INTRODUCTION TO ALGORITHMS

LECTURE 5: MERGE SORTING ALGORITHM

Yao-Chung Fan
yfan@nchu.edu.tw

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 20th century in science and engineering.

Mergesort. [this lecture]

Quicksort. [next lecture]

Mergesort

- ▸ *Top-down mergesort*
- ▸ *bottom-up mergesort*
- ▸ *comparators*
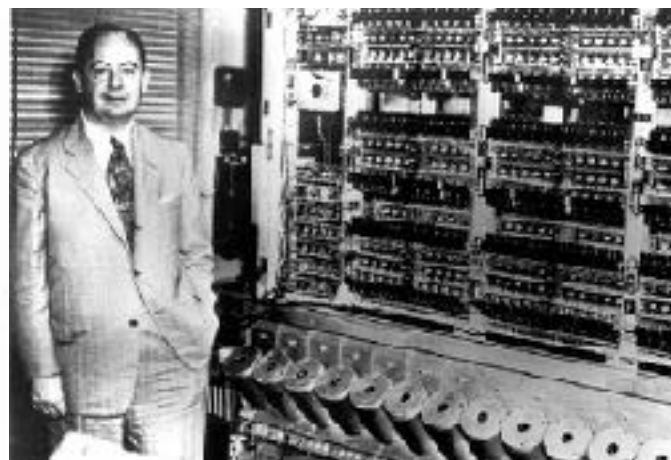- ▸ *stability*

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

First Draft of a Report on the EDVAC

John von Neumann

divide and conquer

A divide and conquer algorithm works by recursively breaking down a problem into two or more sub-problems of the same (or related) type (divide), until these become simple enough to be solved directly (conquer)
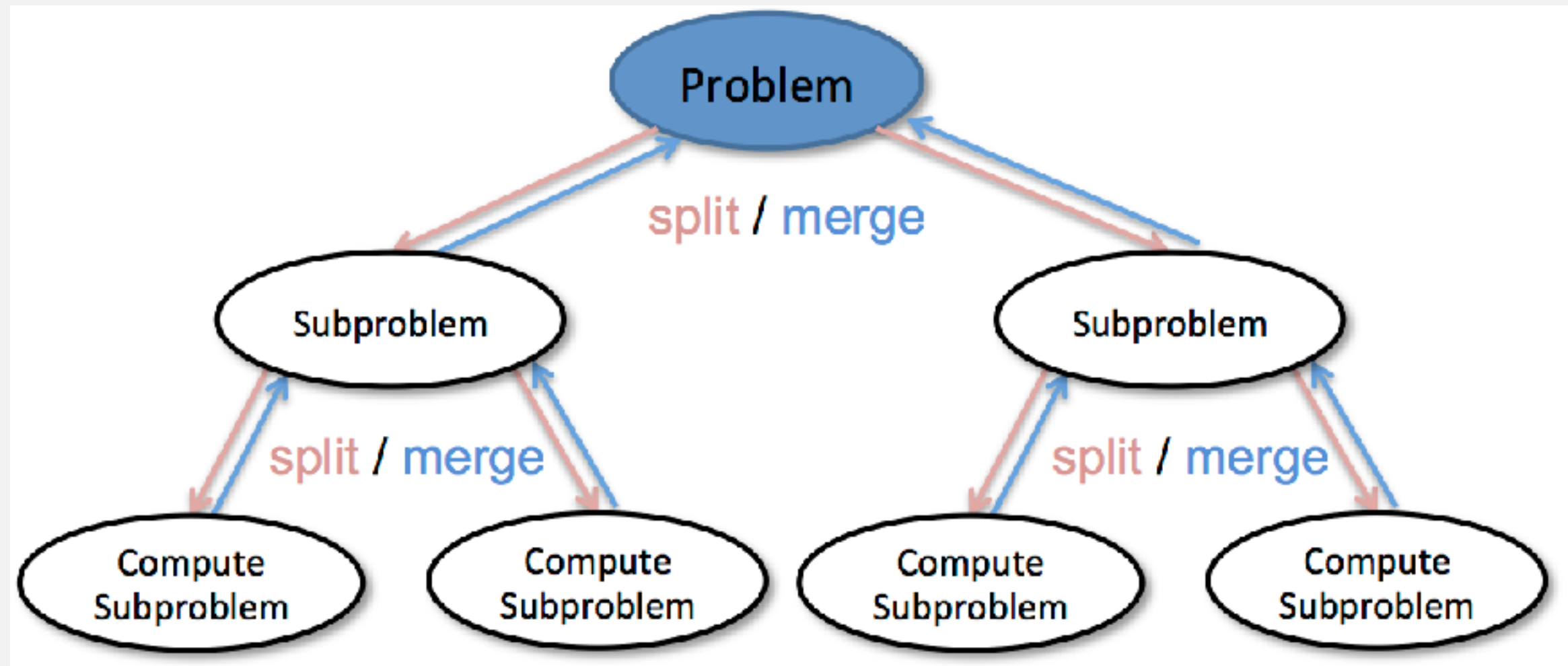
About recursion :)

To understand recursion, one must first understand recursion.

— Stephen Hawking —

AZ QUOTES

divide and conquer (分而治之)

A divide and conquer algorithm works by recursively breaking down a problem into two or more sub-problems of the same (or related) type (divide), until these become simple enough to be solved directly (conquer)
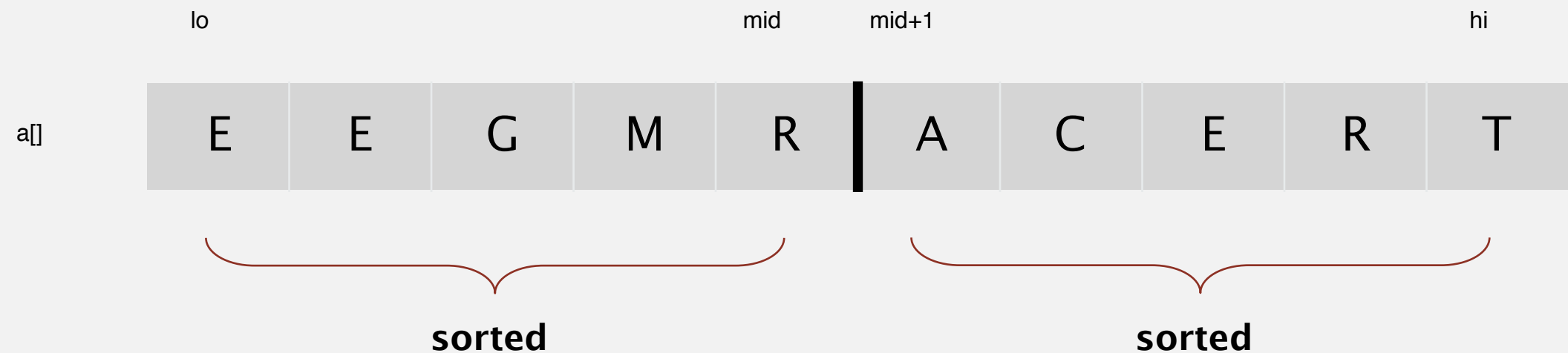

0   1   1   2   3   5   8   13   21   34   55   89   144   233   377   610   987

$$F_n = F_{n-1} + F_{n-2},$$

6

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

```java
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

```java
  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++];
  }
}
```

merge

	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: Java implementation

```java
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);
   }
}
```
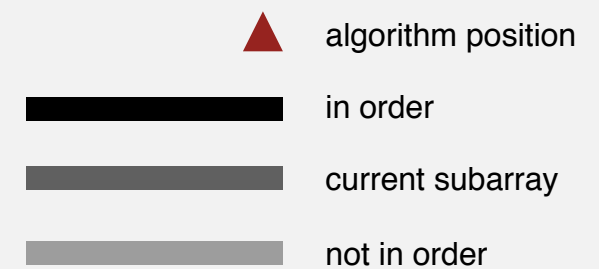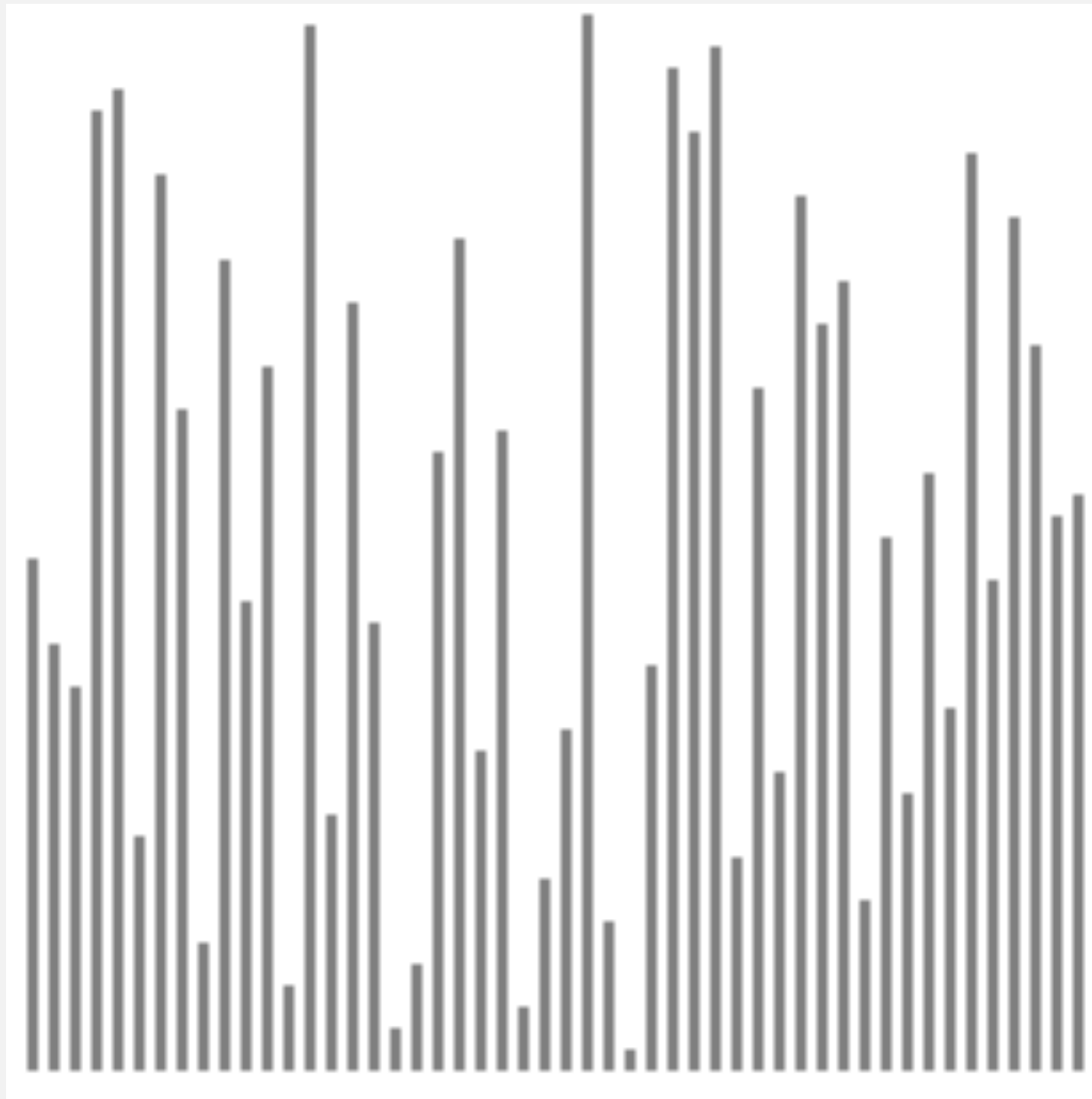
Mergesort: trace

Basic plan:

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

```
                                                          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
```
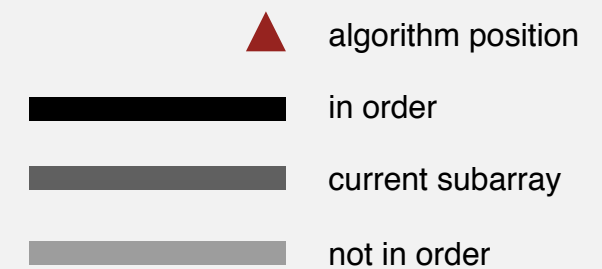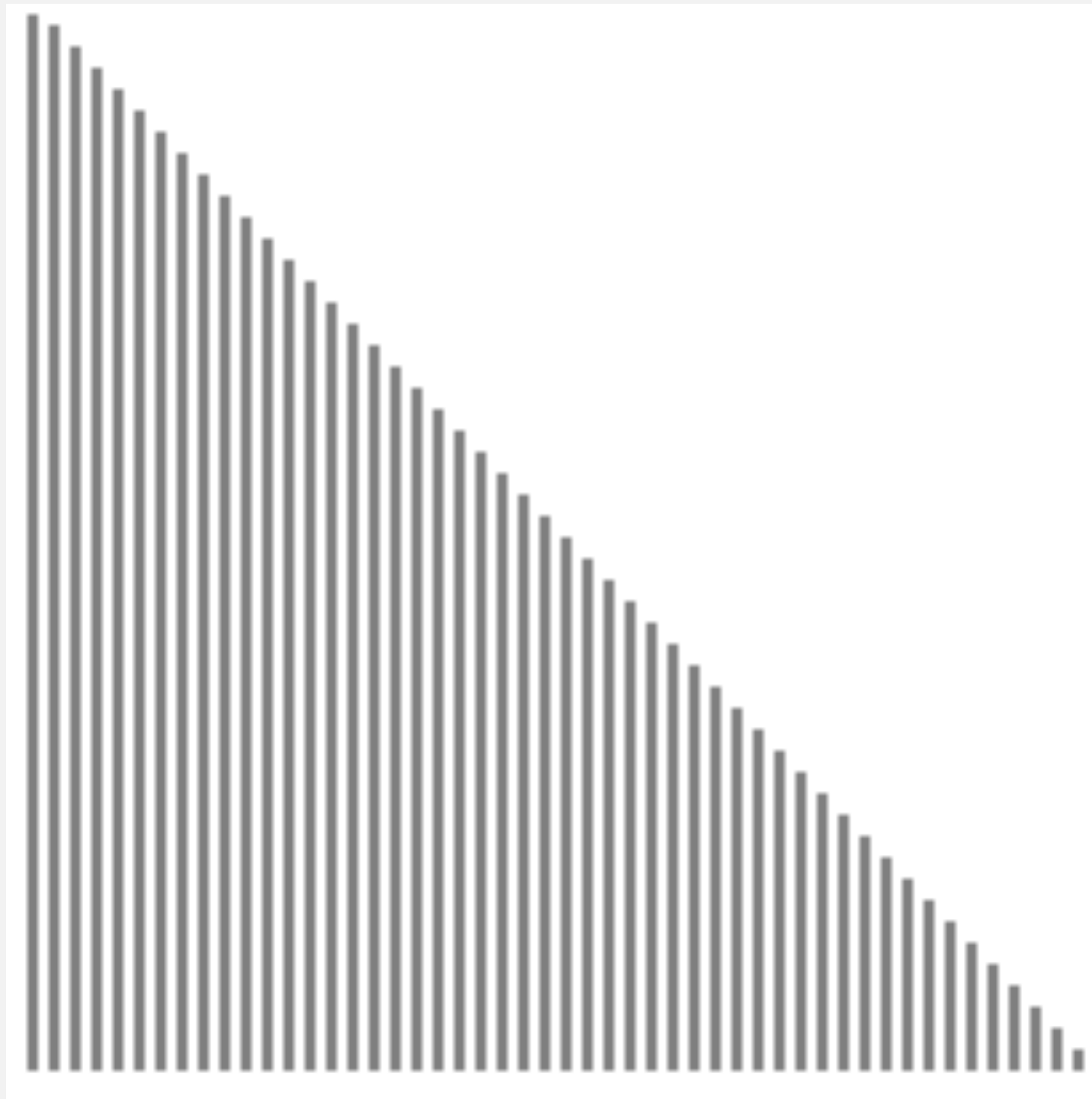
result after recursive call

Mergesort: animation

50 random items

▲ algorithm position

▬ in order

▬ current subarray

▬ not in order

Reason it is slow: excessive data movement.

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

Mergesort: animation

50 reverse-sorted items

▲ algorithm position

━━━ in order

━━━ current subarray

━━━ not in order

Reason it is slow: excessive data movement.

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			mergesort		
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
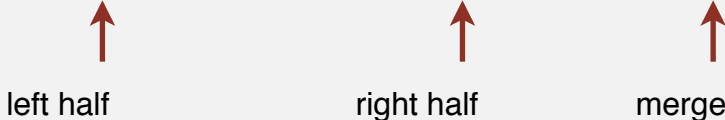
Bottom line. Good algorithms are better than supercomputers.

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 $D(N)$ to mergesort an array of length N satisfies the recurrence:

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

      ↑   ↑   ↑

  left half   right half   merge

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

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

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]

$$D(N) \qquad\qquad N \qquad = N$$
$$D(N/2) \qquad D(N/2) \qquad 2\,(N/2) \qquad = N$$
$$D(N/4) \quad D(N/4) \quad D(N/4) \quad D(N/4) \qquad 4\,(N/4) \qquad = N$$
$$D(N/8) \; D(N/8) \; D(N/8) \; D(N/8) \; D(N/8) \; D(N/8) \; D(N/8) \; D(N/8) \qquad 8\,(N/8) \qquad = N$$

$\lg N$
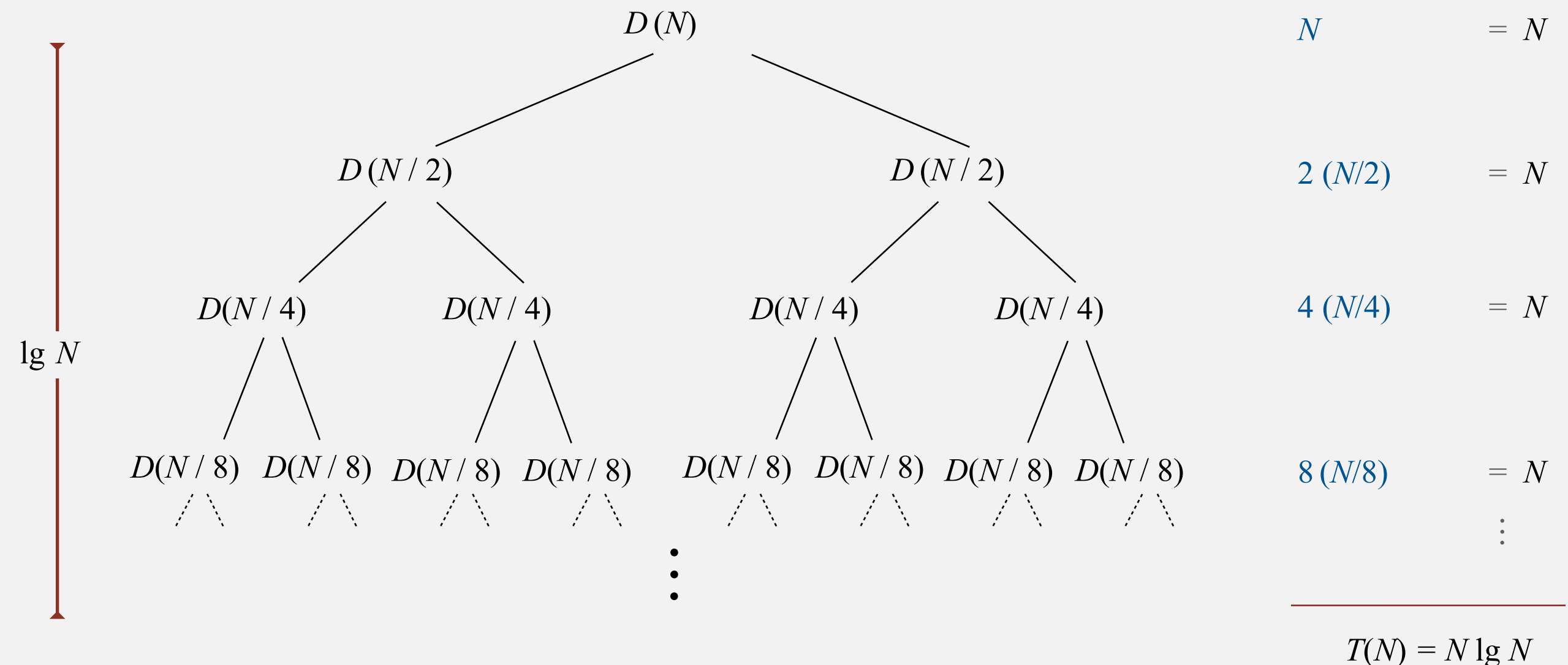
$$T(N) = N \lg N$$

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]

$$D(N) \;=\; 2\,D(N/2) \;+\; N$$ given

$$D(N)/N \;=\; 2\,D(N/2)/N \;+\; 1$$ divide both sides by N

$$=\; D(N/2)/(N/2) \;+\; 1$$ algebra

$$=\; D(N/4)/(N/4) \;+\; 1 \;+\; 1$$ apply to first term

$$=\; D(N/8)/(N/8) \;+\; 1 \;+\; 1 \;+\; 1$$ apply to first term again

$$. \,.\,.$$

stop applying, D(1) = 0

$$=\; D(N/N)/(N/N) + 1 + 1 \;+ ... +\; 1$$

$$=\; \lg N$$

Mer...

```
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];                                        2
                                                                    copy
  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++];           merge
    else if (less(aux[j], aux[i])) a[k] = aux[j++];
    else                        a[k] = aux[i++];    2
  }                              2
}
```

I ... array of

I ...

I ... rrence:

at most 6*N* array accesses (2*N* for the array copy, 2*N* for move back and at most 2*N* comparison)

Key point. Any algorithm with the following structure takes $N \log N$ time:

```
public static void linearithmic(int N)
{
    if (N == 0) return;
    linearithmic(N/2);      ←——— solve two problems
    linearithmic(N/2);      ←——— of half the size
    linear(N);              ←——————— do a linear amount of work
}
```

Mergesort: practical improvements

Use insertion sort for small subarrays.

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

array   10
insertion sort

```
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);
}
```

Visu

first subarray

second subarray

first merge

first half sorted

second half sorted

result
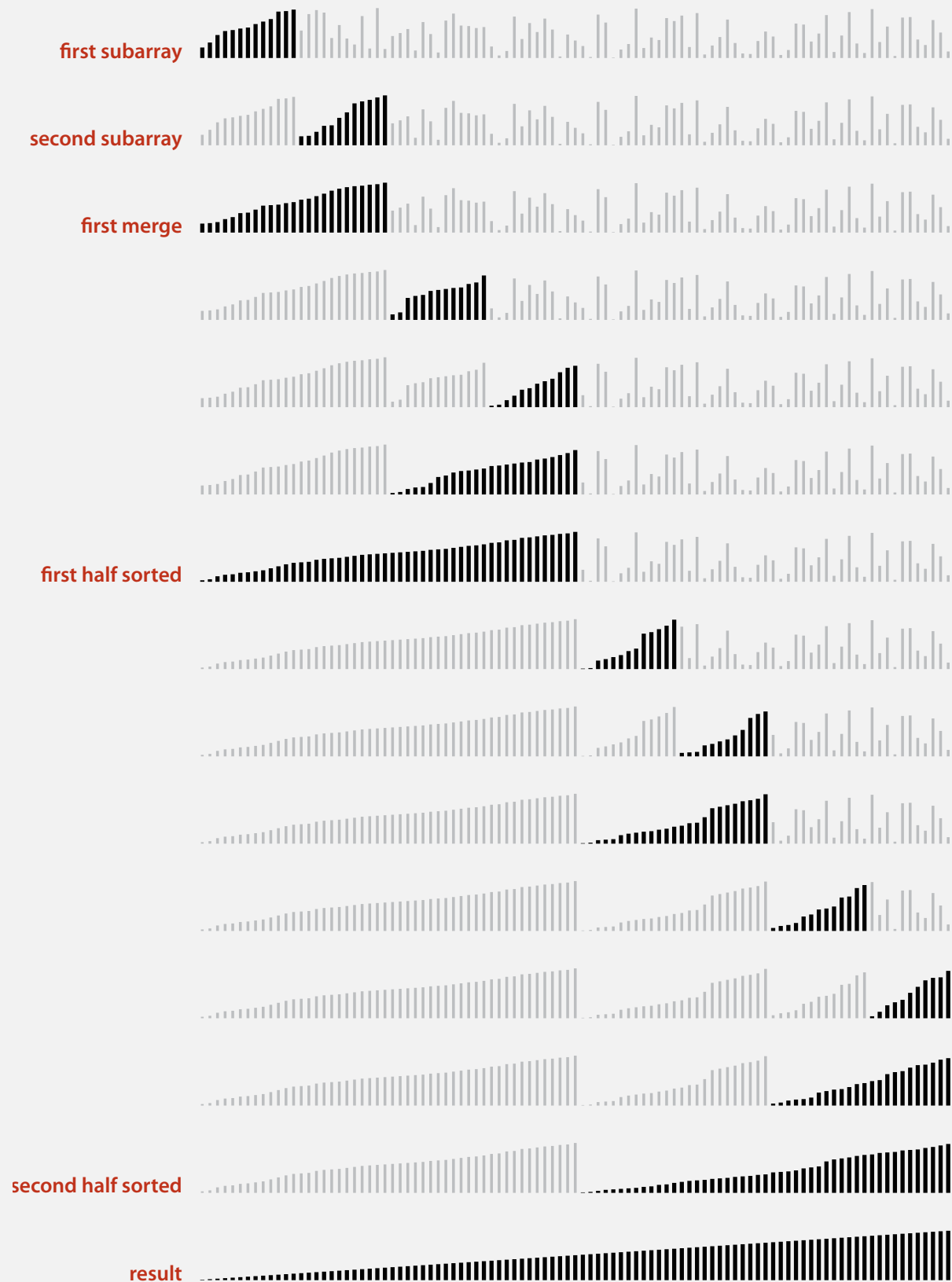
Mergesort: practical improvements

Stop if already sorted.

- 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);
}
```

Java 6 system sort

Basic algorithm for sorting objects = mergesort.

- Cutoff to insertion sort = 7.
- Stop-if-already-sorted test.

Arrays.sort(a)

http://www.java2s.com/Open-Source/Java/6.0-JDK-Modules/j2me/java/util/Arrays.java.html

Mergesort

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

24

Bottom-up mergesort: Java implementation

```java
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.

Visu
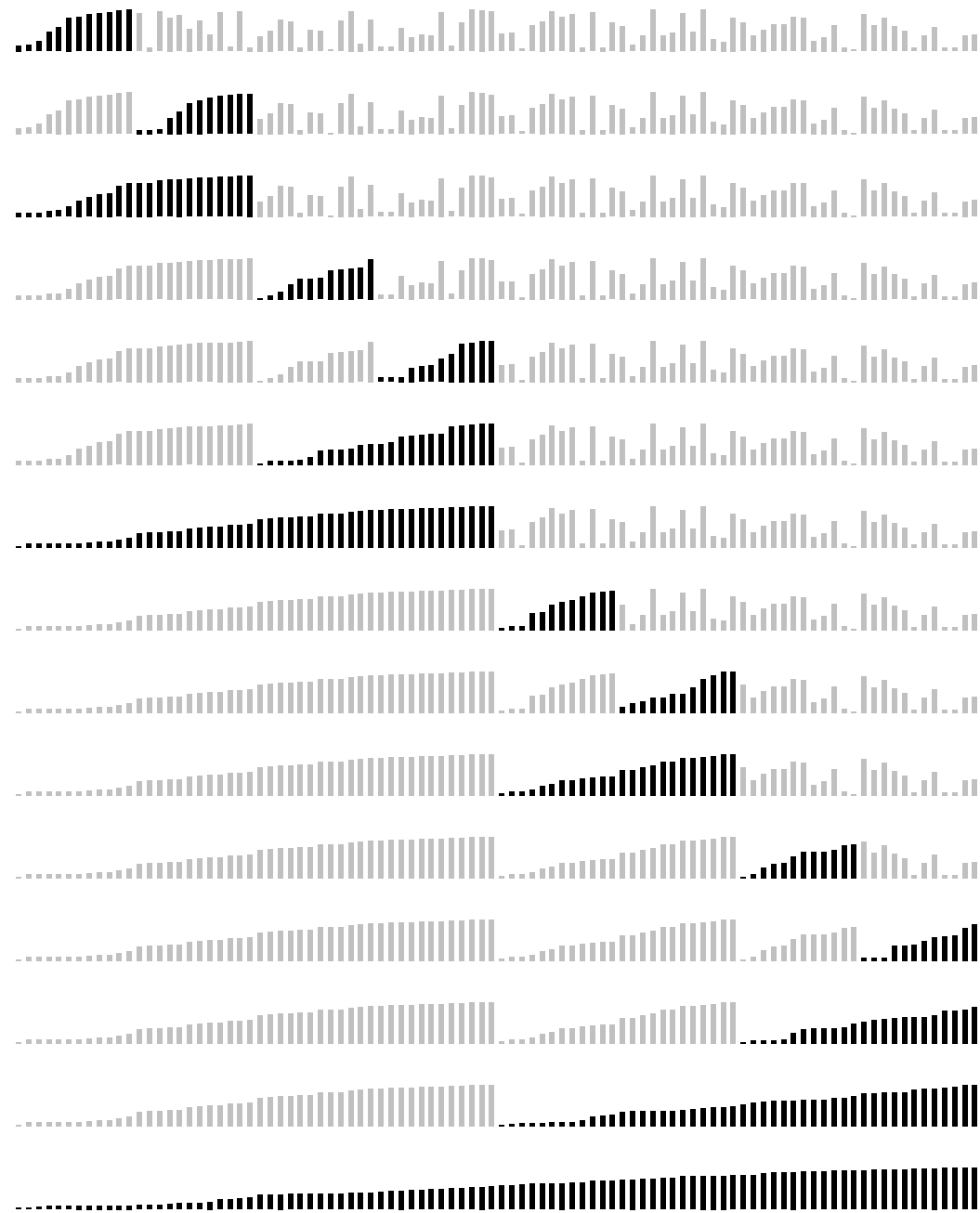
top-down mergesort (cutoff = 12)
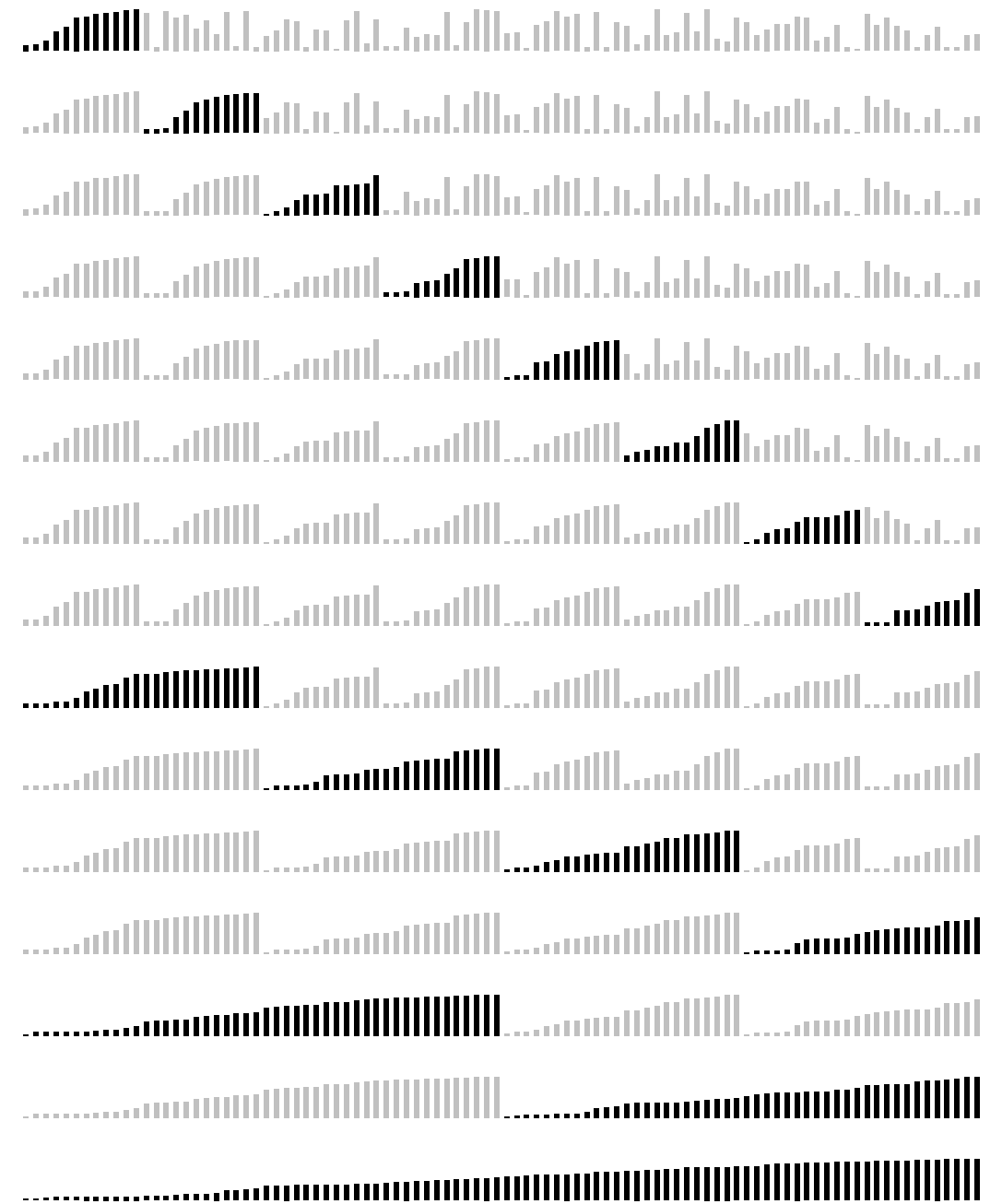
bottom-up mergesort (cutoff = 12)

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

Tim Peters

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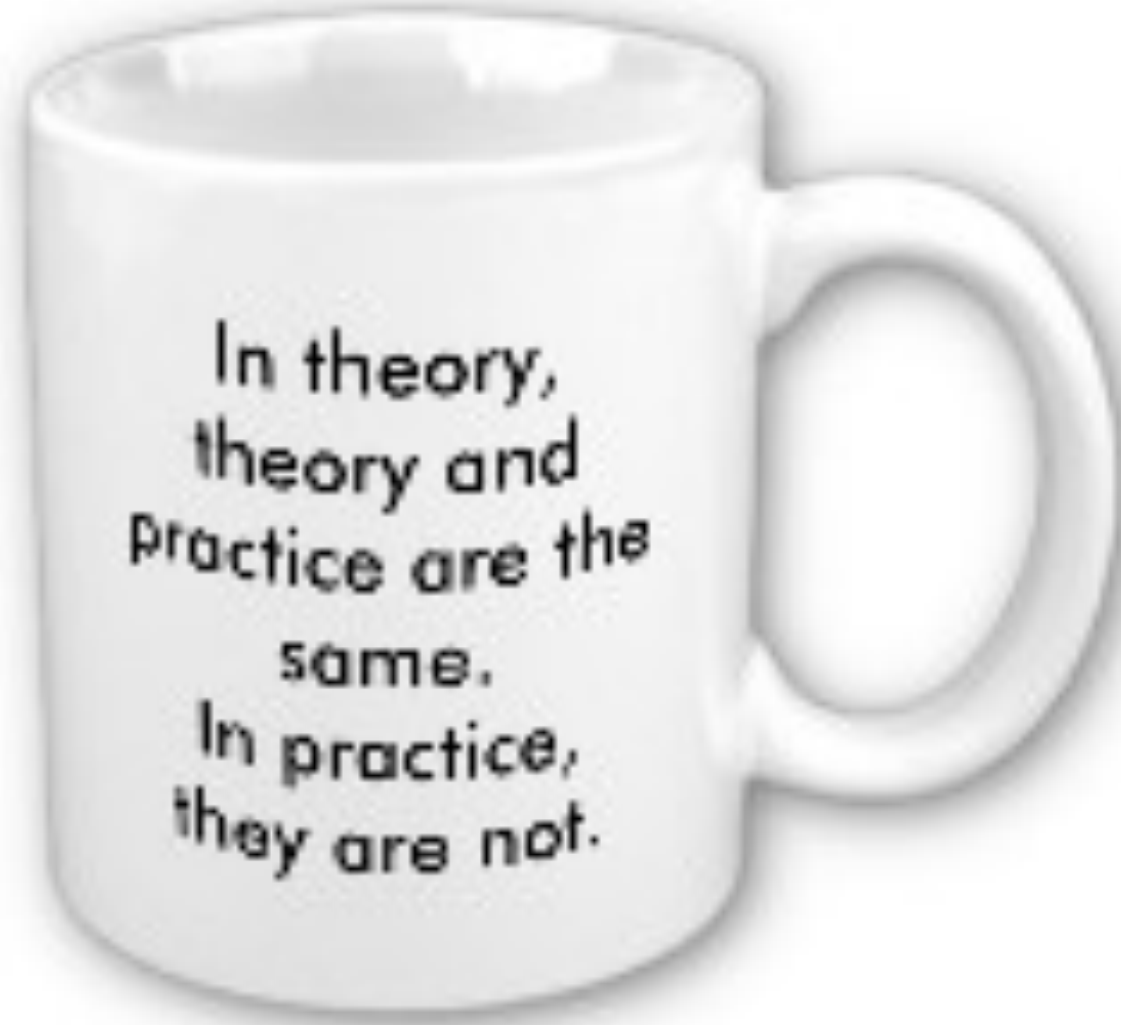
Consequence. Linear time on many arrays with pre-existing order.

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

Mergesort

- ▸ *Top-down mergesort*
- ▸ *bottom-up mergesort*
- ▸ **comparators**
- ▸ *stability*

The difference between theory and practice

Theory is "you know everything but nothing works"

Practice is "everything works but no one know why"

In this class, **theory and practice is combined**:

Nothing works and no one know why

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 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 – …
27	☑ Chaiyya Chaiyya	Sukhwinder Singh	6:11	Bombay Dreams

Sort songs by artist

Sort songs by track name

Comparable interface: review

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

```java
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;
    }
}
```

Comparator interface

Comparator interface: sort using an alternate order.

```
public interface Comparator<Key>

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

string order	example
natural order	Now is the time
case insensitive	is Now the time

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.

```java
public class Student
{
    public static final Comparator<Student> BY_NAME    = new ByName();
    public static final Comparator<Student> BY_SECTION = new BySection();

    private final String name;
    private final int section;

    ...

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

    private 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

Mergesort

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

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_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 (w.r.t sorting key) 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;
         exch(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

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_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

Pf by counterexample. Long-distance exchanges.

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.

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
A₁	A₂	A₃	B	D		A₄	A₅	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 sorting grail

Sleep Sort

```java
1.  public class SleepSort {
2.
3.      public static void main(String[] args) {
4.          int[] a = { 42, 51, 15, 24, 1, 69, 12, 8, 34, 10 };
5.          for (int i : a) {
6.              ThreadPrint t = new ThreadPrint(i);
7.              t.start();
8.          }
9.      }
10.  }
11.
12.  class ThreadPrint extends java.lang.Thread {
13.      private int i;
14.
15.      public ThreadPrint(int pi) {
16.          i = pi;
17.      }
18.
19.      public void run() {
20.          synchronized (this) {
21.              try {
22.                  wait(i);
23.                  notifyAll();
24.              } catch (InterruptedException e) {
25.                  e.printStackTrace();
26.              }
27.              System.out.print(i + ",");
28.          }
29.      }
30.  }
```