2.2 Mergesort

- mergesort
- bottom-up mergesort
- sorting complexity
- comparators
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.
• Java sort for objects.
• Perl, Python stable sort.

Quicksort.
• Java sort for primitive types.
• C qsort, Unix, g++, Visual C++, Python.
- mergesort
- bottom-up mergesort
- sorting complexity
- comparators
Mergesort

Basic plan.
• Divide array into two halves.
• Recursively sort each half.
• Merge two halves.

input: `MERGESORT EXAMPLE`
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 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
**Merging**

**Q.** How to combine two sorted subarrays into a sorted whole.  
**A.** Use an auxiliary array.

<table>
<thead>
<tr>
<th>k</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>input</td>
<td>E</td>
<td>E</td>
<td>G</td>
<td>M</td>
<td>R</td>
<td>A</td>
<td>C</td>
<td>E</td>
<td>R</td>
<td>T</td>
</tr>
<tr>
<td>copy</td>
<td>E</td>
<td>E</td>
<td>G</td>
<td>M</td>
<td>R</td>
<td>A</td>
<td>C</td>
<td>E</td>
<td>R</td>
<td>T</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>i</th>
<th>j</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>aux[]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
<tr>
<td>---</td>
</tr>
<tr>
<td>E</td>
</tr>
<tr>
<td>E</td>
</tr>
</tbody>
</table>

Abstract in-place merge trace

merged result

<table>
<thead>
<tr>
<th>A</th>
<th>C</th>
<th>E</th>
<th>E</th>
<th>E</th>
<th>G</th>
<th>M</th>
<th>R</th>
<th>R</th>
<th>T</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>A</th>
<th>C</th>
<th>E</th>
<th>E</th>
<th>E</th>
<th>G</th>
<th>M</th>
<th>R</th>
<th>R</th>
<th>T</th>
</tr>
</thead>
</table>

| T |
Merging: Java implementation

```java
private static void merge(Comparable[] a, 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++)
        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++];
    }

    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 an exception unless boolean condition is true.

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

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

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

**Best practices.** Use to check internal invariants. Assume assertions will be disabled in production code (e.g., don't use for external argument-checking).
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, m, hi);
   }

   public static void sort(Comparable[] a)
   {
      aux = new Comparable[a.length];
      sort(a, 0, a.length - 1);
   }
}
Mergesort trace

Trace of merge results for top-down mergesort

result after recursive call
Mergesort animation

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

50 reverse-sorted elements

http://www.sorting-algorithms.com/merge-sort
Mergesort: empirical analysis

Running time estimates:
- Home pc executes $10^8$ comparisons/second.
- Supercomputer executes $10^{12}$ comparisons/second.

<table>
<thead>
<tr>
<th></th>
<th>insertion sort ($N^2$)</th>
<th>mergesort ($N \log N$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>computer</td>
<td>thousand</td>
<td>million</td>
</tr>
<tr>
<td>home</td>
<td>instant</td>
<td>2.8 hours</td>
</tr>
<tr>
<td>super</td>
<td>instant</td>
<td>1 second</td>
</tr>
</tbody>
</table>

Bottom line. Good algorithms are better than supercomputers.
**Mergesort: mathematical analysis**

**Proposition.** Mergesort uses $\sim 2\, N \lg N$ data moves to sort any array of size $N$.

**Def.** $D(N) =$ number of data moves to mergesort an array of size $N$.

\[
D(N) = D(N/2) + D(N/2) + 2\, N
\]

**Mergesort recurrence.** $D(N) = 2\, D(N/2) + 2\, N$ for $N > 1$, with $T(1) = 0$.

- Not quite right for odd $N$.
- Similar recurrence holds for many divide-and-conquer algorithms.

**Solution.** $D(N) \sim 2\, N \lg N$.

- For simplicity, we'll prove when $N$ is a power of 2.
- True for all $N$. [see COS 340]
**Mergesort recurrence:** proof 1

**Mergesort recurrence.** \( D(N) = 2 \ D(N/2) + 2 \ N \) for \( N > 1 \), with \( D(1) = 0 \).

**Proposition.** If \( N \) is a power of 2, then \( D(N) = 2 \ N \lg N \).

**Pf.**

\[
\begin{align*}
2N & = 2N \\
2 (2N/2) & = 2N \\
4 (2N/4) & = 2N \\
\cdots & \\
2^k (2N/2^k) & = 2N \\
\cdots & \\
N/2 (4) & = 2N \\
\hline
2N \lg N &
\end{align*}
\]
Mergesort recurrence: proof 2

Mergesort recurrence. \( D(N) = 2 \, D(N/2) + 2 \, N \) for \( N > 1 \), with \( D(1) = 0 \).

Proposition. If \( N \) is a power of 2, then \( D(N) = 2 \, N \, \lg \, N \).

Pf.

\[
\begin{align*}
D(N) &= 2 \, D(N/2) + 2N \\
D(N) / N &= 2 \, D(N/2) / N + 2 \\
&= D(N/2) / (N/2) + 2 \\
&= D(N/4) / (N/4) + 2 + 2 \\
&= D(N/8) / (N/8) + 2 + 2 + 2 \\
&\vdots \\
&= D(N/N) / (N/N) + 2 + 2 + \ldots + 2 \\
&= 2 \, \lg \, N
\end{align*}
\]

- given
- divide both sides by \( N \)
- algebra
- apply to first term
- apply to first term again
- stop applying, \( T(1) = 0 \)
Mergesort recurrence: proof 3

**Mergesort recurrence.** $D(N) = 2 \, D(N/2) + 2 \, N$ for $N > 1$, with $D(1) = 0$.

**Proposition.** If $N$ is a power of 2, then $D(N) = 2 \, N \, \lg N$.

**Pf.** [by induction on $N$]
- **Base case:** $N = 1$.
- **Inductive hypothesis:** $D(N) = 2N \, \lg N$.
- **Goal:** show that $D(2N) = 2(2N)\lg (2N)$.

\[
D(2N) = 2 \, D(N) + 4N \\
= 4 \, N \, \lg N + 4 \, N \\
= 4 \, N \, (\lg (2N) - 1) + 4N \\
= 4 \, N \, \lg (2N)
\]

given
inductive hypothesis
algebra
QED
**Proposition.** Mergesort uses between \( \frac{1}{2} N \lg N \) and \( N \lg N \) compares to sort any array of size \( N \).

**Pf.** The number of compares for the last merge is between \( \frac{1}{2} N \lg N \) and \( N \).
**Mergesort analysis: memory**

**Proposition G.** Mergesort uses extra space proportional to $N$.

**Pf.** The array `aux[]` needs to be of size $N$ for the last merge.

**Def.** A sorting algorithm is **in-place** if it uses $O(\log N)$ extra memory.

**Ex.** Insertion sort, selection sort, shellsort.

**Challenge for the bored.** In-place merge. [Kronrud, 1969]
Mergesort: practical improvements

Use insertion sort for small subarrays.
- Mergesort has too much overhead for tiny subarrays.
- Cutoff to insertion sort for \( \approx 7 \) elements.

Stop if already sorted.
- Is biggest element in first half \( \leq \) smallest element in second half?
- Helps for partially-ordered arrays.

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.

Ex. See MergeX.java or Arrays.sort().
Mergesort visualization

Visual trace of top-down mergesort for with cutoff for small subarrays
- mergesort
- bottom-up mergesort
- sorting complexity
- comparators
**Bottom-up mergesort**

**Basic plan.**
- Pass through array, merging subarrays of size 1.
- Repeat for subarrays of size 2, 4, 8, 16, ...

**Bottom line.** No recursion needed!
Bottom-up mergesort: Java implementation

```java
public class MergeBU
{
    private static Comparable[] aux;

    private static void merge(Comparable[] a, int lo, int mid, int hi)
    {
        /* as before */
    }

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

**Bottom line.** Concise industrial-strength code, if you have the space.
Bottom-up mergesort: visual trace

Visual trace of bottom-up mergesort
- mergesort
- bottom-up mergesort
- sorting complexity
- comparators
Complexity of sorting

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

**Machine model.** Focus on fundamental operations.

**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 cost guarantee for X.

Example: sorting.

- Machine model = # compares.
- Upper bound = ~ N \lg N from mergesort.
- Lower bound = ~ N \lg N?
- Optimal algorithm = mergesort?
Decision tree (for 3 distinct elements)

```
\begin{align*}
    a &< b \\
    \text{yes} & \quad \text{no} \\
    b &< c \\
    \text{yes} & \quad \text{no} \\
    a &< c \\
    \text{yes} & \quad \text{no} \\
    \text{code between compares} & \\
    \text{(e.g., sequence of exchanges)} \\
\end{align*}
```

height of tree = worst-case number of compares
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 input 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.
Proposition. Any compare-based sorting algorithm must use at least \( \lg N! \sim N \lg N \) compares in the worst-case.

Pf.

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

\[
2^h \geq \# \text{ leaves} \geq N!
\Rightarrow h \geq \lg N! \sim N \lg N
\]
Complexity of sorting

**Machine model.** Focus on fundamental operations.

**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 cost guarantee for X.

**Example: sorting.**

- **Machine model = # compares.**
- **Upper bound = ~ N lg N from mergesort.**
- **Lower bound = ~ N lg N.**
- **Optimal algorithm = mergesort.**

**First goal of algorithm design:** optimal algorithms.
Complexity results in context

**Other operations?** Mergesort optimality is only about number of compares.

**Space?**
- Mergesort is **not optimal** with respect to space usage.
- Insertion sort, selection sort, and shellsort are space-optimal.

**Challenge.** Find an algorithm that is both time- and space-optimal.

**Lessons.** Use theory as a guide.

**Ex.** Don't try to design sorting algorithm that uses \( \frac{1}{2} N \lg N \) compares.
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.

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

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

**Digital properties of keys.** We can use digit/character compares instead of key compares for numbers and strings.
- mergesort
- bottom-up mergesort
- sorting complexity
- comparators
Sort by artist name

<table>
<thead>
<tr>
<th>Name</th>
<th>Artist</th>
<th>Time</th>
<th>Album</th>
</tr>
</thead>
<tbody>
<tr>
<td>Let It Be</td>
<td>The Beatles</td>
<td>4:03</td>
<td>Let It Be</td>
</tr>
<tr>
<td>Take My Breath Away</td>
<td>BERLIN</td>
<td>4:13</td>
<td>Top Gun - Soundtrack</td>
</tr>
<tr>
<td>Circle Of Friends</td>
<td>Better Than Ezra</td>
<td>3:27</td>
<td>Empire Records</td>
</tr>
<tr>
<td>Dancing With Myself</td>
<td>Billy Idol</td>
<td>4:43</td>
<td>Don't Stop</td>
</tr>
<tr>
<td>Rebel Yell</td>
<td>Billy Idol</td>
<td>4:49</td>
<td>Rebel Yell</td>
</tr>
<tr>
<td>Piano Man</td>
<td>Billy Joel</td>
<td>5:36</td>
<td>Greatest Hits Vol. 1</td>
</tr>
<tr>
<td>Atomic</td>
<td>Blondie</td>
<td>3:50</td>
<td>Atomic: The Very Best Of Blondie</td>
</tr>
<tr>
<td>Sunday Girl</td>
<td>Blondie</td>
<td>3:15</td>
<td>Atomic: The Very Best Of Blondie</td>
</tr>
<tr>
<td>Call Me</td>
<td>Blondie</td>
<td>3:33</td>
<td>Atomic: The Very Best Of Blondie</td>
</tr>
<tr>
<td>Dreaming</td>
<td>Blondie</td>
<td>3:06</td>
<td>Atomic: The Very Best Of Blondie</td>
</tr>
<tr>
<td>Hurricane</td>
<td>Bob Dylan</td>
<td>8:32</td>
<td>Desire</td>
</tr>
<tr>
<td>The Times They Are A-Changin'</td>
<td>Bob Dylan</td>
<td>3:17</td>
<td>Greatest Hits</td>
</tr>
<tr>
<td>Livin' On A Prayer</td>
<td>Bon Jovi</td>
<td>4:11</td>
<td>Cross Road</td>
</tr>
<tr>
<td>Beds Of Roses</td>
<td>Bon Jovi</td>
<td>6:35</td>
<td>Cross Road</td>
</tr>
<tr>
<td>Runaway</td>
<td>Bon Jovi</td>
<td>3:53</td>
<td>Cross Road</td>
</tr>
<tr>
<td>Rasputin</td>
<td>Boney M</td>
<td>5:50</td>
<td>Greatest Hits</td>
</tr>
<tr>
<td>Have You Ever Seen The Rain</td>
<td>Bonnie Tyler</td>
<td>4:10</td>
<td>Faster Than The Speed Of Night</td>
</tr>
<tr>
<td>Total Eclipse Of The Heart</td>
<td>Bonnie Tyler</td>
<td>7:02</td>
<td>Faster Than The Speed Of Night</td>
</tr>
<tr>
<td>Straight From The Heart</td>
<td>Bonnie Tyler</td>
<td>3:41</td>
<td>Faster Than The Speed Of Night</td>
</tr>
<tr>
<td>Holding Out For A Hero</td>
<td>Bonny Tyler</td>
<td>5:49</td>
<td>Meat Loaf And Friends</td>
</tr>
<tr>
<td>Dancing In The Dark</td>
<td>Bruce Springsteen</td>
<td>4:05</td>
<td>Born To Run</td>
</tr>
<tr>
<td>Thunder Road</td>
<td>Bruce Springsteen</td>
<td>4:51</td>
<td>Born To Run</td>
</tr>
<tr>
<td>Born To Run</td>
<td>Bruce Springsteen</td>
<td>4:30</td>
<td>Born To Run</td>
</tr>
<tr>
<td>Jungleland</td>
<td>Bruce Springsteen</td>
<td>9:34</td>
<td>Born To Run</td>
</tr>
<tr>
<td>Born To Run</td>
<td>Bruce Springsteen</td>
<td>4:30</td>
<td>Born To Run</td>
</tr>
<tr>
<td>Total Total</td>
<td>Bruce Springsteen</td>
<td>3:57</td>
<td>Format Every Other Track</td>
</tr>
</tbody>
</table>

34
Sort by song name

<table>
<thead>
<tr>
<th>Name</th>
<th>Artist</th>
<th>Time</th>
<th>Album</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alive</td>
<td>Pearl Jam</td>
<td>5:41</td>
<td>Ten</td>
</tr>
<tr>
<td>All Over The World</td>
<td>Pixies</td>
<td>5:27</td>
<td>Bossanova</td>
</tr>
<tr>
<td>All Through The Night</td>
<td>Cyndi Lauper</td>
<td>4:30</td>
<td>She's So Unusual</td>
</tr>
<tr>
<td>Allison Road</td>
<td>Gin Blossoms</td>
<td>3:19</td>
<td>New Miserable Experience</td>
</tr>
<tr>
<td>Aria, Aria, Aria Y Ensancha El</td>
<td>Extremoduro</td>
<td>2:34</td>
<td>Deltoya (1992)</td>
</tr>
<tr>
<td>And We Danced</td>
<td>Hooters</td>
<td>3:50</td>
<td>Nervous Night</td>
</tr>
<tr>
<td>As I Lay Me Down</td>
<td>Sophie B. Hawkins</td>
<td>4:09</td>
<td>Whaler</td>
</tr>
<tr>
<td>Atomic</td>
<td>Blondie</td>
<td>3:50</td>
<td>Atomic: The Very Best Of Blondie</td>
</tr>
<tr>
<td>Automatic Lover</td>
<td>Jay-Jay Johanson</td>
<td>4:19</td>
<td>Antenna</td>
</tr>
<tr>
<td>Baba O'Reiley</td>
<td>The Who</td>
<td>5:01</td>
<td>Who's Better, Who's Best</td>
</tr>
<tr>
<td>Beautiful Life</td>
<td>Ace Of Base</td>
<td>3:40</td>
<td>The Bridge</td>
</tr>
<tr>
<td>Beds Of Roses</td>
<td>Bon Jovi</td>
<td>6:35</td>
<td>Cross Road</td>
</tr>
<tr>
<td>Black</td>
<td>Pearl Jam</td>
<td>5:44</td>
<td>Ten</td>
</tr>
<tr>
<td>Bleed American</td>
<td>Jimmy Eat World</td>
<td>3:04</td>
<td>Bleed American</td>
</tr>
<tr>
<td>Borderline</td>
<td>Madonna</td>
<td>4:00</td>
<td>The Immaculate Collection</td>
</tr>
<tr>
<td>Born To Run</td>
<td>Bruce Springsteen</td>
<td>4:30</td>
<td>Born To Run</td>
</tr>
<tr>
<td>Both Sides Of The Story</td>
<td>Phil Collins</td>
<td>6:43</td>
<td>Both Sides</td>
</tr>
<tr>
<td>Bouncing Around The Room</td>
<td>Phish</td>
<td>4:09</td>
<td>A Live One (Disc 1)</td>
</tr>
<tr>
<td>Boys Don't Cry</td>
<td>The Cure</td>
<td>2:35</td>
<td>Staring At The Sea: The Singles 1979–1985</td>
</tr>
<tr>
<td>Brat</td>
<td>Green Day</td>
<td>1:43</td>
<td>Insomniac</td>
</tr>
<tr>
<td>Breakdown</td>
<td>Deerheart</td>
<td>3:40</td>
<td>Deerheart</td>
</tr>
<tr>
<td>Bring Me To Life (Kevin Roen Mix)</td>
<td>Enniscise</td>
<td>9:48</td>
<td></td>
</tr>
<tr>
<td>California</td>
<td>Red Hot Chili Peppers</td>
<td>1:40</td>
<td></td>
</tr>
<tr>
<td>Call Me</td>
<td>Blondie</td>
<td>3:33</td>
<td>Atomic: The Very Best Of Blondie</td>
</tr>
<tr>
<td>Can't Get You Out Of My Head</td>
<td>Kylie Minogue</td>
<td>3:50</td>
<td>Fever</td>
</tr>
<tr>
<td>Celebration</td>
<td>Kool &amp; The Gang</td>
<td>3:45</td>
<td>Time Life Music Sounds Of The Seventies + C</td>
</tr>
<tr>
<td>Chasing Changes</td>
<td>Subradores Singh</td>
<td>5:11</td>
<td>Bon Jovi: Dreamer</td>
</tr>
</tbody>
</table>
Comparable interface: sort uses type's natural order.
**Generalized compare**

**Comparable interface:** sort uses type's natural order.

- **Problem 1.** May want to use a non-natural order.
- **Problem 2.** Desired data type may not come with a “natural” order.

**Ex.** Sort strings by:
- Natural order.
- Case insensitive.
- Spanish.
- British phone book.

```java
String[] a;
...
Arrays.sort(a);
Arrays.sort(a, String.CASE_INSENSITIVE_ORDER);
Arrays.sort(a, Collator.getInstance(Locale.SPANISH));
```

Import java.text.Collator;
Comparators

Solution. Use Java's `comparator` interface.

```java
public interface Comparator<Key>
{
   public int compare(Key v, Key w);
}
```

Remark. The `compare()` method implements a total order like `compareTo()`.

Advantages. Decouples the definition of the data type from the definition of what it means to compare two objects of that type.
- Can add any number of new orders to a data type.
- Can add an order to a library data type with no natural order.
Comparator example

Reverse order. Sort an array of strings in reverse order.

```java
public class ReverseOrder implements Comparator<String>
{
    public int compare(String a, String b)
    {
        return b.compareTo(a);
    }
}
```

... 
```java
Arrays.sort(a, new ReverseOrder());
... 
```

client
Sort implementation with comparators

To support comparators in our sort implementations:
• Pass Comparator to sort() and less().
• Use it in less().

Ex. Insertion sort.

```java
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;  }
```
Generalized compare

Comparators enable multiple sorts of a single array (by different keys).

Ex. Sort students by name or by section.

Arrays.sort(students, Student.BY_NAME);
Arrays.sort(students, Student.BY_SECT);
**Generalized compare**

**Ex.** Enable sorting students by name or by section.

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

    private final String name;
    private final int section;
    ...

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

    private static class BySect implements Comparator<Student> {
        public int compare(Student a, Student b) {
            return a.section - b.section;
        }
    }
}
```

*only use this trick if no danger of overflow*
Generalized compare problem

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

Arrays.sort(students, Student.BY_NAME);

Andrews  3  A  664-480-0023  097 Little
Battle  4  C  874-088-1212  121 Whitman
Chen  2  A  991-878-4944  308 Blair
Fox  1  A  884-232-5341  11 Dickinson
Furia  3  A  766-093-9873  101 Brown
Gazsi  4  B  665-303-0266  22 Brown
Kanaga  3  B  898-122-9643  22 Brown
Rohde  3  A  232-343-5555  343 Forbes

Arrays.sort(students, Student.BY_SECT);

Fox  1  A  884-232-5341  11 Dickinson
Chen  2  A  991-878-4944  308 Blair
Kanaga  3  B  898-122-9643  22 Brown
Andrews  3  A  664-480-0023  097 Little
Furia  3  A  766-093-9873  101 Brown
Rohde  3  A  232-343-5555  343 Forbes
Battle  4  C  874-088-1212  121 Whitman
Gazsi  4  B  665-303-0266  22 Brown

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

A stable sort preserves the relative order of records with equal keys.
**Sorting challenge 5**

**Q.** Which sorts are stable?

- Insertion sort?
- Selection sort?
- Shellsort?
- Mergesort?

<table>
<thead>
<tr>
<th>sorted by time</th>
<th>sorted by location (not stable)</th>
<th>sorted by location (stable)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chicago 09:00:00</td>
<td>Chicago 09:25:52</td>
<td>Chicago 09:00:00</td>
</tr>
<tr>
<td>Phoenix 09:00:03</td>
<td>Chicago 09:03:13</td>
<td>Chicago 09:00:59</td>
</tr>
<tr>
<td>Houston 09:00:13</td>
<td>Chicago 09:21:05</td>
<td>Chicago 09:19:32</td>
</tr>
<tr>
<td>Chicago 09:00:59</td>
<td>Chicago 09:19:46</td>
<td>Chicago 09:19:46</td>
</tr>
<tr>
<td>Houston 09:01:10</td>
<td>Chicago 09:19:32</td>
<td>Chicago 09:19:32</td>
</tr>
<tr>
<td>Chicago 09:03:13</td>
<td>Chicago 09:00:00</td>
<td>Chicago 09:21:05</td>
</tr>
<tr>
<td>Seattle 09:10:11</td>
<td>Chicago 09:35:21</td>
<td>Chicago 09:25:52</td>
</tr>
<tr>
<td>Seattle 09:10:25</td>
<td>Chicago 09:00:59</td>
<td>Chicago 09:35:21</td>
</tr>
<tr>
<td>Phoenix 09:14:25</td>
<td>Houston 09:01:10</td>
<td>Houston 09:00:13</td>
</tr>
<tr>
<td>Chicago 09:19:32</td>
<td>Houston 09:00:13</td>
<td>Houston 09:01:10</td>
</tr>
<tr>
<td>Chicago 09:19:46</td>
<td>Phoenix 09:37:44</td>
<td>Phoenix 09:00:03</td>
</tr>
<tr>
<td>Chicago 09:21:05</td>
<td>Phoenix 09:00:03</td>
<td>Phoenix 09:14:25</td>
</tr>
<tr>
<td>Seattle 09:22:54</td>
<td>Seattle 09:10:25</td>
<td>Seattle 09:10:11</td>
</tr>
<tr>
<td>Chicago 09:25:52</td>
<td>Seattle 09:36:14</td>
<td>Seattle 09:10:25</td>
</tr>
<tr>
<td>Seattle 09:36:14</td>
<td>Seattle 09:10:11</td>
<td>Seattle 09:22:54</td>
</tr>
<tr>
<td>Phoenix 09:37:44</td>
<td>Seattle 09:22:54</td>
<td>Seattle 09:36:14</td>
</tr>
</tbody>
</table>

**Stability when sorting on a second key**

- No longer sorted by time
- Still sorted by time
- mergesort
- bottom-up mergesort
- sorting complexity
- comparators
- sorting challenge
Is insertion sort 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);
    }
}
```

A. Yes, equal elements never more past each other.
Q. Is selection sort stable?

A. No, long-distance exchange might move left element to the right of some equal element.
Q. Is shellsort stable?

A. No. Long-distance exchanges.

```java
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;
            }
        }
    }
}
```
Q. Is mergesort stable?

```java
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)
    {
        aux = new Comparable[a.length];
        sort(a, 0, a.length - 1);
    }
}
```
Q. Is mergesort stable?

A. Yes, if merge is stable.
Sorting challenge 5D (continued)

Q. Is merge stable?

A. Yes, if implemented carefully (take from left subarray if equal).
Sorting challenge 5 (summary)

Q. Which sorts are stable?

Yes. Insertion sort, mergesort.
No. Selection sort, shellsort.

Note. Need to carefully check code (“less than” vs “less than or equal”).
Postscript: optimizing mergesort (a short history)

**Goal.** Remove instructions from the inner loop.

```java
private static void merge(Comparable[] a, 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++];
}
```
Postscript: optimizing mergesort (a short history)

**Idea 1 (1960s). Use sentinels.**

```c
a[M] := maxint; b[N] := maxint;
for (int i = 0, j = 0, k = 0; k < M+1; k++)
   if (less(aux[j], aux[i])) aux[k] = a[i++];
   aux[k] = b[j++];
```

**Problem 1.** Still need copy.

**Problem 2.** No good place to put sentinels.

**Problem 3.** Complicates data-type interface (what is infinity for your type?)
Postscript: Optimizing mergesort (a short history)


```java
private static void merge(Comparable[] a, int lo, int mid, int hi) {
    for (int i = lo; i <= mid; i++)
        aux[i] = a[i]; // copy

    for (int j = mid+1; j <= hi; j++)
        aux[j] = a[hi-j+mid+1]; // reverse copy

    int i = lo, j = hi;
    for (int k = lo; k <= hi; k++)
        if (less(aux[j], aux[i])) a[k] = aux[j--];
        else                      a[k] = aux[i++]; // merge
}
```

Problem. Copy still in inner loop.
Postscript: Optimizing mergesort (a short history)


```java
int mid = (lo+hi)/2;
mergesortABr(b, a, lo, mid);
mergesortABr(b, a, mid+1, r);
mergeAB(a, lo, b, lo, mid, b, mid+1, hi);
```

Problem. Complex interactions with reverse copy.
Solution. Go back to sentinels.

Arrays.sort()
Sorting challenge 6

Problem. Choose mergesort for Algs 4th edition. Recursive argument switch is out (recommended only for pros).

Q. Why not use reverse array copy?

```java
private static void merge(Comparable[] a, int lo, int mid, int hi)
{
    for (int i = lo; i <= mid; i++)
       aux[i] = a[i];

    for (int j = mid+1; j <= hi; j++)
       aux[j] = a[hi-j+mid+1];

    int i = lo, j = hi;
    for (int k = lo; k <= hi; k++)
       if (less(aux[j], aux[i])) a[k] = aux[j--];
       else                     a[k] = aux[i++];
}
```