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>Room</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chen</td>
<td>3 A</td>
<td>(991) 878-4944</td>
<td>308 Blair</td>
<td></td>
</tr>
<tr>
<td>Rohde</td>
<td>2 A</td>
<td>(232) 343-5555</td>
<td>343 Forbes</td>
<td></td>
</tr>
<tr>
<td>Gazsi</td>
<td>4 B</td>
<td>(800) 867-5309</td>
<td>101 Brown</td>
<td></td>
</tr>
<tr>
<td>Furia</td>
<td>1 A</td>
<td>(766) 093-9873</td>
<td>101 Brown</td>
<td></td>
</tr>
<tr>
<td>Kanaga</td>
<td>3 B</td>
<td>(898) 122-9643</td>
<td>22 Brown</td>
<td></td>
</tr>
<tr>
<td>Andrews</td>
<td>3 A</td>
<td>(664) 480-0023</td>
<td>097 Little</td>
<td></td>
</tr>
<tr>
<td>Battle</td>
<td>4 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>Name</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>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>
Sample sort client 1

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

**Ex 1.** Sort random real numbers in ascending order.

```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

seems artificial (stay tuned for an application)
Goal. Sort any type of data.

Ex 2. 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]
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.

**Ex.**

- Standard order for natural and real numbers.
- Chronological order for dates or times.
- Lexicographic order for strings.

**Not transitive.** Ro-sham-bo.

**Not total.** PU course prerequisites.
**Callbacks**

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

**Q.** How can `sort()` compare data of type `Double`, `String`, 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()` function calls object's `compareTo()` method 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.** Specifies a set of methods that a concrete class can provide.

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

**Concrete class.** Can provide the set of methods in the interface.

```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 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

```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: no dependence on type of data to be sorted
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 a run-time exception.
E. I don't know.
java.lang.Comparable API

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

- Defines a total order.
- Returns a negative integer, zero, or positive integer if `v` is less than, equal to, or 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;
    }
}
```

Implementing the `Comparable` interface 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 deck of cards](image)

*initial*
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 Selection.java
Note: Selection.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.
```

```
% javac -Xlint:unchecked 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 */
    }
}
```

The type variable `<Key>` must extend `Comparable<Key>`, and the method `sort()` is used to sort an array `a`.

**Remark.** Use type-safe version in system code (but not in lecture).

[Link to source code](http://algs4.cs.princeton.edu/21elementary/SelectionPedantic.java.html)
Selection sort: animations

20 random items

http://www.sorting-algorithms.com/selection-sort
Selection sort: animations

20 partially-sorted items

algorithm position
- in final order
- not in final order

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

A. $\sim N$
B. $\sim 1/4 \ N^2$
C. $\sim 1/2 \ N^2$
D. $\sim N^2$
E. I don't know.
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>(a[\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>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></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></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></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></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></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></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></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></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></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></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></td>
</tr>
</tbody>
</table>

Trace of selection sort (array contents just after each exchange)

**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=ROalU379l3U
**Insertion sort**

**Algorithm.**  \(\uparrow\) scans from left to right.

**Invariants.**
- Entries to the left of \(\uparrow\) (including \(\uparrow\)) are in ascending order.
- Entries to the right of \(\uparrow\) 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.

```
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 */ }
}
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.

\[
\begin{array}{cccccccccccc}
  i & j & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\
  \hline
  & & \text{S O R T E X A M P L E} & & & & & & & & & & \\
  1 & 0 & O S R T E X A M P L E & & & & & & & & & & \\
  3 & 3 & O R S T E X A M P L E & & & & & & & & & & \\
  4 & 0 & E O R S T X A M P L E & & & & & & & & & & \\
  5 & 5 & E O R S T X A M P L E & & & & & & & & & & \\
  6 & 0 & A E O R S T X A M P L E & & & & & & & & & & \\
  8 & 4 & A E M O P R S T X L E & & & & & & & & & & \\
\end{array}
\]

Trace of insertion sort (array contents just after each insertion)
Elementary sorts: quiz 2

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. $\sim N^2$

E. I don't know.
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.

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

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

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

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


(6 inversions)

**Def.** An array 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.** For partially-sorted arrays, insertion sort runs in linear time.

**Pf.** Number of exchanges equals the 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.

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

tmp =

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

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
Elementary sorts: quiz 3

Which is faster in practice, selection sort or insertion sort?

A. Selection sort.
B. Insertion sort.
C. No significant difference.
D. I don't know.
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
Interview question: shuffle an array

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>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.

![Card images with corresponding numbers]

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.
War story (Microsoft)

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

http://www.browserchoice.eu
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;
}
```
Knuth shuffle

- In iteration $i$, pick integer $r$ between 0 and $i$ uniformly at random.
- Swap $a[i]$ and $a[r]$.

**Proposition.** [Fisher-Yates 1938] Knuth shuffling algorithm produces a uniformly random permutation of the input array in linear time.
Knuth shuffle

- In iteration $i$, pick integer $r$ between 0 and $i$ uniformly at random.
- Swap $a[i]$ and $a[r]$.

**common bug**: between 0 and $N - 1$  
**correct variant**: between $i$ and $N - 1$

```java
public class Knuth {
    public static void shuffle(Object[] a) {
        int N = a.length;
        for (int i = 0; i < N; i++) {
            int r = StdRandom.uniform(i + 1);
            exch(a, i, r);
        }
    }
}
```

http://algs4.cs.princeton.edu/11model/Knuth.java.html
### Broken Knuth shuffle

**Q.** What happens if integer is chosen between 0 and $n-1$?  
**A.** Not uniformly random!

<table>
<thead>
<tr>
<th>permutation</th>
<th>Knuth shuffle</th>
<th>broken shuffle</th>
</tr>
</thead>
<tbody>
<tr>
<td>A B C</td>
<td>1 / 6</td>
<td>4 / 27</td>
</tr>
<tr>
<td>A C B</td>
<td>1 / 6</td>
<td>5 / 27</td>
</tr>
<tr>
<td>B A C</td>
<td>1 / 6</td>
<td>5 / 27</td>
</tr>
<tr>
<td>B C A</td>
<td>1 / 6</td>
<td>5 / 27</td>
</tr>
<tr>
<td>C A B</td>
<td>1 / 6</td>
<td>4 / 27</td>
</tr>
<tr>
<td>C B A</td>
<td>1 / 6</td>
<td>4 / 27</td>
</tr>
</tbody>
</table>

Probability of each permutation when shuffling $\{A, B, C\}$. 

*Note: instead of between 0 and $i$*
Texas hold'em poker. Software must shuffle electronic cards.

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

Shuffling algorithm in FAQ at www.planetpoker.com

```
for i := 1 to 52 do begin
    r := random(51) + 1;  \[ between 1 and 51 \]
    swap := card[r];
    card[r] := card[i];
    card[i] := swap;
end;
```

**Bug 1.** Random number $r$ never 52 $\Rightarrow$ 52\textsuperscript{nd} card can't end up in 52\textsuperscript{nd} place.

**Bug 2.** Shuffle not uniform (should be between 1 and $i$).

**Bug 3.** `random()` uses 32-bit seed $\Rightarrow$ $2^{32}$ possible shuffles.

**Bug 4.** Seed = milliseconds since midnight $\Rightarrow$ 86.4 million shuffles.

“The generation of random numbers is too important to be left to chance.”

— Robert R. Coveyou
War story (online poker)

Best practices for shuffling (if your business depends on it).

- Use a hardware random-number generator that has passed both the FIPS 140-2 and the NIST statistical test suites.
- Continuously monitor statistic properties: hardware random-number generators are fragile and fail silently.
- Use an unbiased shuffling algorithm.

Bottom line. Shuffling a deck of cards is hard!
2.1 Elementary Sorts

- rules of the game
- selection sort
- insertion sort
- shuffling
- comparators
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</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>Jungeland</td>
<td>Bruce Springsteen</td>
<td>9:34</td>
<td>Born To Run</td>
</tr>
<tr>
<td>Twist, Twist, Twist (To Save Me)</td>
<td>The Budos</td>
<td>3:57</td>
<td>Forrest Gump: 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>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 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’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>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>Evanescence Vs. Pa...</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 -</td>
</tr>
</tbody>
</table>
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;
    }
}
```
Comparator interface

Comparator interface: sort using an alternate order.

```java
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>
<tr>
<td></td>
<td>churro nube ñoño</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
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;
    ...

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

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

    public static Comparator<Student> bySectionOrder()
    { return new SectionOrder(); }

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

This trick works here since no danger of overflow.
Comparator interface: implementing

To implement a comparator:

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

```java
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.

```
Stability

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

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

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

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

<table>
<thead>
<tr>
<th>Student</th>
<th>Section</th>
<th>Phone</th>
<th>Room</th>
</tr>
</thead>
<tbody>
<tr>
<td>Furia</td>
<td>1</td>
<td>(766) 093-9873</td>
<td>101 Brown</td>
</tr>
<tr>
<td>Rohde</td>
<td>2</td>
<td>(232) 343-5555</td>
<td>343 Forbes</td>
</tr>
<tr>
<td>Chen</td>
<td>3</td>
<td>(991) 878-4944</td>
<td>308 Blair</td>
</tr>
<tr>
<td>Fox</td>
<td>3</td>
<td>(884) 232-5341</td>
<td>11 Dickinson</td>
</tr>
<tr>
<td>Andrews</td>
<td>3</td>
<td>(664) 480-0023</td>
<td>097 Little</td>
</tr>
<tr>
<td>Kanaga</td>
<td>3</td>
<td>(898) 122-9643</td>
<td>22 Brown</td>
</tr>
<tr>
<td>Gazsi</td>
<td>4</td>
<td>(800) 867-5309</td>
<td>101 Brown</td>
</tr>
<tr>
<td>Battle</td>
<td>4</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.
Elementary sorts: quiz 4

Which sorting algorithms are stable?

A. Selection sort.
B. Insertion sort.
C. Both A and B.
D. Neither A nor B.
E. I don't know.
Stability: insertion sort

Proposition. Insertion sort is stable.

```
public class Insertion {
    public static void sort(Comparable[] a) {
        int N = a.length;
        for (int i = 0; i < N; i++)
            for (int j = i; j > 0 && less(a[j], a[j-1]); j--)
                exch(a, j, j-1);
    }
}
```

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);
        }
    }
}
```

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