Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

Algorithms

FOURTH EDITION

ROBERT SEDGEWICK | KEVIN WAYNE

https://algs4.cs.princeton.edu

2.2 MERGESORT

- *mergesort*
- *sorting complexity*

Last updated on 9/18/24 10:28PM

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

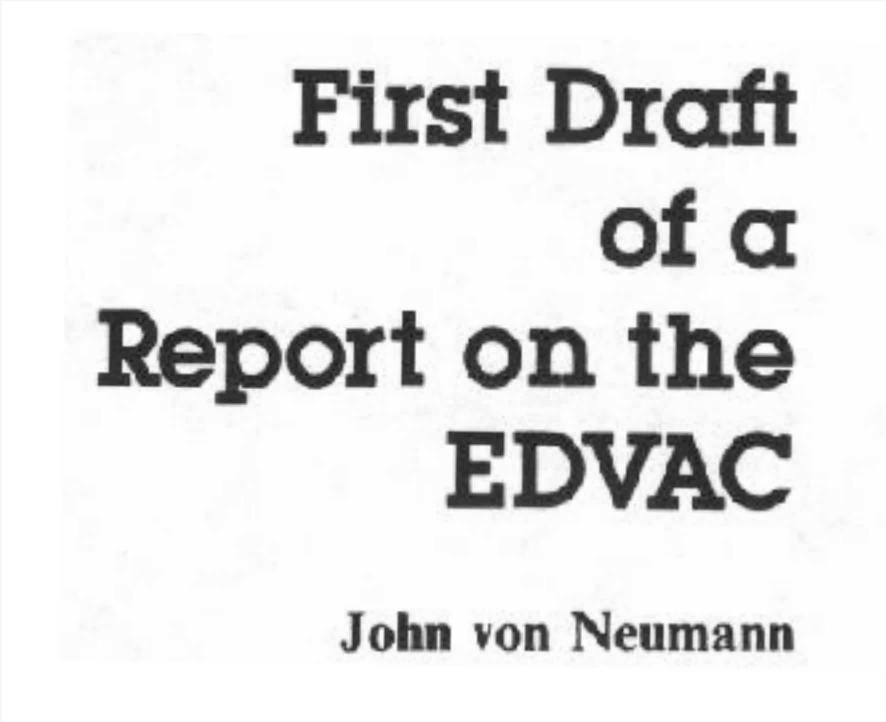
2.2 MERGESORT

- *mergesort*
- *sorting complexity*

Mergesort overview

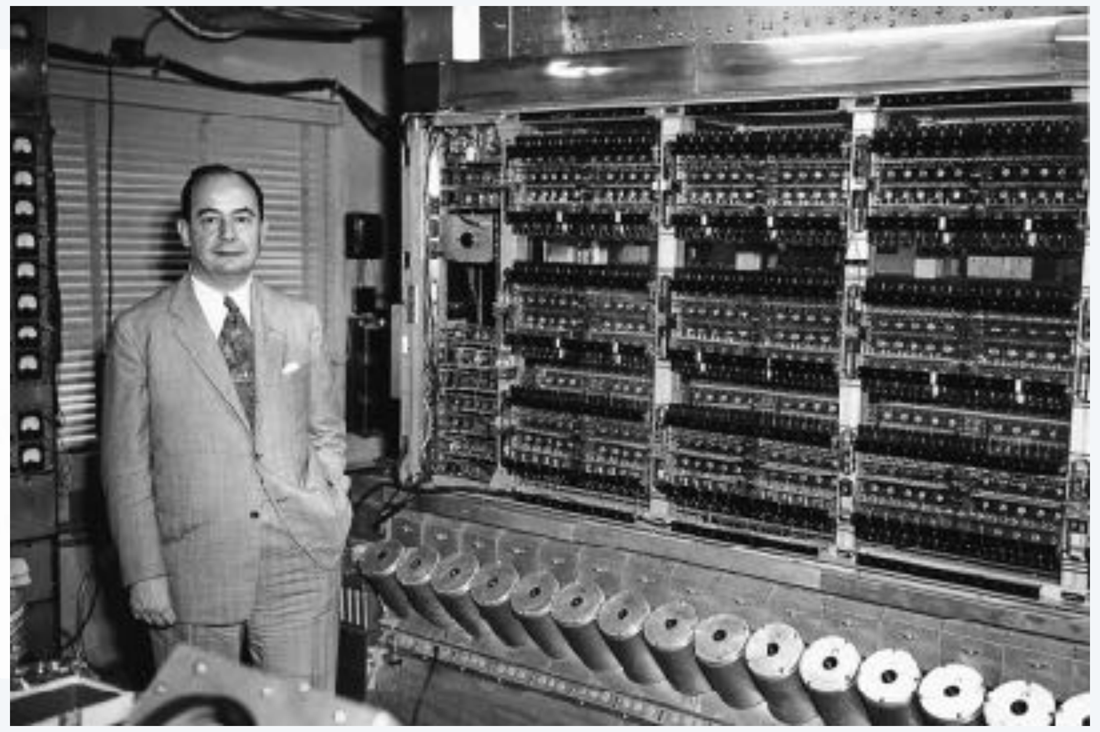
Basic plan.

- 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

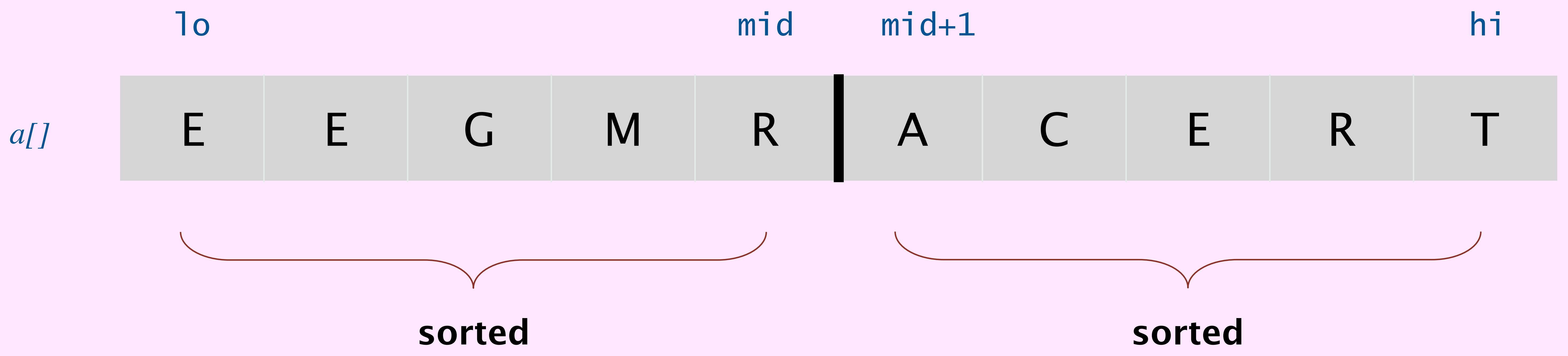
John von Neumann



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]`.



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++];
   }

}
```

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

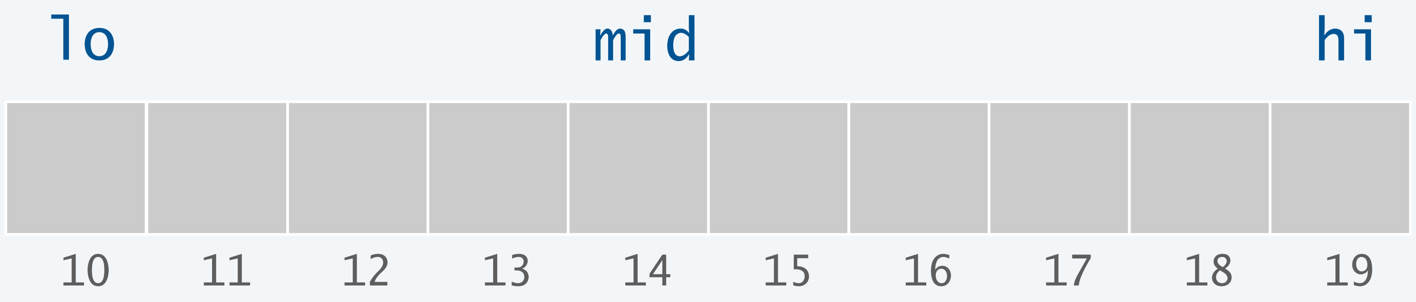
Mergesort: Java implementation

```
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	← *result after recursive call*
`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	

Mergesort quiz 2

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

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

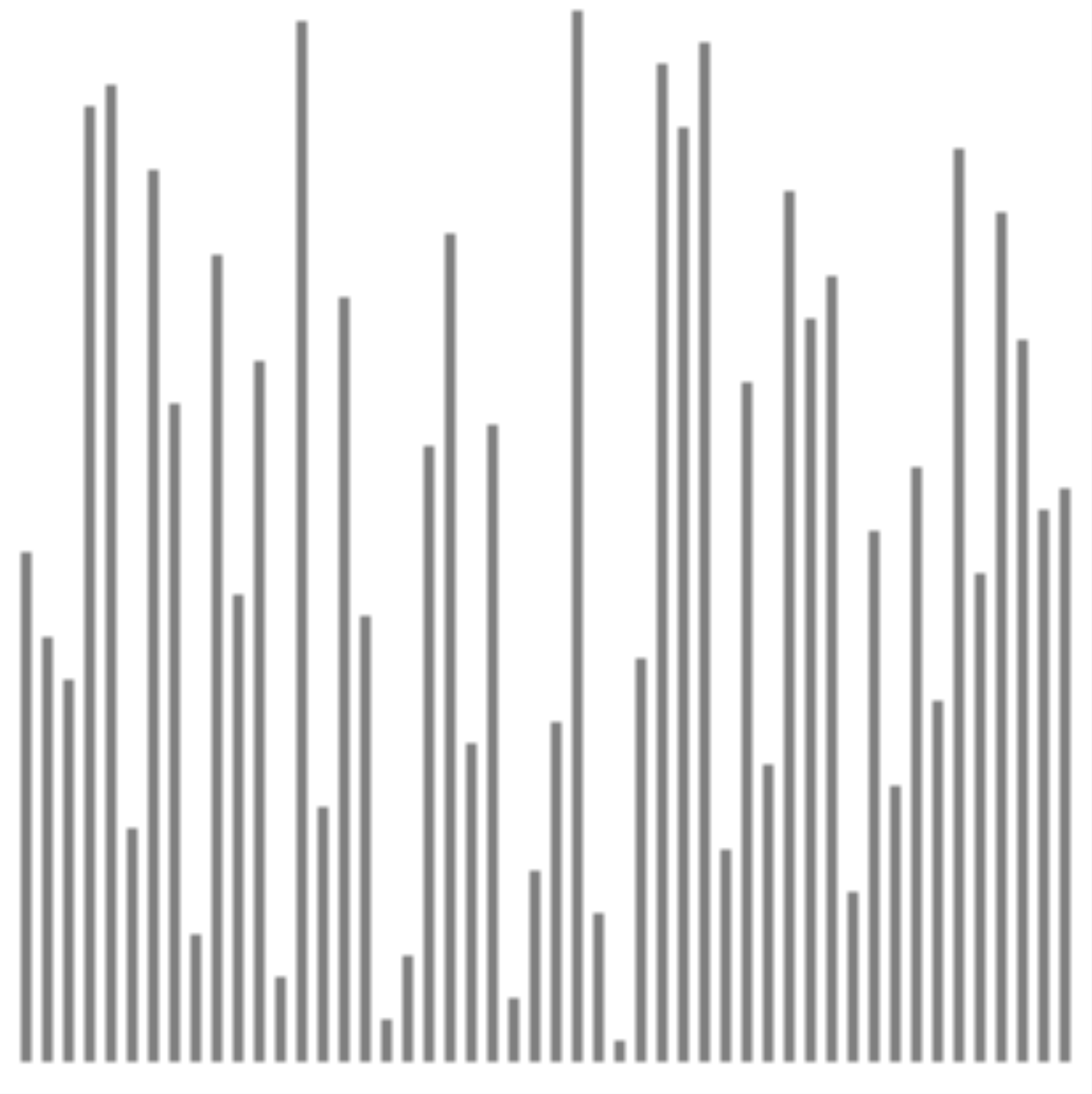
B. $\{\ 1,\ 2,\ 3,\ 6,\ 12\ \}$

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

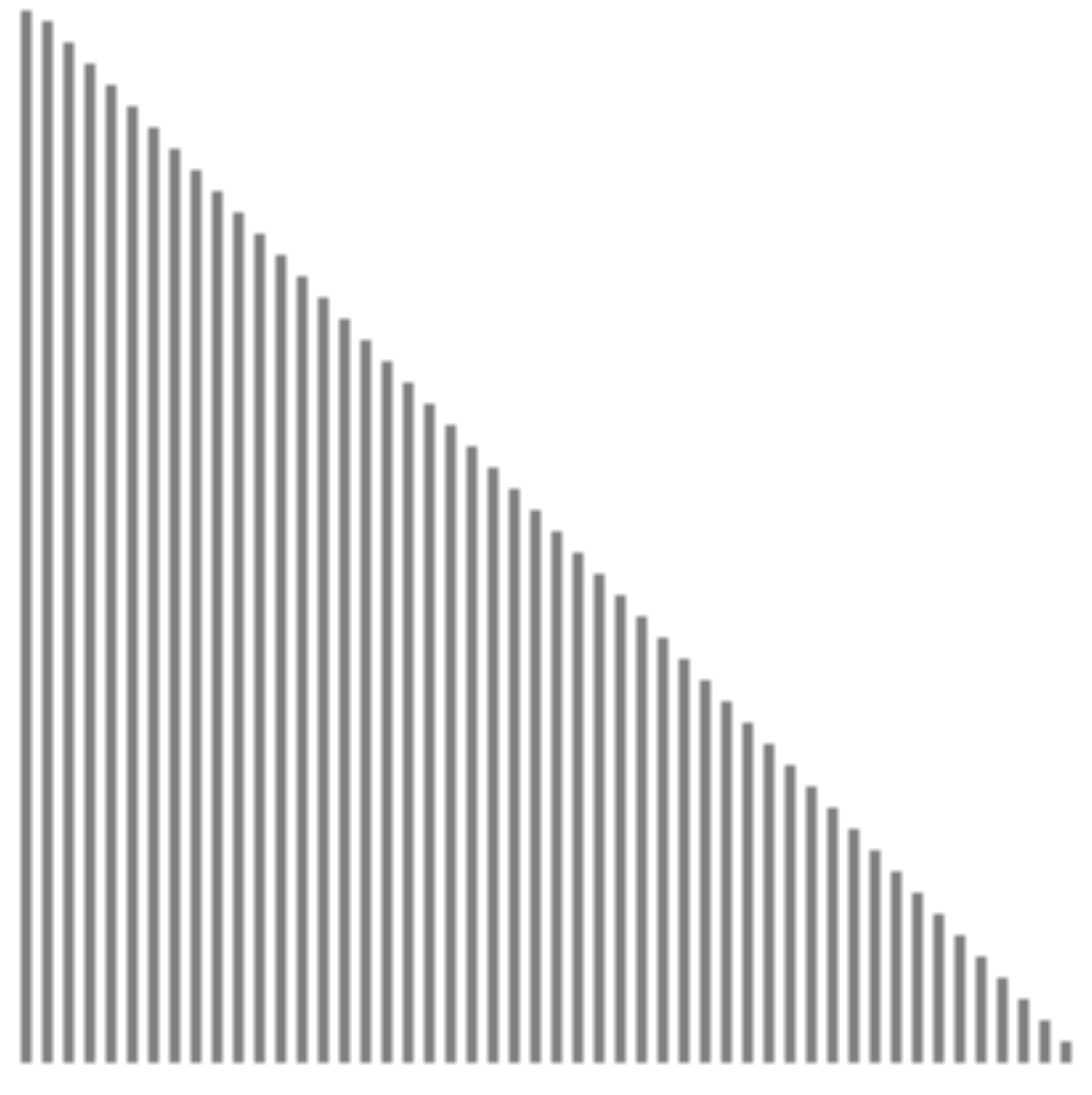
▬ in order

▬ 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:

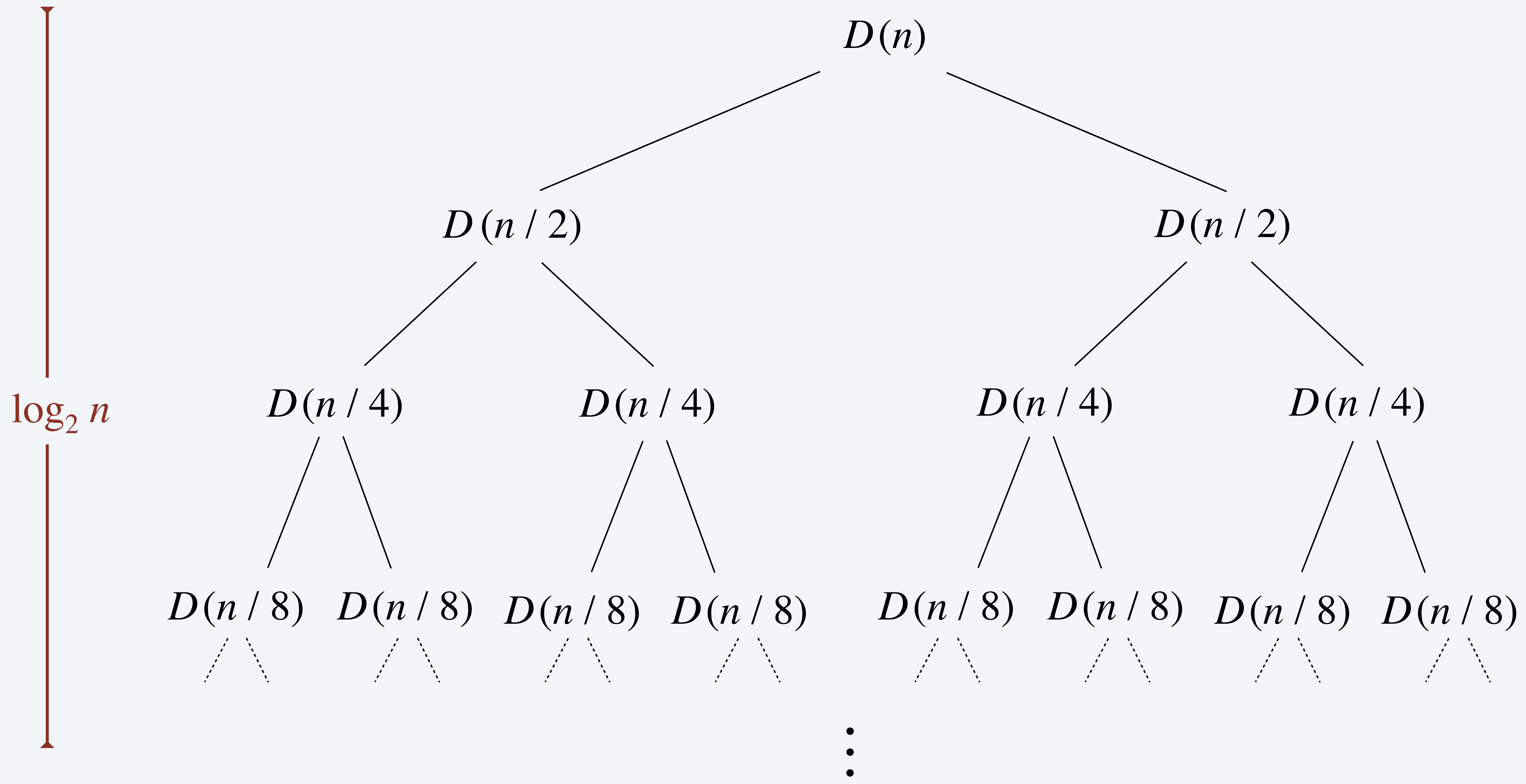
$$D(n) \;=\; 2\,D(n/2) \;+\; n, \text{ for } n > 1, \text{ with } D(1) = 0.$$

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]

Q: how about $D(n) = 3\,D(n/3) + 5n$?



$n = n$

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

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

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

⋮

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

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

Mergesort quiz 4

Is our implementation of mergesort stable?

A. Yes.

B. No, but it can be easily modified to be stable.

C. No, mergesort is inherently unstable.

D. *I don’t remember what stability means.* ←

a sorting algorithm is stable if it preserves the relative order of equal keys

input	C	A_1	B	A_2	A_3
sorted	A_3	A_1	A_2	B	C

not stable

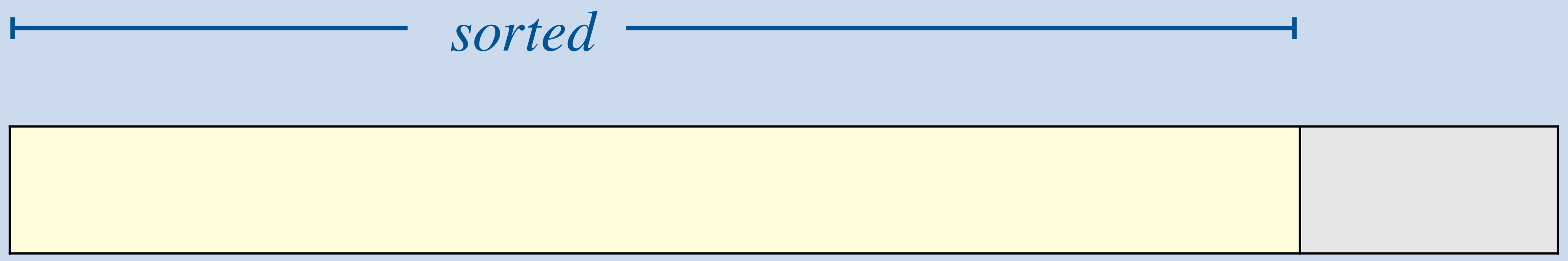
Sorting summary

	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 $n \log_2 n$	$n \log_2 n$	$n \log_2 n$	$\Theta(n \log n)$ *guarantee; stable*
?	✓	✓	n	$n \log_2 n$	$n \log_2 n$	*holy sorting grail*

number of compares to sort an array of n elements

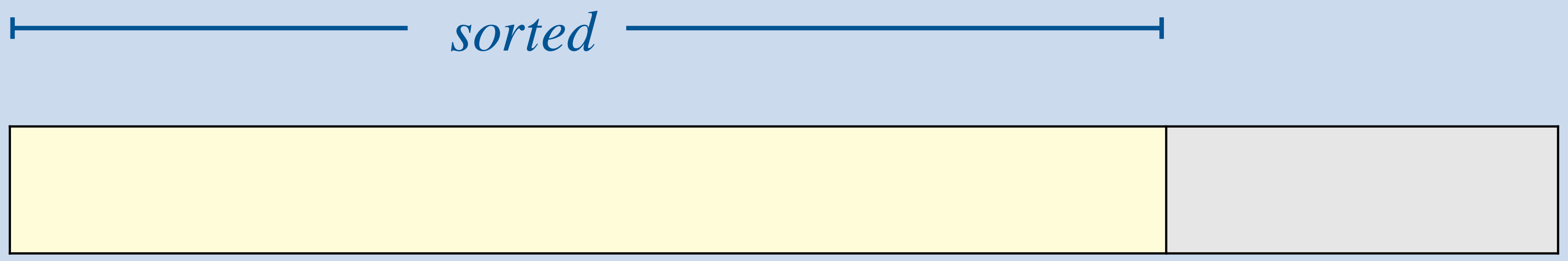
Partially sorted arrays

Version 1. Given an array of n integers where the first $n - 100$ entries are already in sorted order, sort the entire array in $\Theta(n)$ time.



Partially sorted arrays

Version 2. Given an array of n integers where the first $n - \sqrt{n}$ entries are already in sorted order, sort the entire array in $\Theta(n)$ time.




Algorithms

Robert Sedgewick | Kevin Wayne

https://algs4.cs.princeton.edu

2.2 MERGESORT

- *mergesort*
- *sorting complexity*

Computational complexity

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

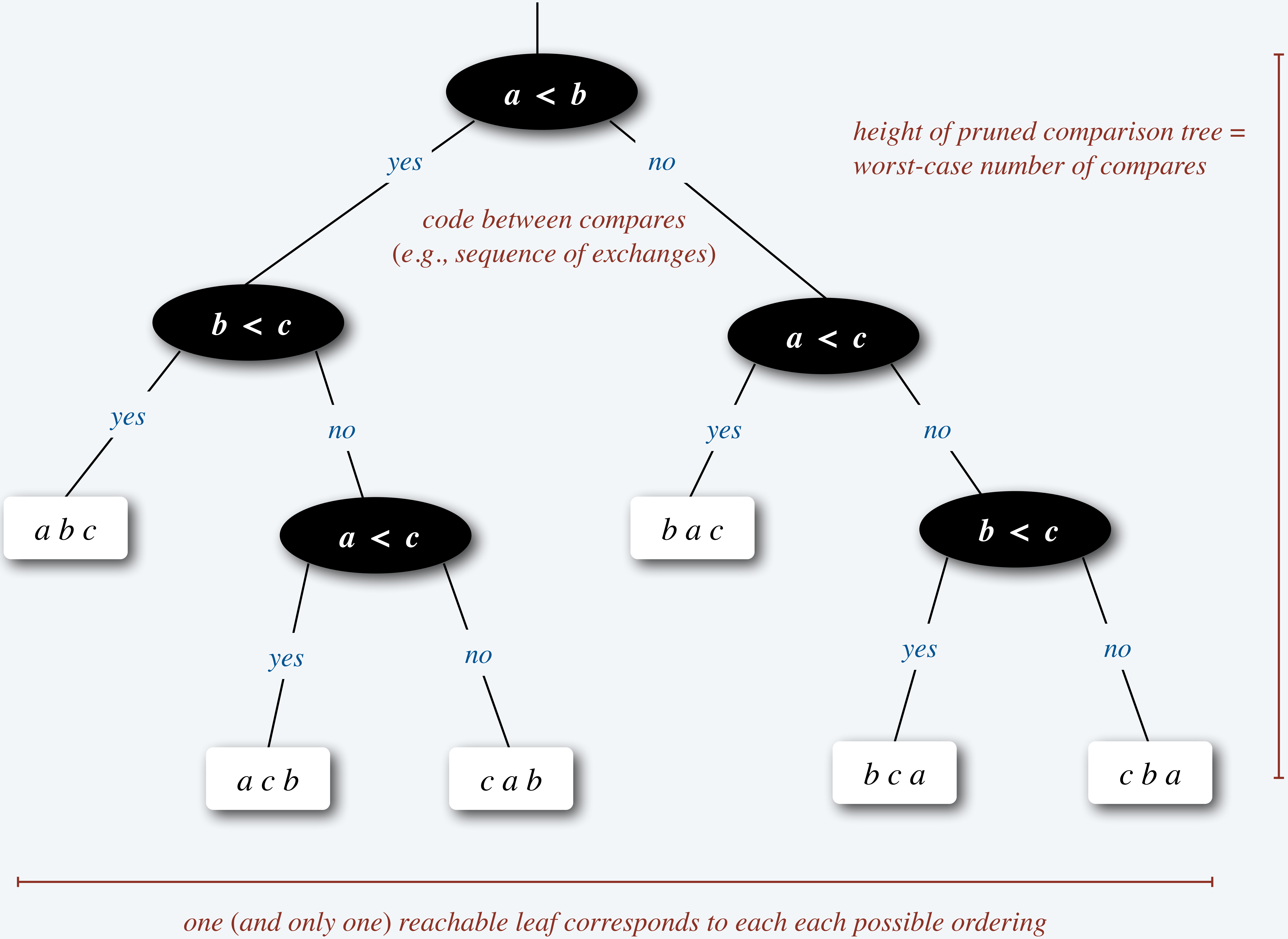
term	**description**	**example (X = sorting)**
model of computation	*specifies memory and primitive operations*	comparison tree
cost model	*primitive operation counts*	# compares
upper bound	*cost guarantee provided by some algorithm for a problem*	$\sim n \log_2 n$
lower bound	*proven limit on cost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

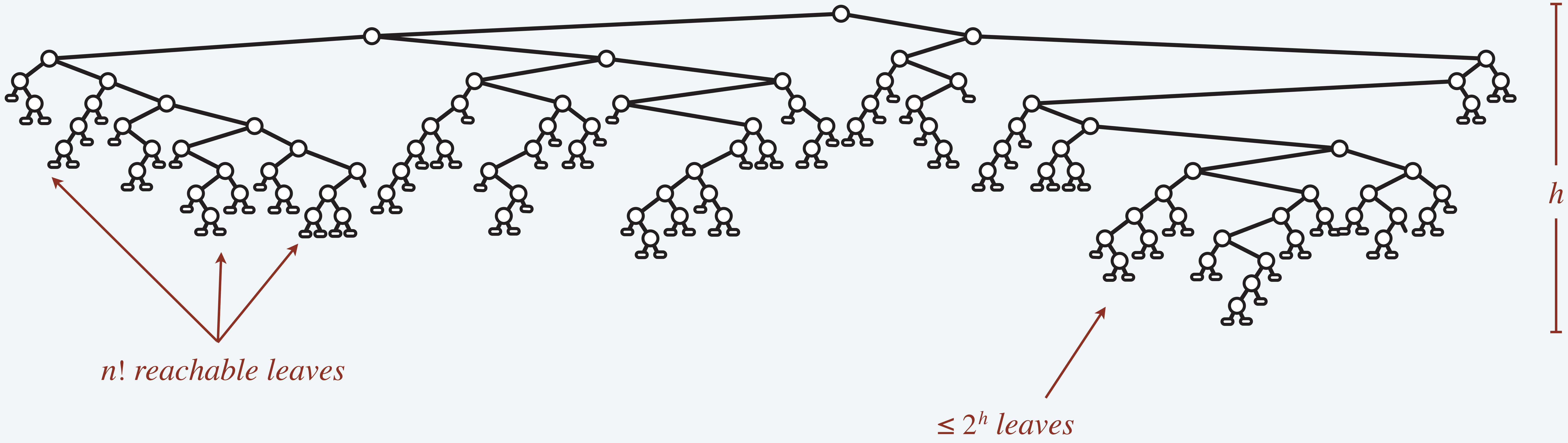
one (and only one) reachable leaf corresponds to each each possible ordering

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.

$$\begin{aligned} 2^h &\geq \#\,\text{reachable leaves} = n! \\ \Rightarrow\ h &\geq \log_2(n!) \\ &\underset{\uparrow}{\sim} n \log_2 n \end{aligned}$$

Stirling's formula

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*	$\sim n \log_2 n$
optimal algorithm	*algorithm with best possible cost guarantee for a problem*	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.

Sorting with few values

Version 1. Is it possible to sort an array of n integers ranging from 0 to $n-1$ in $\Theta(n)$ time?

Sorting with few values

Version 2. Is it possible to sort an array of n *elements* with integer keys ranging from 0 to $n-1$ in $\Theta(n)$ time?

Sorting with few values

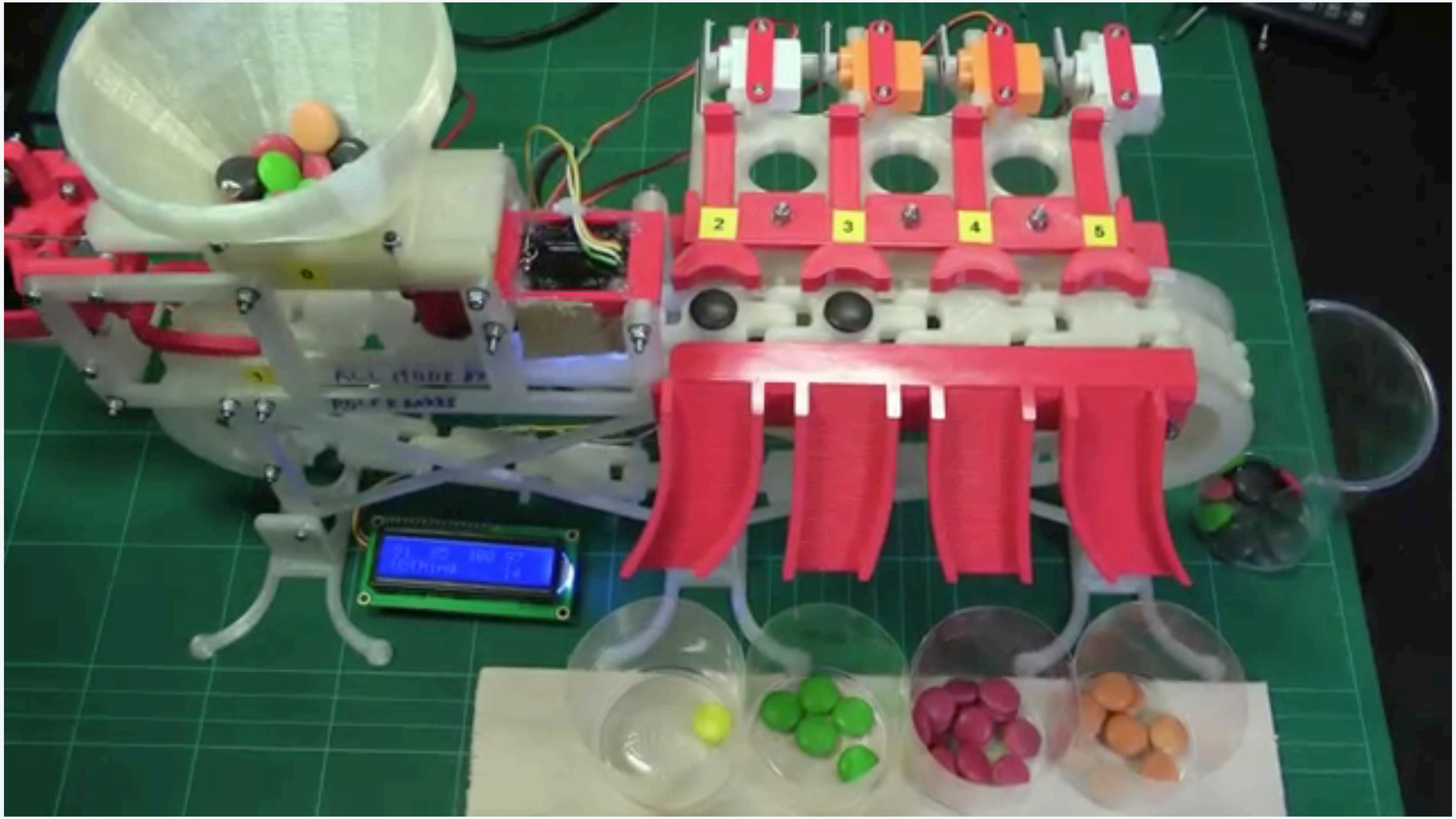
Version 3. Is it possible to sort an array of n integers ranging from 0 to $n^2 - 1$ in $\Theta(n)$ time?

Hint 1. Express each integer as $an + b$, where $0 \leq a, b \leq n - 1$.

Hint 2. The algorithm from Version 2 can be made stable (e.g., insert new elements at the end of the linked list).

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.

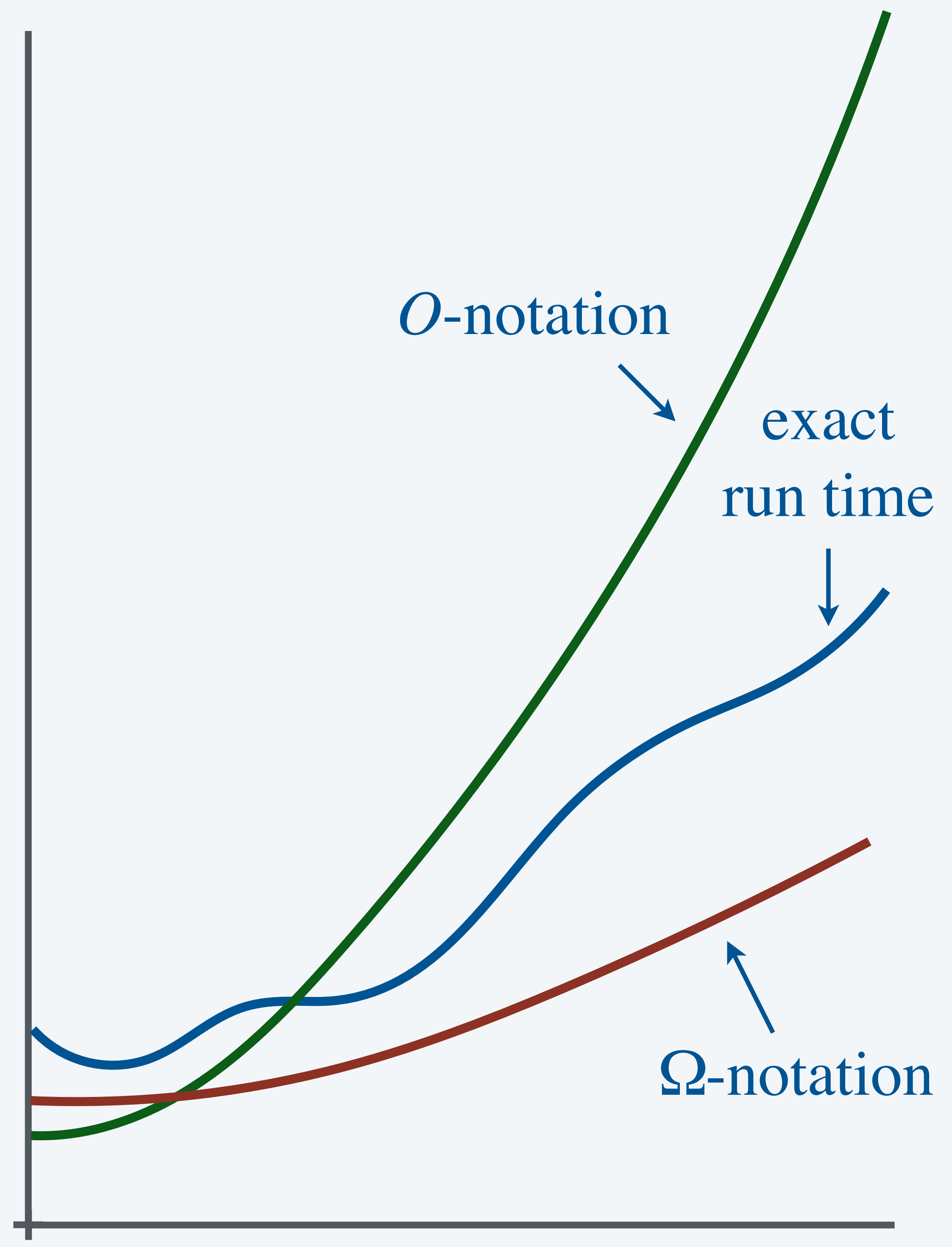
notation	provides	example	shorthand for
tilde (~)	*leading term*	$\sim \frac{1}{2} n^2$	$\frac{1}{2} n^2$ $\frac{1}{2} n^2 + 3n + 22$ $\frac{1}{2} n^2 + n \log_2 n$
big Theta (Θ)	*order of growth*	$\Theta(n^2)$	$\frac{1}{2} n^2$ $7n^2 + n^{1/2}$ $5n^2 - 3n$
big O (O)	*upper bound*	$O(n^2)$	$10n^2$ $22n$ $\log_2 n$
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 $f(n) = 10 \log n + 2\, n \log n + 0.1\, n$?

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

B. $O(2^n)$

C. $O(n \log n)$

D. $\Omega(n)$

E. $All\ of\ the\ above.$

Mergesort quiz 6

Which of the following statements is implied by the sorting lower bound?

A. Any sorting algorithm runs in time at least $O(n \log n)$ on any large enough input.

B. Any compare-based sorting algorithm makes $\Theta(n \log n)$ compares or uses $\Theta(n)$ memory.

C. In the worst case, any compare-based sorting algorithm makes $O(n \log n)$ compares.

D. In the worst case, any compare-based sorting algorithm makes $\Omega(n \log n)$ compares.

E. *None of the above.*

Sorting a linked list

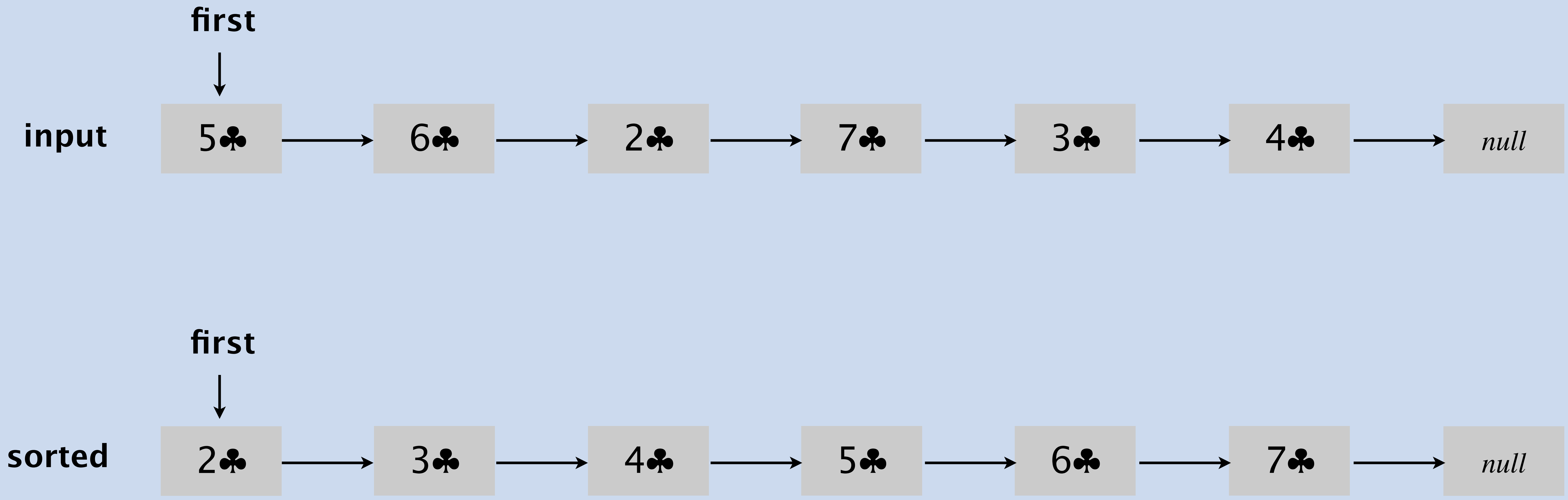
Problem. Given a singly linked list, rearrange its nodes in sorter order.

Application. Sort list of inodes to garbage collect in Linux kernel.

Version 0. $\Theta(n \log n)$ time, $\Theta(n)$ extra space.

Version 1. $\Theta(n \log n)$ time, $\Theta(\log n)$ extra space.

Version 2. $\Theta(n \log n)$ time, $\Theta(1)$ extra space.



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

Merging demo (Transylvanian–Saxon folk dance)

https://www.youtube.com/watch?v=XaqR3G_NVoo