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>Key Value</th>
<th>Phone</th>
<th>Room</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chen</td>
<td>3</td>
<td>991-878-4944</td>
<td>308 Blair</td>
<td></td>
</tr>
<tr>
<td>Rohde</td>
<td>2</td>
<td>232-343-5555</td>
<td>343 Forbes</td>
<td></td>
</tr>
<tr>
<td>Gazsi</td>
<td>4</td>
<td>800-867-5309</td>
<td>101 Brown</td>
<td></td>
</tr>
</tbody>
</table>

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

Library of Congress numbers

FedEx packages

playing cards

Hogwarts houses

contacts
**Sample sort client 1**

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

**Ex 1.** 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 client 2

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]
Sample sort client 3

Goal. Sort any type of data.

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());
    }
}
```
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 (only) the compareTo() method.
- Enables **callbacks**.
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 String data type
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 only compares dates to other dates. See: 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](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;
}
```
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 */ }
}
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 fix?
Generic methods

**Pedantic (type-safe) version.** Compiles cleanly.

```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
Elementary sorts: quiz 1

How many compares does selection sort make to sort an array of $N$ keys?

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.
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>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>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>M</td>
<td>P</td>
<td>L</td>
<td>E</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>S</td>
<td>M</td>
<td>P</td>
<td>L</td>
<td>E</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>T</td>
<td>X</td>
<td>S</td>
<td>M</td>
<td>P</td>
<td>L</td>
<td>R</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<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>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<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>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<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>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<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>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<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>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<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>
<td></td>
<td></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.
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.
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.

```cpp
i++;
```

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

```cpp
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++)
        {
            int j = i;
            while (j > 0 && less(a[j], a[j-1]))
            {
                exch(a, j, j-1);
                j--;
            }
        }
    }

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

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

http://algs4.cs.princeton.edu/21elementary/Insertion.java.html
Insertion sort: animation

40 random items

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

40 reverse-sorted 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>X</td>
<td>A</td>
<td>M</td>
<td>P</td>
<td>L</td>
<td>E</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>M</td>
<td>P</td>
<td>L</td>
<td>E</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>E</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>E</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>E</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>X</td>
</tr>
</tbody>
</table>

Entries in gray do not move. Entries in black moved one position right for insertion. Entry in red is $a[j]$.

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: 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.

\[ \text{X T S R P O M L F E A} \]

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

\[ \text{A E E L M O P R S T X} \]
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.

\[ A \ E \ E \ L \ M \ O \ T \ R \ X \ P \ S \]

\[ T-R \ T-P \ T-S \ R-P \ X-P \ X-S \]

(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
```

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.

```
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.

![Card deck]({})
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.

\[
\begin{array}{cccccccc}
8 & 6 & 9 & 7 & 2 & 4 & 10 & 5 & 3 \\
0.1419 & 0.1576 & 0.4218 & 0.4854 & 0.8003 & 0.9157 & 0.9572 & 0.9649 & 0.9706
\end{array}
\]
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 in Windows 7.

http://www.browserchoice.eu

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 Internet.
- **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
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 demo

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

![Knuth shuffle cards](image-url)
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 $a[r]$.

```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 \}
War story (online poker)

Texas hold'em poker. Software must shuffle electronic cards.

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

War story (online poker)

Shuffling algorithm in FAQ at www.planetpoker.com

```plaintext
for i := 1 to 52 do begin
    r := random(51) + 1;
    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
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
Sort music library by song name
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;
    }
}
```
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 churro nube ñoño</td>
</tr>
<tr>
<td>British phone book</td>
<td>McKinley Mackintosh</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 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);
        }
    }
    ...
}
```
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.

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Grade</th>
<th>Year</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</td>
</tr>
<tr>
<td>Battle</td>
<td>4</td>
<td>C</td>
<td>(874) 088-1212</td>
<td>121</td>
</tr>
<tr>
<td>Chen</td>
<td>3</td>
<td>A</td>
<td>(991) 878-4944</td>
<td>308</td>
</tr>
<tr>
<td>Fox</td>
<td>3</td>
<td>A</td>
<td>(884) 232-5341</td>
<td>11</td>
</tr>
<tr>
<td>Furia</td>
<td>1</td>
<td>A</td>
<td>(766) 093-9873</td>
<td>101</td>
</tr>
<tr>
<td>Gazsi</td>
<td>4</td>
<td>B</td>
<td>(800) 867-5309</td>
<td>101</td>
</tr>
<tr>
<td>Kanaga</td>
<td>3</td>
<td>B</td>
<td>(898) 122-9643</td>
<td>22</td>
</tr>
<tr>
<td>Rohde</td>
<td>2</td>
<td>A</td>
<td>(232) 343-5555</td>
<td>343</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Grade</th>
<th>Year</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</td>
</tr>
<tr>
<td>Rohde</td>
<td>2</td>
<td>A</td>
<td>(232) 343-5555</td>
<td>343</td>
</tr>
<tr>
<td>Andrews</td>
<td>3</td>
<td>A</td>
<td>(664) 480-0023</td>
<td>097</td>
</tr>
<tr>
<td>Chen</td>
<td>3</td>
<td>A</td>
<td>(991) 878-4944</td>
<td>308</td>
</tr>
<tr>
<td>Fox</td>
<td>3</td>
<td>A</td>
<td>(884) 232-5341</td>
<td>11</td>
</tr>
<tr>
<td>Fox</td>
<td>3</td>
<td>A</td>
<td>(884) 232-5341</td>
<td>11</td>
</tr>
<tr>
<td>Kanaga</td>
<td>3</td>
<td>B</td>
<td>(898) 122-9643</td>
<td>22</td>
</tr>
<tr>
<td>Battle</td>
<td>4</td>
<td>C</td>
<td>(874) 088-1212</td>
<td>121</td>
</tr>
<tr>
<td>Gazsi</td>
<td>4</td>
<td>B</td>
<td>(800) 867-5309</td>
<td>101</td>
</tr>
</tbody>
</table>
Stability

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

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

<table>
<thead>
<tr>
<th>Name</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>

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

<table>
<thead>
<tr>
<th>Name</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**.

```java
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>A1</td>
<td>B1</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>B₁</td>
<td>B₂</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 one equal item past another one.