2.1 Elementary Sorts

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

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

### Ex. Student records in a university.

<table>
<thead>
<tr>
<th>Item</th>
<th>Key</th>
<th>Name</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>
</tr>
<tr>
<td>Rohde</td>
<td>2</td>
<td>A</td>
<td>(232) 343–5555</td>
<td>343 Forbes</td>
</tr>
<tr>
<td>Gazsi</td>
<td>4</td>
<td>B</td>
<td>(800) 867–5309</td>
<td>101 Brown</td>
</tr>
<tr>
<td>Furia</td>
<td>1</td>
<td>A</td>
<td>(766) 093–9873</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>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>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Key</th>
<th>Item</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>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>
Sorting applications

Library of Congress numbers

FedEx packages

playing cards

Hogwarts houses

contacts
Sample sort clients

**Goal.** Sort *any* type of data.

**Ex 1.** Sort strings in alphabetical order.

```java
public class StringSorter {
    public static void main(String[] args) {
        String[] a = StdIn.readLineStrings();
        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. Sort any type of data.

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.** Sort any type of data.

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

```java
import java.io.File;

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());
    }
}
```

% java FileSorter .
Insertion.class
Insertion.java
InsertionX.class
InsertionX.java
Selection.class
Selection.java
Shell.class
Shell.java
ShellX.class
ShellX.java
Total order

Goal. Sort any type of data (for which sorting is well defined).

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

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

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
- chronological order
- lexicographic order
Total order

**Goal.** Sort any type of data (for which sorting is well defined).

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

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

**Non-examples.**

- **Ro–sham–bo order** (violates transitivity)
- **course prerequisites** (violates totality)
- **predator–prey** (violates antisymmetry)
Callbacks

**Goal.** Sort *any* type of data (for which sorting is well defined).

**Q.** How can a `sort()` function compare data of type `String`, `Double`, and `java.io.File` without hardwiring in type-specific information.

**Callback = reference to executable code.**
- Client passes array of objects to `sort()` function.
- The `sort()` method 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.
Callbacks: 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**.

“polymorphism”
Callbacks: 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

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

data type implementation (String.java)

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

key point: no dependence on type of data to be sorted

sort implementation (Insertion.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;
}
```
Suppose that the Java architects leave out implements Comparable<String> in the class declaration for String. What 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.
java.lang.Comparable API

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

- Defines a total order.
- Returns a \{ negative integer, zero, positive integer \} if 
  \{ v is less than, equal to, greater than \} w, respectively.
- Throws an exception if incompatible types (or either is `null`).

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;
    }
}
```

can compare Date objects only to other Date objects

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

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

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

![Initial state of selection sort with cards](initial-state.png)
Selection sort

Algorithm. ▲ scans from left to right.

Invariants.
- 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;
}
```
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 */
    }
}
```

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

**Oops.** The compiler complains.

```
% 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 silence the compiler?
Generic methods

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

```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 Object 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
Selection sort: animations

20 partially sorted items

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

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

<table>
<thead>
<tr>
<th>(i)</th>
<th>(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>(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>

Entries in black are examined to find the minimum, entries in red are \(a[\text{min}]\) and entries in gray are in final position.

**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
Insertion sort demo

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

https://www.youtube.com/watch?v=ROalU379I3U
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.

\[
i++;
\]

- Moving from right to left, exchange \(a[i]\) with each larger entry to its left.

\[
\text{for (int } j = i; j > 0; j--) \\
\text{ if (less(a[j], a[j-1]))} \\
\text{ exch(a, j, j-1);} \\
\text{ else break;}
\]
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 */  }
}
Insertion sort: animation

40 random items

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

**Proposition.** To sort a randomly ordered array with distinct keys, insertion sort uses $\sim \frac{1}{4} n^2$ compares and $\sim \frac{1}{4} n^2$ exchanges on average.

**Pf.** Expect each entry to move halfway back.

<table>
<thead>
<tr>
<th>i</th>
<th>j</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></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>0</td>
<td>O</td>
<td>S</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>2</td>
<td>1</td>
<td>O</td>
<td>R</td>
<td>S</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>3</td>
<td>3</td>
<td>O</td>
<td>R</td>
<td>S</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>4</td>
<td>0</td>
<td>E</td>
<td>O</td>
<td>R</td>
<td>S</td>
<td>T</td>
<td>X</td>
<td>A</td>
<td>M</td>
<td>P</td>
<td>L</td>
<td>E</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>E</td>
<td>O</td>
<td>R</td>
<td>S</td>
<td>T</td>
<td>X</td>
<td>A</td>
<td>M</td>
<td>P</td>
<td>L</td>
<td>E</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>A</td>
<td>E</td>
<td>O</td>
<td>R</td>
<td>S</td>
<td>T</td>
<td>X</td>
<td>A</td>
<td>M</td>
<td>P</td>
<td>L</td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td>A</td>
<td>E</td>
<td>M</td>
<td>O</td>
<td>R</td>
<td>S</td>
<td>T</td>
<td>X</td>
<td>P</td>
<td>L</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>4</td>
<td>A</td>
<td>E</td>
<td>M</td>
<td>O</td>
<td>P</td>
<td>R</td>
<td>S</td>
<td>T</td>
<td>X</td>
<td>L</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>2</td>
<td>A</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>
<td></td>
</tr>
<tr>
<td>10</td>
<td>2</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></td>
</tr>
</tbody>
</table>

Trace of insertion sort (array contents just after each insertion)
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: animation

40 reverse-sorted items

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

Worst case. If the array is in descending order (and no duplicates), insertion sort makes $\sim \frac{1}{2} n^2$ compares and $\sim \frac{1}{2} n^2$ exchanges.

X T S R P O M L F E A

Best case. If the array is in ascending order, insertion sort makes $n - 1$ compares and 0 exchanges.

A E E L M O P R S T X
Which is faster in practice, selection sort or insertion sort?

A. Selection sort.
B. Insertion sort.
C. No significant difference.
Insertion sort: animation

40 partially sorted items

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.

\[
\begin{array}{cccccccccc}
A & E & E & L & M & O & T & R & X & P & S \\
\end{array}
\]

(6 inversions)

**Def.** A family of arrays is **partially sorted** if the number of inversions is $\leq cn$.

- Ex 1. A sorted array has 0 inversions.
- Ex 2. A subarray of size 10 appended to a sorted subarray of size $n$.

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

**Pf.** Number of exchanges in insertion sort = number of inversions.

\[
\text{number of compares} \leq \text{exchanges} + (n - 1)
\]
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.

\[
\begin{array}{cccccccccccc}
\end{array}
\]

Binary insertion sort. Use binary search to find insertion point.

- Number of compares \( \sim n \lg n \).
- But still a quadratic number of array accesses.

\[
\begin{array}{cccccccccccc}
\end{array}
\]

\[\text{binary search for first key} > K\]
2.1 Elementary Sorts

- rules of the game
- selection sort
- insertion sort
- shuffling
- comparators
Interview question: shuffle an array

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

all \( n! \) permutations equally likely
Goal. Rearrange array so that result is a uniformly random permutation.

all n! permutations equally likely
Shuffle sort

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

<table>
<thead>
<tr>
<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>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
</tr>
<tr>
<td>0.8003</td>
<td>0.9706</td>
<td>0.9157</td>
<td>0.9649</td>
<td>0.1576</td>
<td>0.4854</td>
<td>0.1419</td>
<td>0.4218</td>
<td>0.9572</td>
</tr>
</tbody>
</table>
Shuffle sort

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

<table>
<thead>
<tr>
<th>8</th>
<th>6</th>
<th>9</th>
<th>7</th>
<th>2</th>
<th>4</th>
<th>10</th>
<th>5</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>♦8</td>
<td>♦6</td>
<td>♦9</td>
<td>♦7</td>
<td>♦2</td>
<td>♦4</td>
<td>♦10</td>
<td>♦5</td>
<td>♦3</td>
</tr>
</tbody>
</table>

0.1419 0.1576 0.4218 0.4854 0.8003 0.9157 0.9572 0.9649 0.9706
Shuffle sort

- 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.
Microsoft antitrust probe by EU. Microsoft agreed to provide a randomized ballot screen for users to select browser.
War story (Microsoft)

Microsoft antitrust probe by EU.  Microsoft agreed to provide a randomized ballot screen for users to select browser in Windows 7.

Solution?  Implement shuffle sort by making comparator always return a random answer.

```java
public int compareTo(Browser that) {
    double r = Math.random();
    if (r < 0.5) return -1;
    if (r > 0.5) return +1;
    return 0;
}
```

(browser comparator (fails to implement a total order))
2.1 Elementary Sorts

- rules of the game
- selection sort
- insertion sort
- shuffling
- comparators
Different orderings

Q. When might we need to define different sort orderings?
<table>
<thead>
<tr>
<th>Name</th>
<th>Artist</th>
<th>Time</th>
<th>Album</th>
</tr>
</thead>
<tbody>
<tr>
<td>12 Let It Be</td>
<td>The Beatles</td>
<td>4:03</td>
<td>Let It Be Soundtrack</td>
</tr>
<tr>
<td>13 Take My Breath Away</td>
<td>BERLIN</td>
<td>4:13</td>
<td>Top Gun Soundtrack</td>
</tr>
<tr>
<td>14 Circle Of Friends</td>
<td>Better Than Ezra</td>
<td>3:27</td>
<td>Empire Records</td>
</tr>
<tr>
<td>15 Dancing With Myself</td>
<td>Billy Idol</td>
<td>4:43</td>
<td>Don't Stop</td>
</tr>
<tr>
<td>16 Rebel Yell</td>
<td>Billy Idol</td>
<td>4:49</td>
<td>Rebel Yell</td>
</tr>
<tr>
<td>17 Piano Man</td>
<td>Billy Joel</td>
<td>5:36</td>
<td>Greatest Hits Vol. 1</td>
</tr>
<tr>
<td>20 Atomic</td>
<td>Blondie</td>
<td>3:50</td>
<td>Atomic: The Very Best Of Blondie</td>
</tr>
<tr>
<td>21 Sunday Girl</td>
<td>Blondie</td>
<td>3:15</td>
<td>Atomic: The Very Best Of Blondie</td>
</tr>
<tr>
<td>22 Call Me</td>
<td>Blondie</td>
<td>3:33</td>
<td>Atomic: The Very Best Of Blondie</td>
</tr>
<tr>
<td>23 Dreaming</td>
<td>Blondie</td>
<td>3:06</td>
<td>Atomic: The Very Best Of Blondie</td>
</tr>
<tr>
<td>24 Hurricane</td>
<td>Bob Dylan</td>
<td>8:32</td>
<td>Desire</td>
</tr>
<tr>
<td>25 The Times They Are A-Changin'</td>
<td>Bob Dylan</td>
<td>3:17</td>
<td>Greatest Hits</td>
</tr>
<tr>
<td>26 Livin' On A Prayer</td>
<td>Bon Jovi</td>
<td>4:11</td>
<td>Cross Road</td>
</tr>
<tr>
<td>27 Beds Of Roses</td>
<td>Bon Jovi</td>
<td>6:35</td>
<td>Cross Road</td>
</tr>
<tr>
<td>28 Runaway</td>
<td>Bon Jovi</td>
<td>3:53</td>
<td>Cross Road</td>
</tr>
<tr>
<td>29 Rasputin (Extended Mix)</td>
<td>Boney M</td>
<td>5:50</td>
<td>Greatest Hits</td>
</tr>
<tr>
<td>30 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>31 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>32 Straight From The Heart</td>
<td>Bonnie Tyler</td>
<td>3:41</td>
<td>Faster Than The Speed Of Night</td>
</tr>
<tr>
<td>33 Holding Out For A Hero</td>
<td>Bonny Tyler</td>
<td>5:49</td>
<td>Meat Loaf And Friends</td>
</tr>
<tr>
<td>34 Dancing In The Dark</td>
<td>Bruce Springsteen</td>
<td>4:05</td>
<td>Born In The U.S.A.</td>
</tr>
<tr>
<td>35 Thunder Road</td>
<td>Bruce Springsteen</td>
<td>4:51</td>
<td>Born To Run</td>
</tr>
<tr>
<td>36 Born To Run</td>
<td>Bruce Springsteen</td>
<td>4:30</td>
<td>Born To Run</td>
</tr>
<tr>
<td>37 Jungleland</td>
<td>Bruce Springsteen</td>
<td>9:34</td>
<td>Born To Run</td>
</tr>
<tr>
<td>38 Born To Run</td>
<td>Bruce Springsteen</td>
<td>4:17</td>
<td>Forget About The Landlady (Disc 3)</td>
</tr>
</tbody>
</table>
Sort music library by song name
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;
    }
}
```

http://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>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>natural order</td>
<td>Now is the time</td>
<td></td>
</tr>
<tr>
<td>case insensitive</td>
<td>is Now the time</td>
<td></td>
</tr>
<tr>
<td>Spanish language</td>
<td>café cafetero cuarto churro nube ñoño</td>
<td></td>
</tr>
<tr>
<td>British phone book</td>
<td>McKinley Mackintosh</td>
<td></td>
</tr>
</tbody>
</table>
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: using with our sorting libraries

To support comparators in our sort implementations:

- Pass Comparator to both sort() and less(), and use it in less().
- Use Object instead of Comparable.

```java
import java.util.Comparator;

public class Insertion {
    ...  

    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 comparator, Object v, Object w) {
        return comparator.compare(v, w) < 0;  
    }
}
```

http://algs4.cs.princeton.edu/21elementary/Insertion.java.html
Comparator interface: implementing

To implement a comparator:

- Define a (nested) class that implements the Comparator interface.
- Implement the `compareTo()` 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();
    }
}
```

http://algs4.cs.princeton.edu/12oop/Student.java.html
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 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 `compareTo()` method.
- Provide client access to Comparator.

```java
Insertion.sort(a, Student.byNameOrder());

<table>
<thead>
<tr>
<th>Name</th>
<th>ID</th>
<th>Grade</th>
<th>Phone</th>
<th>Office</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>ID</th>
<th>Grade</th>
<th>Phone</th>
<th>Office</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>
Stability

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

\[
\text{Selection.sort(a, Student.byNameOrder());}
\]

<table>
<thead>
<tr>
<th>Name</th>
<th>Sect</th>
<th>Grade</th>
<th>Phone</th>
<th>Room</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>

\[
\text{Selection.sort(a, Student.bySectionOrder());}
\]

<table>
<thead>
<tr>
<th>Name</th>
<th>Sect</th>
<th>Grade</th>
<th>Phone</th>
<th>Room</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.