

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.

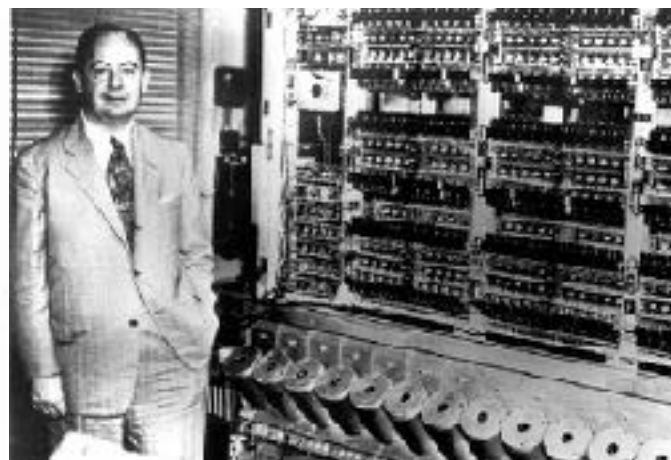
- Divide array into two halves.
- **Recursively** sort each half.
- Merge two halves.

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 :)



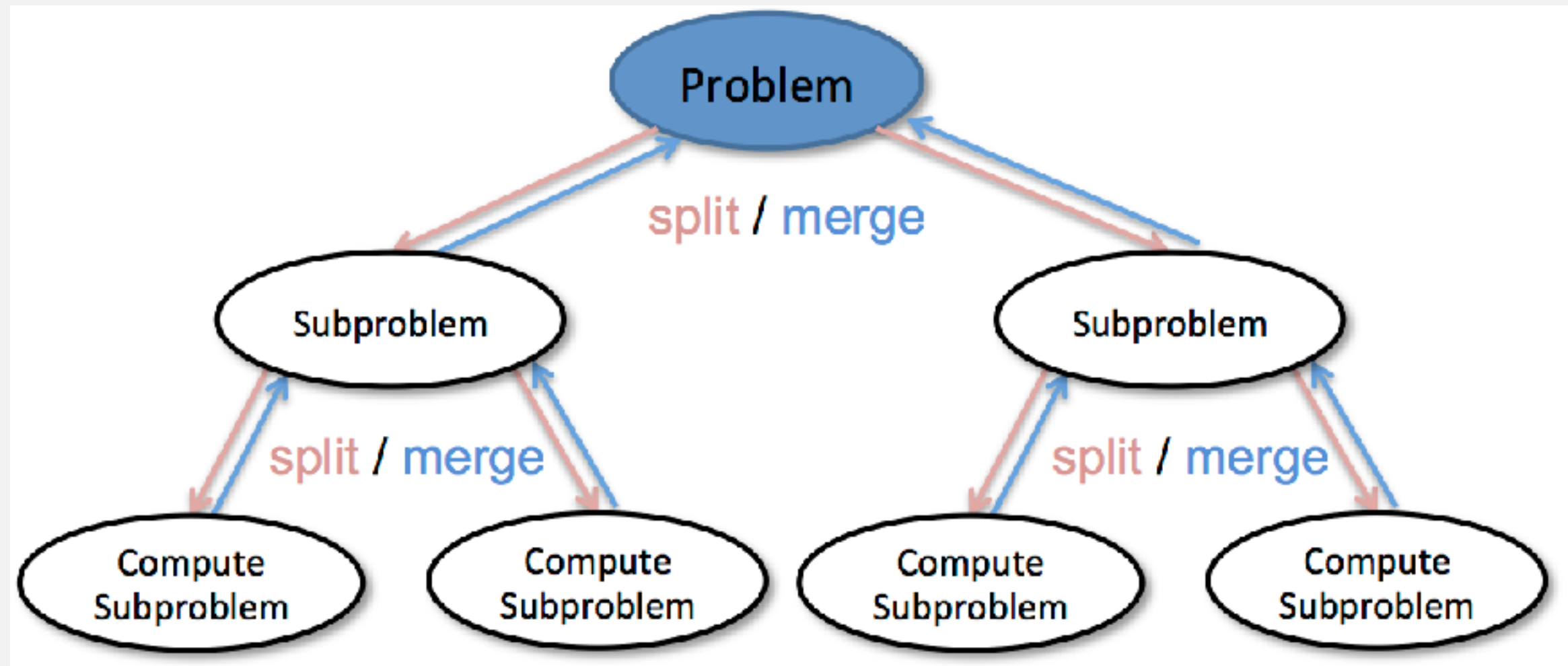
To understand recursion, one must
first understand recursion.

— *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)

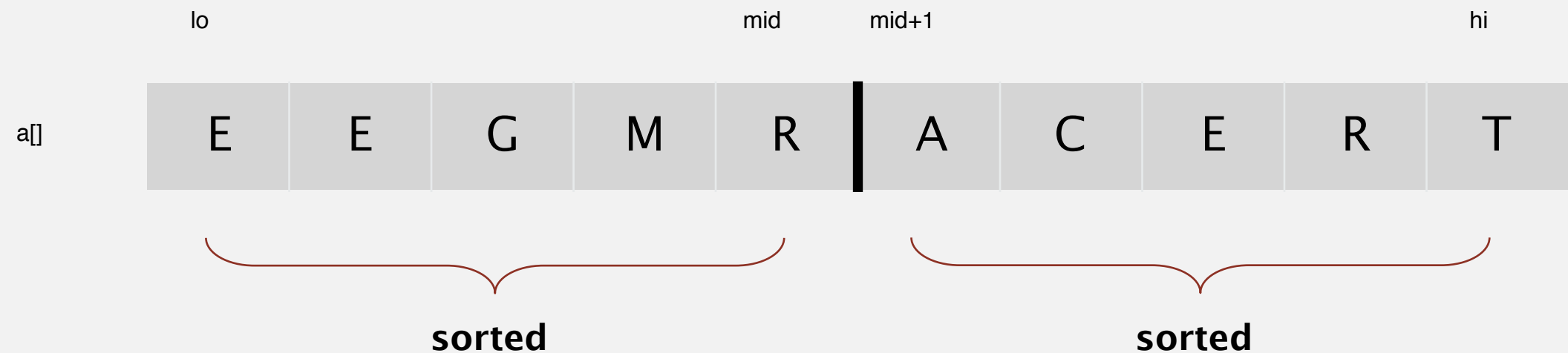


0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987

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

Merging demo

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



Merging: Java implementation

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

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

copy

```
    int i = lo, j = mid+1;
    for (int k = lo; k <= hi; k++)
    {
        if (i > mid)        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



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

Basic plan:

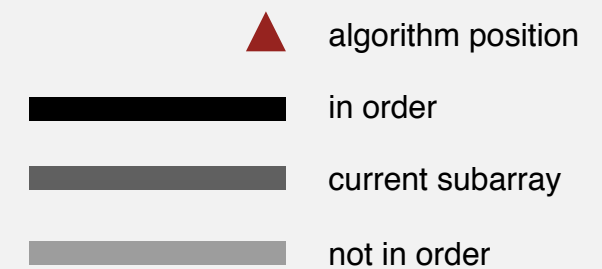
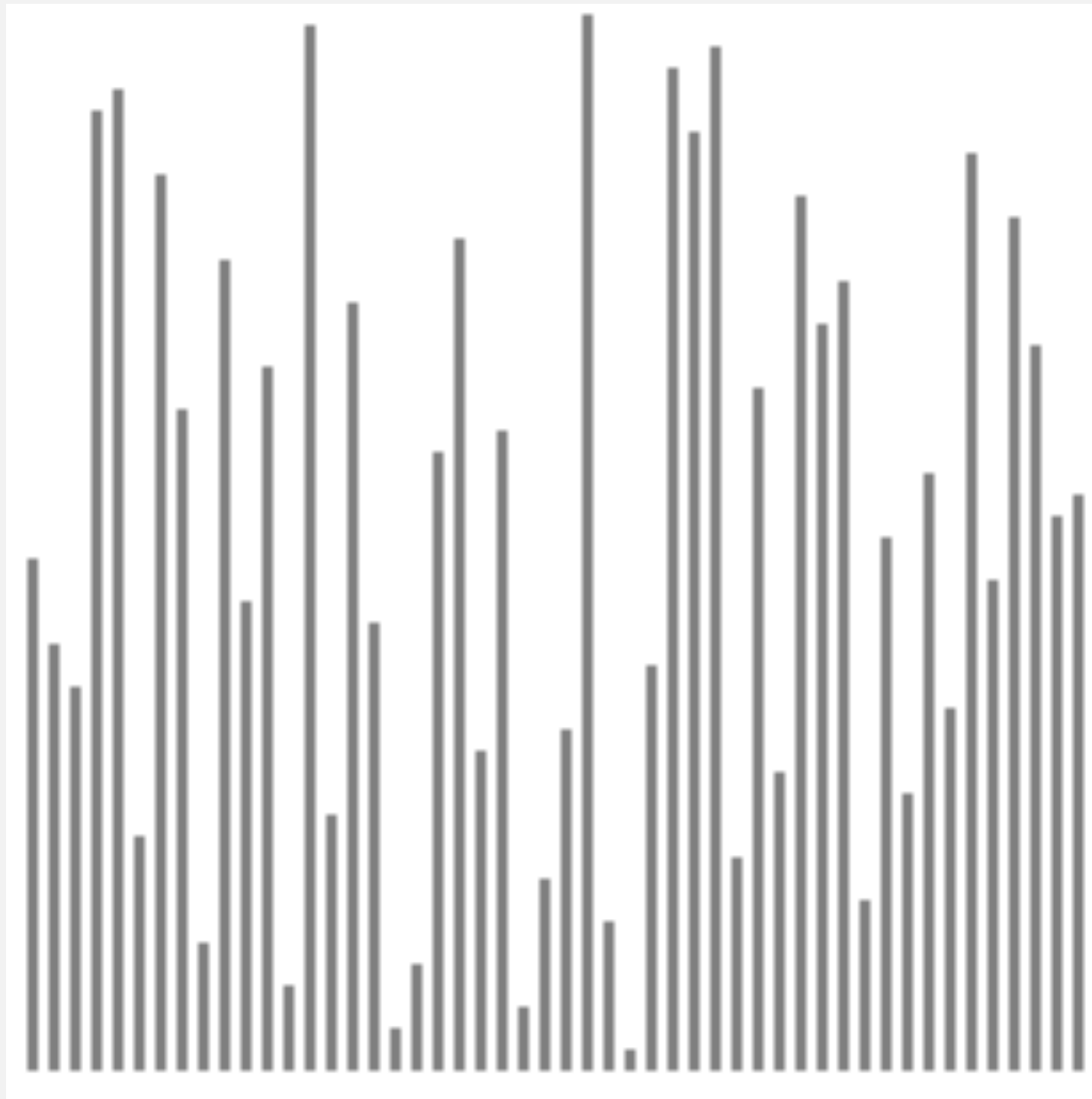
- Divide array into two halves.
- **Recursively** sort each half.
- Merge two halves.

	a[]															
	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
	E	M	R	G	E	S	O	R	T	E	X	A	M	P	L	E
	E	M	G	R	E	S	O	R	T	E	X	A	M	P	L	E
	E	G	M	R	E	S	O	R	T	E	X	A	M	P	L	E
	E	G	M	R	E	S	O	R	T	E	X	A	M	P	L	E
	E	G	M	R	E	S	O	R	T	E	X	A	M	P	L	E
	E	G	M	R	E	O	R	S	T	E	X	A	M	P	L	E
	E	E	G	M	O	R	R	S	T	E	X	A	M	P	L	E
	E	E	G	M	O	R	R	S	E	T	X	A	M	P	L	E
	E	E	G	M	O	R	R	S	E	T	A	X	M	P	L	E
	E	E	G	M	O	R	R	S	A	E	T	X	M	P	L	E
	E	E	G	M	O	R	R	S	A	E	T	X	M	P	E	L
	E	E	G	M	O	R	R	S	A	E	T	X	E	L	M	P
	E	E	G	M	O	R	R	S	A	E	E	L	M	P	T	X
	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

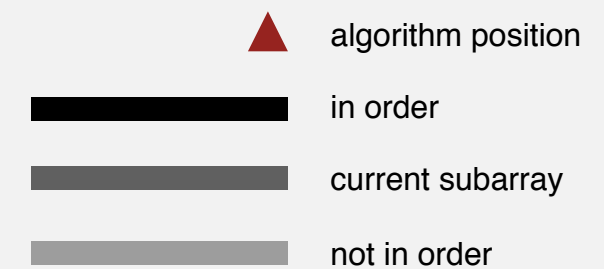
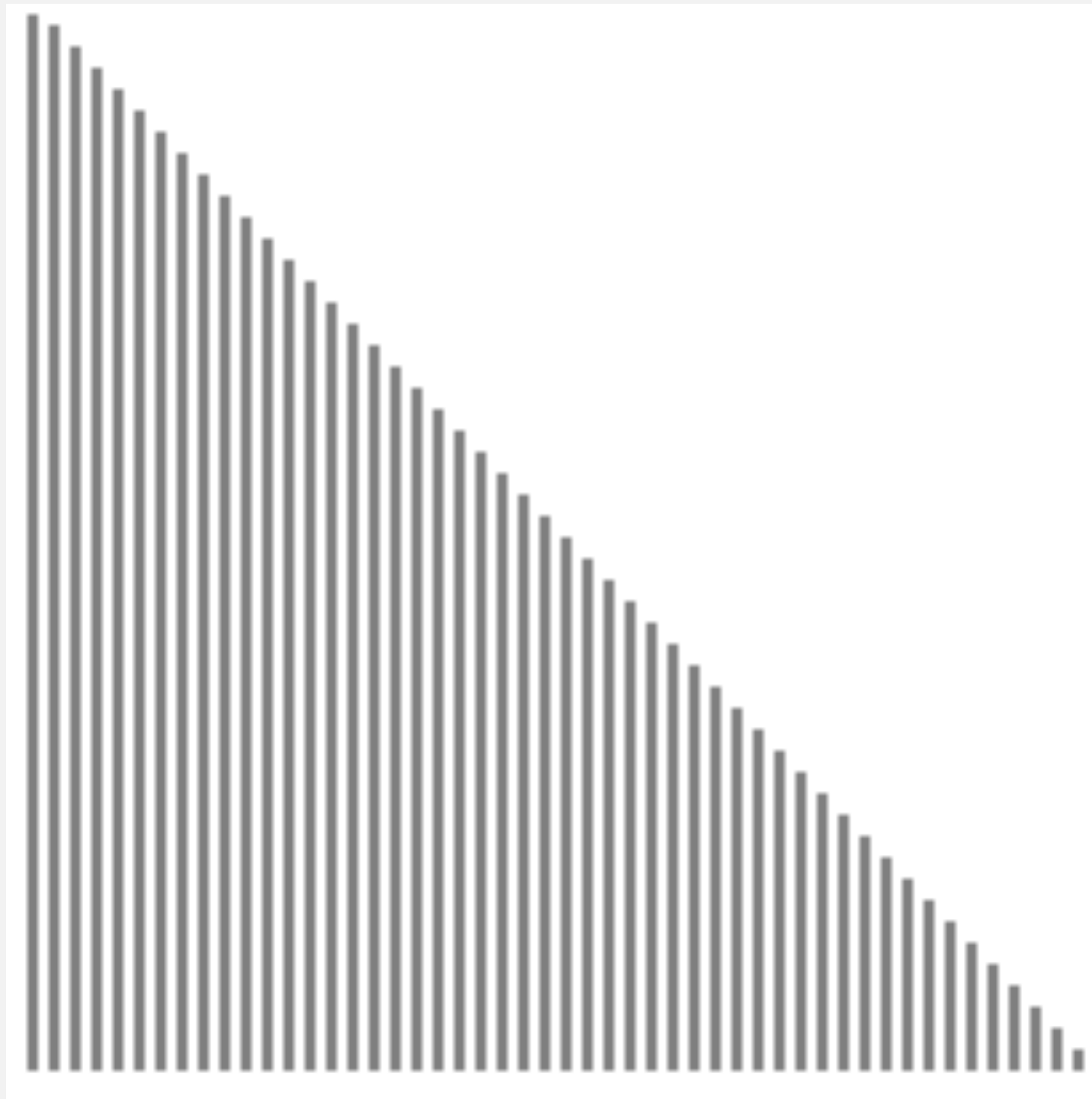


Reason it is slow: excessive data movement.

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

Mergesort: animation

50 reverse-sorted items



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

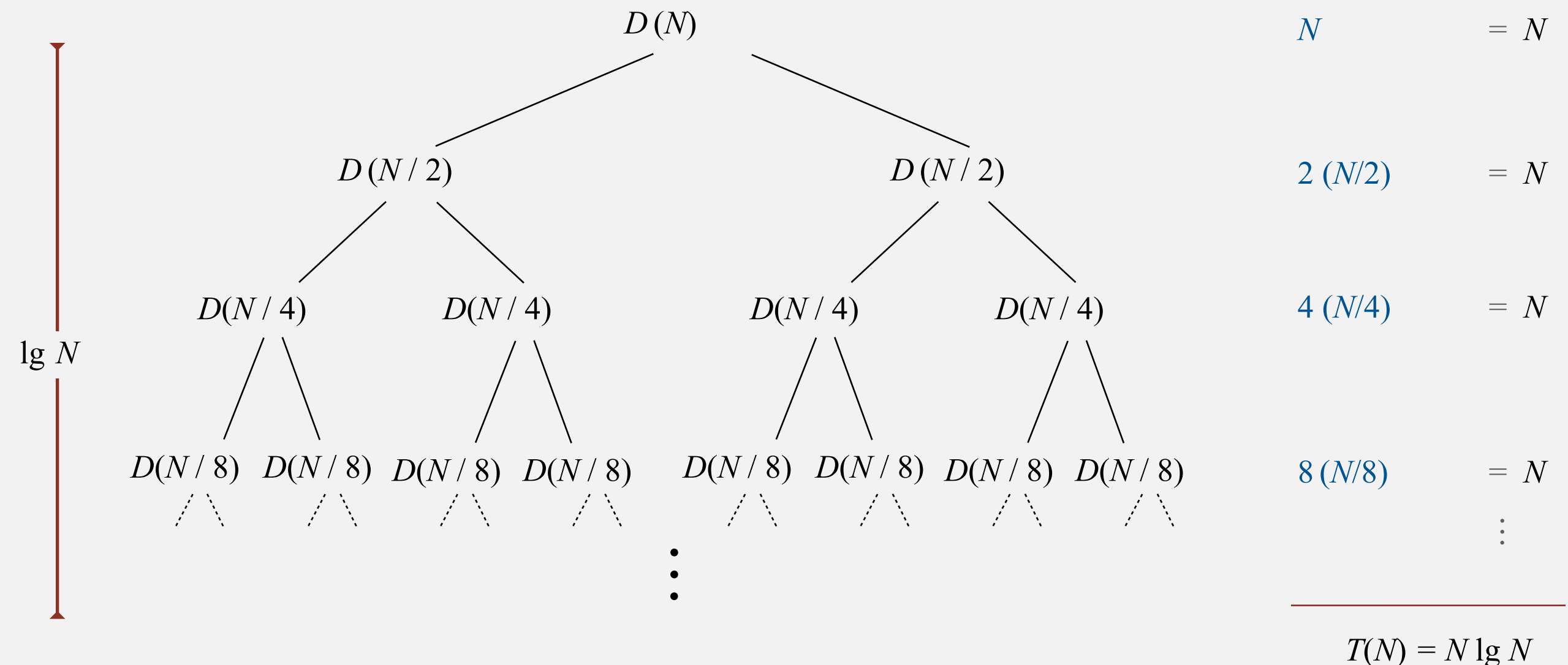
We solve the recurrence when N is a power of 2:

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

Divide-and-conquer recurrence: proof by picture

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]



Divide-and-conquer recurrence: proof by expansion

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, $D(1) = 0$

$$= D(N/N) / (N/N) + 1 + 1 + \dots + 1$$

$$= \lg N$$

Merge

```
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

merge

array of

reference:

at most $6N$ array accesses ($2N$ for the array copy, $2N$ for move back and at most $2N$ 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);
    linearithmic(N/2);
    linear(N);
}
```

← solve two problems

← of half the size

← do a linear amount of work

Mergesort: practical improvements

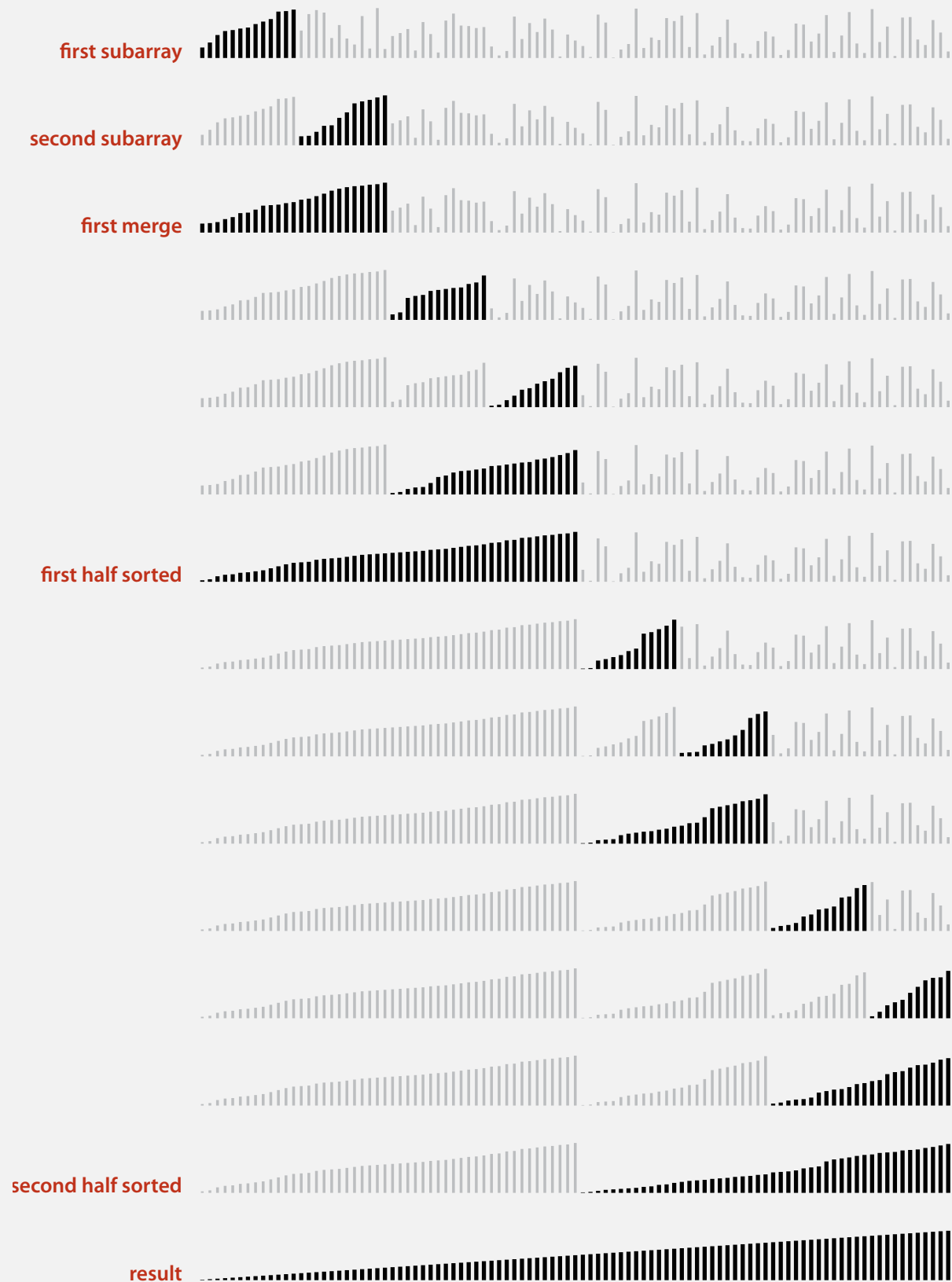
Use insertion sort for small subarrays.

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

```
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

Visu



Mergesort: practical improvements

Stop if already sorted.

- Is largest item in first half \leq 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

- ▶ *Top-down 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
sz = 1	M	E	R	G	E	S	O	R	T	E	X	A	M	P	L	E
merge(a, aux, 0, 0, 1)	E	M	R	G	E	S	O	R	T	E	X	A	M	P	L	E
merge(a, aux, 2, 2, 3)	E	M	G	R	E	S	O	R	T	E	X	A	M	P	L	E
merge(a, aux, 4, 4, 5)	E	M	G	R	E	S	O	R	T	E	X	A	M	P	L	E
merge(a, aux, 6, 6, 7)	E	M	G	R	E	S	O	R	T	E	X	A	M	P	L	E
merge(a, aux, 8, 8, 9)	E	M	G	R	E	S	O	R	E	T	X	A	M	P	L	E
merge(a, aux, 10, 10, 11)	E	M	G	R	E	S	O	R	E	T	A	X	M	P	L	E
merge(a, aux, 12, 12, 13)	E	M	G	R	E	S	O	R	E	T	A	X	M	P	L	E
merge(a, aux, 14, 14, 15)	E	M	G	R	E	S	O	R	E	T	A	X	M	P	E	L
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 */ }

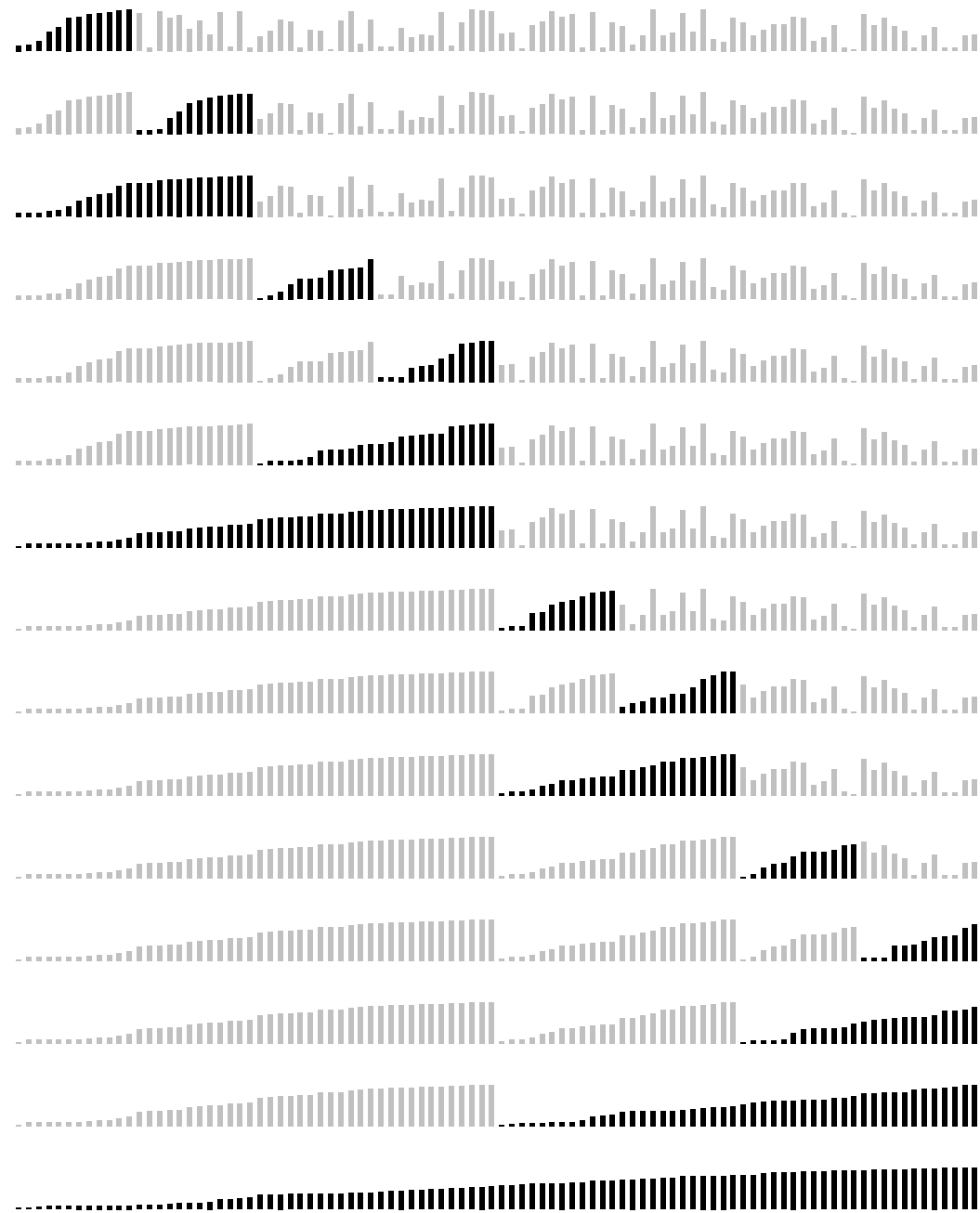
    public static void sort(Comparable[] a)
    {
        int N = a.length;
        Comparable[] aux = new Comparable[N];
        for (int sz = 1; sz < N; sz = sz+sz)
            for (int lo = 0; lo < N-sz; lo += sz+sz)
                merge(a, aux, lo, lo+sz-1, Math.min(lo+sz+sz-1, N-1));
    }
}
```

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

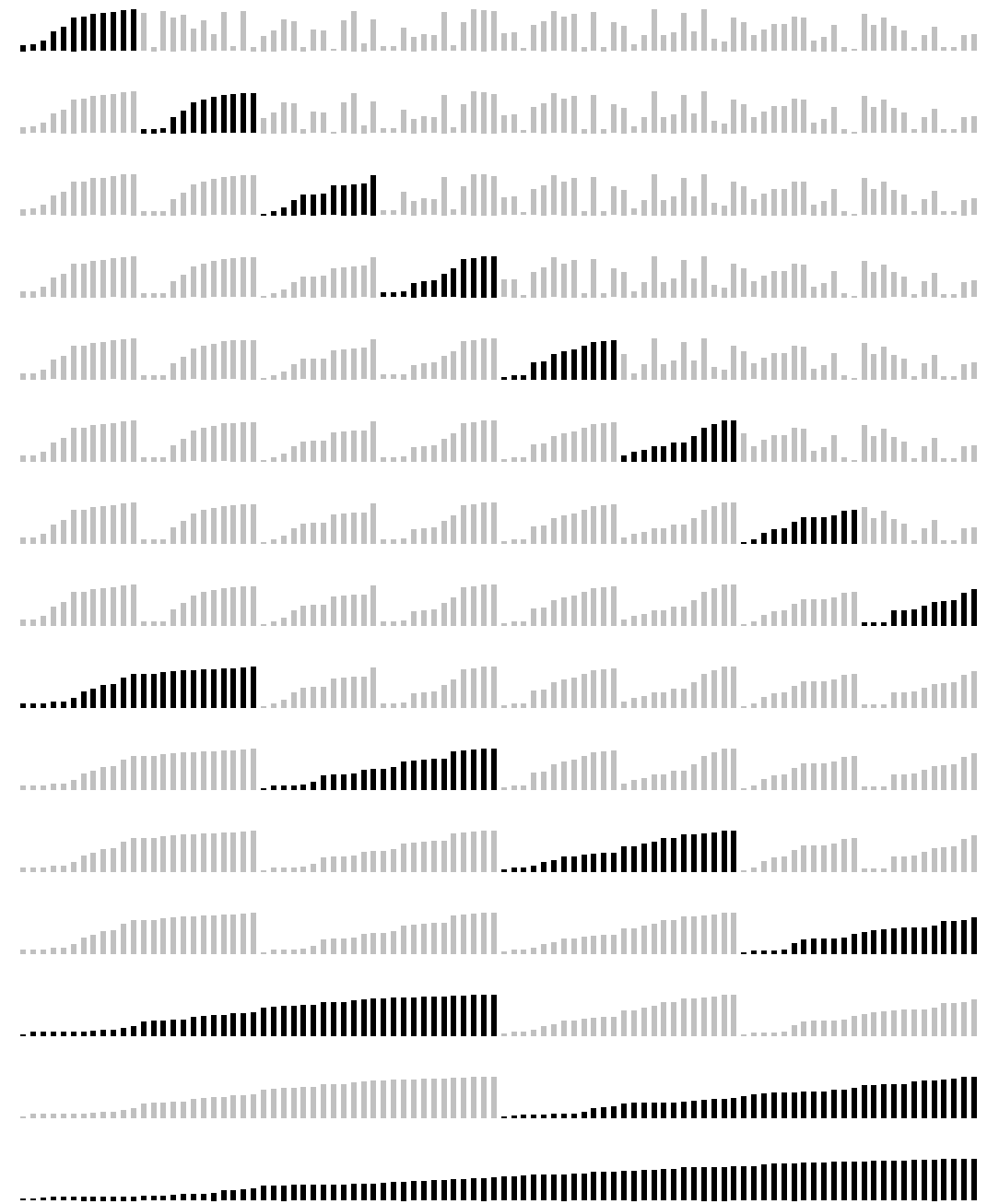
Bottom line. Simple and non-recursive version of mergesort.

Mergesort: visualizations

Visu



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 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.

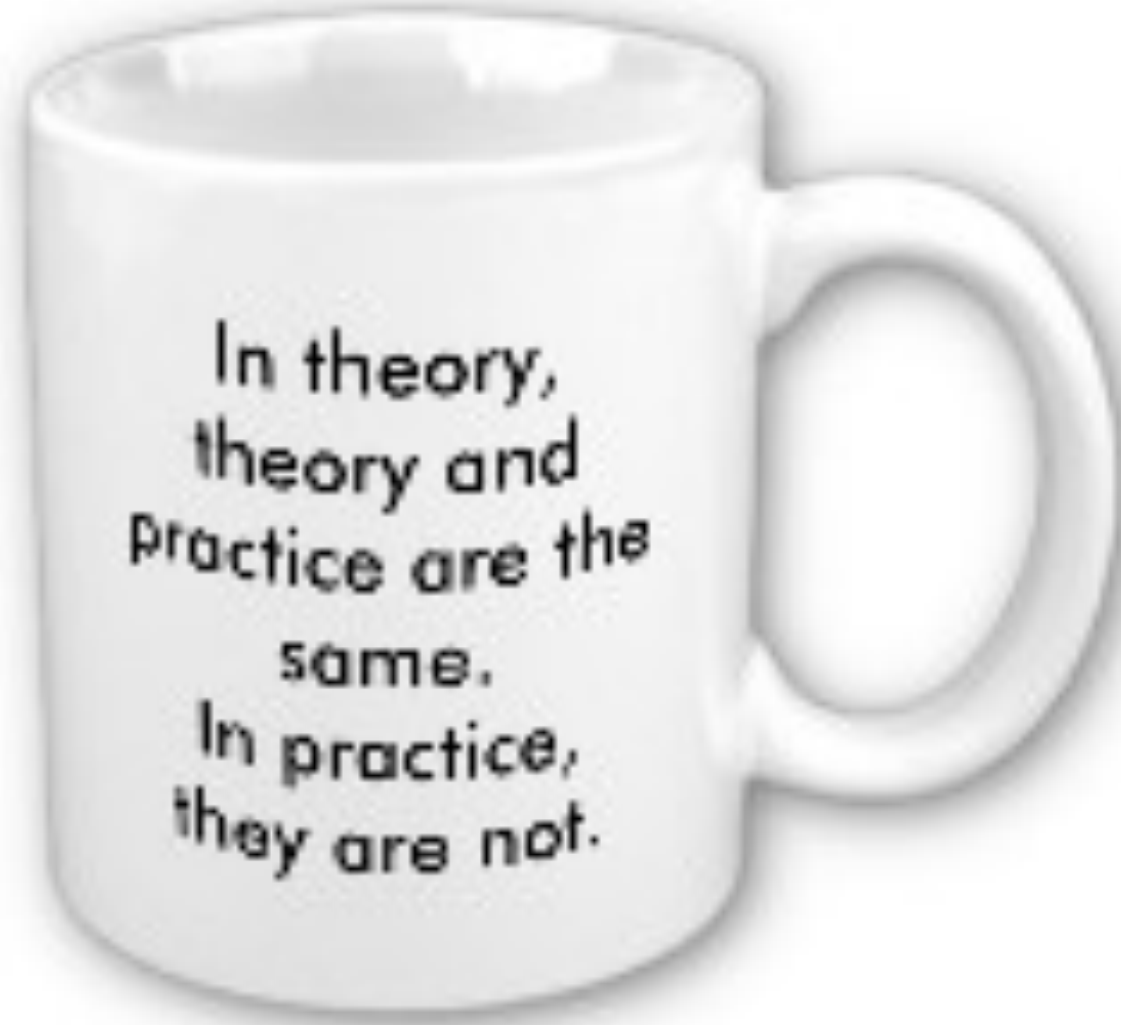
In a nutshell, the main routine marches over the array once, left to right, alternately identifying the next run, then merging it into the previous runs "intelligently". Everything else is complication for speed, and some hard-won measure of memory efficiency.

...

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*



**In theory,
theory and
Practice are the
same.
In practice,
they are not.**

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	<input checked="" type="checkbox"/> Alive	Pearl Jam	5:41	Ten
2	<input checked="" type="checkbox"/> All Over The World	Pixies	5:27	Bossanova
3	<input checked="" type="checkbox"/> All Through The Night	Cyndi Lauper	4:30	She's So Unusual
4	<input checked="" type="checkbox"/> Allison Road	Gin Blossoms	3:19	New Miserable Experience
5	<input checked="" type="checkbox"/> Ama, Ama, Ama Y Ensancha El ...	Extremoduro	2:34	Deltoya (1992)
6	<input checked="" type="checkbox"/> And We Danced	Hooters	3:50	Nervous Night
7	<input checked="" type="checkbox"/> As I Lay Me Down	Sophie B. Hawkins	4:09	Whaler
8	<input checked="" type="checkbox"/> Atomic	Blondie	3:50	Atomic: The Very Best Of Blondie
9	<input checked="" type="checkbox"/> Automatic Lover	Jay-Jay Johanson	4:19	Antenna
10	<input checked="" type="checkbox"/> Baba O'Riley	The Who	5:01	Who's Better, Who's Best
11	<input checked="" type="checkbox"/> Beautiful Life	Ace Of Base	3:40	The Bridge
12	<input checked="" type="checkbox"/> Beds Of Roses	Bon Jovi	6:35	Cross Road
13	<input checked="" type="checkbox"/> Black	Pearl Jam	5:44	Ten
14	<input checked="" type="checkbox"/> Bleed American	Jimmy Eat World	3:04	Bleed American
15	<input checked="" type="checkbox"/> Borderline	Madonna	4:00	The Immaculate Collection
16	<input checked="" type="checkbox"/> Born To Run	Bruce Springsteen	4:30	Born To Run
17	<input checked="" type="checkbox"/> Both Sides Of The Story	Phil Collins	6:43	Both Sides
18	<input checked="" type="checkbox"/> Bouncing Around The Room	Phish	4:09	A Live One (Disc 1)
19	<input checked="" type="checkbox"/> Boys Don't Cry	The Cure	2:35	Staring At The Sea: The Singles 1979-1985
20	<input checked="" type="checkbox"/> Brat	Green Day	1:43	Insomniac
21	<input checked="" type="checkbox"/> Breakdown	Deerheart	3:40	Deerheart
22	<input checked="" type="checkbox"/> Bring Me To Life (Kevin Roen Mix)	Evanescence Vs. Pa...	9:46	
23	<input checked="" type="checkbox"/> Californication	Red Hot Chili Pepp...	1:40	
24	<input checked="" type="checkbox"/> Call Me	Blondie	3:33	Atomic: The Very Best Of Blondie
25	<input checked="" type="checkbox"/> Can't Get You Out Of My Head	Kylie Minogue	3:50	Fever
26	<input checked="" type="checkbox"/> Celebration	Kool & The Gang	3:15	Time Life Music Sounds Of The Seventies - C
27	<input checked="" type="checkbox"/> Choke, Choke	Goldfinger Single	5:11	Backyard Breeze

Sort songs by artist

The screenshot shows a music player interface with a playlist of 50 tracks. The tracks are sorted by artist, with 'ASAP Rocky' appearing first. The current track playing is 'Shield for your eyes, a Beast in the well on your hand' by Melt Banana. The interface includes a sidebar with playlist options, a main track list, and a playback control bar at the bottom.

Track	Artist	Time	Album
Goldie	ASAP Rocky	3:15	Goldie
Peso	ASAP Rocky	2:51	Peso
Purple Swag	ASAP Rocky	1:59	Purple Swag
Alpha Beta Gaga	Air	4:40	Talkie Walkie
Avril 14th	Aphex Twin	2:06	Drukqs
Bela Lugosi's Dead	Bauhaus	9:40	Bauhaus - 1979-1
Nitemare Hippy Girl	Beck	2:56	Mellow Gold
The Lovely Universe	Circulatory System	3:24	Circulatory System
My Mind Went Blank - Feat. Point Blank	DJ Screw	6:38	Bigtyme Recordz '9
First class 77	Fantastic Plastic Mac...	5:49	Hôtel Costes by St
Hercules Theme	Hercules And Love A...	4:30	Hercules And Love
Bright Whites	Kishi Bashi	4:16	Room For Dream
Gloria	Laura Branigan	4:52	Gloria / Living A Li
Dance Yrself Clean	LCD Soundsystem	8:58	This Is Happening
Hour Fortress	Light Asylum	4:47	Light Asylum
Shield for your eyes, a Beast in the well on your hand	Melt Banana	4:03	Cell-Scape
You Could Easily Have Me	Metronomy	3:07	Pip Paine (Pay The

Sort songs by track name

The screenshot shows a music player interface with a playlist on the left and a track list on the right. The track list is sorted by track name. The current track playing is 'Purple Swag' by A\$AP Rocky.

Left Panel:

- + New Playlist
- Triscuit Party Playlist go n...
- Paper Bag
- Random
- Graceland 25th Anniversary Edi...
- terrible, horrible, no good, very...
- Bobby Conn - The Golden Age
- Windows Media Player

Track List:

Track	Artist	Time	Album
1 String Strung	Metronomy	2:44	Pip Paine (Pay The
1979	The Smashing Pump...	4:26	Mellon Collie And
A New Error	Moderat	6:08	Moderat
Allah Hoo Allah Hoo Allah Hoo	Nusrat Fateh Ali Khan	27:57	Anthology - Nusra
Alpha Beta Gaga	Air	4:40	Talkie Walkie
Another Me To Mother You - Bonus Track	Metronomy	4:15	Pip Paine (Pay The
Are Mums Mates - Bonus Track	Metronomy	2:15	Pip Paine (Pay The
Avril 14th	Aphex Twin	2:06	Drukqs
Bearcan	Metronomy	6:39	Pip Paine (Pay The
Bela Lugosi's Dead	Bauhaus	9:40	Bauhaus - 1979-1
Black Eye/Burnt Thumb	Metronomy	4:43	Pip Paine (Pay The
Bright Whites	Kishi Bashi	4:16	Room For Dream
Dance Yrself Clean	LCD Soundsystem	8:58	This Is Happening
Danger Song	Metronomy	4:42	Pip Paine (Pay The
Debaser	Pixies	2:53	Doolittle
Gloria	Laura Branigan	4:52	Gloria / Living A Li
Goldie	EXPLICIT A\$AP Rocky	3:15	Goldie
Hear To Wear - Bonus Track	Metronomy	3:28	Pip Paine (Pay The

Current Track: Purple Swag, A\$AP Rocky

Progress Bar: 0:21

Comparable interface: review

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

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

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

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.

```
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

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

Stability

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

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

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

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

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

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

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

Stability

Q. Which sorts are stable?

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--)
                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

Stability: shellsort

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

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

h	0	1	2	3	4
	B 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.

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
<hr/>					<hr/>					
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 sorting grail

Sleep Sort

```
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. }
```