Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

https://algs4.cs.princeton.edu

2.2 MERGESORT

- *mergesort*
- *bottom-up mergesort*
- *sorting complexity*

Last updated on 9/21/23 9:01 AM

Two classic sorting algorithms: mergesort and quicksort

Critical components in our computational infrastructure.

Mergesort. [this lecture]

…

Quicksort. [next lecture]

…


Algorithms

Robert Sedgewick | Kevin Wayne

https://algs4.cs.princeton.edu

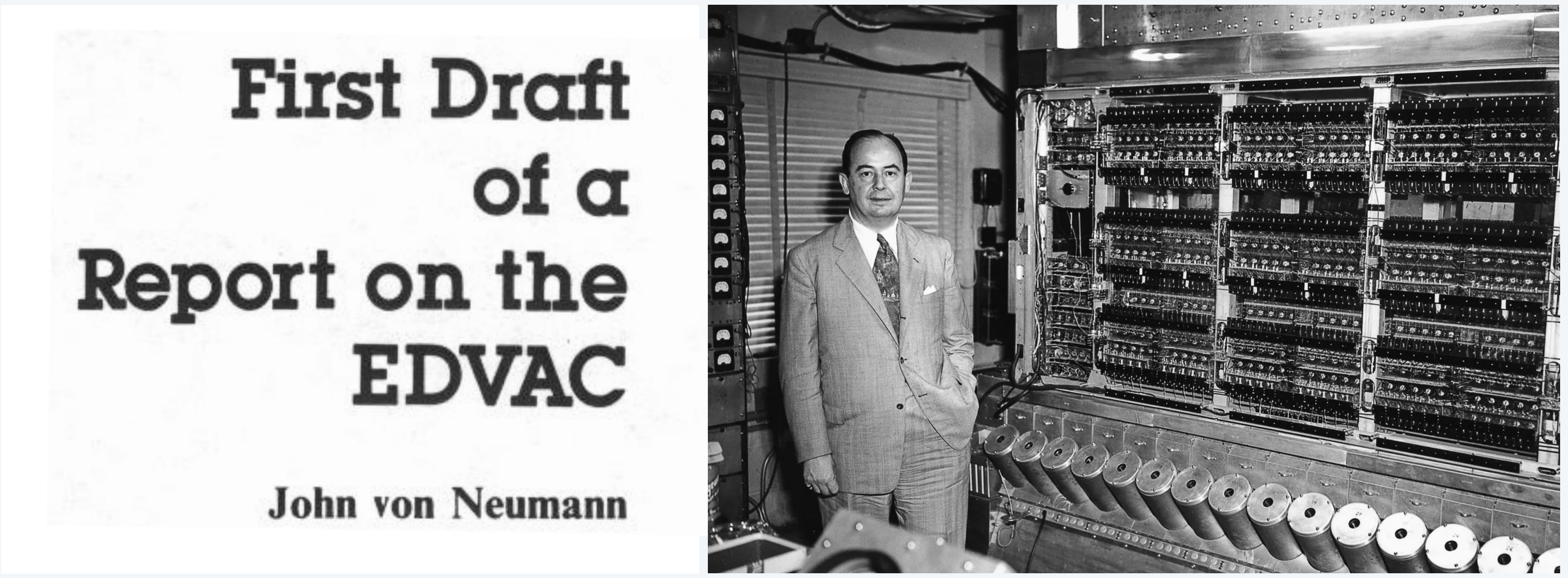
2.2 MERGESORT

- *mergesort*
- *bottom-up mergesort*
- *sorting complexity*

Mergesort overview

Basic plan.

- Divide array into two halves.
- Recursively sort left half.
- Recursively sort right half.
- Merge two sorted halves.



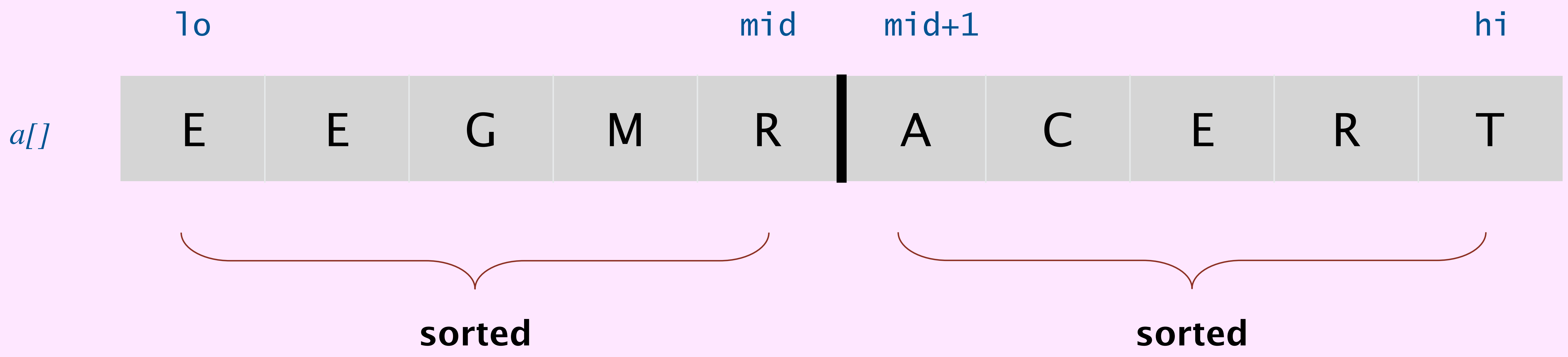
First Draft
of a
Report on the
EDVAC

John von Neumann

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

Abstract in-place merge 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]`.



	`lo`				`mid`	`mid+1`				`hi`
a[]	E	E	G	M	R	A	C	E	R	T

sorted

sorted

Merging: Java implementation

```
private static void merge(Comparable[] a, Comparable[] aux, int lo, int mid, int hi) {
```

copy

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

merge

```
   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++];
   }

}
```

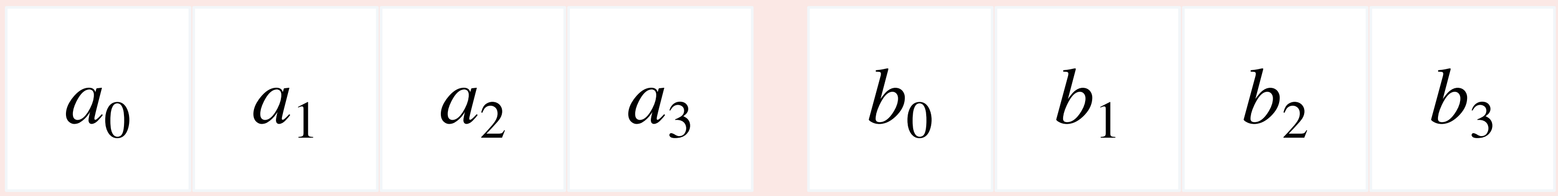
	lo			i	mid			j		hi
aux[]	A	G	L	O	R	H	I	M	S	T
						k				
a[]	A	G	H	I	L	M				

Mergesort quiz 1

How many calls does `merge()` make to `less()` when merging two sorted subarrays, each of length $n/2$, into a sorted array of length n?

- **A.** $\sim \frac{1}{4} n$ to $\sim \frac{1}{2} n$
- **B.** $\sim \frac{1}{2} n$
- **C.** $\sim \frac{1}{2} n$ to $\sim n$
- **D.** $\sim n$

merging two sorted arrays, each of length n/2



a_0 a_1 a_2 a_3 b_0 b_1 b_2 b_3

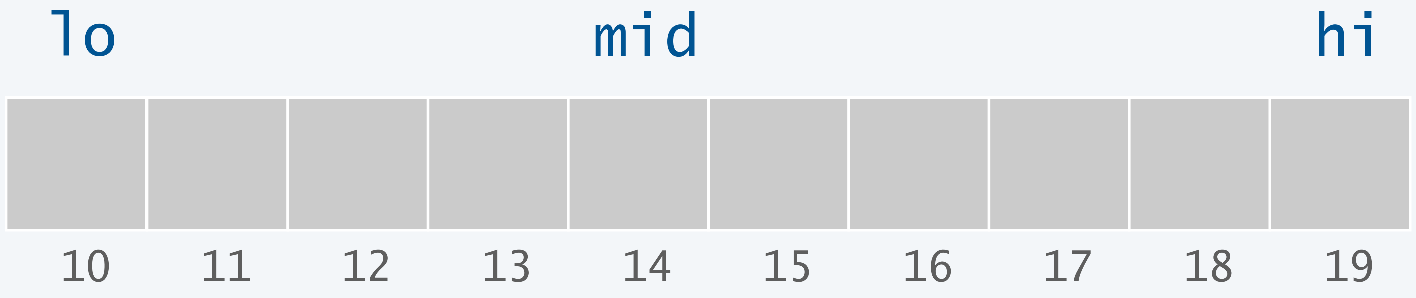
Mergesort: Java implementation

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

avoid allocating arrays within recursive function calls



Mergesort: trace

a[]

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

← *result after recursive call*

Mergesort quiz 2

Which subarray lengths will arise when mergesorting an array of length 12?

A. $\{\ 1,\ 2,\ 3,\ 4,\ 6,\ 8,\ 12\ \}$

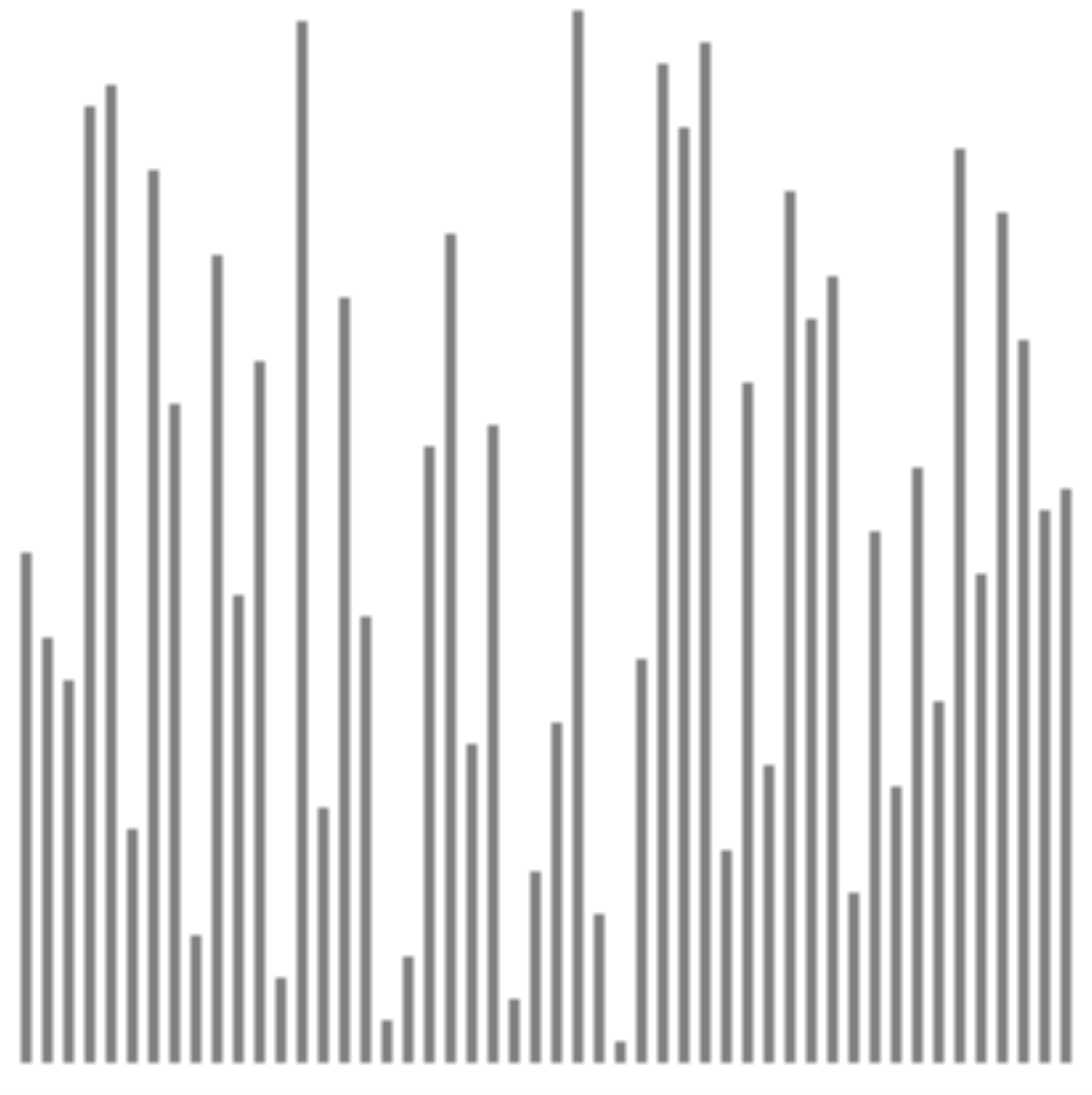
B. $\{\ 1,\ 2,\ 3,\ 6,\ 12\ \}$

C. $\{\ 1,\ 2,\ 4,\ 8,\ 12\ \}$

D. $\{\ 1,\ 3,\ 6,\ 9,\ 12\ \}$

Mergesort: animation

50 random items



https://www.toptal.com/developers/sorting-algorithms/merge-sort

▲ algorithm position

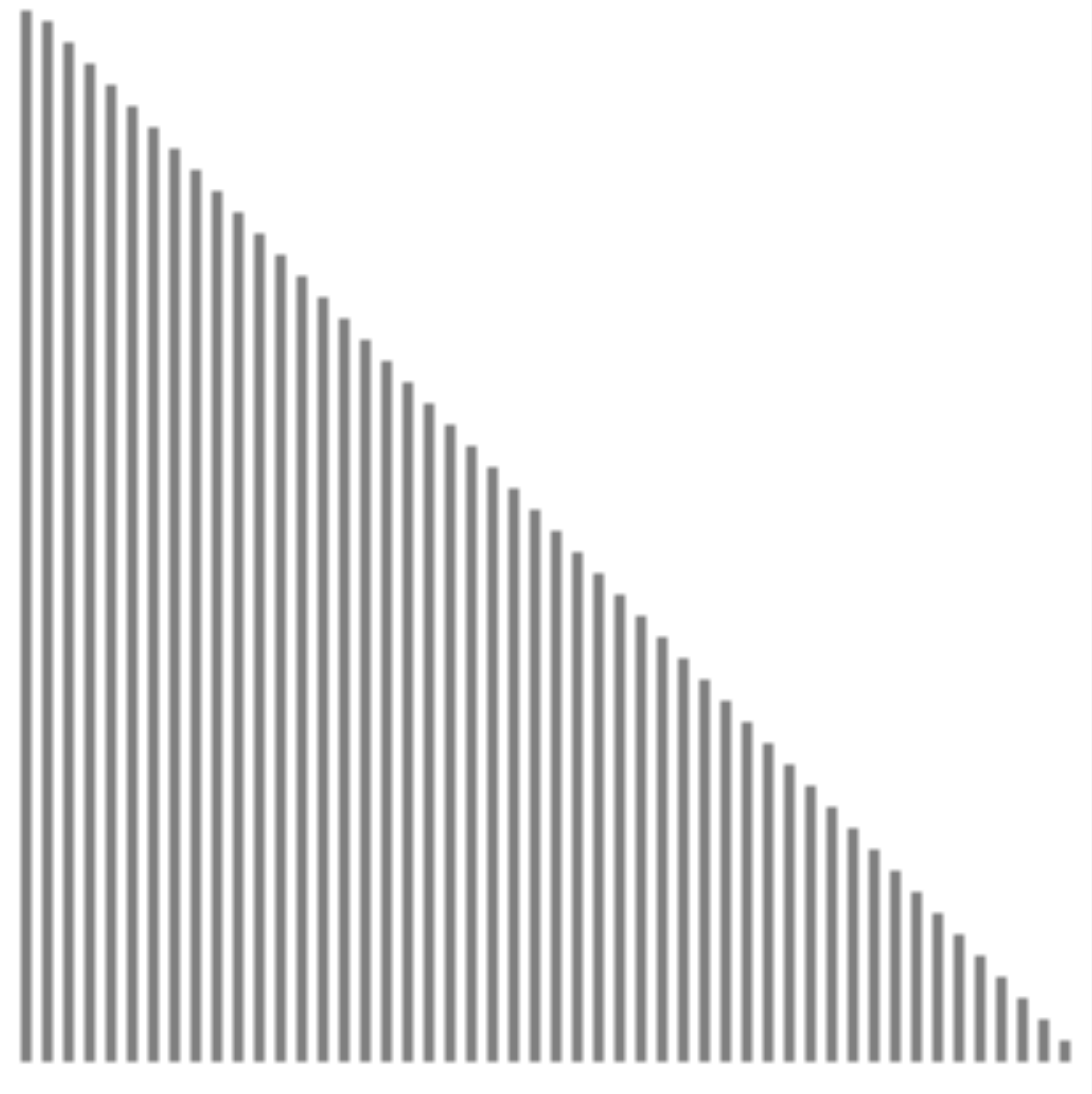
▬ in order

▬ current subarray

▬ not in order

Mergesort: animation

50 reverse-sorted items



https://www.toptal.com/developers/sorting-algorithms/merge-sort

▲ **algorithm position**

■ **in order**

▩ **current subarray**

▨ **not in order**

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 analysis: number of compares

Proposition. Mergesort uses $\le n \log_2 n$ compares to sort any array of length n.

Pf sketch. The number of compares $C(n)$ to mergesort any array of length n satisfies the recurrence:

$$C(n) \;\le\; \underset{\substack{\uparrow \\ \textit{sort} \\ \textit{left half}}}{C(\lceil n/2 \rceil)} \;+\; \underset{\substack{\uparrow \\ \textit{sort} \\ \textit{right half}}}{C(\lfloor n/2 \rfloor)} \;+\; \underset{\substack{\uparrow \\ \textit{merge}}}{n-1} \quad \text{for } n > 1, \text{ with } C(1) = 0.$$

proposition holds even when n is not a power of 2
(but analysis cleaner in this case)

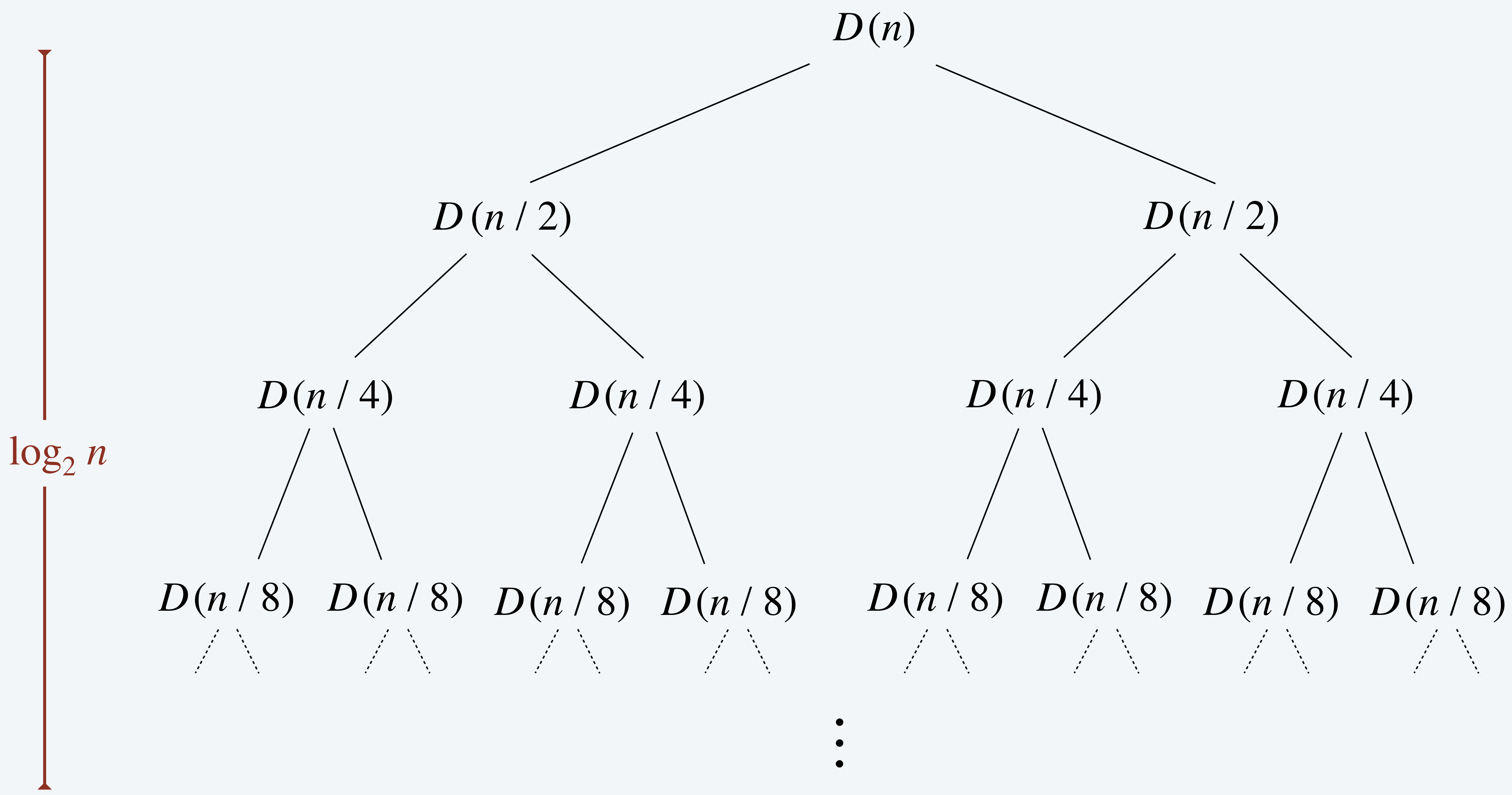
For simplicity. Assume n is a power of 2 and solve this recurrence:

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

Divide-and-conquer recurrence

Proposition. If $D(n)$ satisfies $D(n) = 2\,D(n/2) + n$ for $n > 1$, with $D(1) = 0$, then $D(n) = n \log_2 n$.

Pf by picture. [assuming n is a power of 2]



$n = n$

$2\,(n/2) = n$

$4\,(n/4) = n$

$8\,(n/8) = n$

$\vdots$

$$\boxed{D(n) = n \log_2 n}$$

Mergesort analysis: number of array accesses

Proposition. Mergesort makes $\Theta(n \log n)$ array accesses.

Pf sketch. The number of array accesses $A(n)$ satisfies the recurrence:

$$A(n) \ = \ A(\lceil n/2 \rceil) \ + \ A(\lfloor n/2 \rfloor) \ + \ \Theta(n) \quad \text{for } n > 1, \text{ with } A(1) = 0.$$

Key point. Any algorithm with the following structure takes $\Theta(n \log n)$ time:

```
public static void f(int n) {
    if (n == 0) return;
    f(n/2);
    f(n/2);
    linear(n);
}
```

solve two problems of half the size

do $\Theta(n)$ work

Famous examples. FFT, closest pair, hidden-line removal, Kendall-tau distance, …

Mergesort analysis: memory

Proposition. Mergesort uses $\Theta(n)$ extra space.

Pf. The length of the `aux[]` array is n, to handle the last merge.

two sorted subarrays

A	C	D	G	H	I	M	N	U	V

B	E	F	J	O	P	Q	R	S	T

merged result

A	B	C	D	E	F	G	H	I	J	M	N	O	P	Q	R	S	T	U	V

essentially negligible

Def. A sorting algorithm is in-place if it uses $\Theta(\log n)$ extra space (or less).

Ex. Insertion sort and selection sort.

Challenge 1 (not hard). Get by with an `aux[]` array of length $\sim \frac{1}{2} n$ (instead of n).

Challenge 2 (very hard). In-place merge. [Kronrod 1969]

Mergesort quiz 3

Consider the following modified version of mergesort.

How much total memory is allocated over all recursive calls?

A. $\Theta(n)$

B. $\Theta(n \log n)$

C. $\Theta(n^2)$

D. $\Theta(2^n)$

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

Mergesort: practical improvement

Use insertion sort for small subarrays.

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

```
private static void sort(...) {

   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);
}
```

makes mergesort about 20% faster

Algorithms

Robert Sedgewick | Kevin Wayne

https://algs4.cs.princeton.edu

2.2 MERGESORT

- ▸ *mergesort*
- ▸ *bottom-up mergesort*
- ▸ *sorting complexity*

Bottom-up mergesort

Basic plan.

- Pass through array, merging subarrays of length 1.
- Repeat for subarrays of length $2, 4, 8, \ldots$

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

Bottom-up mergesort: Java implementation

```
public class MergeBU {

   private static void merge(...) {
      /* 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));
   }                                  ↑           ↑

}                               mid          hi
```

Proposition. At most $n \log_2 n$ compares; $\Theta(n)$ extra space.

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

Mergesort quiz 4

Which is faster in practice for $n = 2^{20}$, top-down mergesort or bottom-up mergesort?

A. Top-down (recursive) mergesort.

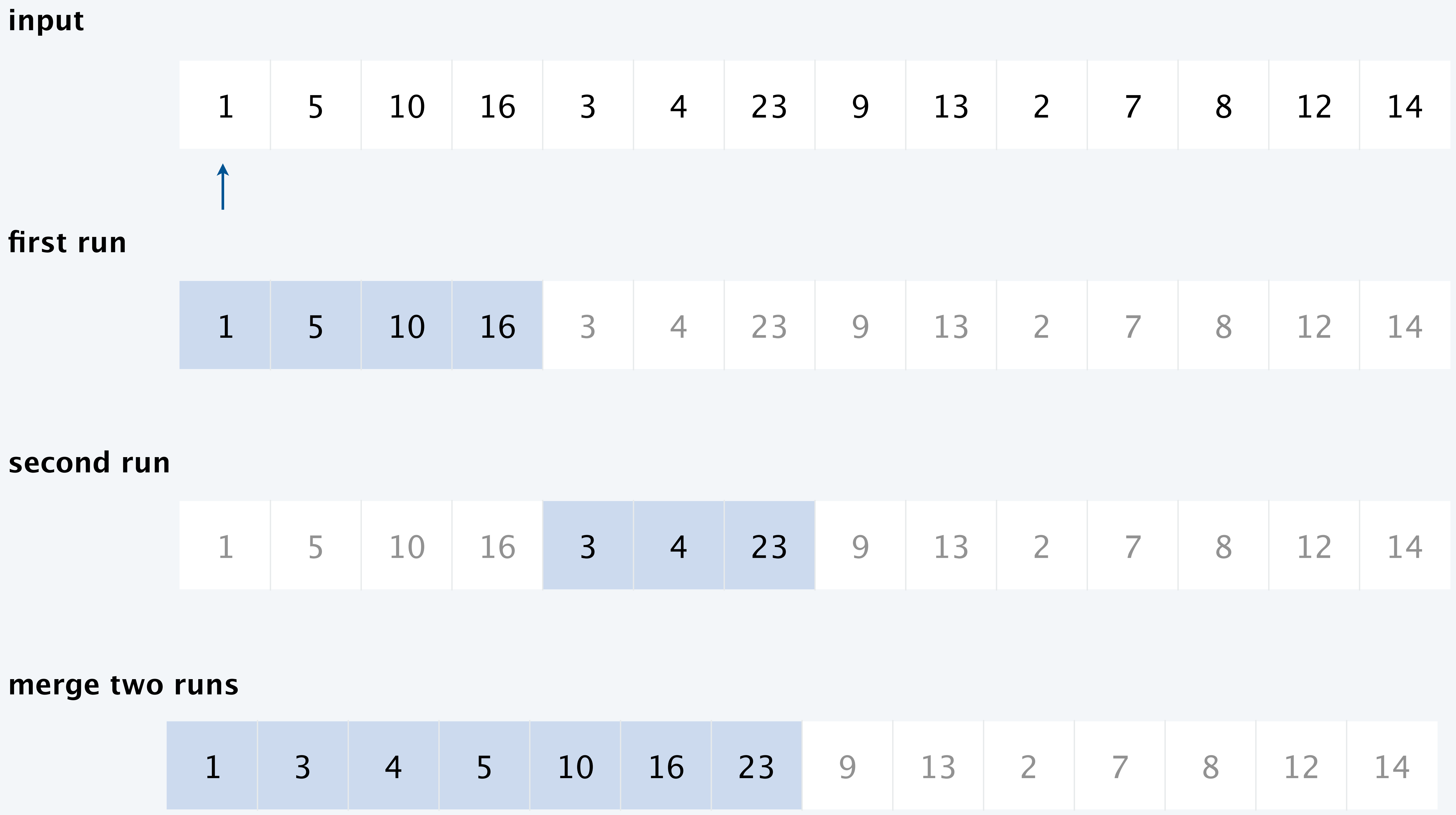
B. Bottom-up (non-recursive) mergesort.

C. No difference.

D. *I don't know.*

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.

Timsort (2002)

- Natural mergesort.
- Use binary insertion sort to make initial runs (if needed).
- A few more clever optimizations.

```
This describes an adaptive, stable, natural mergesort, modestly called
timsort (hey, I earned it <wink>).  It has supernatural performance on many
kinds of partially ordered arrays (less than lg(n!) comparisons needed, and
as few as n-1), yet as fast as Python's previous highly tuned samplesort
hybrid on random arrays.

In a nutshell, the main routine marches over the array once, left to right,
alternately identifying the next run, then merging it into the previous
runs "intelligently".  Everything else is complication for speed, and some
hard-won measure of memory efficiency.
...
```

Tim Peters

Consequence. Only $\Theta(n)$ compares on many arrays with pre-existing order.

Widely used. Python, Java, GNU Octave, Android, …

Timsort bug (February 2015)

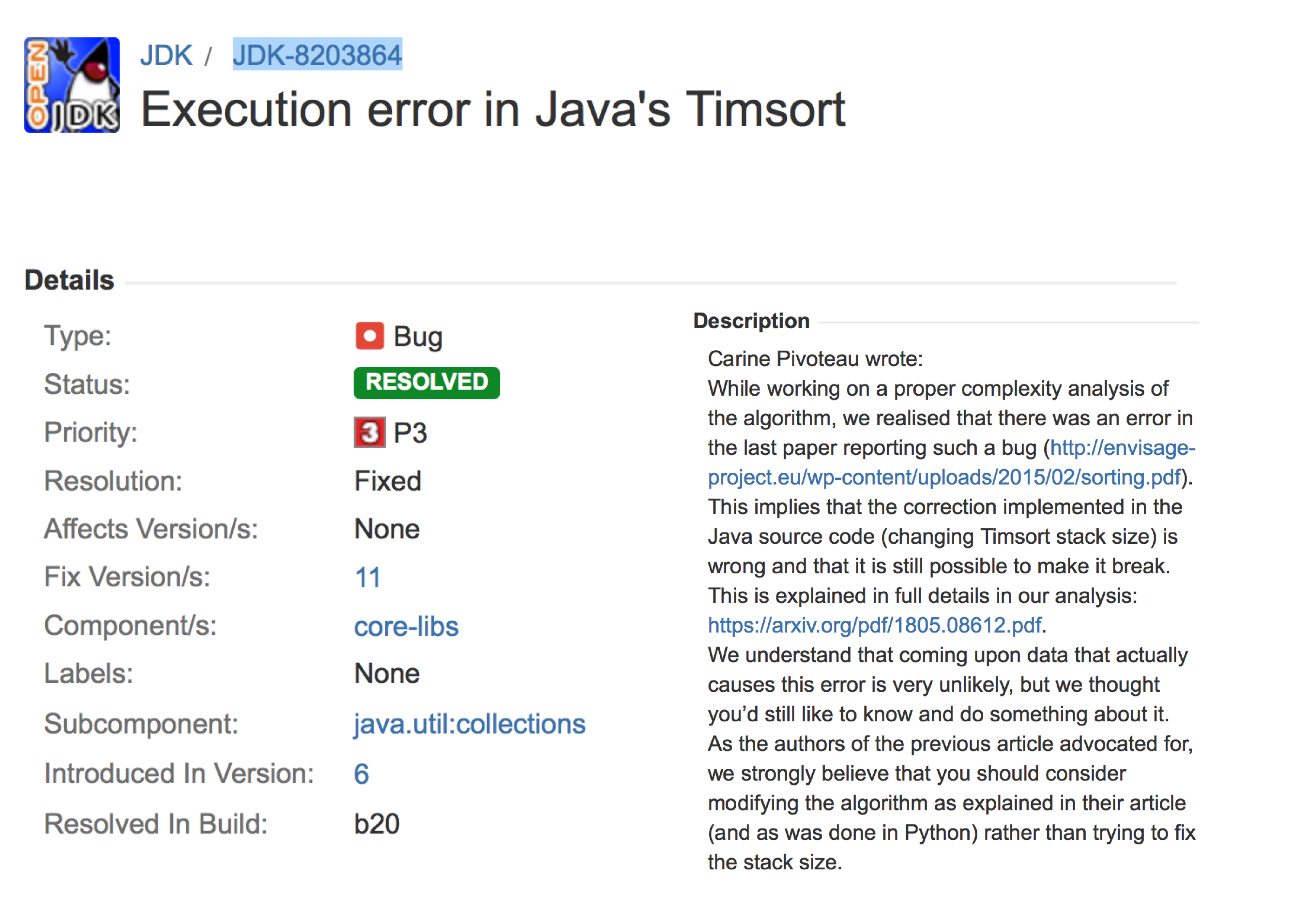
Proving that Android's, Java's and Python's sorting algorithm is broken (and showing how to fix it)

February 24, 2015 Envisage Written by Stijn de Gouw. $s

Tim Peters developed the Timsort hybrid sorting algorithm in 2002. It is a clever combination of ideas from merge sort and insertion sort, and designed to perform well on real world data. TimSort was first developed for Python, but later ported to Java (where it appears as java.util.Collections.sort and java.util.Arrays.sort) by Joshua Bloch (the designer of Java Collections who also pointed out that most binary search algorithms were broken). TimSort is today used as the default sorting algorithm for Android SDK, Sun's JDK and OpenJDK. Given the popularity of these platforms this means that the number of computers, cloud services and mobile phones that use TimSort for sorting is well into the billions.

http://envisage-project.eu/proving-android-java-and-python-sorting-algorithm-is-broken-and-how-to-fix-it

Timsort bug (May 2018)



JDK / JDK-8203864

Execution error in Java's Timsort

Details

Type:	Bug
Status:	RESOLVED
Priority:	P3
Resolution:	Fixed
Affects Version/s:	None
Fix Version/s:	11
Component/s:	core-libs
Labels:	None
Subcomponent:	java.util:collections
Introduced In Version:	6
Resolved In Build:	b20

Description

Carine Pivoteau wrote:
While working on a proper complexity analysis of the algorithm, we realised that there was an error in the last paper reporting such a bug (http://envisage-project.eu/wp-content/uploads/2015/02/sorting.pdf).
This implies that the correction implemented in the Java source code (changing Timsort stack size) is wrong and that it is still possible to make it break.
This is explained in full details in our analysis: https://arxiv.org/pdf/1805.08612.pdf.
We understand that coming upon data that actually causes this error is very unlikely, but we thought you'd still like to know and do something about it.
As the authors of the previous article advocated for, we strongly believe that you should consider modifying the algorithm as explained in their article (and as was done in Python) rather than trying to fix the stack size.

https://bugs.openjdk.java.net/browse/JDK-8203864

Sorting summary

	in-place?	stable?	best	average	worst	remarks
selection	✓		½ n^2	½ n^2	½ n^2	*n exchanges*
insertion	✓	✓	n	¼ n^2	½ n^2	*use for small n* *or partially sorted*
merge		✓	½ $n \log_2 n$	$n \log_2 n$	$n \log_2 n$	Θ(*n log n*) *guarantee;* *stable*
timsort		✓	n	$n \log_2 n$	$n \log_2 n$	*improves mergesort* *when pre-existing order*
?	✓	✓	n	$n \log_2 n$	$n \log_2 n$	*holy sorting grail*

number of compares to sort an array of n elements

https://algs4.cs.princeton.edu

2.2 MERGESORT

- *mergesort*
- *bottom-up mergesort*
- *sorting complexity*

Computational complexity

A framework to study efficiency of algorithms for solving a particular problem X.

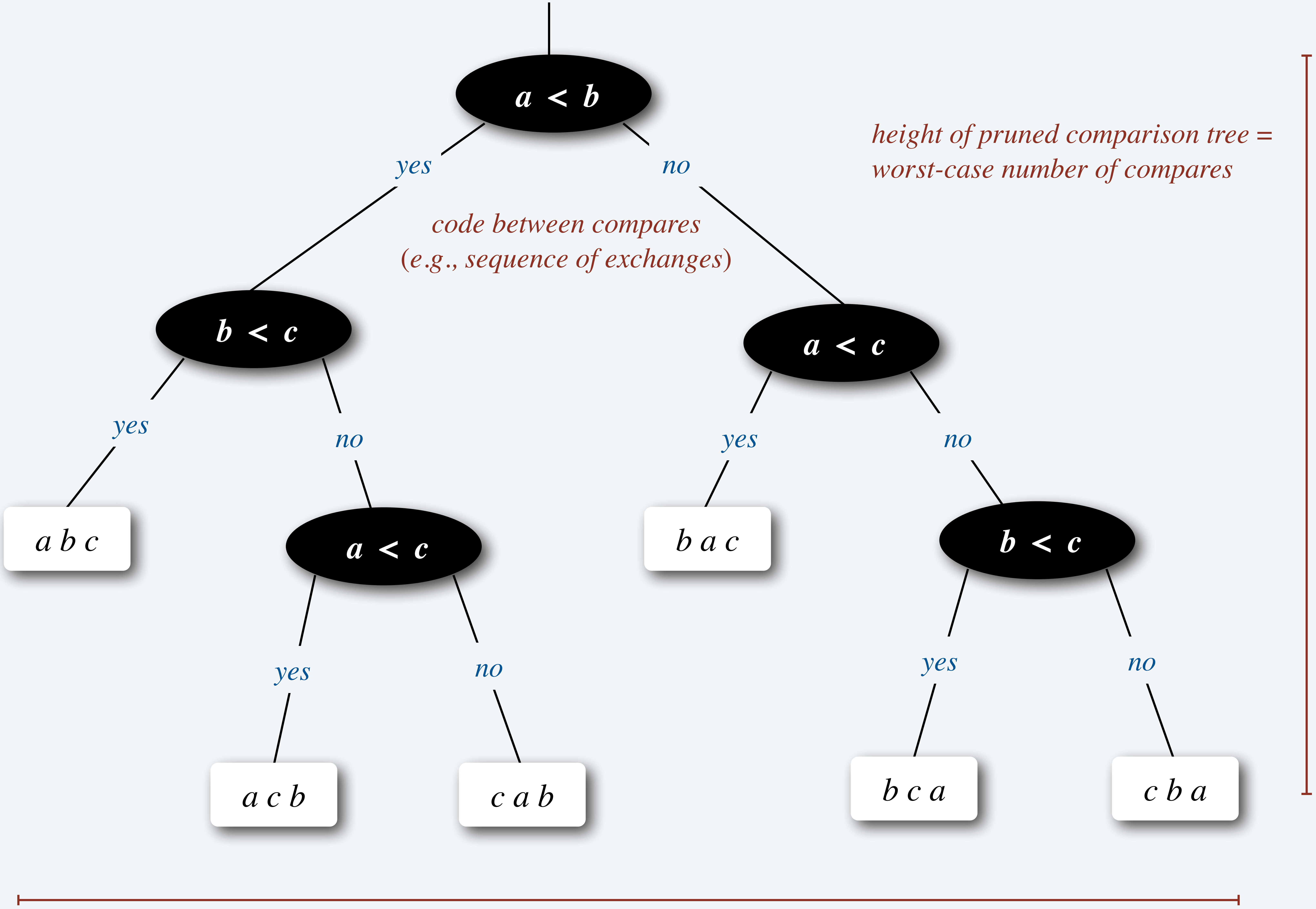
term	description	example (X = sorting)
model of computation	*specifies memory and primitive operations*	comparison tree
cost model	*primitive operation counts*	# compares
upper bound	*cost guarantee provided by some algorithm for a problem*	$\sim n \log_2 n$
lower bound	*proven limit on cost guarantee for all algorithms for a problem*	?
optimal algorithm	*algorithm with best possible cost guarantee for a problem*	?

can gain knowledge about input only through pairwise compares (e.g., Java's `Comparable` *framework)*

from mergesort

lower bound ~ upper bound

Comparison tree (for 3 distinct keys a, b, and c)



code between compares (e.g., sequence of exchanges)

height of pruned comparison tree = worst-case number of compares

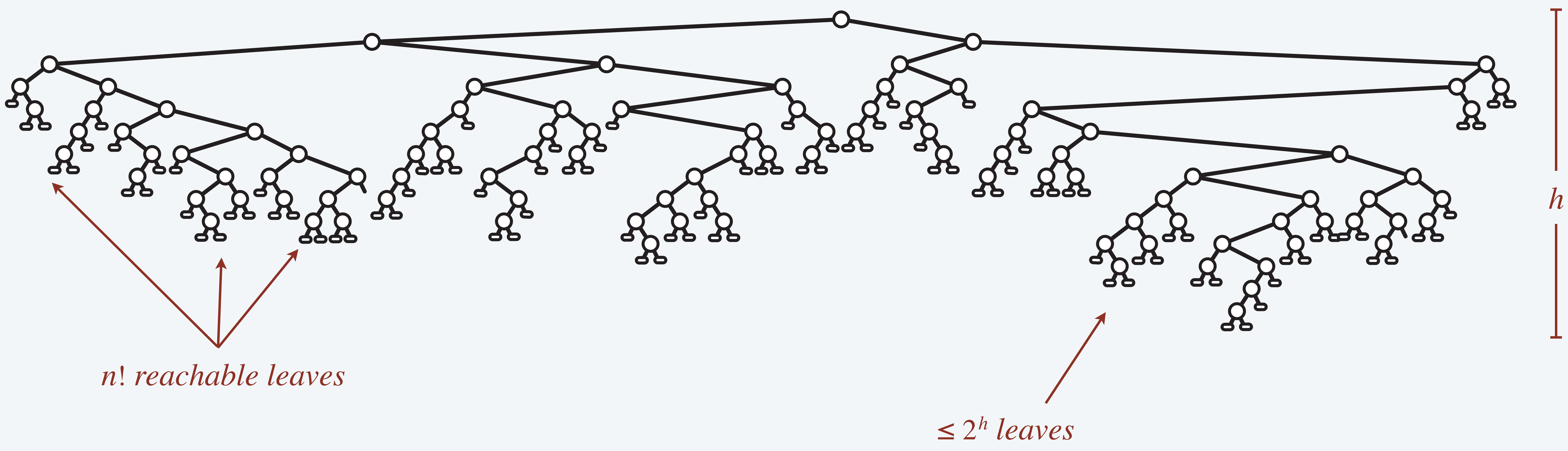
one (and only one) reachable leaf corresponds to each each possible ordering

Compare-based lower bound for sorting

Proposition. In the worst case, any compare-based sorting algorithm must make at least $\log_2(n!) \sim n \log_2 n$ compares.

Pf.

- Assume array consists of n distinct values a_1 through a_n.
- $n!$ different orderings $\Rightarrow$ $n!$ reachable leaves.
- Worst-case number of compares = height h of pruned comparison tree.
- Binary tree of height h has $\leq 2^h$ leaves.



Compare-based lower bound for sorting

Proposition. In the worst case, any compare-based sorting algorithm must make at least $\log_2(n!) \sim n \log_2 n$ compares.

Pf.

- Assume array consists of n distinct values a_1 through a_n.
- $n!$ different orderings $\Rightarrow$ $n!$ reachable leaves.
- Worst-case number of compares = height h of pruned comparison tree.
- Binary tree of height h has $\leq 2^h$ leaves.

$$\begin{aligned} 2^h &\geq \text{\# reachable leaves} = n! \\ \Rightarrow \quad h &\geq \log_2(n!) \\ &\underset{\substack{\uparrow \\ \textit{Stirling's formula}}}{\sim} n \log_2 n \end{aligned}$$

Computational complexity

A framework to study efficiency of algorithms for solving a particular problem X.

term	description	example (X = sorting)
model of computation	*specifies memory and primitive operations*	comparison tree
cost model	*primitive operation counts*	# compares
upper bound	*cost guarantee provided by some algorithm for a problem*	$\sim n \log_2 n$
lower bound	*proven limit on cost guarantee for all algorithms for a problem*	$\boxed{\sim n \log_2 n}$
optimal algorithm	*algorithm with best possible cost guarantee for a problem*	$\boxed{\text{mergesort}}$

First goal of algorithm design: optimal algorithms.

Computational complexity results in context

Compares? Mergesort is optimal with respect to number compares.

Space? Mergesort is not optimal with respect to space usage.

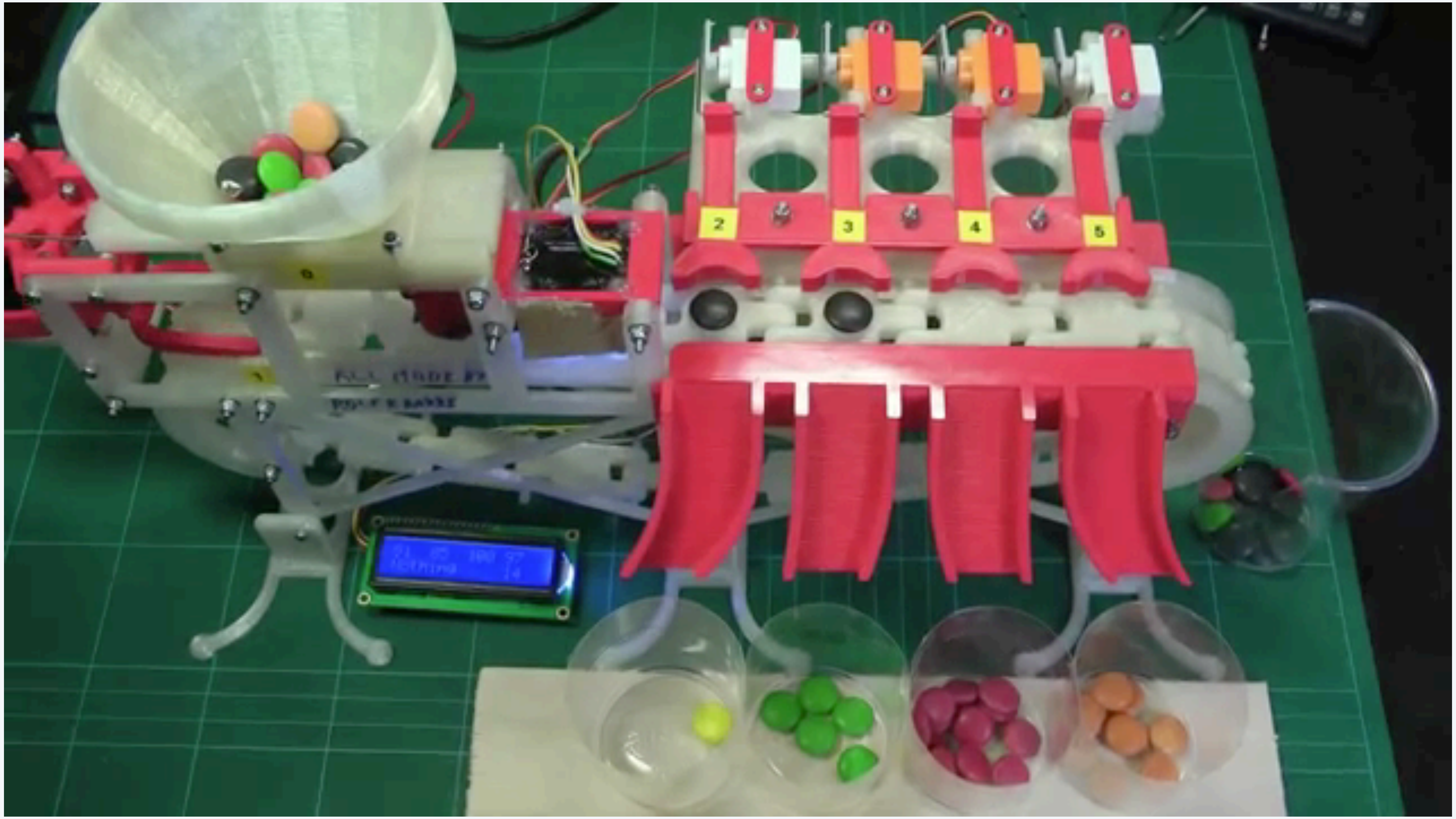
Lesson. Use theory as a guide.

Ex. ~~Design sorting algorithm that makes $\sim \frac{1}{2} n \log_2 n$ compares in worst case?~~

Ex. Design sorting algorithm that makes $\Theta(n \log n)$ compares and uses $\Theta(1)$ extra space.

Commercial break (sponsored by Skittles)

Q. Why doesn't this Skittles sorter violate the sorting lower bound?



https://www.youtube.com/watch?v=tSEHDBSynVo

Complexity results in context (continued)

Lower bound may not hold if the algorithm can exploit:

- The initial order of the input array.

  Ex: insertion sort makes only $\Theta(n)$ compares on partially sorted arrays.

- The distribution of key values.

  Ex: 3-way quicksort makes only $\Theta(n)$ compares on arrays with a small number of distinct keys. [next lecture]

- The representation of the keys.

  Ex: radix sorts do not make any key compares; they access the data via individual characters/digits.

Asymptotic notations

Warning: many programmers misuse O to mean Θ.

notation	provides	example	shorthand for
tilde (~)	*leading term*	$\sim \frac{1}{2} n^2$	$\frac{1}{2} n^2$ $\frac{1}{2} n^2 + 3n + 22$ $\frac{1}{2} n^2 + n \log_2 n$
big Theta (Θ)	*order of growth*	$\Theta(n^2)$	$\frac{1}{2} n^2$ $7n^2 + n^{\frac{1}{2}}$ $5n^2 - 3n$
big O (O)	*upper bound*	$O(n^2)$	$10n^2$ $22n$ $\log_2 n$
big Omega (Ω)	*lower bound*	$\Omega(n^2)$	$\frac{1}{2} n^2$ $n^3 + 3n$ 2^n

← *ignore lower-order terms*

← *also ignore leading coefficient*

← $\Theta(n^2)$ *or smaller*

← $\Theta(n^2)$ *or larger*

Mergesort quiz 5

Which of the following correctly describes the function $f(n) = 3n^2 + 30n$?

A. $\sim n^2$

B. $\Theta(n)$

C. $O(n^3)$

D. $\mathit{All\ of\ the\ above}.$

E. $\mathit{None\ of\ the\ above}.$

Sorting lower bound

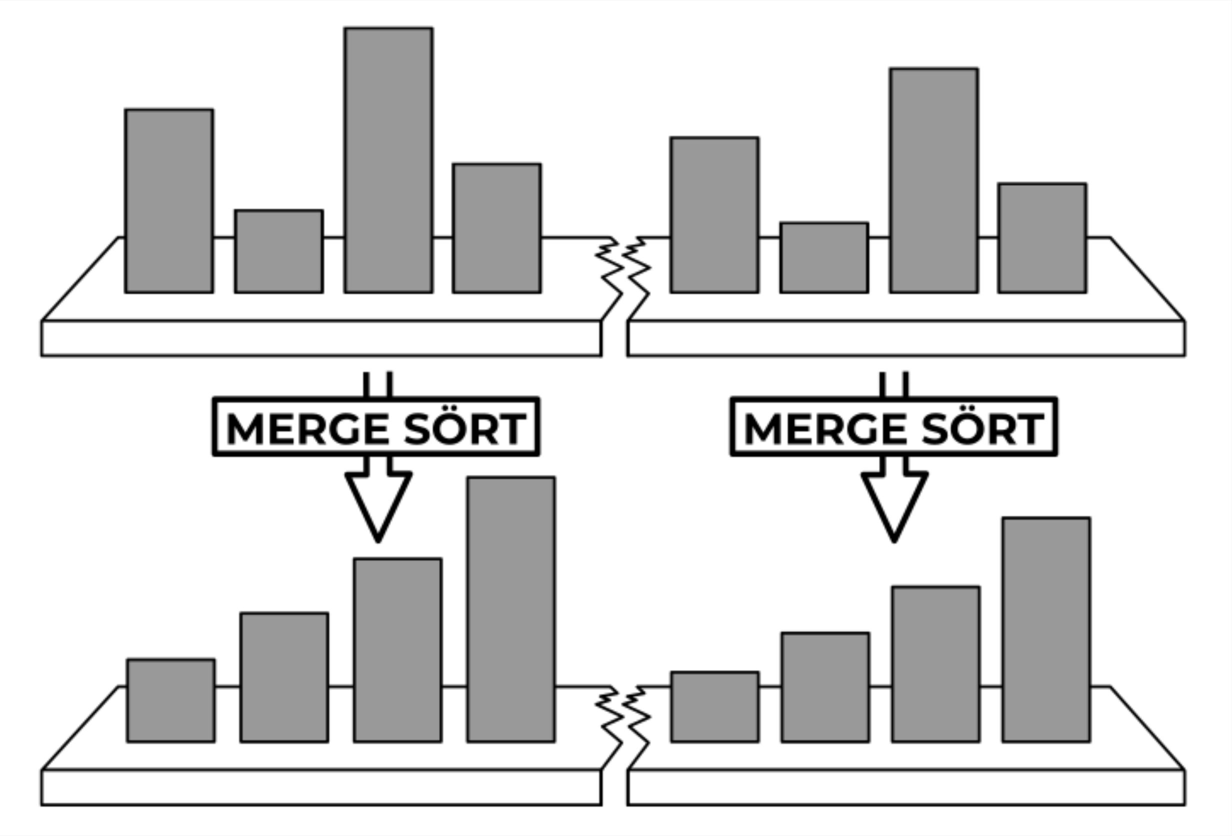
Interviewer. Give a formal description of the sorting lower bound for sorting arrays of n elements.

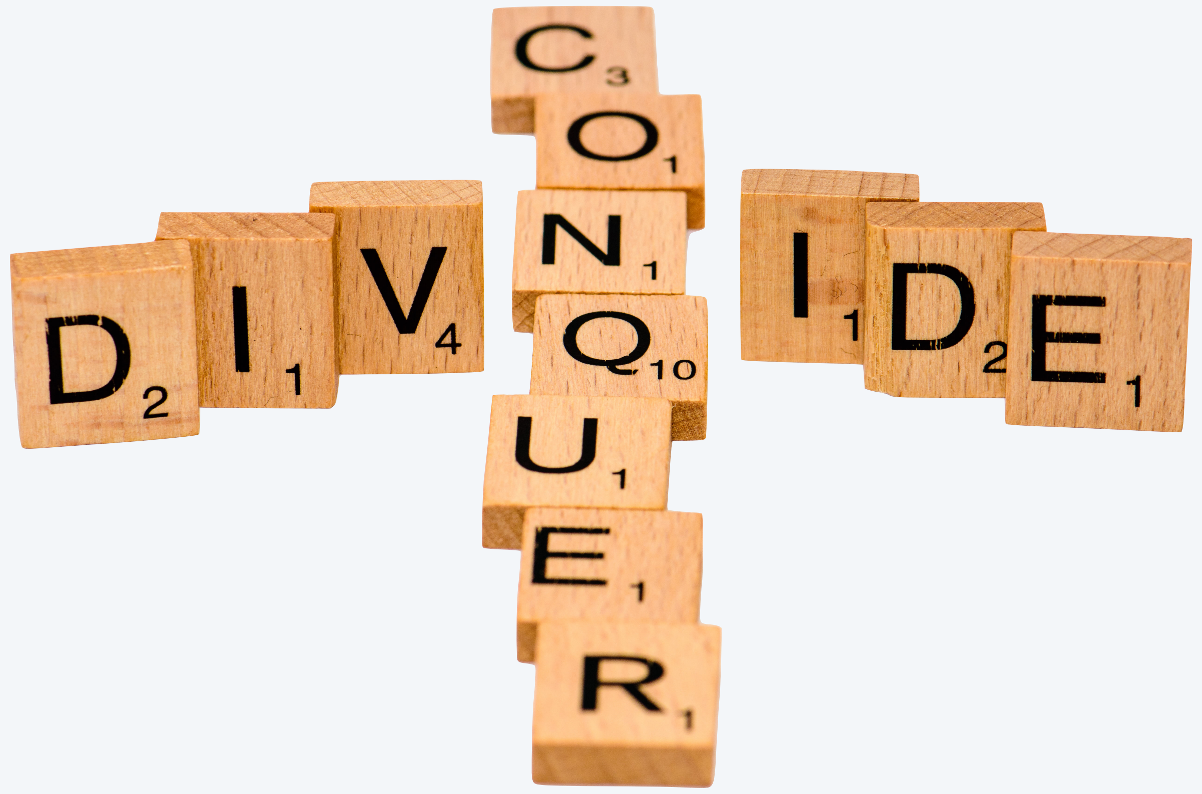
Summary

Mergesort. Makes $\Theta(n \log n)$ compares (and array accesses) in the worst case.

Sorting lower bound. No compare-based sorting algorithm makes fewer than $\Theta(n \log n)$ compares in the worst case.

Divide-and-conquer. Divide a problem into two (or more) subproblems; solve each subproblem independently; combine results.





Credits

image/video	source	license
Jon von Neumann	IAS / Alan Richards	
Tim Peters	unknown	
Theory vs. Practice	Ela Sjolie	
Skittles Sorting Machine	Rolf R. Bakke	
Fast Skittles Sorting Machine	Kazumichi Moriyama	
Impossible Stamp	Adobe Stock	education license
Divide-and-Conquer	wallpapercrafter.com	
Mergesort Instructions	IDEA	CC BY-NC-SA 4.0

A final thought