2.1 Elementary Sorts

- rules of the game
- selection sort
- insertion sort
- shuffling
- comparators
- stability
2.1 Elementary Sorts

- rules of the game
- selection sort
- insertion sort
- shuffling
- comparators
- stability
### Sorting problem

**Ex.** Student records in a university.

<table>
<thead>
<tr>
<th>Item</th>
<th>Key</th>
<th>Item</th>
<th>Key</th>
<th>Phone</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chen</td>
<td>3</td>
<td>A</td>
<td>(991) 878–4944</td>
<td>308 Blair</td>
<td></td>
</tr>
<tr>
<td>Rohde</td>
<td>2</td>
<td>A</td>
<td>(232) 343–5555</td>
<td>343 Forbes</td>
<td></td>
</tr>
<tr>
<td>Gazsi</td>
<td>4</td>
<td>B</td>
<td>(800) 867–5309</td>
<td>101 Brown</td>
<td></td>
</tr>
<tr>
<td>Furia</td>
<td>1</td>
<td>A</td>
<td>(766) 093–9873</td>
<td>101 Brown</td>
<td></td>
</tr>
<tr>
<td>Kanaga</td>
<td>3</td>
<td>B</td>
<td>(898) 122–9643</td>
<td>22 Brown</td>
<td></td>
</tr>
<tr>
<td>Andrews</td>
<td>3</td>
<td>A</td>
<td>(664) 480–0023</td>
<td>097 Little</td>
<td></td>
</tr>
<tr>
<td>Battle</td>
<td>4</td>
<td>C</td>
<td>(874) 088–1212</td>
<td>121 Whitman</td>
<td></td>
</tr>
</tbody>
</table>

**Sort.** Rearrange array of $n$ items in ascending order by key.

<table>
<thead>
<tr>
<th>Item</th>
<th>Key</th>
<th>Item</th>
<th>Key</th>
<th>Phone</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>Andrews</td>
<td>3</td>
<td>A</td>
<td>(664) 480–0023</td>
<td>097 Little</td>
<td></td>
</tr>
<tr>
<td>Battle</td>
<td>4</td>
<td>C</td>
<td>(874) 088–1212</td>
<td>121 Whitman</td>
<td></td>
</tr>
<tr>
<td>Chen</td>
<td>3</td>
<td>A</td>
<td>(991) 878–4944</td>
<td>308 Blair</td>
<td></td>
</tr>
<tr>
<td>Furia</td>
<td>1</td>
<td>A</td>
<td>(766) 093–9873</td>
<td>101 Brown</td>
<td></td>
</tr>
<tr>
<td>Gazsi</td>
<td>4</td>
<td>B</td>
<td>(800) 867–5309</td>
<td>101 Brown</td>
<td></td>
</tr>
<tr>
<td>Kanaga</td>
<td>3</td>
<td>B</td>
<td>(898) 122–9643</td>
<td>22 Brown</td>
<td></td>
</tr>
<tr>
<td>Rohde</td>
<td>2</td>
<td>A</td>
<td>(232) 343–5555</td>
<td>343 Forbes</td>
<td></td>
</tr>
</tbody>
</table>
Total order

Sorting is a well-defined problem if and only if there is a total order.

A total order is a binary relation ≤ that satisfies:
- Totality: either \( v \leq w \) or \( w \leq v \) or both.
- Transitivity: if both \( v \leq w \) and \( w \leq x \), then \( v \leq x \).
- Antisymmetry: if both \( v \leq w \) and \( w \leq v \), then \( v = w \).

Examples.

<table>
<thead>
<tr>
<th>Video name</th>
<th>Views*</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Despacito&quot;[6]</td>
<td>2,993,700,000</td>
</tr>
<tr>
<td>&quot;See You Again&quot;[11]</td>
<td>2,894,000,000</td>
</tr>
<tr>
<td>&quot;Gangnam Style&quot;[17]</td>
<td>803,700,000</td>
</tr>
<tr>
<td>&quot;Baby&quot;[41]</td>
<td>245,400,000</td>
</tr>
<tr>
<td>&quot;Bad Romance&quot;[146]</td>
<td>178,400,000</td>
</tr>
<tr>
<td>&quot;Charlie Bit My Finger&quot;[136]</td>
<td>128,900,000</td>
</tr>
<tr>
<td>&quot;Evolution of Dance&quot;[131]</td>
<td>118,900,000</td>
</tr>
</tbody>
</table>

numerical order (descending)  chronological order  lexicographic order
Total order

Sorting is a well-defined problem if and only if there is a total order.

A total order is a binary relation $\leq$ that satisfies:

- **Totality:** either $v \leq w$ or $w \leq v$ or both.
- **Transitivity:** if both $v \leq w$ and $w \leq x$, then $v \leq x$.
- **Antisymmetry:** if both $v \leq w$ and $w \leq v$, then $v = w$.

Non-examples.

- **Course prerequisites:** (violates totality)
- **Ro–sham–bo order:** (violates transitivity)
- **Predator–prey:** (violates antisymmetry)
Goal. Single function that sorts any type of data (that has a total order).

Ex 1. Sort strings in alphabetical order.

```java
public class StringSorter {
    public static void main(String[] args) {
        String[] a = StdIn.readAllStrings();
        Insertion.sort(a);
        for (int i = 0; i < a.length; i++)
            StdOut.println(a[i]);
    }
}

% more words3.txt
bed bug dad yet zoo ... all bad yes

% java StringSorter < words3.txt
all bad bed bug dad ... yes yet zoo
[suppressing newlines]
```
Sample sort clients

**Goal.** Single function that sorts *any* type of data (that has a total order).

**Ex 2.** Sort random real numbers in ascending order.

seems artificial (stay tuned for an application)

```java
public class Experiment {
    public static void main(String[] args) {
        int n = Integer.parseInt(args[0]);
        Double[] a = new Double[n];
        for (int i = 0; i < n; i++)
            a[i] = StdRandom.uniform();
        Insertion.sort(a);
        for (int i = 0; i < n; i++)
            StdOut.println(a[i]);
    }
}
```

% java Experiment 10
0.08614716385210452
0.09054270895414829
0.10708746304898642
0.21166190071646818
0.363292849257276
0.460954145685913
0.5340026311350087
0.7216129793703496
0.9003500354411443
0.9293994908845686
Sample sort clients

**Goal.** Single function that sorts any type of data (that has a total order).

**Ex 3.** Sort the files in a given directory by filename.

```java
import java.io.File;

public class FileSorter
{
    public static void main(String[] args)
    {
        File directory = new File(args[0]);
        File[] files = directory.listFiles();
        Insertion.sort(files);
        for (int i = 0; i < files.length; i++)
            StdOut.println(files[i].getName());
    }
}
```
How can a single function sort any type of data?

Goal. Single function that sorts any type of data (that has a total order).

Solution. **Callback** = reference to executable code.

Please sort these Japanese names for me:
あゆみ, アユミ, Ayumi, 歩美, ....

But I don’t speak Japanese and I don’t know how words are ordered.

No problem. Whenever you need to compare two words, give me a call back.

オーケー. Just make sure to use a total order.
Callbacks

**Goal.** Single function that sorts *any* type of data (that has a total order).

**Solution.** Callback = reference to executable code.
- Client passes array of objects to `sort()` function.
- The `sort()` function calls object’s `compareTo()` function as needed.

**Implementing callbacks.**
- *Java:* interfaces.
- *C:* function pointers.
- *C++:* class-type functors.
- *C#:* delegates.
- *Python, Perl, ML, Javascript:* first-class functions.
**Java interfaces**

**Interface.** A type that defines a set of methods that a class can provide.

```java
public interface Comparable<Item>
{
    public int compareTo(Item that);
}
```

**Class that implements interface.** Must implement all interface methods.

```java
public class String implements Comparable<String>
{
    ...

    public int compareTo(String that)
    {
        ...
    }
}
```

**Impact.**
- You can treat any `String` object as an object of type `Comparable`.
- On a `Comparable` object, you can invoke (only) the `compareTo()` method.
- Enables **callbacks**.
Callbacks in Java: roadmap

client (StringSorter.java)

```java
public class StringSorter {
    public static void main(String[] args) {
        String[] a = StdIn.readAllStrings();
        Insertion.sort(a);
        for (int i = 0; i < a.length; i++)
            StdOut.println(a[i]);
    }
}
```

java.lang.Comparable interface

```java
public interface Comparable<Item> {
    public int compareTo(Item that);
}
```

data type implementation (String.java)

```java
public class String implements Comparable<String> {
    ... 
    public int compareTo(String that) {
        ... 
    }
}
```

sort implementation (Insertion.java)

```java
public static void sort(Comparable[] a) {
    int n = a.length;
    for (int i = 0; i < n; i++)
        for (int j = i; j > 0; j--)
            if (a[j].compareTo(a[j-1]) < 0)
                exch(a, j, j-1);
            else break;
}
```

key point: client code does not depend upon type of data to be sorted
Suppose that the Java architects left out `implements Comparable<String>` in the class declaration for `String`. Which would be the effect?

A. `String.java` won’t compile.

B. `StringSorter.java` won’t compile.

C. `Insertion.java` won’t compile.

D. `Insertion.java` will throw an exception.
Comparable API

Implement `compareTo()` so that `v.compareTo(w)`

- Returns a
  - negative integer if `v` is less than `w`
  - positive integer if `v` is greater than `w`
  - zero if `v` is equal to `w`

- Defines a total order.
- Throws an exception if incompatible types (or either is `null`).

```
\{ 
  v.compareTo(w) <= 0 means v is less than or equal to w
\}
```

Built-in comparable types. Integer, Double, String, Date, File, ...

User-defined comparable types. Implement the Comparable interface.
Implementing the Comparable interface

**Date data type.** Simplified version of java.util.Date.

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

    public Date(int m, int d, int y)
    {
        month = m;
        day = d;
        year = y;
    }

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

https://algs4.cs.princeton.edu/12oop/Date.java.html
2.1 **Elementary Sorts**

- rules of the game
- *selection sort*
- *insertion sort*
- shuffling
- comparators
- stability
Selection sort demo

- In iteration $i$, find index $\min$ of smallest remaining entry.
- Swap $a[i]$ and $a[\min]$.  

![Initial cards](initial)
Selection sort

Algorithm. ↑ scans from left to right.

Insets. Entries the left of ↑ (including ↑) fixed and in ascending order.
- No entry to right of ↑ is smaller than any entry to the left of ↑.
Selection sort inner loop

To maintain algorithm invariants:

- Move the pointer to the right.
  
  ```
  i++;  
  ```

- Identify index of minimum entry on right.
  
  ```
  int min = i;
  for (int j = i+1; j < n; j++)
      if (less(a[j], a[min]))
          min = j;
  ```

- Exchange into position.
  
  ```
  exch(a, i, min);
  ```
Two useful sorting abstractions

**Helper functions.** Refer to data only through *compares* and *exchanges*.

**Less.** Is item $v$ less than $w$?

```java
private static boolean less(Comparable v, Comparable w) {
    return v.compareTo(w) < 0;
}
```

**Exchange.** Swap item in array $a[]$ at index $i$ with the one at index $j$.

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

*polymorphism: you can treat any object as an object of type Object*
Selection sort: Java implementation

```java
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);
        }
    }

    private static boolean less(Comparable v, Comparable w) {
        /* see previous slide */
    }

    private static void exch(Object[] a, int i, int j) {
        /* see previous slide */
    }
}
```

https://algs4.cs.princeton.edu/21elementary/Selection.java.html
Generic methods

**Oops.** The compiler complains.

```bash
% javac-algs4 Selection.java
Selection.java:83: warning: [unchecked] unchecked call to compareTo(T) as a member of the raw type java.lang.Comparable
    return (v.compareTo(w) < 0);
    ^
1 warning
```

**Q.** How to appease the compiler?
Generic methods

Pedantic (type-safe) version. Compiles without any warnings.

generic type variable (for a static method)
(type inferred from argument; must be Comparable)

```java
public class SelectionPedantic {
    public static <Key extends Comparable<Key>> void sort(Key[] a) {
        /* as before */
    }

    private static <Key extends Comparable<Key>> boolean less(Key v, Key w) {
        /* as before */
    }

    private static void exch(Object[] a, int i, int j) {
        /* as before */
    }
}
```


Remark. Use type-safe version in system code (but not in lecture).
Selection sort: animations

20 random items

http://www.sorting-algorithms.com/selection-sort
How many compares does selection sort make to sort an array of $n$ distinct items in reverse order?

A. $\sim n$

B. $\sim \frac{1}{4} n^2$

C. $\sim \frac{1}{2} n^2$

D. $\sim n^2$
Selection sort: mathematical analysis

Proposition. Selection sort makes \((n - 1) + (n - 2) + \ldots + 1 + 0 \sim n^2 / 2\) compares and \(n\) exchanges to sort any array of \(n\) items.

<table>
<thead>
<tr>
<th>(i)</th>
<th>(i_{\text{min}})</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>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>a[]</td>
<td>S</td>
<td>O</td>
<td>R</td>
<td>T</td>
<td>E</td>
<td>X</td>
<td>A</td>
<td>M</td>
<td>P</td>
<td>L</td>
<td>E</td>
</tr>
<tr>
<td>0</td>
<td>6</td>
<td>S</td>
<td>O</td>
<td>R</td>
<td>T</td>
<td>E</td>
<td>X</td>
<td>A</td>
<td>M</td>
<td>P</td>
<td>L</td>
<td>E</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>A</td>
<td>O</td>
<td>R</td>
<td>T</td>
<td>E</td>
<td>X</td>
<td>S</td>
<td>M</td>
<td>P</td>
<td>L</td>
<td>E</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>A</td>
<td>E</td>
<td>R</td>
<td>T</td>
<td>O</td>
<td>X</td>
<td>S</td>
<td>M</td>
<td>P</td>
<td>L</td>
<td>E</td>
</tr>
<tr>
<td>3</td>
<td>9</td>
<td>A</td>
<td>E</td>
<td>E</td>
<td>T</td>
<td>O</td>
<td>X</td>
<td>S</td>
<td>M</td>
<td>P</td>
<td>L</td>
<td>E</td>
</tr>
<tr>
<td>4</td>
<td>7</td>
<td>A</td>
<td>E</td>
<td>E</td>
<td>L</td>
<td>O</td>
<td>X</td>
<td>S</td>
<td>M</td>
<td>P</td>
<td>T</td>
<td>R</td>
</tr>
<tr>
<td>5</td>
<td>7</td>
<td>A</td>
<td>E</td>
<td>E</td>
<td>L</td>
<td>M</td>
<td>X</td>
<td>S</td>
<td>O</td>
<td>P</td>
<td>T</td>
<td>R</td>
</tr>
<tr>
<td>6</td>
<td>8</td>
<td>A</td>
<td>E</td>
<td>E</td>
<td>L</td>
<td>M</td>
<td>O</td>
<td>S</td>
<td>X</td>
<td>P</td>
<td>T</td>
<td>R</td>
</tr>
<tr>
<td>7</td>
<td>10</td>
<td>A</td>
<td>E</td>
<td>E</td>
<td>L</td>
<td>M</td>
<td>O</td>
<td>P</td>
<td>X</td>
<td>S</td>
<td>T</td>
<td>R</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>A</td>
<td>E</td>
<td>E</td>
<td>L</td>
<td>M</td>
<td>O</td>
<td>P</td>
<td>R</td>
<td>S</td>
<td>T</td>
<td>X</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>A</td>
<td>E</td>
<td>E</td>
<td>L</td>
<td>M</td>
<td>O</td>
<td>P</td>
<td>R</td>
<td>S</td>
<td>T</td>
<td>X</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>A</td>
<td>E</td>
<td>E</td>
<td>L</td>
<td>M</td>
<td>O</td>
<td>P</td>
<td>R</td>
<td>S</td>
<td>T</td>
<td>X</td>
</tr>
</tbody>
</table>

Running time insensitive to input. Quadratic time, even if input is sorted. Data movement is minimal. Linear number of exchanges—exactly \(n\).
2.1 Elementary Sorts

- rules of the game
- selection sort
- insertion sort
- shuffling
- comparators
- stability
Insertion sort demo

- In iteration $i$, swap $a[i]$ with each larger entry to its left.

https://www.youtube.com/watch?v=ROalU379l3U
Insertion sort

**Algorithm.** ↑ scans from left to right.

**Invariants.**

- Entries to the left of ↑ (including ↑) are in ascending order.
- Entries to the right of ↑ have not yet been seen.
Insertion sort: inner loop

To maintain algorithm invariants:

- Move the pointer to the right.
  
  ```java
  i++;
  ```

- Moving from right to left, exchange `a[i]` with each larger entry to its left.
  
  ```java
  for (int j = i; j > 0; j--)
      if (less(a[j], a[j-1]))
          exch(a, j, j-1);
  else break;
  ```
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; j--)
                if (less(a[j], a[j-1]))
                    exch(a, j, j-1);
            else break;
    }

    private static boolean less(Comparable v, Comparable w)
    { /* as before */ }

    private static void exch(Object[] a, int i, int j)
    { /* as before */ }
}
Elementary sorts: quiz 3

How many compares does insertion sort make to sort an array of \( n \) distinct keys in reverse order?

A. \( \sim n \)
B. \( \sim \frac{1}{4} \ n^2 \)
C. \( \sim \frac{1}{2} \ n^2 \)
D. \( \sim n^2 \)
Insertion sort: analysis

Worst case. Insertion sort makes \( \sim \frac{1}{2} n^2 \) compares and \( \sim \frac{1}{2} n^2 \) exchanges to sort an array of \( n \) distinct keys in reverse order.

Pf. Exactly \( i \) compares and exchanges in iteration \( i \).

http://www.sorting-algorithms.com/insertion-sort
Which is faster in practice to sort an array of n items, selection sort or insertion sort?

A. Selection sort.
B. Insertion sort.
C. No significant difference.
D. It depends.
**Insertion sort: analysis**

**Best case.** Insertion sort makes \( n - 1 \) compares and 0 exchanges to sort an array of \( n \) distinct keys in ascending order.

http://www.sorting-algorithms.com/insertion-sort
**Insertion sort: analysis**

**Good case.** Insertion sort is linear time for “partially sorted” arrays.

**Q.** What do we mean by partially sorted?

![Diagram showing the algorithm position in order and not yet seen.](http://www.sorting-algorithms.com/insertion-sort)
Insertion sort: partially sorted arrays

**Def.** An inversion is a pair of keys that are out of order.

\[
\begin{array}{ccccccc}
A & E & E & L & M & O & T & R & X & P & S \\
\hline
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11
\end{array}
\]

**Def.** A family of arrays is **partially sorted** if the number of inversions is \( \leq c n \).

- Ex 1. A sorted array.
- Ex 2. A subarray of length 10 appended to a sorted subarray of length \( n \).

\[ \leq 10n + 45 \text{ inversions} \]

**Proposition.** Insertion sort runs in linear time on partially sorted arrays.

**Pf.**

- Number of exchanges in insertion sort = number of inversions.
- Number of compares \( \leq \) number of exchanges + \((n - 1)\).

exchange decreases number of inversions by 1

\( n \) compares in iteration \( i \) triggers one exchange
(\( \text{except possibly last one in iteration} \)
Insertion sort: practical improvements

Half exchanges. Shift items over (instead of exchanging).
- Eliminates unnecessary data movement.
- No longer uses only `less()` and `exch()` to access data.

```
A C H H I M N N P Q X Y K B I N A R Y
```

Binary insertion sort. Use binary search to find insertion point.
- Number of compares $\sim n \log_2 n$.
- But still a quadratic number of array accesses.

```
A C H H I M N N P Q X Y K B I N A R Y
```

binary search for first key $> K$
2.1 Elementary Sorts

- rules of the game
- selection sort
- insertion sort
- shuffling
- comparators
- stability
Goal. Rearrange array so that result is a uniformly random permutation.

all $n!$ permutations equally likely
Shuffling by sorting

- Generate a random real number for each array entry.
- Sort the array.
Shuffling by sorting

- Generate a random real number for each array entry.
- Sort the array.

Proposition. Shuffle sort produces a uniformly random permutation.

Application. Shuffle columns in a spreadsheet.

assuming real numbers are uniformly random (and no ties)
**Interview Question:** Shuffle an Array

**Goal.** Rearrange array so that result is a uniformly random permutation.

Shuffling by sorting.
- Quadratic time (with insertion sort or selection sort).
- Linearithmic time (with mergesort).

**Challenge.** Design a linear-time algorithm (without sorting).
Microsoft antitrust probe by EU. Microsoft agreed to provide a randomized ballot screen for users to select browser.

Select your web browser(s)

- **Google Chrome**: A fast new browser from Google. Try it now!
- **Safari**: Safari for Windows from Apple, the world’s most innovative browser.
- **Mozilla Firefox**: Your online security is Firefox’s top priority. Firefox is free, and made to help you get the most out of the web.
- **Opera**: The fastest browser on Earth. Secure, powerful and easy to use, with excellent privacy protection.
- **Internet Explorer**: Designed to help you take control of your privacy and browse with confidence. Free from Microsoft.

appeared last 50% of the time
Texas hold’em poker. Software must shuffle electronic cards.

How We Learned to Cheat at Online Poker: A Study in Software Security

Ivy league school room draw. Students assigned random room draw times.

U. claims error in room draw process, provides compensation to affected students

By Rebecca Han | Apr 26, 2019

Photo Credit: Jon Ort / The Daily Princetonian
2.1 Elementary Sorts

- rules of the game
- selection sort
- insertion sort
- shuffling
- comparators
- stability
Q. When might we need to define different sort orderings?
Sort music library by artist

<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 - Soundtrack</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 (Extended Mix)</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>Bonnie 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 In The U.S.A.</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>Born To Run</td>
<td>Bruce Springsteen</td>
<td>9:34</td>
<td>Born To Run</td>
</tr>
<tr>
<td>Downtown Train To Columbus</td>
<td>Bruce Springsteen</td>
<td>2:57</td>
<td>Streets Of Asbury-The Soundtrack (Disc 3)</td>
</tr>
</tbody>
</table>
Sort music library by song name

<table>
<thead>
<tr>
<th>Name</th>
<th>Artist</th>
<th>Time</th>
<th>Album</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cross Road</td>
<td>Bon Jovi</td>
<td>6:33</td>
<td>Cross Road</td>
</tr>
<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>Ama, Ama, Ama Y Ensanda 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>Whieler</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’Riley</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>Blood American</td>
<td>Jimmy Eat World</td>
<td>3:04</td>
<td>Blood 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>Brit</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 Rens Mix)</td>
<td>Evanescence Vs. Pat Benatar</td>
<td>9:48</td>
<td></td>
</tr>
<tr>
<td>Californication</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 – Part One</td>
</tr>
<tr>
<td>Chicago Chanteuse</td>
<td>Sultans-Era</td>
<td>5:11</td>
<td>Bombay Dreams</td>
</tr>
</tbody>
</table>
Comparable interface: review

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

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

    public Date(int m, int d, int y)
    {
        month = m;
        day = d;
        year = y;
    }
    ...

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

https://algs4.cs.princeton.edu/12oop/Date.java.html
Comparator interface

**Comparator interface:** sort using an alternate order.

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

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

<table>
<thead>
<tr>
<th>string order</th>
<th>example</th>
</tr>
</thead>
<tbody>
<tr>
<td>natural order</td>
<td>Now is the time</td>
</tr>
<tr>
<td>case insensitive</td>
<td>is Now the time</td>
</tr>
<tr>
<td>Spanish language</td>
<td>café cafetero cuarto</td>
</tr>
<tr>
<td>British phone book</td>
<td>McKinley Mackintosh</td>
</tr>
</tbody>
</table>

- Pre-1994 order for digraphs ch and ll and rr
- churro nube ñoño
Comparator interface: system sort

To use with Java system sort:

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

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

**Bottom line.** Decouples the definition of the data type from the definition of what it means to compare two objects of that type.
Comparator interface: implementing

To implement a comparator:

- Define a (nested) class that implements the Comparator interface.
- Implement the `compare()` method.
- Provide client access to Comparator.

```java
import java.util.Comparator;

public class Student
{
    private final String name;
    private final int section;
    ...

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

    public static Comparator<Student> byNameOrder()
    {
        return new NameOrder();
    }
}
```

https://algs4.cs.princeton.edu/12oop/Student.java.html
Comparator interface: implementing

To implement a comparator:

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

```java
import java.util.Comparator;

public class Student
{
    private final String name;
    private final int section;
    ...

    private static class SectionOrder implements Comparator<Student>
    {
        public int compare(Student v, Student w)
        {
            return Integer.compare(v.section, w.section);
        }
    }

    public static Comparator<Student> bySectionOrder()
    { return new SectionOrder(); }
}
```
Comparator interface: implementing

To implement a comparator:

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

**Insertion.sort(a, Student.byNameOrder());**

<table>
<thead>
<tr>
<th>Name</th>
<th>Rank</th>
<th>Grade</th>
<th>Phone</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>Andrews</td>
<td>3</td>
<td>A</td>
<td>(664) 480–0023</td>
<td>097 Little</td>
</tr>
<tr>
<td>Battle</td>
<td>4</td>
<td>C</td>
<td>(874) 088–1212</td>
<td>121 Whitman</td>
</tr>
<tr>
<td>Chen</td>
<td>3</td>
<td>A</td>
<td>(991) 878–4944</td>
<td>308 Blair</td>
</tr>
<tr>
<td>Fox</td>
<td>3</td>
<td>A</td>
<td>(884) 232–5341</td>
<td>11 Dickinson</td>
</tr>
<tr>
<td>Furia</td>
<td>1</td>
<td>A</td>
<td>(766) 093–9873</td>
<td>101 Brown</td>
</tr>
<tr>
<td>Gazsi</td>
<td>4</td>
<td>B</td>
<td>(800) 867–5309</td>
<td>101 Brown</td>
</tr>
<tr>
<td>Kanaga</td>
<td>3</td>
<td>B</td>
<td>(898) 122–9643</td>
<td>22 Brown</td>
</tr>
<tr>
<td>Rohde</td>
<td>2</td>
<td>A</td>
<td>(232) 343–5555</td>
<td>343 Forbes</td>
</tr>
</tbody>
</table>

**Insertion.sort(a, Student.bySectionOrder());**

<table>
<thead>
<tr>
<th>Name</th>
<th>Rank</th>
<th>Grade</th>
<th>Phone</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>Furia</td>
<td>1</td>
<td>A</td>
<td>(766) 093–9873</td>
<td>101 Brown</td>
</tr>
<tr>
<td>Rohde</td>
<td>2</td>
<td>A</td>
<td>(232) 343–5555</td>
<td>343 Forbes</td>
</tr>
<tr>
<td>Andrews</td>
<td>3</td>
<td>A</td>
<td>(664) 480–0023</td>
<td>097 Little</td>
</tr>
<tr>
<td>Chen</td>
<td>3</td>
<td>A</td>
<td>(991) 878–4944</td>
<td>308 Blair</td>
</tr>
<tr>
<td>Fox</td>
<td>3</td>
<td>A</td>
<td>(884) 232–5341</td>
<td>11 Dickinson</td>
</tr>
<tr>
<td>Kanaga</td>
<td>3</td>
<td>B</td>
<td>(898) 122–9643</td>
<td>22 Brown</td>
</tr>
<tr>
<td>Battle</td>
<td>4</td>
<td>C</td>
<td>(874) 088–1212</td>
<td>121 Whitman</td>
</tr>
<tr>
<td>Gazsi</td>
<td>4</td>
<td>B</td>
<td>(800) 867–5309</td>
<td>101 Brown</td>
</tr>
</tbody>
</table>
2.1 Elementary Sorts

- rules of the game
- selection sort
- insertion sort
- shuffling
- comparators
- stability

skipped in lecture (see precept)
Stability

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

```
Selection.sort(a, Student.byNameOrder());
```

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Andrews</td>
<td>3</td>
<td>A</td>
<td>(664) 480–0023</td>
<td>097 Little</td>
</tr>
<tr>
<td>Battle</td>
<td>4</td>
<td>C</td>
<td>(874) 088–1212</td>
<td>121 Whitman</td>
</tr>
<tr>
<td>Chen</td>
<td>3</td>
<td>A</td>
<td>(991) 878–4944</td>
<td>308 Blair</td>
</tr>
<tr>
<td>Fox</td>
<td>3</td>
<td>A</td>
<td>(884) 232–5341</td>
<td>11 Dickinson</td>
</tr>
<tr>
<td>Furia</td>
<td>1</td>
<td>A</td>
<td>(766) 093–9873</td>
<td>101 Brown</td>
</tr>
<tr>
<td>Gazsi</td>
<td>4</td>
<td>B</td>
<td>(800) 867–5309</td>
<td>101 Brown</td>
</tr>
<tr>
<td>Kanaga</td>
<td>3</td>
<td>B</td>
<td>(898) 122–9643</td>
<td>22 Brown</td>
</tr>
<tr>
<td>Rohde</td>
<td>2</td>
<td>A</td>
<td>(232) 343–5555</td>
<td>343 Forbes</td>
</tr>
</tbody>
</table>

```
Selection.sort(a, Student.bySectionOrder());
```

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Furia</td>
<td>1</td>
<td>A</td>
<td>(766) 093–9873</td>
<td>101 Brown</td>
</tr>
<tr>
<td>Rohde</td>
<td>2</td>
<td>A</td>
<td>(232) 343–5555</td>
<td>343 Forbes</td>
</tr>
<tr>
<td>Chen</td>
<td>3</td>
<td>A</td>
<td>(991) 878–4944</td>
<td>308 Blair</td>
</tr>
<tr>
<td>Fox</td>
<td>3</td>
<td>A</td>
<td>(884) 232–5341</td>
<td>11 Dickinson</td>
</tr>
<tr>
<td>Andrews</td>
<td>3</td>
<td>A</td>
<td>(664) 480–0023</td>
<td>097 Little</td>
</tr>
<tr>
<td>Kanaga</td>
<td>3</td>
<td>B</td>
<td>(898) 122–9643</td>
<td>22 Brown</td>
</tr>
<tr>
<td>Gazsi</td>
<td>4</td>
<td>B</td>
<td>(800) 867–5309</td>
<td>101 Brown</td>
</tr>
<tr>
<td>Battle</td>
<td>4</td>
<td>C</td>
<td>(874) 088–1212</td>
<td>121 Whitman</td>
</tr>
</tbody>
</table>

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

A **stable** sort preserves the relative order of items with equal keys.
Which sorting algorithm(s) are stable?

A. Selection sort.
B. Insertion sort.
C. Both A and B.
D. Neither A nor B.
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);
    }
}
```

<table>
<thead>
<tr>
<th>i</th>
<th>j</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>B₁</td>
<td>A₁</td>
<td>A₂</td>
<td>A₃</td>
<td>B₂</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>A₁</td>
<td>B₁</td>
<td>A₂</td>
<td>A₃</td>
<td>B₂</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>A₁</td>
<td>A₂</td>
<td>B₁</td>
<td>A₃</td>
<td>B₂</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>A₁</td>
<td>A₂</td>
<td>A₃</td>
<td>B₁</td>
<td>B₂</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>A₁</td>
<td>A₂</td>
<td>A₃</td>
<td>B₁</td>
<td>B₂</td>
</tr>
<tr>
<td></td>
<td></td>
<td>A₁</td>
<td>A₂</td>
<td>A₃</td>
<td>B₁</td>
<td>B₂</td>
</tr>
</tbody>
</table>

Pf. Equal items never move past each other.
Stability: selection sort

**Proposition.** Selection sort is **not stable**.

```java
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);
        }
    }
}
```

<table>
<thead>
<tr>
<th>i</th>
<th>min</th>
<th>0</th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2</td>
<td>B₂</td>
<td>B₁</td>
<td>A</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>A</td>
<td>B₂</td>
<td>B₁</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>A</td>
<td>B₂</td>
<td>B₁</td>
</tr>
</tbody>
</table>

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