http://algs4.cs.princeton.edu

2.2 Mergesort

- *mergesort*
- *bottom-up mergesort*
- *sorting complexity*
- *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^{th} century in science and engineering.

Mergesort. [this lecture]

...

Quicksort. [next lecture]

...

http://algs4.cs.princeton.edu

2.2 Mergesort

- *mergesort*
- *bottom-up mergesort*
- *sorting complexity*
- *comparators*
- *stability*

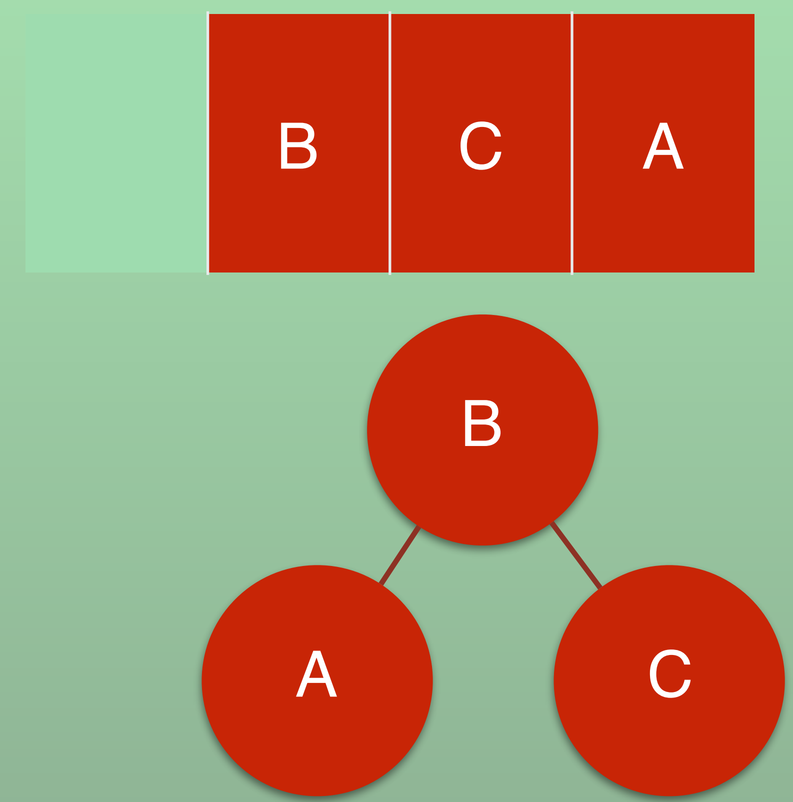
How to improve sorting?

- Q. what’s the easiest non-empty list to sort?
  - A. a list of length one!
- Q. What is the magic secret of binary search?
  - A. divide-and-conquer
- Q. Why does divide-and-conquer work?
  - A. it folds a list into a tree, thus converting a two-dimensional problem (quadratic) into a one-and-a-half-dimensional problem (linearithmic).

Quadratic vs. Linearithmic

	B	C	A
A	?	?	?
B	?	?	?
C	?	?	?

3x3=9 potential comparisons



3x2=6 potential comparisons

Mergesort

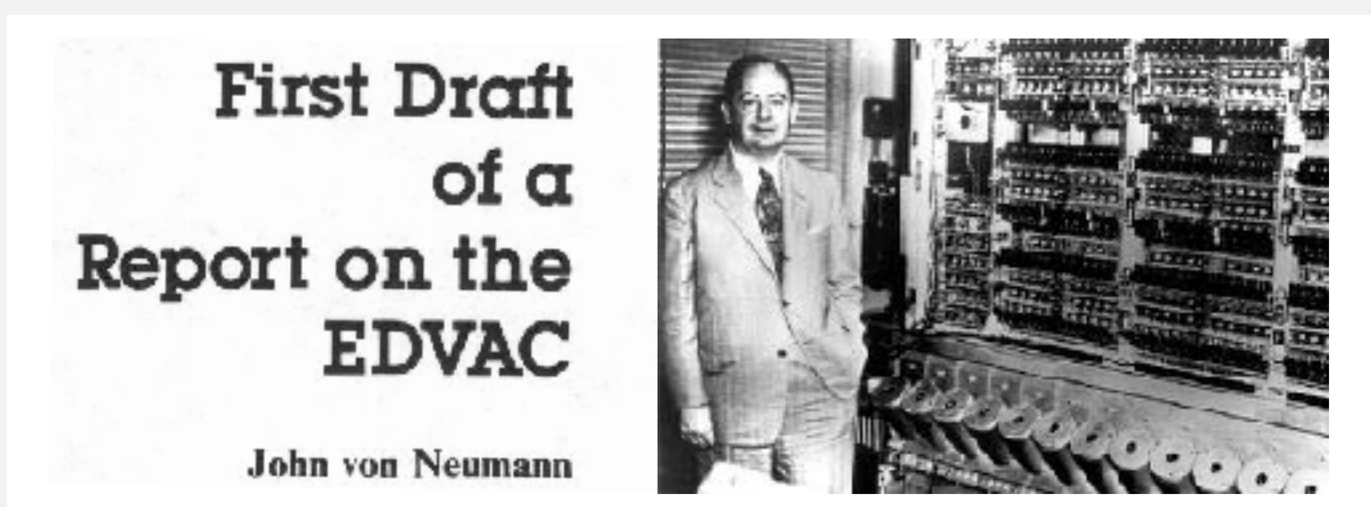
Basic plan.

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

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

Mergesort overview

1945:



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**

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

sorted

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];

  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++];
  }
}
```

copy a to aux

merge aux into a

Four cases:
(1) Nothing left on left;
(2) Nothing left on right;
(3) Left element is smaller;
(4) Right element is smaller.

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);
   }
}
```

Mergesort: trace

a[]

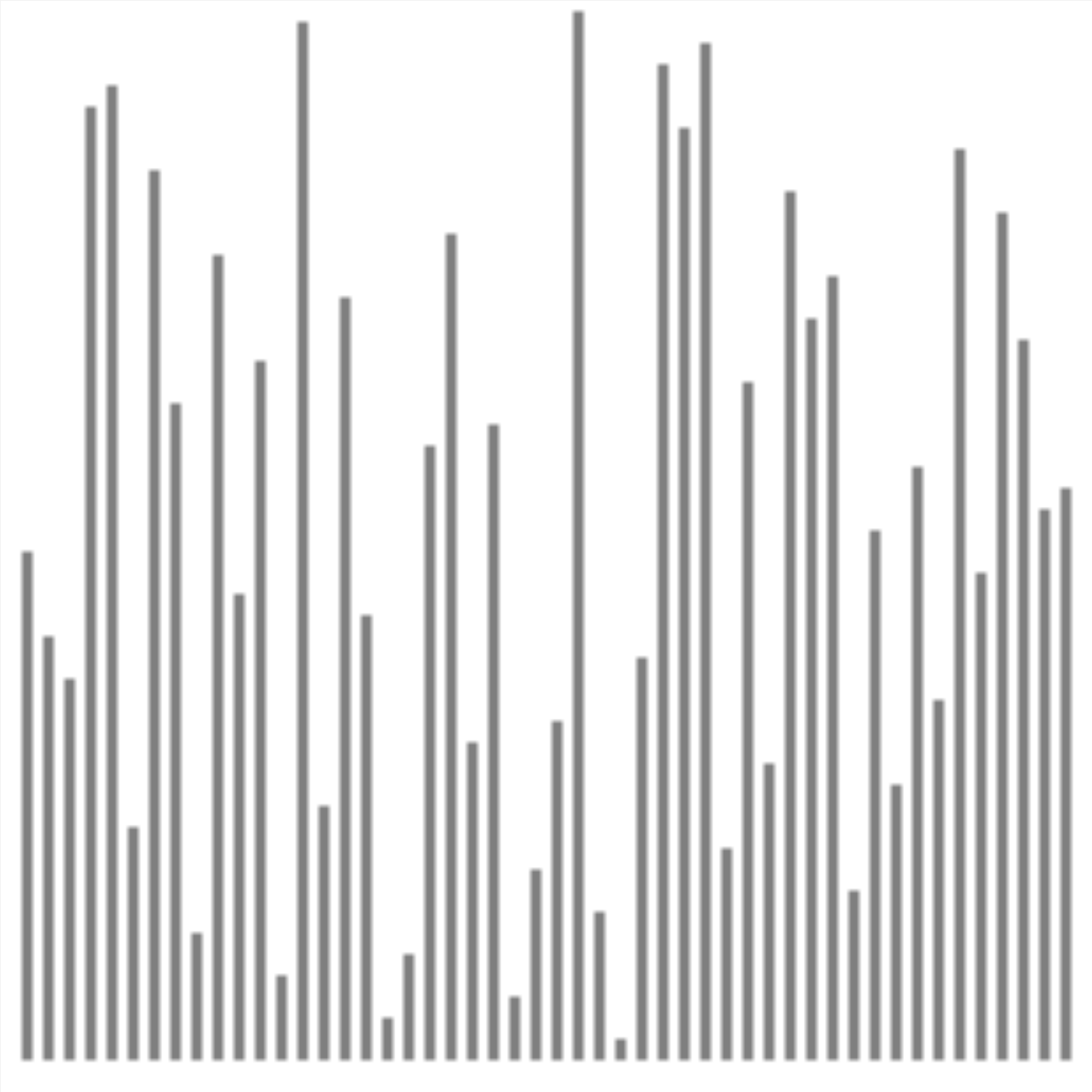
call (lo, mid, hi)	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	M	E	R	G	E	S	O	R	T	E	X	A	M	P	L	E
merge(a, aux, 0, 0, 1)	E	M	R	G	E	S	O	R	T	E	X	A	M	P	L	E
merge(a, aux, 2, 2, 3)	E	M	G	R	E	S	O	R	T	E	X	A	M	P	L	E
merge(a, aux, 0, 1, 3)	E	G	M	R	E	S	O	R	T	E	X	A	M	P	L	E
merge(a, aux, 4, 4, 5)	E	G	M	R	E	S	O	R	T	E	X	A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

lo → first argument after aux; hi → last argument

result after recursive call

Mergesort: animation

50 random items



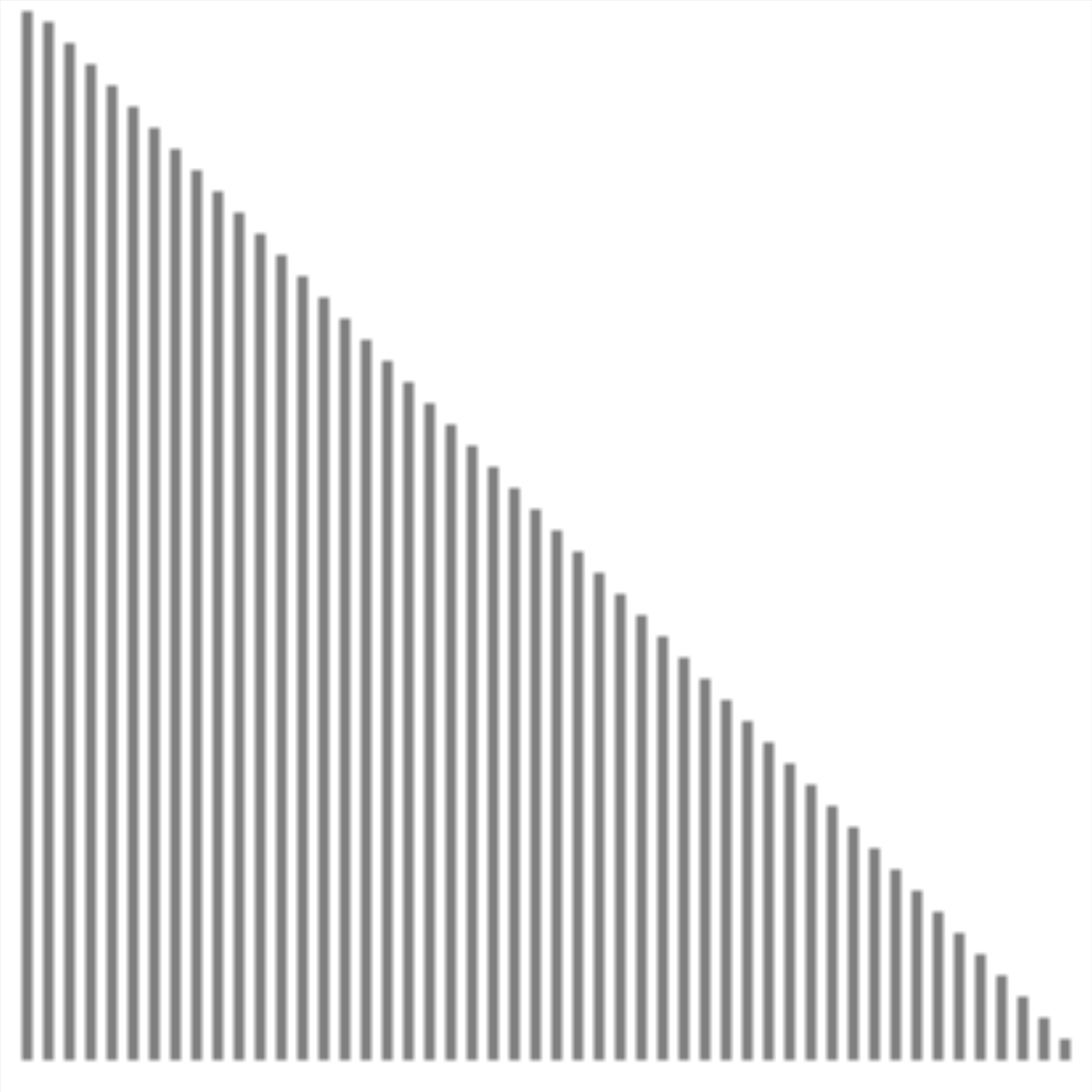
http://www.sorting-algorithms.com/merge-sort

14.5 secs

▲ algorithm position
in order
current subarray
not in order

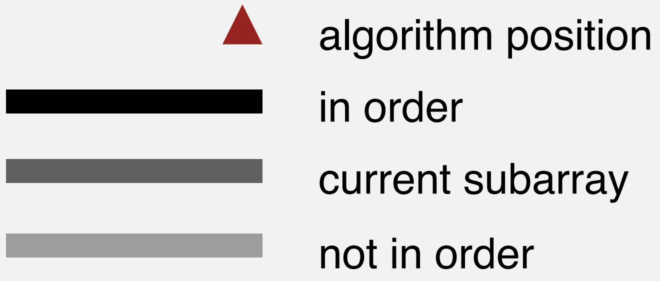
Mergesort: animation

50 reverse-sorted items



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

14.5 secs



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 $\mathbf{C}(N)$ to mergesort an array of length N satisfies the recurrence:

$$\mathbf{C}(N) \leq \underbrace{\mathbf{C}(\lceil N/2 \rceil)}_{\text{left half}} + \underbrace{\mathbf{C}(\lfloor N/2 \rfloor)}_{\text{right half}} + \underbrace{N}_{\text{merge}} \quad \text{for } N > 1, \text{ with } \mathbf{C}(1) = 0.$$

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

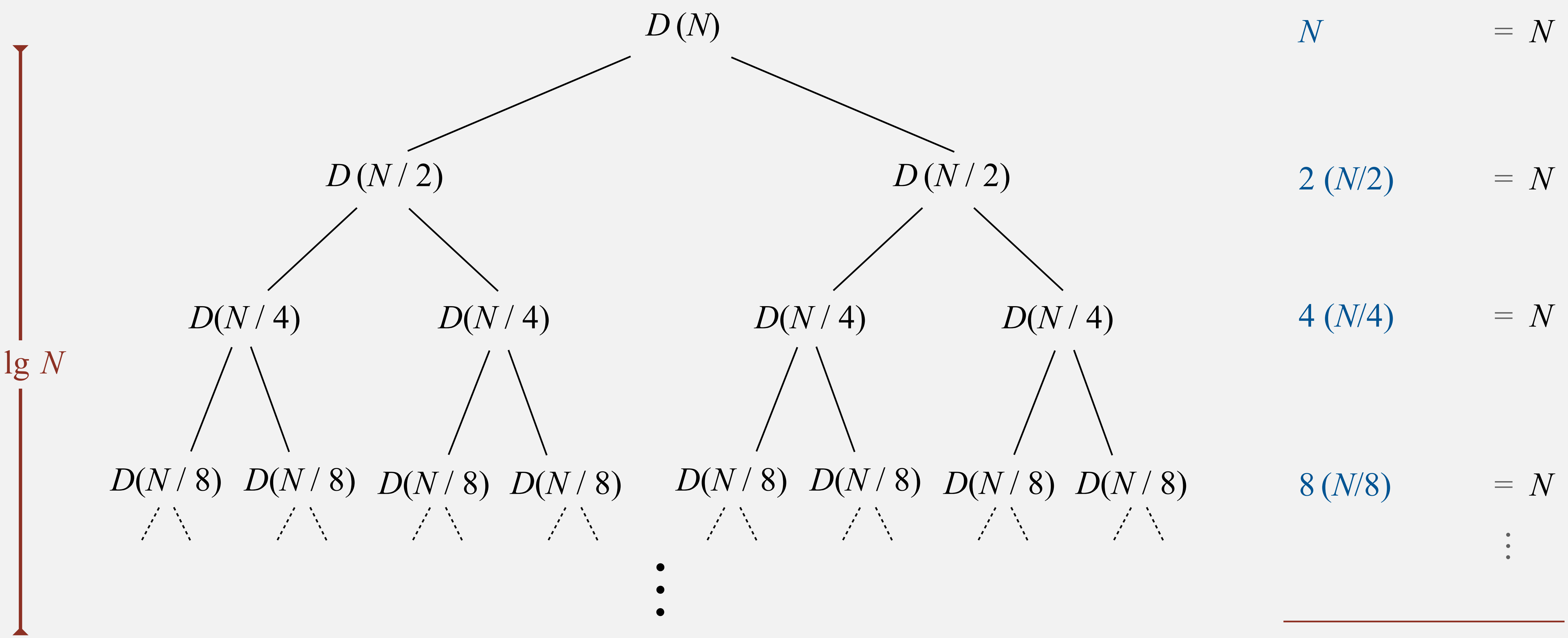
$$\boldsymbol{D}(N) = 2\,\boldsymbol{D}(N/2) + N, \text{ for } N > 1, \text{ with } \boldsymbol{D}(1) = 0.$$

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

Divide-and-conquer: proof by visualization

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

Pf 1. [assuming N is a power of 2] The number of elements involved at each level is still N. But the number of levels is *lg N* (not *N* as in the case of a N^2 algorithm).



Total merge compares $D(N) = N \lg N$

Divide-and-conquer recurrence: proof by induction

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

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

- Base case: $N = 1$.
- Inductive hypothesis: $D(N) = N \lg N$.
- Goal: show that $D(2N) = (2N) \lg (2N)$.

$$
\begin{aligned}
D(2N) &= 2\,D(N) + 2N && \text{given} \\
&= 2\,N \lg N + 2N && \text{inductive hypothesis: } D(N) = N \lg N \\
&= 2\,N\,(\lg (2N) - 1) + 2N && \text{algebra: } \lg 2N = \lg N + 1 \\
&= 2\,N \lg (2N) && \text{QED}
\end{aligned}
$$

Mergesort: number of array accesses

Proposition. Mergesort uses $\le 6\,N \lg N$ array accesses to sort an array of length N.

where does this factor of 6 come from?

Look again at the merge algorithm itself.

Pf sketch. The number of array accesses $\mathbf{A}(N)$ satisfies the recurrence:

$$\mathbf{A}(N) \le \mathbf{A}(\lceil N/2 \rceil) + \mathbf{A}(\lfloor N/2 \rfloor) + 6N \text{ for } N > 1, \text{ with } \mathbf{A}(1) = 0.$$

Key point. Any algorithm with the following structure takes $c\,N \log_k N$ time:

```
public static void linearithmic(int N)
{
    if (done(N)) return;
    linearithmic(N/k);
          // …
    linearithmic(N/k);
    linear(N);
}
```

terminating condition

solve k problems, each of 1/kth the size

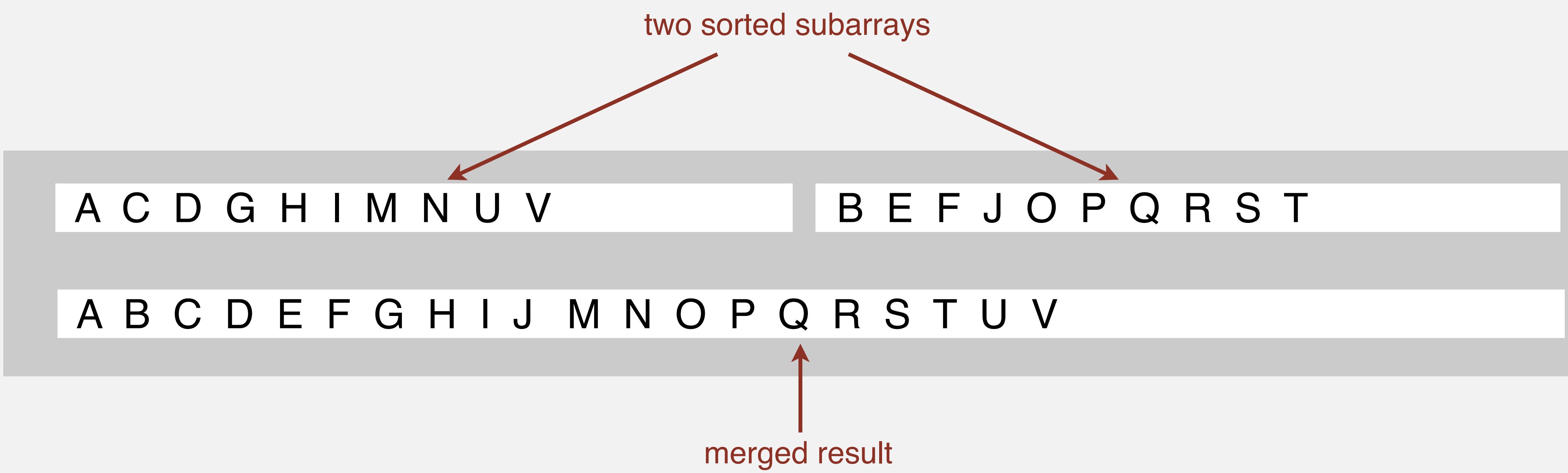
do a linear amount of work cN

Notable examples. FFT, hidden-line removal, Kendall-tau distance, …

Mergesort analysis: memory

Proposition. Mergesort uses extra space proportional to N.

Pf. The array aux[] needs to be of length N for the last merge.



Def. A sorting algorithm is in-place if it uses $\leq c \log N$ extra memory.

Ex. Insertion sort, selection sort, shellsort.

Peer discussion: How can we improve upon mergesort? Come up with some ideas between yourselves (no books or internet!!). 5 minutes.

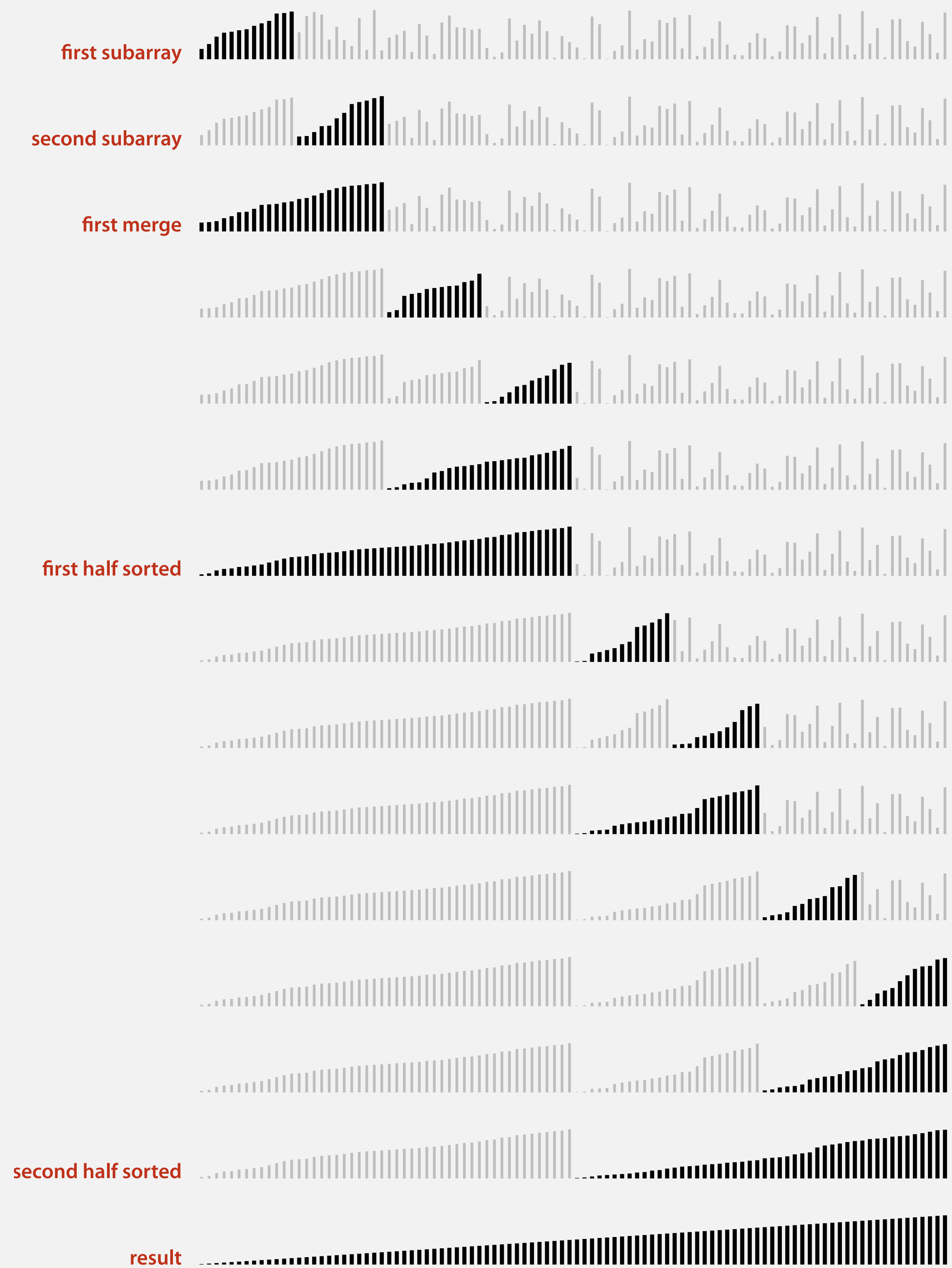
Mergesort: practical improvements (1)

Use insertion sort for small subarrays.

- Mergesort has too much overhead for tiny subarrays.
- Cutoff to insertion sort for ≈ 7 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 (2)

Skip merge if already in order.

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

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

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

```
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);
}
```

Mergesort: practical improvements (3)

Eliminate the copy to the auxiliary array. Save time (but not space) by switching the role of the input and auxiliary array in each recursive call.

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

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

merge from a[] to aux[]

assumes aux[] is initialized to a[] once, before recursive calls

switch roles of aux[] and a[]

Java 6 system sort

Basic algorithm for sorting objects = mergesort.

- Cutoff to insertion sort = 7.
- Stop-if-already-sorted test.
- Eliminate-the-copy-to-the-auxiliary-array trick.

Arrays.sort(a)

http://hg.openjdk.java.net/jdk8u/jdk8u/jdk/file/be44bff34df4/src/share/classes/java/util/Arrays.java

http://algs4.cs.princeton.edu

2.2 MERGESORT

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

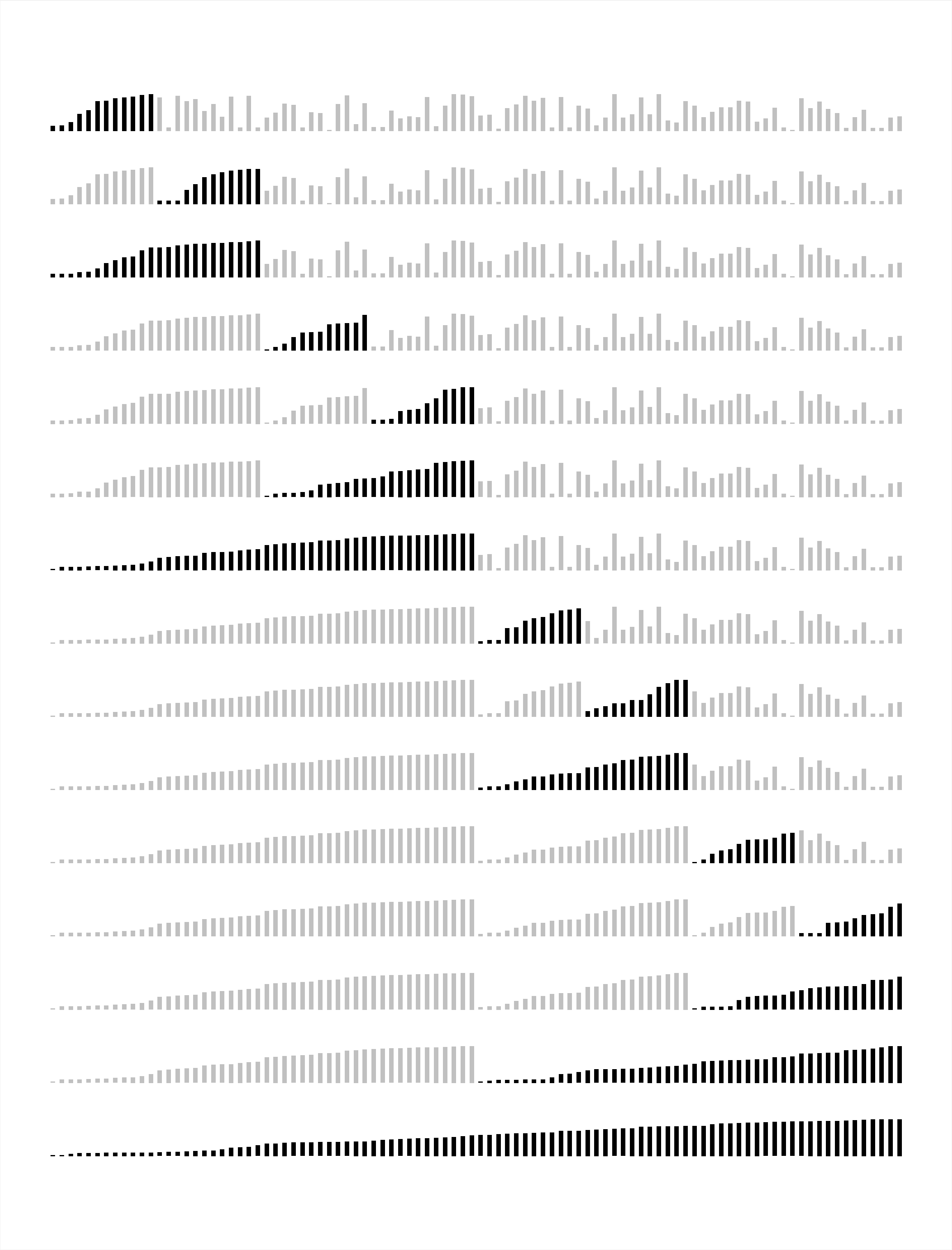
```
public class MergeBU
{
   private static void merge(...)
   {  /* as before */  }

   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));
   }
}
```

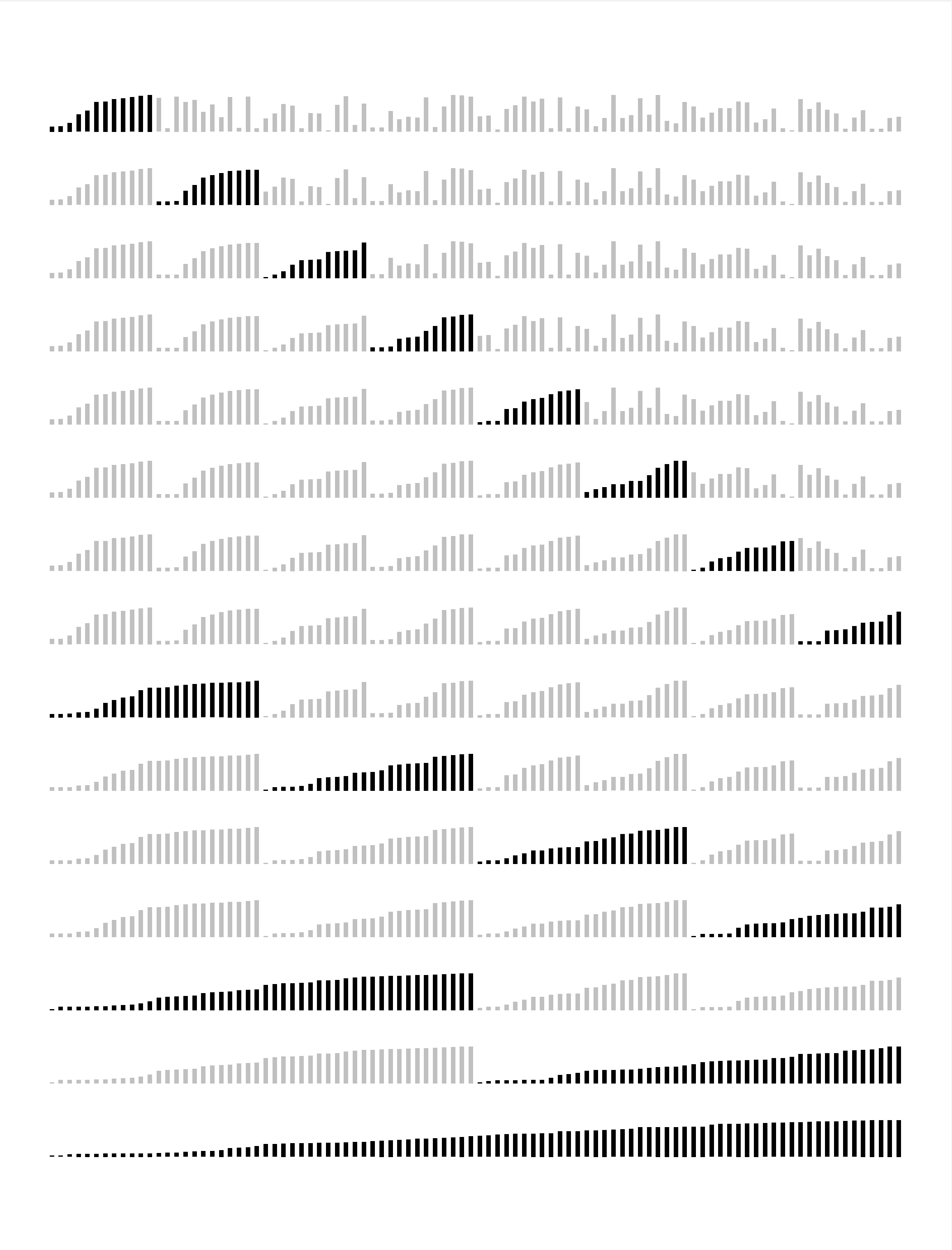
but about 10% slower than recursive,
top-down mergesort on typical systems

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.

Timsort

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

Tim Peters

Well, maybe—but it uses ideas that were originally published 9 years earlier by Peter McIlroy

Intro

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.

Consequence. Linear time on many arrays with pre-existing order.

Now widely used. Python, Java 7, GNU Octave, Android, ….

http://algs4.cs.princeton.edu

2.2 Mergesort

- mergesort
- bottom-up mergesort
- **sorting complexity**
- comparators
- stability

Complexity of sorting

Computational complexity. Framework to study efficiency of algorithms for solving a particular problem X.

Model of computation. Allowable operations.

Cost model. Operation count(s).

Upper bound. Cost guarantee provided by some algorithm for X.

Lower bound. Proven limit on cost guarantee of all algorithms for X.

Optimal algorithm. Algorithm with best possible cost guarantee for X.

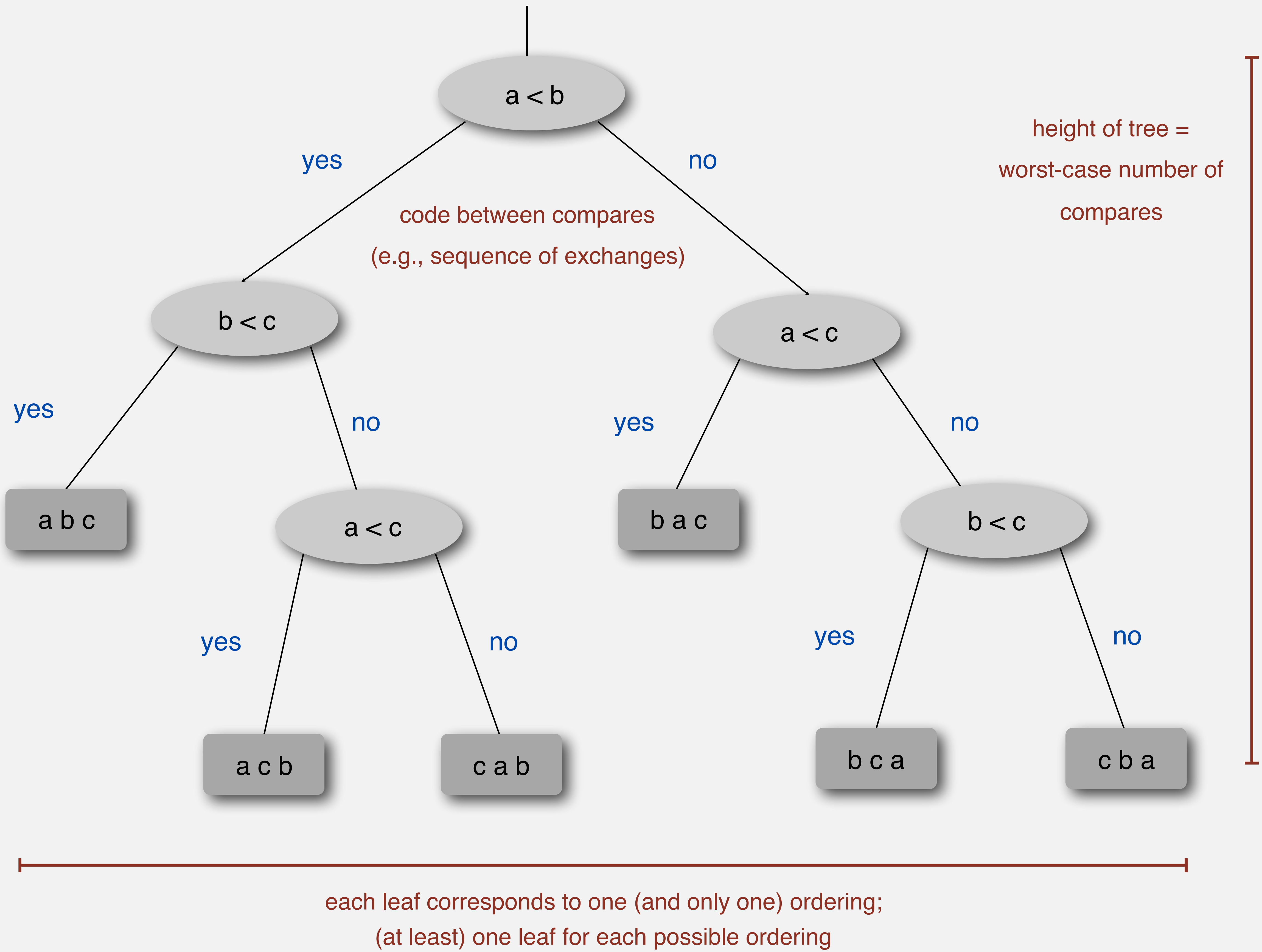
lower bound ~ upper bound

Example: sorting.

- Model of computation: decision tree.
- Cost model: # compares.
- Upper bound: $\sim N \lg N$ from mergesort.
- Lower bound:
- Optimal algorithm:

can access information only through compares (e.g., Java Comparable framework)

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



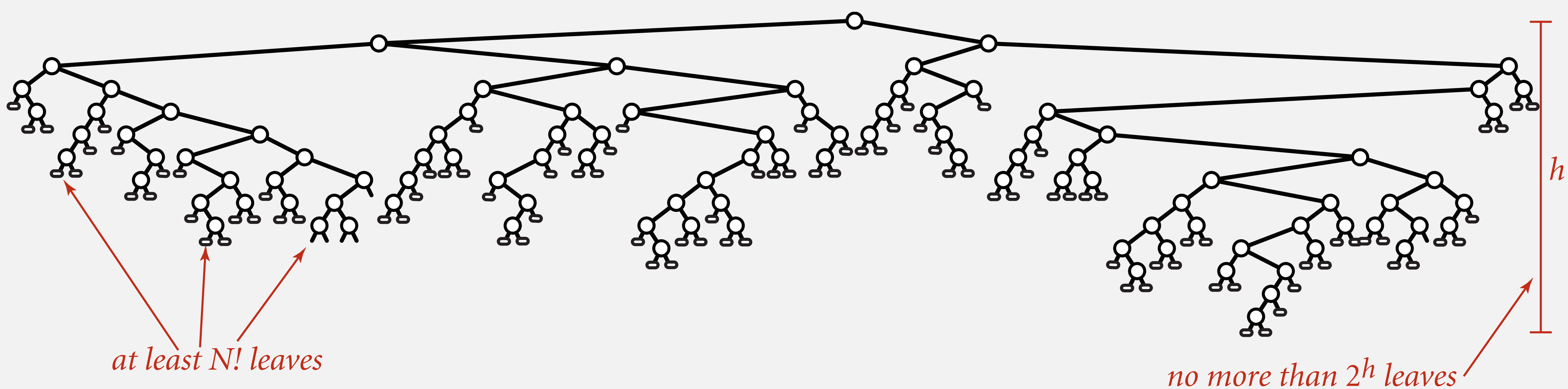
Compare-based lower bound for sorting

Proposition. Any compare-based sorting algorithm must use at least $\lg(N!) \sim N \lg N$ compares in the worst-case.

This relates back to the earlier module on entropy.

Pf.

- Assume array consists of N distinct values a_1 through a_N.
- Worst case dictated by height h of decision tree.
- Binary tree of height h has at most 2^h leaves.
- N ! different orderings $\Rightarrow$ at least N ! leaves.



Compare-based lower bound for sorting

Proposition. Any compare-based sorting algorithm must use at least $\lg(N!) \sim N \lg N$ compares in the worst-case.

Pf.

- Assume array consists of N distinct values a_1 through a_N.
- Worst case dictated by height h of decision tree.
- Binary tree of height h has at most 2^h leaves.
- N ! different orderings $\Rightarrow$ at least N ! leaves.

$$2^h \geq \#\text{ leaves} \geq N!$$
$$\Rightarrow h \geq \lg(N!) \sim N \lg N$$

Stirling's formula

Complexity results in context

Compares? Mergesort is optimal with respect to number compares.

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

Lessons. Use theory as a guide.

Ex. Design sorting algorithm that guarantees $\frac{1}{2} N \lg N$ compares?

Ex. Design sorting algorithm that is both time- and space-optimal?

Complexity results in context (continued)

Lower bound may not hold if the algorithm can take advantage of:

- The initial order of the input.

  Ex: insertion sort requires only a linear number of compares on partially-sorted arrays.

- The distribution of key values.

  Ex: 3-way quicksort requires only a linear number of compares on arrays with a constant number of distinct keys. [stay tuned]

- The representation of the keys.

  Ex: radix sort requires no key compares — it accesses the data via character/digit compares.

http://algs4.cs.princeton.edu

2.2 Mergesort

- mergesort
- bottom-up mergesort
- sorting complexity
- **comparators**
- stability

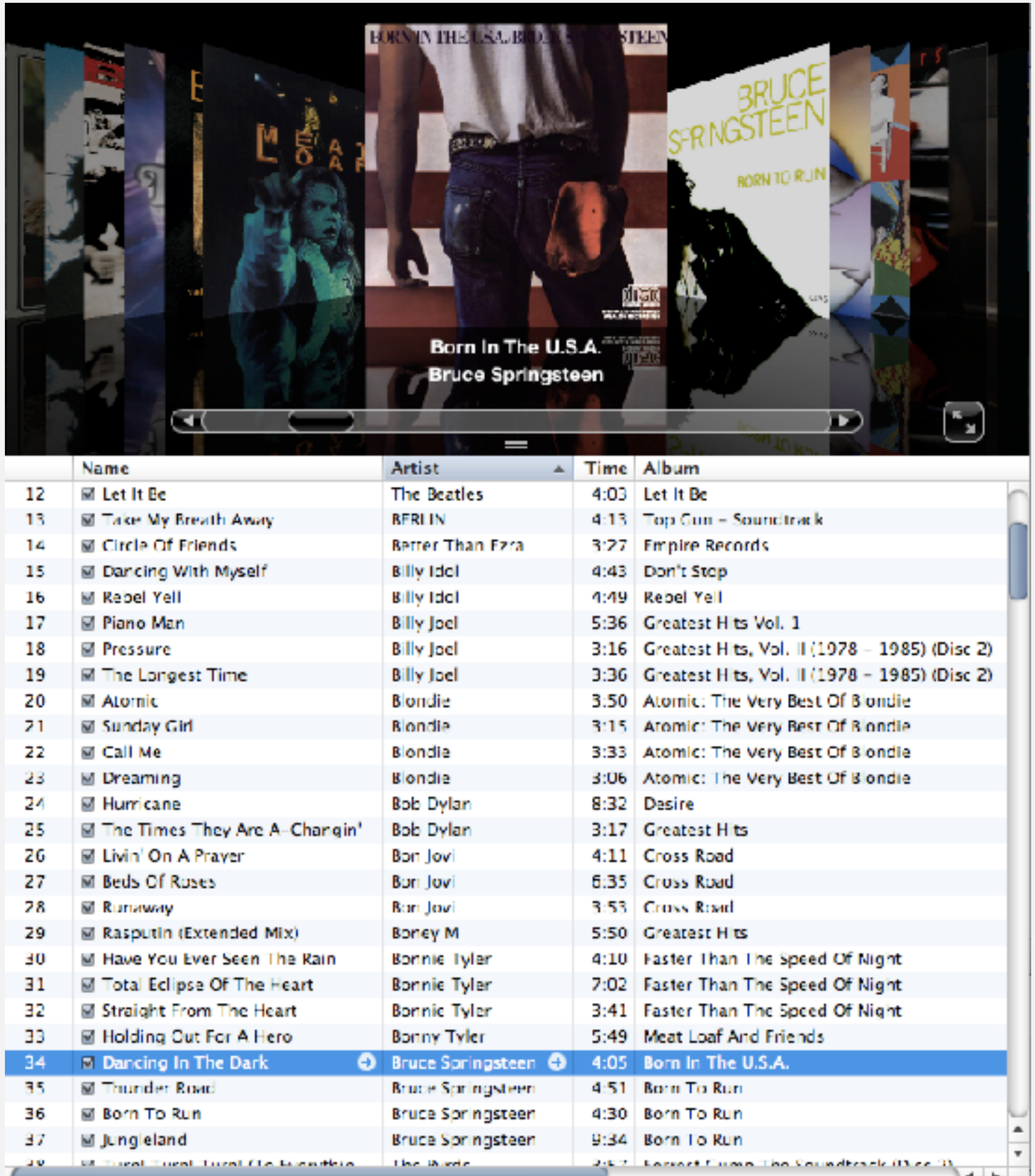
Sort countries by gold medals

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



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

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	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	Chaiyya Chaiyya	Sukhwinder 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)
   {
      int cfy = Integer.compare(this.year, that.year);
      if (cfy != 0) return cfy;
      int cfm = Integer.compare(this.month, that.month);
      if (cfm != 0) return cfm;
      return Integer.compare(this.day, that.day);
   }
}
```

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
natural order	Now is the time
case insensitive	is Now the time
Spanish language	café cafetero cuarto churro nube ñoño
British phone book	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().

uses natural order

uses alternate order defined by Comparator<String> object

```
String[] a;
...
Arrays.sort(a);
...
Arrays.sort(a, String.CASE_INSENSITIVE_ORDER);
...
Arrays.sort(a, Collator.getInstance(new Locale("es")));
...
Arrays.sort(a, new BritishPhoneBookOrder());
...
```

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--)
         swap(a, j, j-1);
}

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

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

Which sort is this?

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;  }
   }
}
```

this trick works here
since no danger of overflow

Comparator interface: implementing

To implement a comparator:

- Define a (inner) 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

http://algs4.cs.princeton.edu

2.2 Mergesort

- mergesort
- bottom-up mergesort
- sorting complexity
- 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?

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

no longer sorted by time

still sorted by time

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--)
            swap(a, j, j-1);
   }
}
```

i	j	0	1	2	3	4
0	0	B_1	A_1	A_2	A_3	B_2
1	0	A_1	B_1	A_2	A_3	B_2
2	1	A_1	A_2	B_1	A_3	B_2
3	2	A_1	A_2	A_3	B_1	B_2
4	4	A_1	A_2	A_3	B_1	B_2
		A_1	A_2	A_3	B_1	B_2

Pf. Equal items never move past each other.

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;
         swap(a, i, min);
      }
   }
}
```

i	min	0	1	2
0	2	B_1	B_2	A
1	1	A	B_2	B_1
2	2	A	B_2	B_1
		A	B_2	B_1

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

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)
               swap(a, j, j-h);
         }
         h = h/3;
      }
   }
}
```

h	0	1	2	3	4
	B_1	B_2	B_3	B_4	A_1
4	A_1	B_2	B_3	B_4	B_1
1	A_1	B_2	B_3	B_4	B_1
	A_1	B_2	B_3	B_4	B_1

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_1	A_2	A_3	B	D		A_4	A_5	C	E	F	G

Pf. Takes from left subarray if equal keys.

Sorting summary

	inplace?	stable?	best	average	worst	remarks
selection	✔		$\frac{1}{2} N^2$	$\frac{1}{2} N^2$	$\frac{1}{2} N^2$	N exchanges
insertion	✔	✔	N	$\frac{1}{4} N^2$	$\frac{1}{2} N^2$	use for small N or partially ordered
shell	✔		$N \log_3 N$	?	$c N^{3/2}$	tight code; subquadratic
merge		✔	$\frac{1}{2} N \lg N$	$N \lg N$	$N \lg N$	$N \log N$ guarantee; stable
timsort		✔	N	$N \lg N$	$N \lg N$	improves mergesort when preexisting order
?	✔	✔	N	$N \lg N$	$N \lg N$	*holy grail* of sorting