

<http://algs4.cs.princeton.edu>

2.2 MERGESORT

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

Two classic sorting algorithms

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]

- Java sort for objects.
- Perl, C++ stable sort, Python stable sort, Firefox JavaScript, ...

Quicksort. [next lecture]

- Java sort for primitive types.
- C qsort, Unix, Visual C++, Python, Matlab, Chrome JavaScript, ...

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

2.2 MERGESORT

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

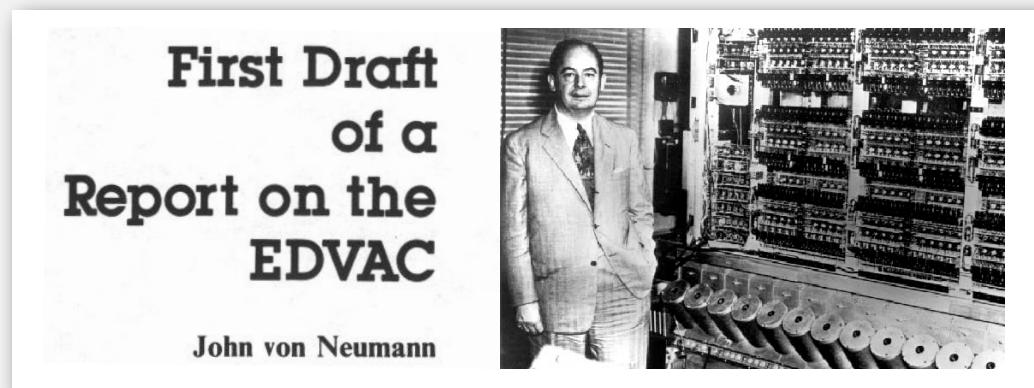
Mergesort

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 |

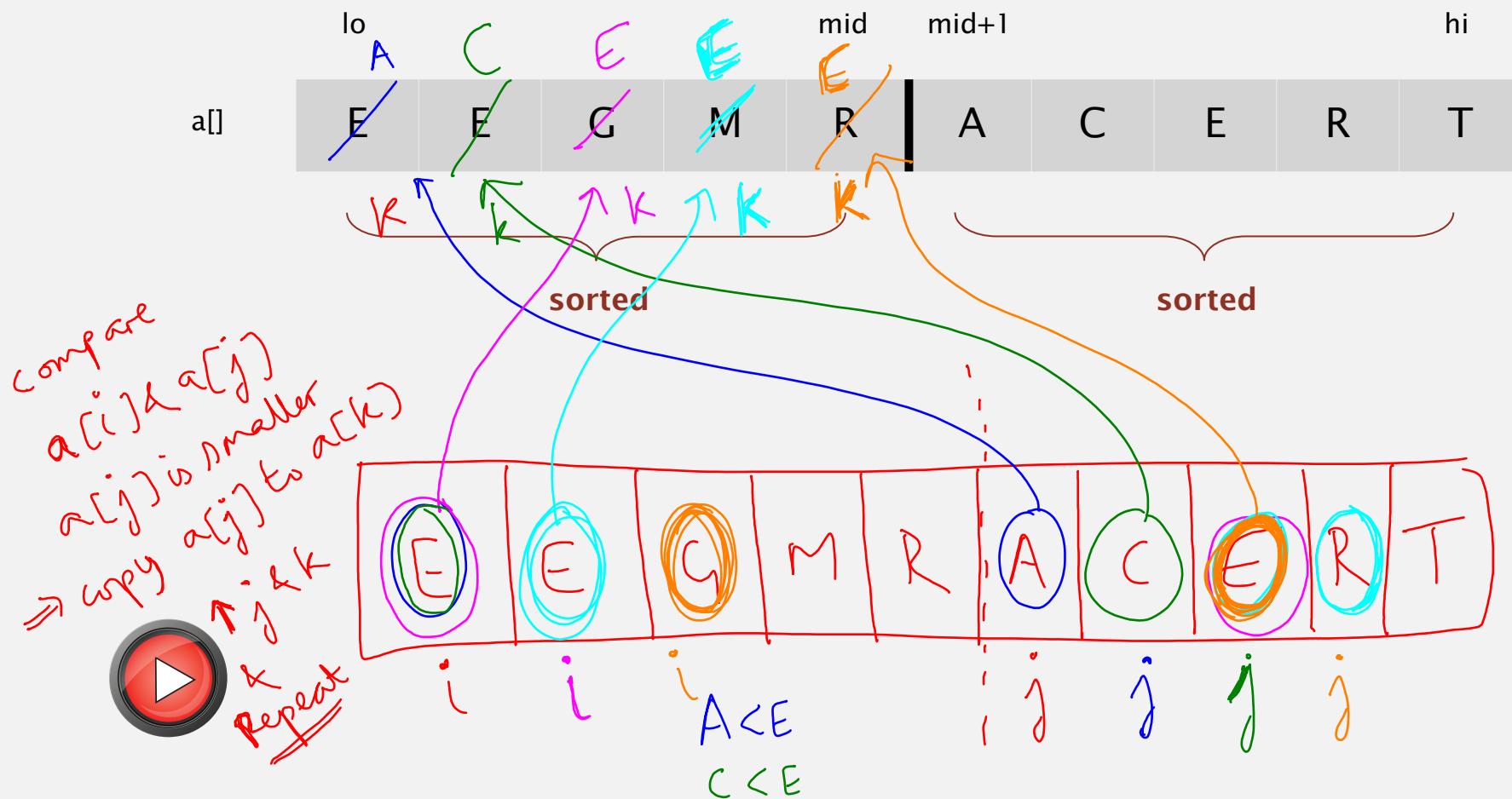
Mergesort overview



Abstract in-place merge demo

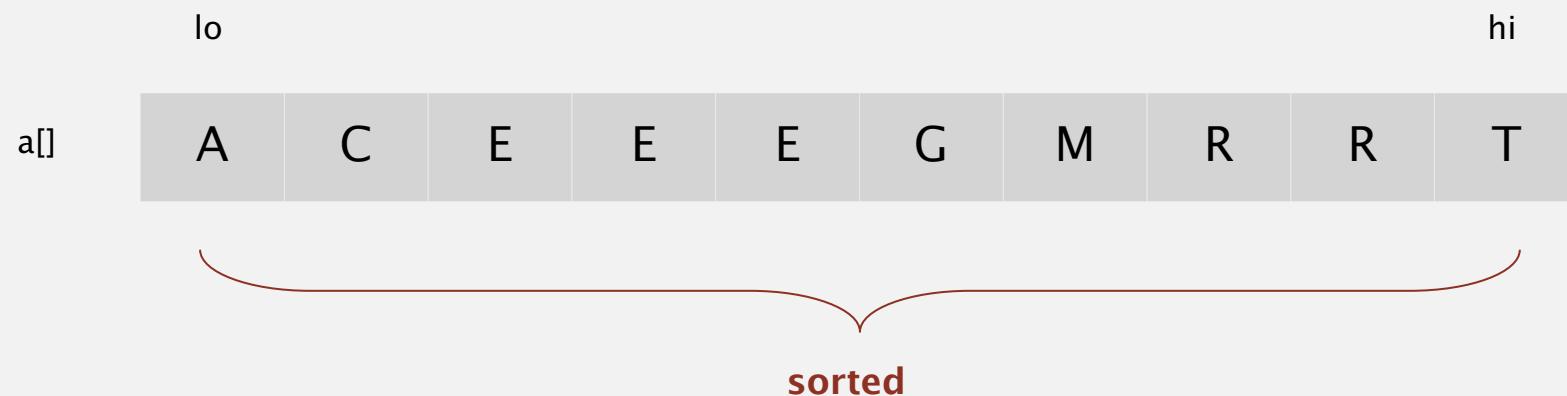
→ See
demo
it
need
be

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



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 $a[hi]$.



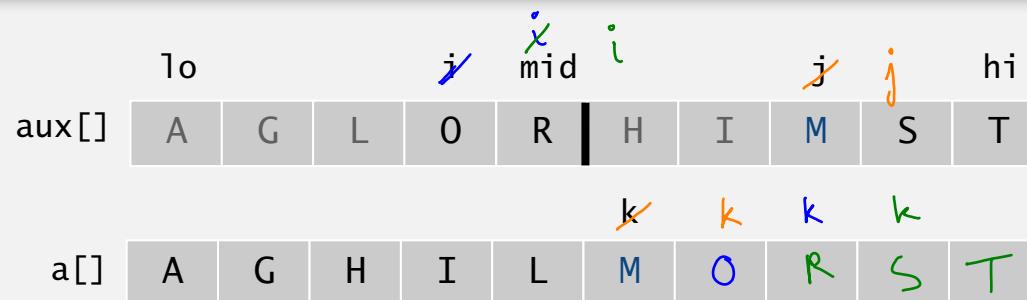
Merging: Java implementation

```
private static void merge(Comparable[] a, Comparable[] aux, int lo, int mid, int hi)
{
    assert isSorted(a, lo, mid);      // precondition: a[lo..mid] sorted
    assert isSorted(a, mid+1, hi);    // precondition: a[mid+1..hi] sorted

    for (int k = lo; k <= hi; k++)  → copy a into aux           copy
        aux[k] = a[k];

    int i = lo, j = mid+1;
    for (int k = lo; k <= hi; k++)  if you reach 1st half, copy j till the end.      merge
    {
        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++];
    }

    assert isSorted(a, lo, hi);      // postcondition: a[lo..hi] sorted
}
```



Assertions

Assertion. Statement to test assumptions about your program.

- Helps detect logic bugs.
- Documents code.

Java assert statement. Throws exception unless boolean condition is true.

```
assert isSorted(a, lo, hi);
```

✓ Can enable or disable at runtime. ⇒ No cost in production code.



```
java -ea MyProgram    // enable assertions
java -da MyProgram    // disable assertions (default)
```

Best practices. Use assertions to check internal invariants;
assume assertions will be disabled in production code. ←

do not use for external
argument checking

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

So no one is actually sorting anything
→ Sorting occurs because you keep Array
merging smaller in sorted order

create Aux here, not above



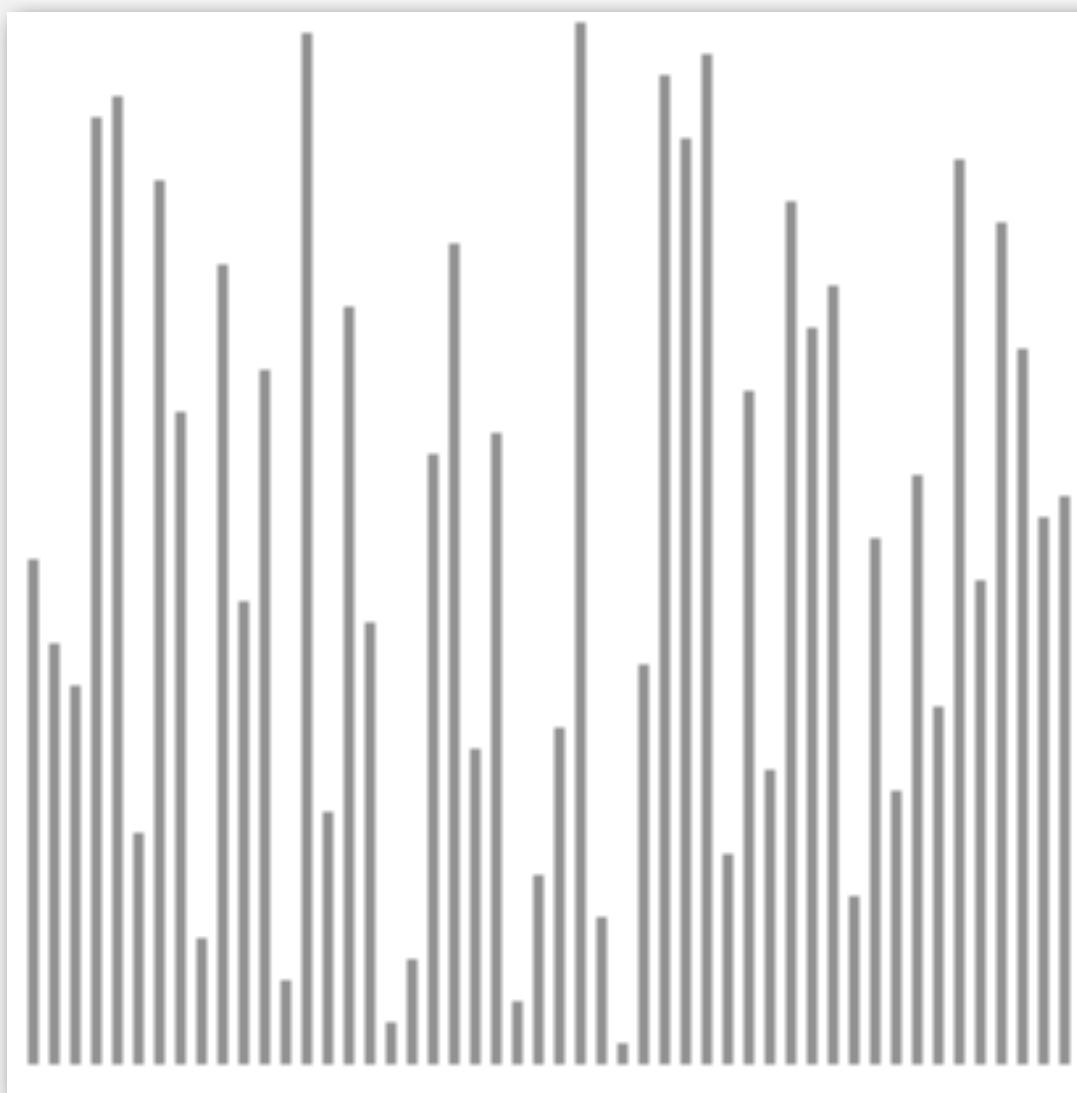
Mergesort: trace

| | a[] | | | | | | | | | | | | | | | | |
|---------------------------|-----|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|---|
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | |
| lo | M | E | R | G | E | S | O | R | T | E | X | A | M | P | L | E | |
| hi | E | M | 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 | E | G | L | M | M | O | P | R | R | S | T | X |

result after recursive call

Mergesort: animation

50 random items

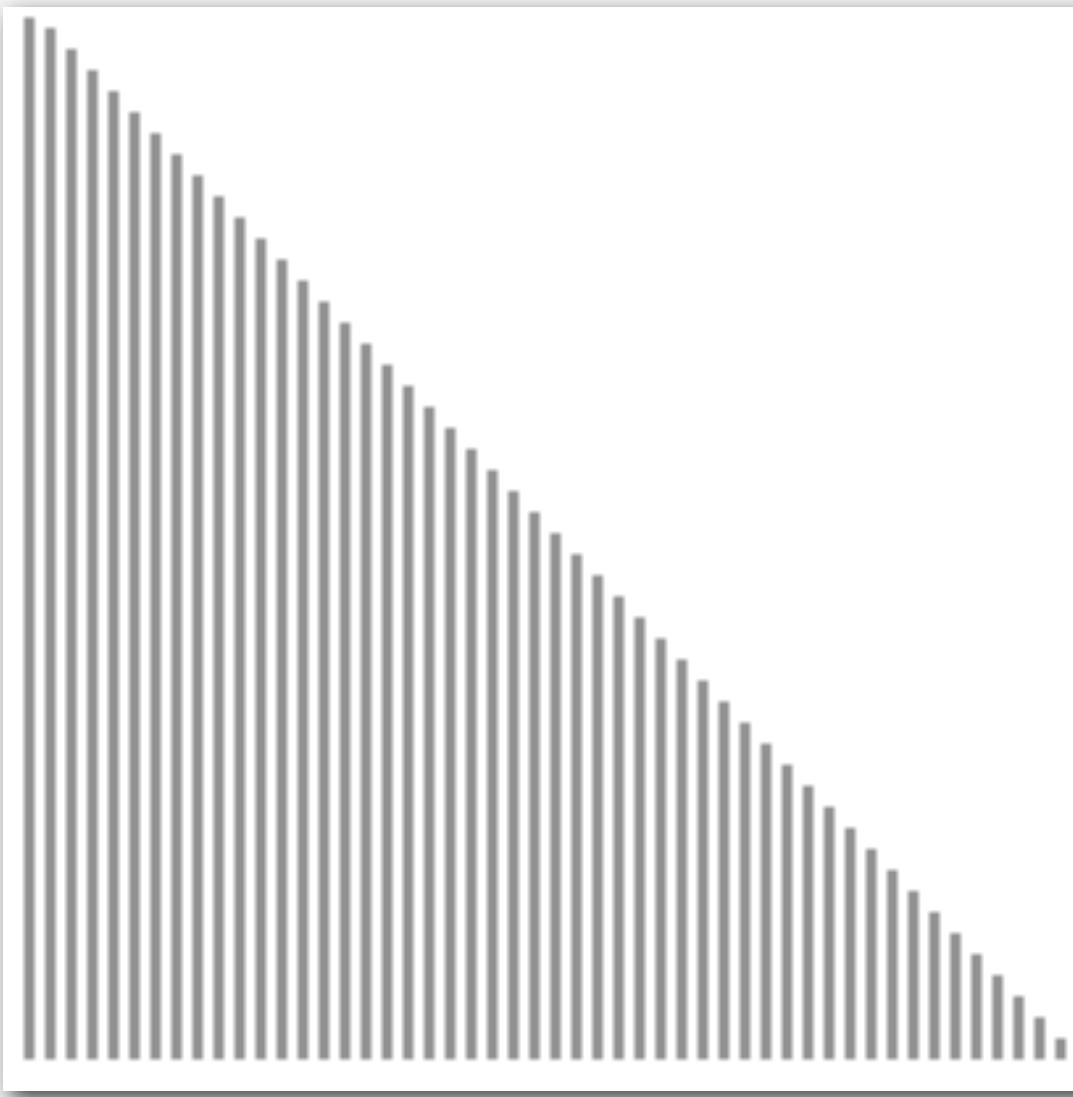


<http://www.sorting-algorithms.com/merge-sort>

- ▲ algorithm position
- in order
- current subarray
- not in order

Mergesort: animation

50 reverse-sorted items



<http://www.sorting-algorithms.com/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.

| computer | insertion sort (N^2) | | | mergesort ($N \log N$) | | |
|----------|--------------------------|-----------|-----------|--------------------------|----------|---------|
| | 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 and array accesses

Proposition. Mergesort uses at most $N \lg N$ compares and $6N \lg N$ array accesses to sort any array of size N . *reduced from N^2 .*

Pf sketch. The number of compares $C(N)$ and array accesses $A(N)$ to mergesort an array of size N satisfy the recurrences:

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

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

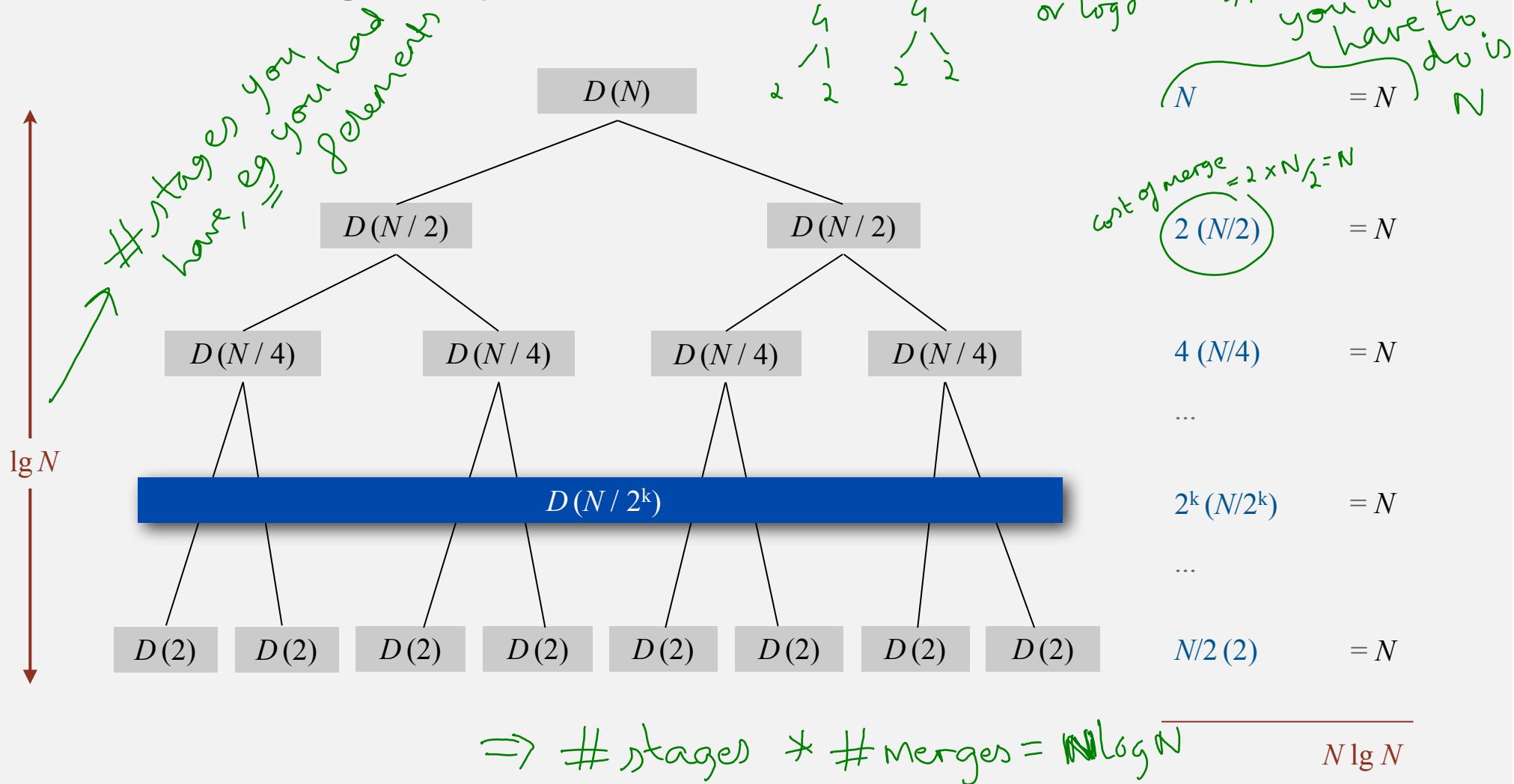
We solve the recurrence when N is a power of 2. \leftarrow result holds for all N

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

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]



$$D(N/2) = 2D(N/4) + 1$$

Divide-and-conquer recurrence: proof by expansion

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

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

$$\begin{aligned} D(N) &= 2D(N/2) + N \\ D(N)/N &= 2D(N/2)/N + 1 \\ \boxed{D(N)/N} &= \boxed{D(N/2)/(N/2) + 1} \\ &= D(N/4)/(N/4) + 1 + 1 \\ &= D(N/8)/(N/8) + 1 + 1 + 1 \\ &\quad \dots \\ &= D(N/N)/(N/N) + 1 + 1 + \dots + 1 \\ \underline{\underline{N}} \quad D(N) &= \lg N \Rightarrow D(N) = N \lg N \end{aligned}$$

Apply to $D(N/2)$ RHS

$$\begin{aligned} 2D(N/4) &+ N/2 \quad \times 1 \\ (N/2) & \\ = 2D(N/4) &+ 1 + 1 \\ = D(N/4) &+ 1 + 1 \end{aligned}$$

given

divide both sides by N

algebra

apply to first term

apply to first term again

stop applying, $D(1) = 0$

Divide-and-conquer recurrence: proof by induction

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

Pf 3. [assuming N is a power of 2]

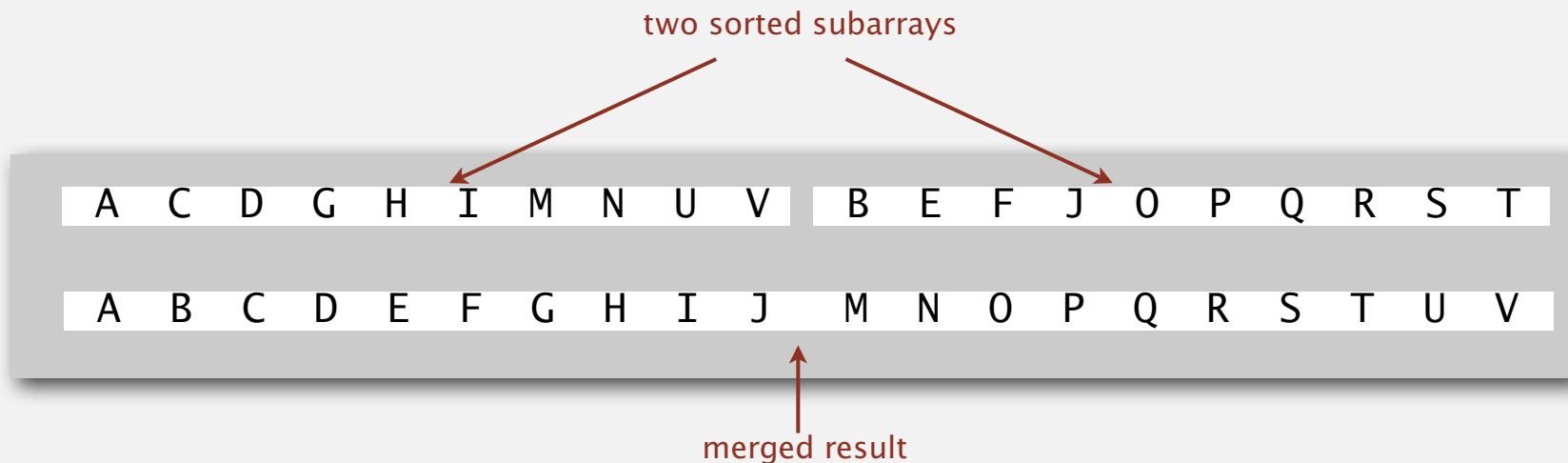
- Base case: $N = 1$.
- Inductive hypothesis: $D(N) = N \lg N$.
- Goal: show that $D(2N) = (2N) \lg (2N)$.

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

Mergesort analysis: memory

Proposition. Mergesort uses extra space proportional to N .

Pf. The array $\text{aux}[]$ needs to be of size 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.

Challenge for the bored. In-place merge. [Kronrod, 1969]

Mergesort: practical improvements

Use insertion sort for small subarrays.

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

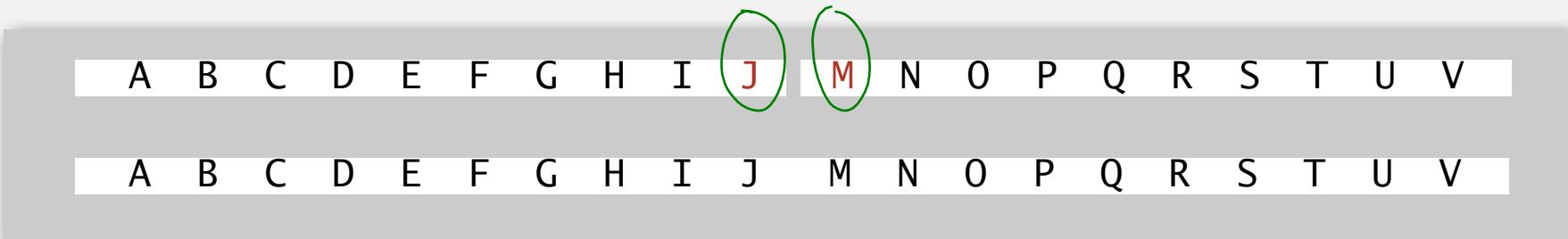
```
private static void sort(Comparable[] a, Comparable[] aux, int lo, int hi)
{
    if (hi <= lo + CUTOFF - 1)
    {
        Insertion.sort(a, lo, hi);
        return;
    }
    int mid = lo + (hi - lo) / 2;
    sort (a, aux, lo, mid);
    sort (a, aux, mid+1, hi);
    merge(a, aux, lo, mid, hi);
}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

Mergesort: practical improvements

Stop if already sorted.

- Is biggest item in first half \leq smallest item in second half?
- Helps for partially-ordered arrays.

Assumption in Merge Sort is already sorted sub arrays



```
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

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.

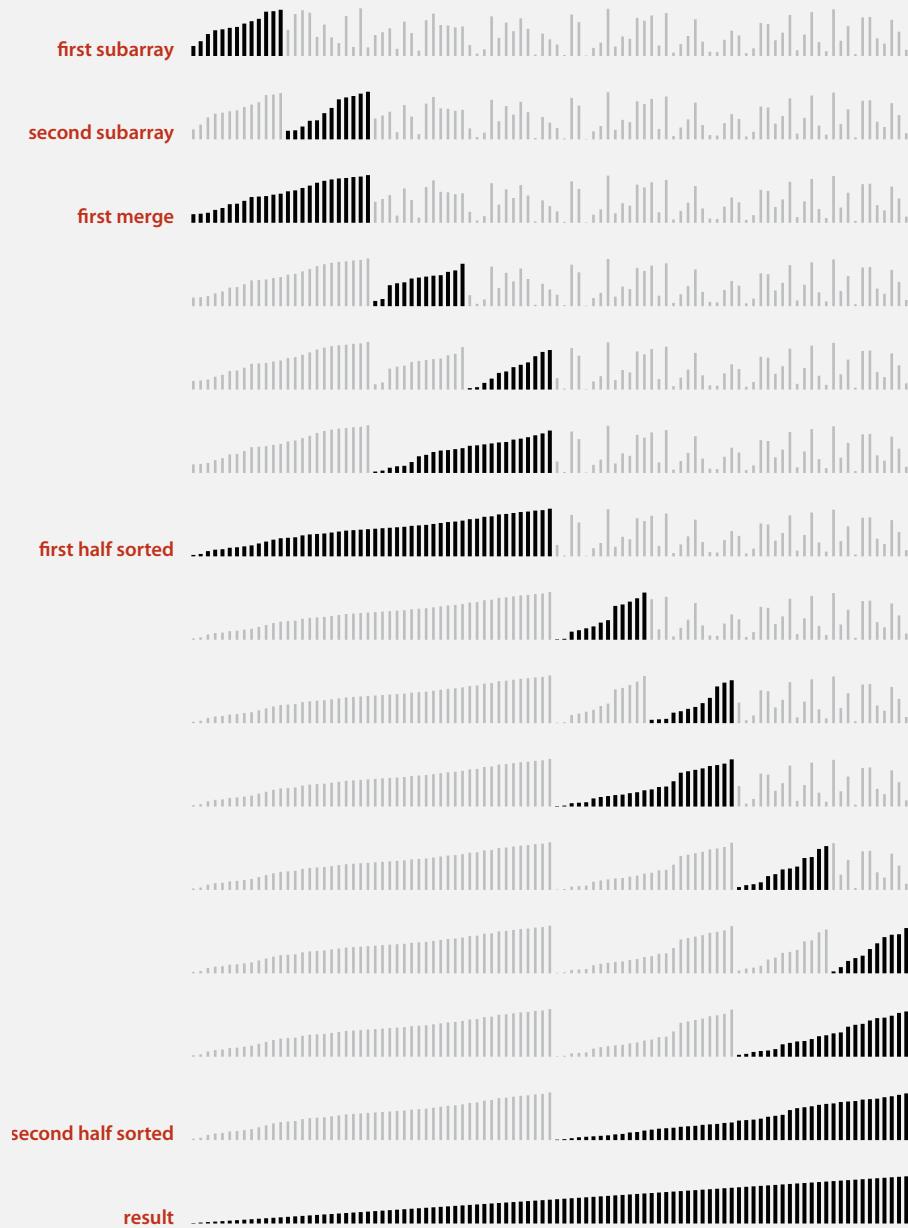
```
private static void merge(Comparable[] a, Comparable[] aux, int lo, int mid, int hi)
{
    int i = lo, j = mid+1;
    for (int k = lo; k <= hi; k++)
    {
        if (i > mid)          aux[k] = a[j++];
        else if (j > hi)      aux[k] = a[i++];
        else if (less(a[j], a[i])) aux[k] = a[j++];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

Note: sort(a) initializes aux[] and sets
aux[i] = a[i] for each i.

switch roles of aux[] and a[]
```

Mergesort: visualization



Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

2.2 MERGESORT

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

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<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, 16,

Traditionally
we would have
stopped at ~~*~~
& proceeded to
Sort
"EMGR"
& "ESOP"
but now we bleed
over to RHS

| a[i] | | | | | | | | | | | | | | | | | |
|---------------------------|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | | |
| 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 | E | G | L | M | M | O | P | R | R | S | T | X |

Not
understood

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 *= 2*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));
    }
}
```

consider elements in
(1,2) (1,4) (1,8) etc
Stopping condition is
2, 4, 8, 16 etc

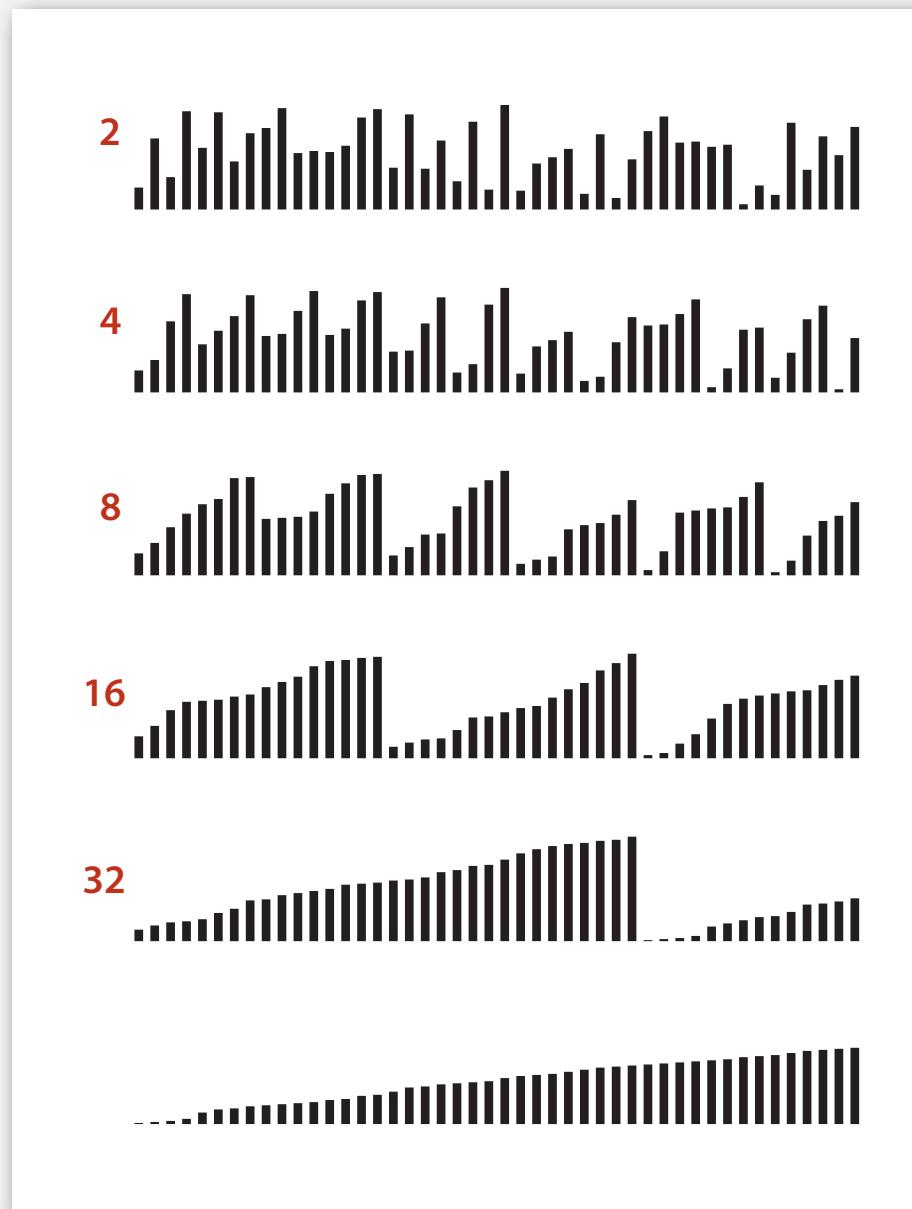
$\lceil N \rceil$

Increase
by
 2×2^k
 2^k

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

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

Bottom-up mergesort: visual trace



Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

2.2 MERGESORT

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

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<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 .

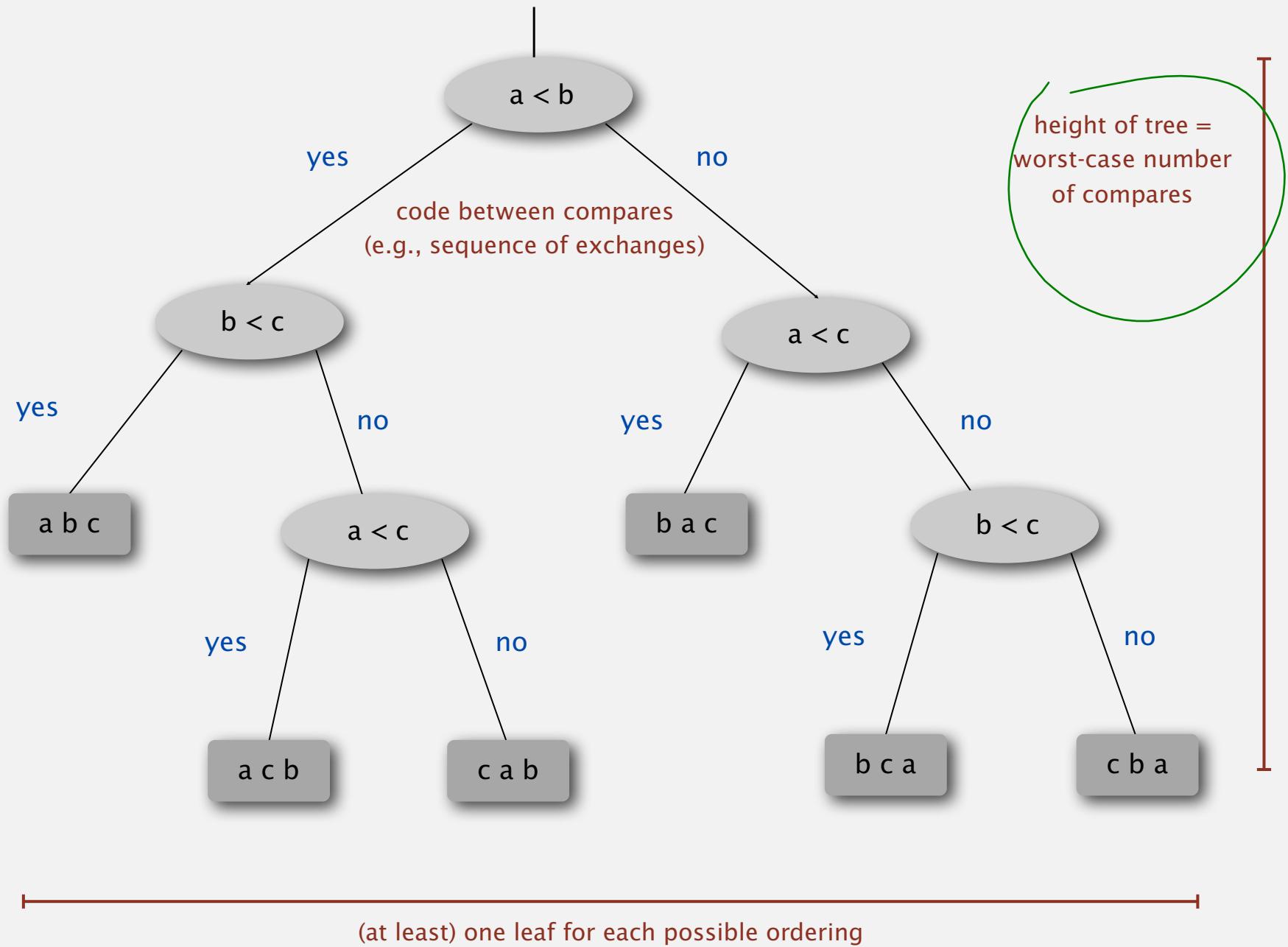
Example: sorting.

- 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)

lower bound for sorting

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



→ \Rightarrow worst case, we would do atleast $N!$ comparisons

N! Comparisons
as $N!$ is least # leaves
 \Rightarrow least # comparisons.

Compare-based lower bound for sorting

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

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.

for N distinct elements, which is true

full binary tree

$$2^h \geq \# \text{leaves} \geq N!$$

$$\Rightarrow h \geq \log(N!) \sim N \log(N) \quad (\text{Stirling's formula})$$

high a_N 

leaves
At least $N^{\frac{1}{h}}$
No more than 2^h

$$N! < 2^h$$

$$\log(N!) < \log(2^h)$$

1. *W*

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.

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

↑
Stirling's formula

Complexity of sorting

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 .

Example: sorting.

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

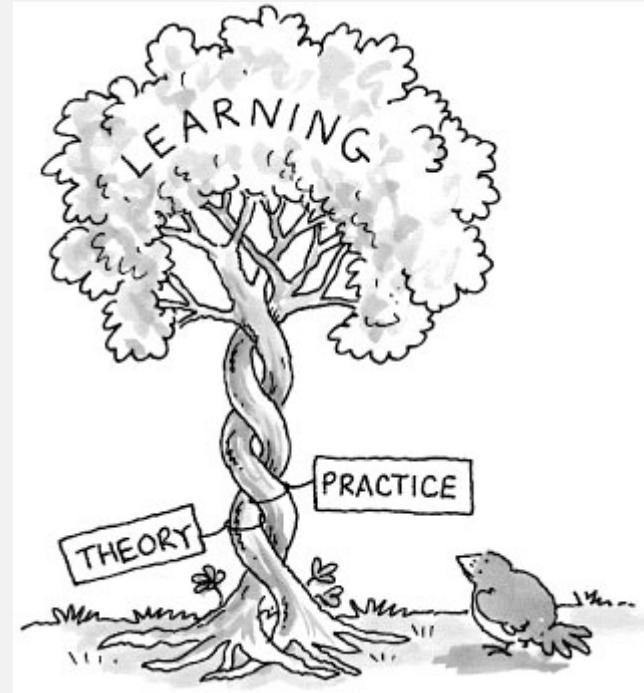


First goal of algorithm design: optimal algorithms.

Complexity results in context

Compares? Mergesort **is** optimal with respect to number compares.

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



takes
twice the
amt of space

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 has information about:

- The initial order of the input.
- The distribution of key values.
- The representation of the keys.

If the Array is sorted, then Insertion Sort may take $\sim N$ time

Partially-ordered arrays. Depending on the initial order of the input, we may not need $N \lg N$ compares.

insertion sort requires only $N-1$ compares if input array is sorted

Duplicate keys. Depending on the input distribution of duplicates, we may not need $N \lg N$ compares.

stay tuned for 3-way quicksort

Digital properties of keys. We can use digit/character compares instead of key compares for numbers and strings.

stay tuned for radix sorts

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

2.2 MERGESORT

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

Algorithms

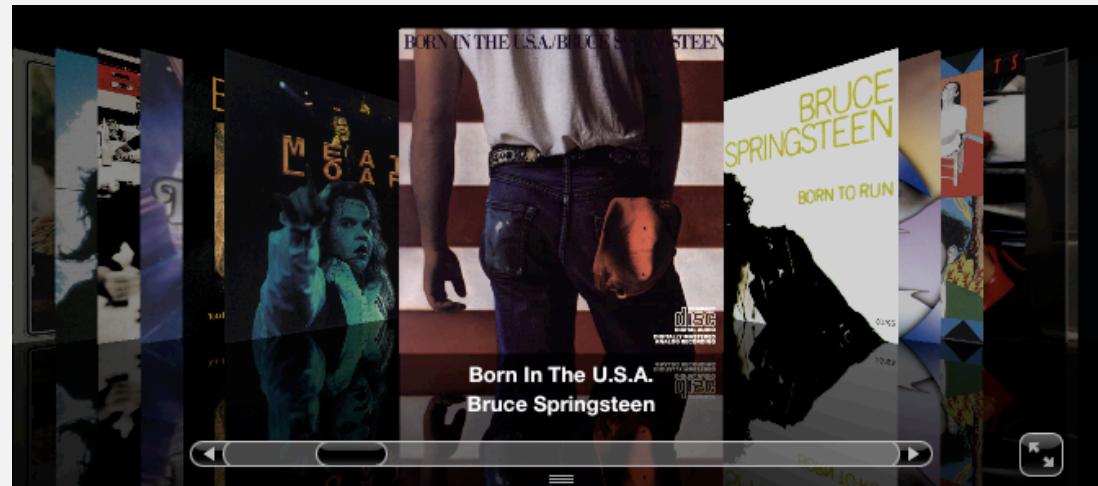
ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

2.2 MERGESORT

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

Sort music library by artist name



The image shows a music library interface with a list of songs and an album cover for 'Born In The U.S.A.' by Bruce Springsteen. The list is sorted by artist name, with Bruce Springsteen appearing at the top. The album cover for 'Born In The U.S.A.' is displayed prominently at the top of the interface.

| | Name | Artist | Time | Album |
|----|---|-------------------|------|---|
| 12 | <input checked="" type="checkbox"/> Let It Be | The Beatles | 4:03 | Let It Be |
| 13 | <input checked="" type="checkbox"/> Take My Breath Away | BERLIN | 4:13 | Top Gun – Soundtrack |
| 14 | <input checked="" type="checkbox"/> Circle Of Friends | Better Than Ezra | 3:27 | Empire Records |
| 15 | <input checked="" type="checkbox"/> Dancing With Myself | Billy Idol | 4:43 | Don't Stop |
| 16 | <input checked="" type="checkbox"/> Rebel Yell | Billy Idol | 4:49 | Rebel Yell |
| 17 | <input checked="" type="checkbox"/> Piano Man | Billy Joel | 5:36 | Greatest Hits Vol. 1 |
| 18 | <input checked="" type="checkbox"/> Pressure | Billy Joel | 3:16 | Greatest Hits, Vol. II (1978 – 1985) (Disc 2) |
| 19 | <input checked="" type="checkbox"/> The Longest Time | Billy Joel | 3:36 | Greatest Hits, Vol. II (1978 – 1985) (Disc 2) |
| 20 | <input checked="" type="checkbox"/> Atomic | Blondie | 3:50 | Atomic: The Very Best Of Blondie |
| 21 | <input checked="" type="checkbox"/> Sunday Girl | Blondie | 3:15 | Atomic: The Very Best Of Blondie |
| 22 | <input checked="" type="checkbox"/> Call Me | Blondie | 3:33 | Atomic: The Very Best Of Blondie |
| 23 | <input checked="" type="checkbox"/> Dreaming | Blondie | 3:06 | Atomic: The Very Best Of Blondie |
| 24 | <input checked="" type="checkbox"/> Hurricane | Bob Dylan | 8:32 | Desire |
| 25 | <input checked="" type="checkbox"/> The Times They Are A-Changin' | Bob Dylan | 3:17 | Greatest Hits |
| 26 | <input checked="" type="checkbox"/> Livin' On A Prayer | Bon Jovi | 4:11 | Cross Road |
| 27 | <input checked="" type="checkbox"/> Beds Of Roses | Bon Jovi | 6:35 | Cross Road |
| 28 | <input checked="" type="checkbox"/> Runaway | Bon Jovi | 3:53 | Cross Road |
| 29 | <input checked="" type="checkbox"/> Rasputin (Extended Mix) | Boney M | 5:50 | Greatest Hits |
| 30 | <input checked="" type="checkbox"/> Have You Ever Seen The Rain | Bonnie Tyler | 4:10 | Faster Than The Speed Of Night |
| 31 | <input checked="" type="checkbox"/> Total Eclipse Of The Heart | Bonnie Tyler | 7:02 | Faster Than The Speed Of Night |
| 32 | <input checked="" type="checkbox"/> Straight From The Heart | Bonnie Tyler | 3:41 | Faster Than The Speed Of Night |
| 33 | <input checked="" type="checkbox"/> Holding Out For A Hero | Bonny Tyler | 5:49 | Meat Loaf And Friends |
| 34 | <input checked="" type="checkbox"/> Dancing In The Dark | Bruce Springsteen | 4:05 | Born In The U.S.A. |
| 35 | <input checked="" type="checkbox"/> Thunder Road | Bruce Springsteen | 4:51 | Born To Run |
| 36 | <input checked="" type="checkbox"/> Born To Run | Bruce Springsteen | 4:30 | Born To Run |
| 37 | <input checked="" type="checkbox"/> Jungleland | Bruce Springsteen | 9:34 | Born To Run |
| 38 | <input checked="" type="checkbox"/> Turn! Turn! Turn! (To Everything) | The Byrds | 3:57 | Forrest Gump The Soundtrack (Disc 2) |

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... | 9:48 | |
| 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 | 3:45 | Time Life Music Sounds Of The Seventies – C |
| 27 | <input checked="" type="checkbox"/> Chaiwa Chaiwa | Sukhwinder Singh | 5:11 | Bombay Dreams |

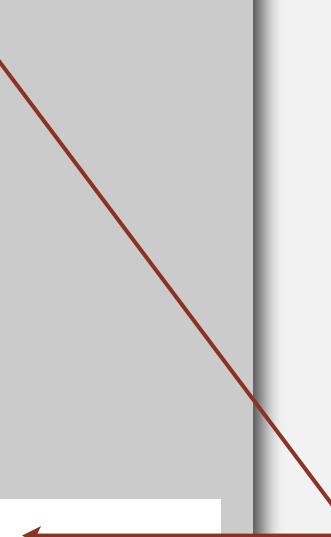
Comparable interface: review

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)
    {
        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;
    }
}
```



natural order

Comparator interface

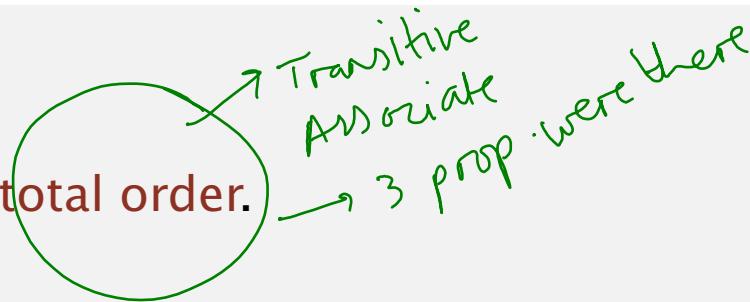
Comparator interface: sort using an alternate order.

```
public interface Comparator<Key>
```

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

compare keys v and w

Required property. Must be a **total order**.



Ex. Sort strings by:

- Natural order. Now is the time
- Case insensitive. is Now the time
- Spanish. café cafetero cuarto churro nube ñoño
- British phone book. McKinley Mackintosh
- . . .

pre-1994 order for
digraphs ch and ll and rr

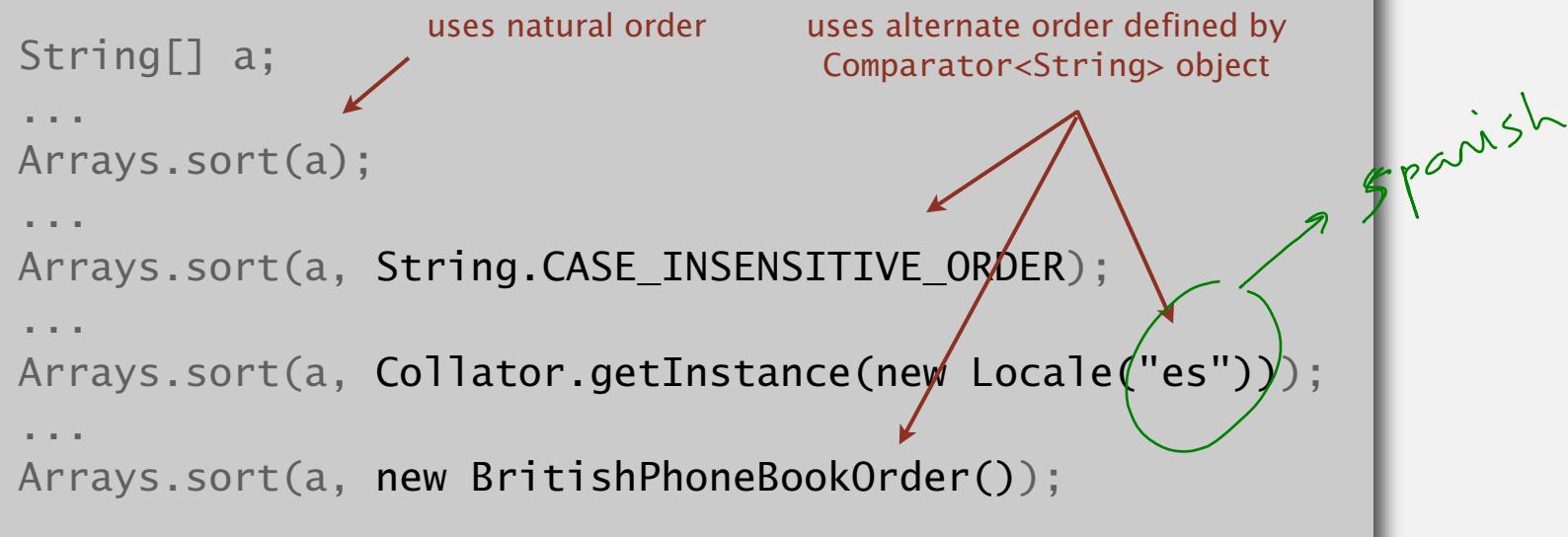


Comparator interface: system sort

To use with Java system sort:

- Create Comparator object.
- Pass as second argument to Arrays.sort().

```
String[] a;           uses natural order
...
Arrays.sort(a);       uses alternate order defined by
...
Arrays.sort(a, String.CASE_INSENSITIVE_ORDER);    Comparator<String> object
...
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--)
            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; }
    }
}
```

one Comparator for the class

Compares string

Compares string in a diff way

this technique works here since no danger of overflow

Comparator interface: implementing

To implement a comparator:

- Define a (nested) class that implements the Comparator interface.
- Implement the compare() method.

`Arrays.sort(a, Student.BY_NAME);`

| | | | | |
|---------|---|---|--------------|--------------|
| 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 |

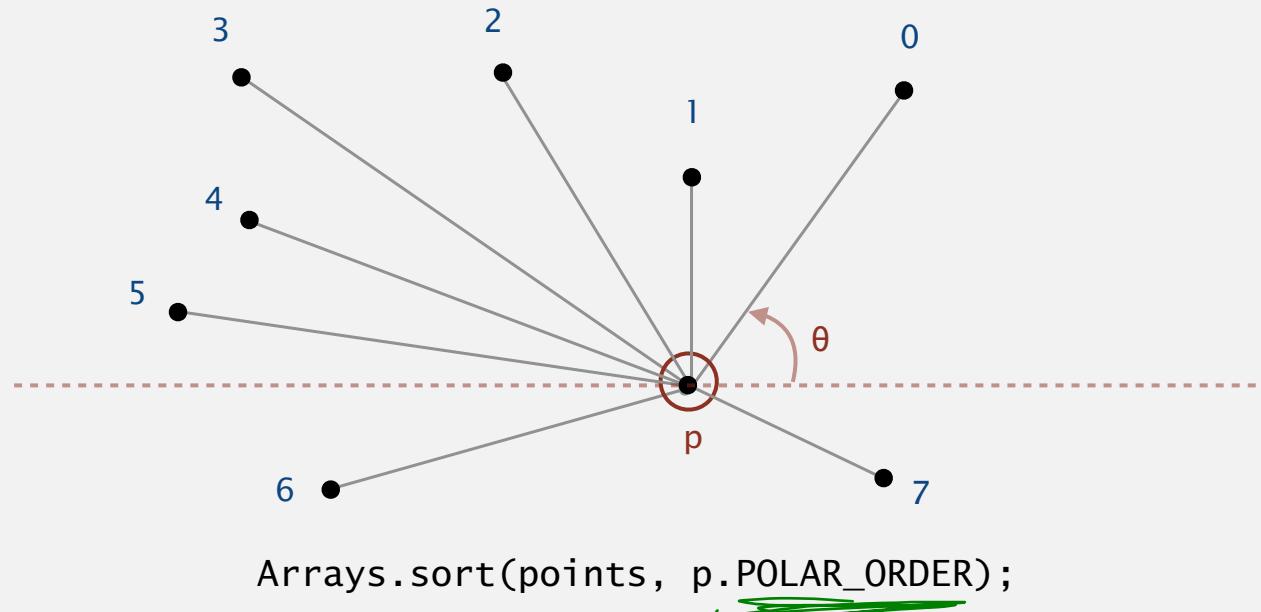
`Arrays.sort(a, Student.BY_SECTION);`

| | | | | |
|---------|---|---|--------------|--------------|
| 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 |

Polar order

→ sorting points

Polar order. Given a point p , order points by polar angle they make with p .



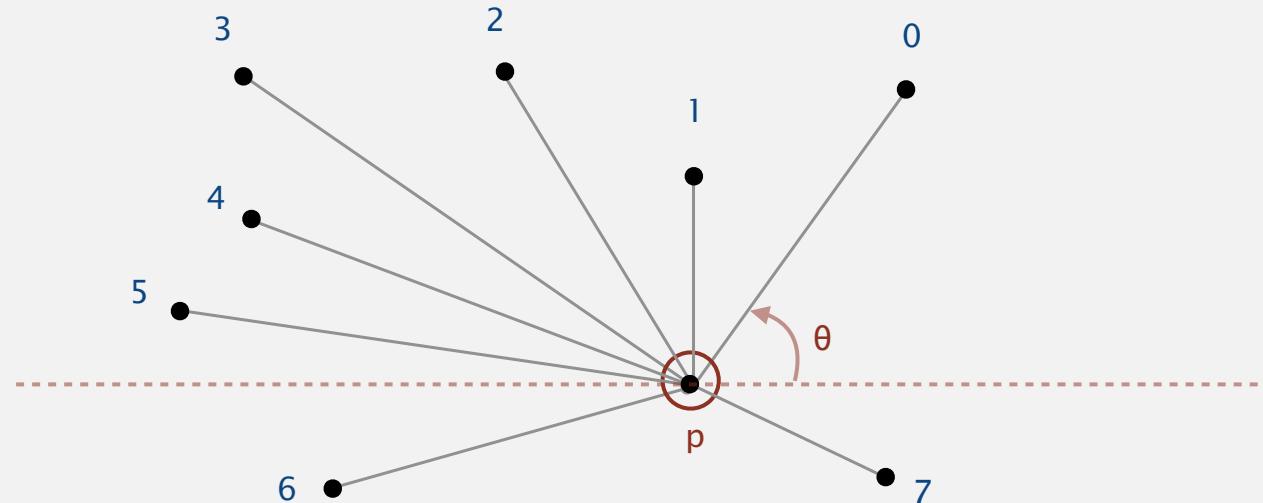
Application. Graham scan algorithm for convex hull. [see previous lecture]

High-school trig solution. Compute polar angle θ w.r.t. p using `atan2()`.

Drawback. Evaluating a trigonometric function is expensive.

Polar order

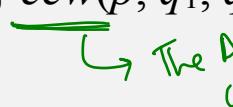
Polar order. Given a point p , order points by polar angle they make with p .



```
Arrays.sort(points, p.POLAR_ORDER);
```

A ccw-based solution.

- If q_1 is above p and q_2 is below p , then q_1 makes smaller polar angle.
- If q_1 is below p and q_2 is above p , then q_1 makes larger polar angle.
- Otherwise, $\text{ccw}(p, q_1, q_2)$ identifies which of q_1 or q_2 makes larger angle.

 The Area calculation

Comparator interface: polar order

```
public class Point2D
{
    public final Comparator<Point2D> POLAR_ORDER = new PolarOrder();
    private final double x, y;                         one Comparator for each point (not static)
    ...
    private static int ccw(Point2D a, Point2D b, Point2D c)
    { /* as in previous lecture */ }

    private class PolarOrder implements Comparator<Point2D>
    {
        public int compare(Point2D q1, Point2D q2)
        {
            double dy1 = q1.y - y;
            double dy2 = q2.y - y;

            if (dy1 == 0 && dy2 == 0) { ... }           p, q1, q2 horizontal
            else if (dy1 >= 0 && dy2 < 0) return -1;    q1 above p; q2 below p
            else if (dy2 >= 0 && dy1 < 0) return +1;    q1 below p; q2 above p
            else return -ccw(Point2D.this, q1, q2);       both above or below p
        }
    }
}
```

to access invoking point from within inner class

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

2.2 MERGESORT

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

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<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, Student.BY_NAME);`

| | | | | |
|---------|---|---|--------------|--------------|
| 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, Student.BY_SECTION);`

| | | | | |
|---------|---|---|--------------|--------------|
| 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 |

→ LOL

@#%&@! 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. Insertion sort and mergesort (but not selection sort or shellsort).

sorted by time

| | |
|---------|----------|
| Chicago | 09:00:00 |
| Phoenix | 09:00:03 |
| Houston | 09:00:13 |
| Chicago | 09:00:59 |
| Houston | 09:01:10 |
| Chicago | 09:03:13 |
| Seattle | 09:10:11 |
| Seattle | 09:10:25 |
| Phoenix | 09:14:25 |
| Chicago | 09:19:32 |
| Chicago | 09:19:46 |
| Chicago | 09:21:05 |
| Seattle | 09:22:43 |
| Seattle | 09:22:54 |
| Chicago | 09:25:52 |
| Chicago | 09:35:21 |
| Seattle | 09:36:14 |
| Phoenix | 09:37:44 |

sorted by location (not stable)

| | |
|---------|----------|
| Chicago | 09:25:52 |
| Chicago | 09:03:13 |
| Chicago | 09:21:05 |
| Chicago | 09:19:46 |
| Chicago | 09:19:32 |
| Chicago | 09:00:00 |
| Chicago | 09:35:21 |
| Chicago | 09:00:59 |
| Houston | 09:01:10 |
| Houston | 09:00:13 |
| Phoenix | 09:37:44 |
| Phoenix | 09:00:03 |
| Phoenix | 09:14:25 |
| Seattle | 09:10:25 |
| Seattle | 09:36:14 |
| Seattle | 09:22:43 |
| Seattle | 09:10:11 |
| Seattle | 09:22:54 |
| Seattle | 09:22:43 |
| Seattle | 09:36:14 |
| Seattle | 09:22:54 |
| Phoenix | 09:22:54 |

sorted by location (stable)

| | |
|---------|----------|
| Chicago | 09:00:00 |
| Chicago | 09:00:59 |
| Chicago | 09:03:13 |
| Chicago | 09:19:32 |
| Chicago | 09:19:46 |
| Chicago | 09:21:05 |
| Chicago | 09:25:52 |
| Chicago | 09:35:21 |
| Houston | 09:00:13 |
| Houston | 09:01:10 |
| Phoenix | 09:00:03 |
| Phoenix | 09:14:25 |
| Phoenix | 09:37:44 |
| Seattle | 09:10:11 |
| Seattle | 09:10:25 |
| Seattle | 09:22:43 |
| Seattle | 09:22:54 |
| Seattle | 09:36:14 |

Note. Need to carefully check code ("less than" vs. "less than or equal to").

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);
    }
}
```

↓ if $i=2$,
↓ then A_2
↓ would have
↓ swapped.

| i | j | 0 | 1 | 2 | 3 | 4 |
|---|---|----------------------|----------------------|----------------------|----------------|----------------------|
| 0 | 0 | B₁ | A ₁ | A ₂ | A ₃ | B ₂ |
| 1 | 0 | A₁ | B ₁ | A ₂ | A ₃ | B ₂ |
| 2 | 1 | A ₁ | A₂ | B ₁ | A ₃ | B ₂ |
| 3 | 2 | A ₁ | A ₂ | A₃ | B ₁ | B ₂ |
| 4 | 4 | A ₁ | A ₂ | A ₃ | B ₁ | B₂ |
| | | A ₁ | A ₂ | A ₃ | B ₁ | B ₂ |

Pf. Equal items never move past each other.

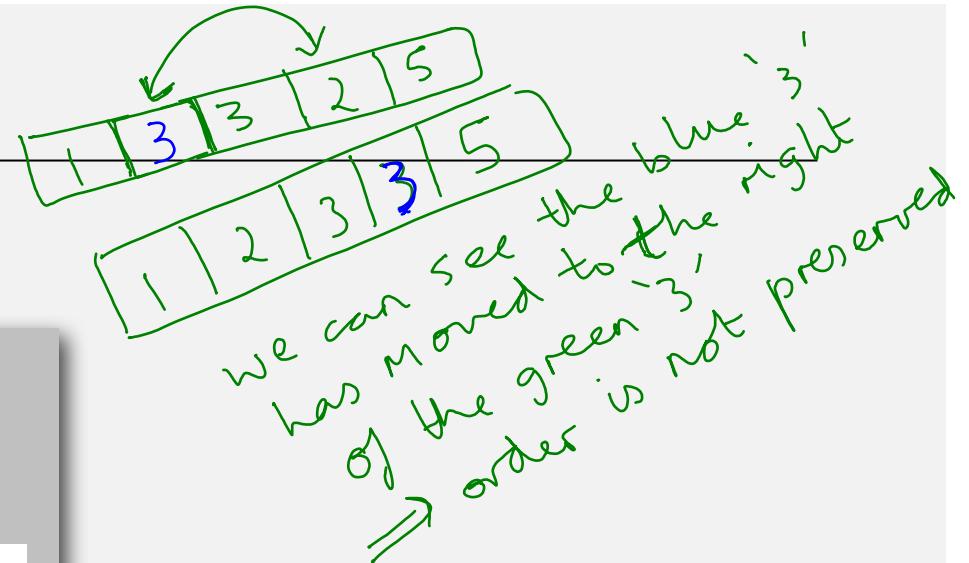
Now when we do less(a[i], a[j]) this gives false a[i] is NOT less than a[j] \Rightarrow items stay in place in terms of the original order

If there were 2 elements were 2 A₃ appears twice, say at pos $i=3$ & $i=4$ say we are at $i=4$ $\therefore A_3 A_3 \dots$

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 |

Pf by counterexample. Long-distance exchange might move an item past some 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 Comparable[] aux;
    private static void merge(Comparable[] a, int lo, int mid, int hi)
    { /* as before */ }

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

If 2 keys have equal value, we pick from the "left" sub array. \rightarrow Stability as order is preserved

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



If $A_i = A_i^4$ then A_i is put into "a" array



Pf. Takes from left subarray if equal keys.

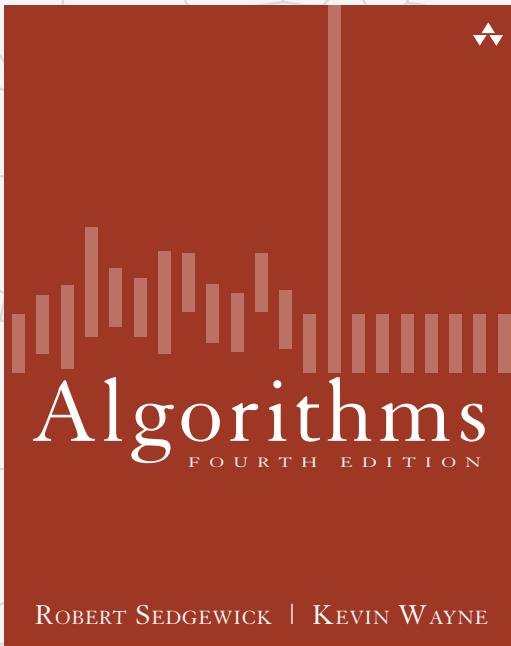
Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

2.2 MERGESORT

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



2.2 MERGESORT

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