2.1 Elementary Sorts

- rules of the game
- selection sort
- insertion sort
- shellsort
- shuffling
- convex hull
rules of the game
- selection sort
- insertion sort
- shellsort
- shuffling
- convex hull
Ex. Student records in a university.

<table>
<thead>
<tr>
<th>Item</th>
<th>Key</th>
<th>Value</th>
<th>Phone</th>
<th>Room</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chen</td>
<td>A</td>
<td>991-878-4944</td>
<td>308 Blair</td>
<td></td>
</tr>
<tr>
<td>Rohde</td>
<td>A</td>
<td>232-343-5555</td>
<td>343 Forbes</td>
<td></td>
</tr>
<tr>
<td>Gazsi</td>
<td>B</td>
<td>766-093-9873</td>
<td>101 Brown</td>
<td></td>
</tr>
<tr>
<td>Furia</td>
<td>A</td>
<td>766-093-9873</td>
<td>101 Brown</td>
<td></td>
</tr>
<tr>
<td>Kanaga</td>
<td>B</td>
<td>898-122-9643</td>
<td>22 Brown</td>
<td></td>
</tr>
<tr>
<td>Andrews</td>
<td>A</td>
<td>664-480-0023</td>
<td>097 Little</td>
<td></td>
</tr>
<tr>
<td>Battle</td>
<td>C</td>
<td>874-088-1212</td>
<td>121 Whitman</td>
<td></td>
</tr>
</tbody>
</table>

Sort. Rearrange array of N items into ascending order.

<table>
<thead>
<tr>
<th>Item</th>
<th>Key</th>
<th>Value</th>
<th>Phone</th>
<th>Room</th>
</tr>
</thead>
<tbody>
<tr>
<td>Andrews</td>
<td>A</td>
<td>664-480-0023</td>
<td>097 Little</td>
<td></td>
</tr>
<tr>
<td>Battle</td>
<td>C</td>
<td>874-088-1212</td>
<td>121 Whitman</td>
<td></td>
</tr>
<tr>
<td>Chen</td>
<td>A</td>
<td>991-878-4944</td>
<td>308 Blair</td>
<td></td>
</tr>
<tr>
<td>Furia</td>
<td>A</td>
<td>766-093-9873</td>
<td>101 Brown</td>
<td></td>
</tr>
<tr>
<td>Gazsi</td>
<td>B</td>
<td>766-093-9873</td>
<td>101 Brown</td>
<td></td>
</tr>
<tr>
<td>Kanaga</td>
<td>B</td>
<td>898-122-9643</td>
<td>22 Brown</td>
<td></td>
</tr>
<tr>
<td>Rohde</td>
<td>A</td>
<td>232-343-5555</td>
<td>343 Forbes</td>
<td></td>
</tr>
</tbody>
</table>
Sample sort client

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

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

seems artificial, but 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]);
    }
}
```

```bash
% java Experiment 10
0.9054270895414829
0.9293994908845686
0.9003500354411443
0.7216129793703496
0.5340026311350087
0.460954145685913
0.363292849257276
0.21166190071646818
0.10708746304898642
0.09054270895414829
0.08614716385210452
```

Goal. Sort any type of data.

Ex 2. Sort strings from file in alphabetical order.

```java
public class StringSorter {
    public static void main(String[] args) {
        String[] a = In.readStrings(args[0]);
        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
Sample sort client

**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());
   }
}
```
Callbacks

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

**Q.** How can `sort()` know how to compare data of type `Double`, `String`, and `java.io.File` without any information about the type of an item's key?

*Callback = reference to executable code.*

- Client passes array of objects to `sort()` function.
- The `sort()` function calls back 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: roadmap

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());
    }
}

Comparable interface (built in to Java)

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

public class File
implements Comparable<File>
{
    ...
    public int compareTo(File b)
    {
        ...
        return -1;
        ...
        return +1;
        ...
        return 0;
    }
}

sort implementation

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;
}
Total order

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

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

**Ex.**

- Standard order for natural numbers and real numbers.
- Alphabetical order for strings.
- Chronological order for dates.
- ...

An intransitive relation
Comparable API

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

- Implements 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;
   }
}
```

only compare dates to other dates
Two useful sorting abstractions

**Helper functions.** Refer to data 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(Comparable[] a, int i, int j) {
    Comparable swap = a[i];
    a[i] = a[j];
    a[j] = swap;
}
```
Goal. Test if an array is sorted.

```java
private static boolean isSorted(Comparable[] a) {
    for (int i = 1; i < a.length; i++)
       if (less(a[i], a[i-1])) return false;
    return true;
}
```

Q. If the sorting algorithm passes the test, did it correctly sort the array?
A.
‣ rules of the game
‣ selection sort
‣ insertion sort
‣ shellsort
‣ shuffling
‣ convex hull
Selection sort demo
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);
  ```
Selection sort: Java implementation

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)
    { /* as before */ }

    private static void exch(Comparable[] a, int i, int j)
    { /* as before */ }
}
Selection sort: mathematical analysis

**Proposition.** Selection sort uses \((N - 1) + (N - 2) + \ldots + 1 + 0 \sim N^2/2\) compares and \(N\) exchanges.

![Trace of selection sort (array contents just after each exchange)](image)

Running time insensitive to input. Quadratic time, even if input array is sorted. Data movement is minimal. Linear number of exchanges.
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
Selection sort: Gypsy folk dance
- rules of the game
- selection sort
- **insertion sort**
- shellsort
- shuffling
- convex hull
Insertion sort demo
**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.

```c
i++;
```

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

```c
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(Comparable[] a, int i, int j)
    { /* as before */ }
}
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>i</td>
<td>j</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</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>2</td>
<td>1</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></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></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></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></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>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>

Trace of insertion sort (array contents just after each insertion)
| i   | j | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33 | 34 |
| 1   | 1 | A | S | O | M | E | W | H | A | T | L | O | N | G | E | R | I | N | S | T | E | R | S | O | R | T | E | X | M | P | L | E  |
Insertion sort: animation

40 random items

http://www.sorting-algorithms.com/insertion-sort
Insertion sort: best and worst case

**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
```

**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 E E A
```
Insertion sort: animation

40 reverse-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.

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

\[
\text{T-R T-P T-S R-P X-P X-S}
\]

(6 inversions)

Def. An array is partially sorted if the number of inversions is \( \leq cN \).

• Ex 1. A subarray of size 10 appended to a sorted subarray of size \( N \).
• Ex 2. An array of size \( N \) with only 10 entries out of place.

Proposition. For partially-sorted arrays, insertion sort runs in linear time.

Pf. Number of exchanges equals the number of inversions.

\[\text{number of compares} = \text{exchanges} + (N - 1)\]
Insertion sort: animation

40 partially-sorted items

http://www.sorting-algorithms.com/insertion-sort
Insertion sort: Romanian folk dance
rules of the game
selection sort
insertion sort
shellsort
shuffling
convex hull
How to shuffle an array

**Shuffling.** Rearrange an array so that result is a uniformly random permutation.
Shuffling. Rearrange an array so that result is a uniformly random permutation.
Knuth shuffle demo
Knuth shuffle

**Knuth shuffle.** [Fisher-Yates 1938]

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

**Proposition.** Knuth shuffling algorithm produces a uniformly random permutation of the input array in linear time.
Knuth shuffle

Knuth shuffle. [Fisher-Yates 1938]

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

Proposition. Knuth shuffling algorithm produces a uniformly random permutation of the input array in linear time.
Knuth shuffle

Knuth shuffle. [Fisher-Yates 1938]

• In iteration \( i \), pick integer \( r \) between 0 and \( i \) uniformly at random.
• Swap \( a[i] \) and \( a[r] \).

```java
public class StdRandom
{
   ...
   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);
      }
   }
}
```
Shuffle sort

Shuffle sort.
• Generate a random real number for each array entry.
• Sort the array.

Proposition. Shuffle sort produces a uniformly random permutation of the input array, provided no duplicate values.
Shuffle sort

Shuffle sort.
• Generate a random real number for each array entry.
• Sort the array.

Proposition. Shuffle sort produces a uniformly random permutation of the input array, provided no duplicate values.
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 web.

Opera: The fastest browser on Earth. Secure, powerful and easy to use, with excellent privacy protection.

Internet Explorer: Designed to help you take control of your privacy and browse with confidence. Free from Microsoft.

appeared last 50% of the time
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 (should implement a total order))
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
http://itmanagement.earthweb.com/entdev/article.php/616221
War story (online poker)

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 \( i \) and 51).
Bug 3. \texttt{random()} uses 32-bit seed \( \Rightarrow \) \( 2^{32} \) possible shuffles.
Bug 4. Seed = milliseconds since midnight \( \Rightarrow \) 86.4 million possible shuffles.

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

“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!
• rules of the game
• selection sort
• insertion sort
• shellsort
• shuffling
• convex hull
Convex hull

The **convex hull** of a set of $N$ points is the smallest perimeter fence enclosing the points.

**Equivalent definitions.**

- Smallest convex set containing all the points.
- Smallest area convex polygon enclosing the points.
- Convex polygon enclosing the points, whose vertices are points in the set.
Convex hull

The **convex hull** of a set of $N$ points is the smallest perimeter fence enclosing the points.

**Convex hull output.** Sequence of vertices in counterclockwise order.
Convex hull: mechanical algorithm

**Mechanical algorithm.** Hammer nails perpendicular to plane; stretch elastic rubber band around points.

http://www.dfanning.com/math_tips/convexhull_1.gif
Convex hull application: motion planning

Robot motion planning. Find shortest path in the plane from $s$ to $t$ that avoids a polygonal obstacle.

Fact. Shortest path is either straight line from $s$ to $t$ or it is one of two polygonal chains of convex hull.
Farthest pair problem. Given $N$ points in the plane, find a pair of points with the largest Euclidean distance between them.

Fact. Farthest pair of points are extreme points on convex hull.
Convex hull: geometric properties

**Fact.** Can traverse the convex hull by making only counterclockwise turns.

**Fact.** The vertices of convex hull appear in increasing order of polar angle with respect to point $p$ with lowest $y$-coordinate.
Graham scan demo
Convex hull: Graham scan

- Choose point \( p \) with smallest \( y \)-coordinate.
- Sort points by polar angle with \( p \).
- Consider points in order, and discard unless that would create a ccw turn.
Graham scan: implementation challenges

Q. How to find point $p$ with smallest $y$-coordinate?
A. Define a total order, comparing $y$-coordinate. [next lecture]

Q. How to sort points by polar angle with respect to $p$?
A. Define a total order for each point $p$. [next lecture]

Q. How to determine whether $p_1 \rightarrow p_2 \rightarrow p_3$ is a counterclockwise turn?
A. Computational geometry. [next two slides]

Q. How to sort efficiently?
A. Mergesort sorts in $N \log N$ time. [next lecture]

Q. How to handle degeneracies (three or more points on a line)?
A. Requires some care, but not hard. [see booksite]
Implementing ccw

**CCW.** Given three points \( a, b, \) and \( c \), is \( a \rightarrow b \rightarrow c \) a counterclockwise turn?

- \( \infty \)-slopes
- Collinear cases

Lesson. Geometric primitives are tricky to implement.
- Dealing with degenerate cases.
- Coping with floating-point precision.
Implementing ccw

CCW. Given three points \( a, b, \) and \( c, \) is \( a \rightarrow b \rightarrow c \) a counterclockwise turn?

- Determinant (or cross product) gives twice signed area of planar triangle.

\[
2 \times \text{Area}(a, b, c) = \begin{vmatrix} a_x & a_y & 1 \\ b_x & b_y & 1 \\ c_x & c_y & 1 \end{vmatrix} = (b_x - a_x)(c_y - a_y) - (b_y - a_y)(c_x - a_x) = (b - a) \times (c - a)
\]

- If signed area \( > 0, \) then \( a \rightarrow b \rightarrow c \) is counterclockwise.
- If signed area \( < 0, \) then \( a \rightarrow b \rightarrow c \) is clockwise.
- If signed area \( = 0, \) then \( a \rightarrow b \rightarrow c \) are collinear.
Immutable point data type

```java
public class Point2D {
    private final double x;
    private final double y;

    public Point(double x, double y) {
        this.x = x;
        this.y = y;
    }

    // ... more code...

    public static int ccw(Point a, Point b, Point c) {
        int area2 = (b.x-a.x)*(c.y-a.y) - (b.y-a.y)*(c.x-a.x);
        if (area2 < 0) return -1; // clockwise
        else if (area2 > 0) return +1; // counter-clockwise
        else return 0; // collinear
    }
}
```