longer sorted by name.

A **stable** sort preserves the relative order of items with equal keys.

# Stability

---

Q. Which sorts are stable?

sorted by time	sorted by location	Stable?
Chicago 09:00:00	Chicago 09:25:52	
Phoenix 09:00:03	Chicago 09:03:13	
Houston 09:00:13	Chicago 09:21:05	
Chicago 09:00:59	Chicago 09:19:46	
Houston 09:01:10	Chicago 09:19:32	
Chicago 09:03:13	Chicago 09:00:00	
Seattle 09:10:11	Chicago 09:35:21	
Seattle 09:10:25	Chicago 09:00:59	
Phoenix 09:14:25	Houston 09:01:10	
Chicago 09:19:32	Houston 09:00:13	
Chicago 09:19:46	Phoenix 09:37:44	
Chicago 09:21:05	Phoenix 09:00:03	
Seattle 09:22:43	Phoenix 09:14:25	
Seattle 09:22:54	Seattle 09:10:25	
Chicago 09:25:52	Seattle 09:36:14	
Chicago 09:35:21	Seattle 09:22:43	
Seattle 09:36:14	Seattle 09:10:11	
Phoenix 09:37:44	Seattle 09:22:54	

# Stability

Q. Which sorts are stable?

**sorted by time**

Chicago	09:00:00
Phoenix	09:00:03
Houston	09:00:13
Chicago	09:00:59
Houston	09:01:10
Chicago	09:03:13
Seattle	09:10:11
Seattle	09:10:25
Phoenix	09:14:25
Chicago	09:19:32
Chicago	09:19:46
Chicago	09:21:05
Seattle	09:22:43
Seattle	09:22:54
Chicago	09:25:52
Chicago	09:35:21
Seattle	09:36:14
Phoenix	09:37:44

**sorted by location (not stable)**

Chicago	09:25:52
Chicago	09:03:13
Chicago	09:21:05
Chicago	09:19:46
Chicago	09:19:32
Chicago	09:00:00
Chicago	09:35:21
Chicago	09:00:59
Houston	09:01:10
Houston	09:00:13
Phoenix	09:37:44
Phoenix	09:00:03
Phoenix	09:14:25
Seattle	09:10:25
Seattle	09:36:14
Seattle	09:22:43
Seattle	09:10:11
Seattle	09:22:54

*no  
longer  
sorted  
by time*

# Stability

Q. Which sorts are stable?

sorted by time

Chicago	09:00:00
Phoenix	09:00:03
Houston	09:00:13
Chicago	09:00:59
Houston	09:01:10
Chicago	09:03:13
Seattle	09:10:11
Seattle	09:10:25
Phoenix	09:14:25
Chicago	09:19:32
Chicago	09:19:46
Chicago	09:21:05
Seattle	09:22:43
Seattle	09:22:54
Chicago	09:25:52
Chicago	09:35:21
Seattle	09:36:14
Phoenix	09:37:44

sorted by location (not stable)

Chicago	09:25:52
Chicago	09:03:13
Chicago	09:21:05
Chicago	09:19:46
Chicago	09:19:32
Chicago	09:00:00
Chicago	09:35:21
Chicago	09:00:59
Houston	09:01:10
Houston	09:00:13
Phoenix	09:37:44
Phoenix	09:00:03
Phoenix	09:14:25
Seattle	09:10:25
Seattle	09:36:14
Seattle	09:22:43
Seattle	09:10:11
Seattle	09:22:54

sorted by location **Stable?**

Chicago	09:00:00
Chicago	09:00:59
Chicago	09:03:13
Chicago	09:19:32
Chicago	09:19:46
Chicago	09:21:05
Chicago	09:25:52
Chicago	09:35:21
Houston	09:00:13
Houston	09:01:10
Phoenix	09:00:03
Phoenix	09:14:25
Phoenix	09:37:44
Seattle	09:10:11
Seattle	09:10:25
Seattle	09:22:43
Seattle	09:22:54
Seattle	09:36:14

no  
longer  
sorted  
by time

# Stability

---

Q. Which sorts are stable?

A. Need to check algorithm (and implementation).

sorted by time		sorted by location (not stable)	sorted by location (stable)
Chicago	09:00:00	Chicago 09:25:52	Chicago 09:00:00
Phoenix	09:00:03	Chicago 09:03:13	Chicago 09:00:59
Houston	09:00:13	Chicago 09:21:05	Chicago 09:03:13
Chicago	09:00:59	Chicago 09:19:46	Chicago 09:19:32
Houston	09:01:10	Chicago 09:19:32	Chicago 09:19:46
Chicago	09:03:13	Chicago 09:00:00	Chicago 09:21:05
Seattle	09:10:11	Chicago 09:35:21	Chicago 09:25:52
Seattle	09:10:25	Chicago 09:00:59	Chicago 09:35:21
Phoenix	09:14:25	Houston 09:01:10	Houston 09:00:13
Chicago	09:19:32	Houston 09:00:13	Houston 09:01:10
Chicago	09:19:46	Phoenix 09:37:44	Phoenix 09:00:03
Chicago	09:21:05	Phoenix 09:00:03	Phoenix 09:14:25
Seattle	09:22:43	Phoenix 09:14:25	Phoenix 09:37:44
Seattle	09:22:54	Seattle 09:10:25	Seattle 09:10:11
Chicago	09:25:52	Seattle 09:36:14	Seattle 09:10:25
Chicago	09:35:21	Seattle 09:22:43	Seattle 09:22:43
Seattle	09:36:14	Seattle 09:10:11	Seattle 09:22:54
Phoenix	09:37:44	Seattle 09:22:54	Seattle 09:36:14

# Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

## INSERTION SORT

---

► *Is it stable?*

```
public class Insertion
{
    public static void sort(Comparable[] a)
    {
        int N = a.length;
        for (int i = 0; i < N; i++)
            for (int j = i; j > 0 && less(a[j], a[j-1]); j--)
                exch(a, j, j-1);
    }
}
```

## Stability: insertion sort

Proposition. Insertion sort is **stable**.

```
public class Insertion
{
    public static void sort(Comparable[] a)
    {
        int N = a.length;
        for (int i = 0; i < N; i++)
            for (int j = i; j > 0 && less(a[j], a[j-1]); j--)
                exch(a, j, j-1);
    }
}
```

i	j	0	1	2	3	4
0	0	B <sub>1</sub>	A <sub>1</sub>	A <sub>2</sub>	A <sub>3</sub>	B <sub>2</sub>
1	0	A <sub>1</sub>	B <sub>1</sub>	A <sub>2</sub>	A <sub>3</sub>	B <sub>2</sub>
2	1	A <sub>1</sub>	A <sub>2</sub>	B <sub>1</sub>	A <sub>3</sub>	B <sub>2</sub>
3	2	A <sub>1</sub>	A <sub>2</sub>	A <sub>3</sub>	B <sub>1</sub>	B <sub>2</sub>
4	4	A <sub>1</sub>	A <sub>2</sub>	A <sub>3</sub>	B <sub>1</sub>	B <sub>2</sub>
		A <sub>1</sub>	A <sub>2</sub>	A <sub>3</sub>	B <sub>1</sub>	B <sub>2</sub>

Pf. Equal items never move past each other.

# Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

## SELECTION SORT

- *Is it stable?*

```
public class Selection
{
    public static void sort(Comparable[] a)
    {
        int N = a.length;
        for (int i = 0; i < N; i++)
        {
            int min = i;
            for (int j = i+1; j < N; j++)
                if (less(a[j], a[min]))
                    min = j;
            exch(a, i, min);
        }
    }
}
```

## Stability: selection sort

Proposition. Selection sort is **not stable**.

```
public class Selection
{
    public static void sort(Comparable[] a)
    {
        int N = a.length;
        for (int i = 0; i < N; i++)
        {
            int min = i;
            for (int j = i+1; j < N; j++)
                if (less(a[j], a[min]))
                    min = j;
            exch(a, i, min);
        }
    }
}
```

i	min	0	1	2
0	2	B <sub>1</sub>	B <sub>2</sub>	A
1	1	A	B <sub>2</sub>	B <sub>1</sub>
2	2	A	B <sub>2</sub>	B <sub>1</sub>
		A	B <sub>2</sub>	B <sub>1</sub>

Pf by counterexample. Long-distance exchange can move one equal item past another one.

# Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

## SHELLSORT

► *Is it stable?*

```
public class Shell
{
    public static void sort(Comparable[] a)
    {
        int N = a.length;
        int h = 1;
        while (h < N/3) h = 3*h + 1;
        while (h >= 1)
        {
            for (int i = h; i < N; i++)
            {
                for (int j = i; j > h && less(a[j], a[j-h]); j -= h)
                    exch(a, j, j-h);
            }
            h = h/3;
        }
    }
}
```

## Stability: shellsort

Proposition. Shellsort sort is **not stable**.

```
public class Shell
{
    public static void sort(Comparable[] a)
    {
        int N = a.length;
        int h = 1;
        while (h < N/3) h = 3*h + 1;
        while (h >= 1)
        {
            for (int i = h; i < N; i++)
            {
                for (int j = i; j > h && less(a[j], a[j-h]); j -= h)
                    exch(a, j, j-h);
            }
            h = h/3;
        }
    }
}
```

h	0	1	2	3	4
	B <sub>1</sub>	B <sub>2</sub>	B <sub>3</sub>	B <sub>4</sub>	A <sub>1</sub>
4	A <sub>1</sub>	B <sub>2</sub>	B <sub>3</sub>	B <sub>4</sub>	B <sub>1</sub>
1	A <sub>1</sub>	B <sub>2</sub>	B <sub>3</sub>	B <sub>4</sub>	B <sub>1</sub>
	A <sub>1</sub>	B <sub>2</sub>	B <sub>3</sub>	B <sub>4</sub>	B <sub>1</sub>

Pf by counterexample. Long-distance exchanges.

# Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

## MERGESORT

---

► *Is it stable?*

```
public class Merge
{
    private static void merge(...)
    { /* as before */ }

    private static void sort(Comparable[] a, Comparable[] aux, int lo, int hi)
    {
        if (hi <= lo) return;
        int mid = lo + (hi - lo) / 2;
        sort(a, aux, lo, mid);
        sort(a, aux, mid+1, hi);
        merge(a, aux, lo, mid, hi);
    }

    public static void sort(Comparable[] a)
    { /* as before */ }
}
```

## Stability: mergesort

---

Proposition. Mergesort is stable.

```
public class Merge
{
    private static void merge(...)
    { /* as before */ }

    private static void sort(Comparable[] a, Comparable[] aux, int lo, int hi)
    {
        if (hi <= lo) return;
        int mid = lo + (hi - lo) / 2;
        sort(a, aux, lo, mid);
        sort(a, aux, mid+1, hi);
        merge(a, aux, lo, mid, hi);
    }

    public static void sort(Comparable[] a)
    { /* as before */ }
}
```

Pf. Suffices to verify that merge operation is stable.

## Stability: mergesort

Proposition. Merge operation is **stable**.

```
private static void merge(...)  
{  
    for (int k = lo; k <= hi; k++)  
        aux[k] = a[k];  
  
    int i = lo, j = mid+1;  
    for (int k = lo; k <= hi; k++)  
    {  
        if (i > mid) a[k] = aux[j++];  
        else if (j > hi) a[k] = aux[i++];  
        else if (less(aux[j], aux[i])) a[k] = aux[j++];  
        else a[k] = aux[i++];  
    }  
}
```

0	1	2	3	4		5	6	7	8	9	10
A <sub>1</sub>	A <sub>2</sub>	A <sub>3</sub>	B	D		A <sub>4</sub>	A <sub>5</sub>	C	E	F	G

Pf. Takes from left subarray if equal keys.