2.1 Elementary Sorts

- rules of the game
- selection sort
- insertion sort
- shellsort
- shuffling
- convex hull
2.1 Elementary Sorts

- rules of the game
- selection sort
- insertion sort
- shellsort
- shuffling
- convex hull
Sorting problem

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

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

**Sort.** Rearrange array of $N$ items into ascending order.

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

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]);
    }
}
```

% 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 from file in alphabetical order.

```java
public class StringSorter {
    public static void main(String[] args) {
        String[] a = In.readString(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 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());
    }
}
```

```bash
% java FileSorter .
Insertion.class
Insertion.java
InsertionX.class
InsertionX.java
Selection.class
Selection.java
Shell.class
Shell.java
ShellX.class
ShellX.java
```
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

**client**

```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());
    }
}
```

**object implementation**

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

**Comparable interface (built in to Java)**

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

**sort implementation**

```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 File data type
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 and real numbers.
- Chronological order for dates or times.
- Alphabetical order for strings.
- ...

\[
\begin{align*}
\text{violates totality: } (\text{Double.NaN} \leq \text{Double.NaN}) & \text{ is false} \\
\text{Surprising but true. The } \leq \text{ operator for double is not a total order. (!)}
\end{align*}
\]
Comparable API

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

- Is 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`).

![Comparison Diagram]

- Less than (return -1)
- Equal to (return 0)
- Greater than (return +1)

**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;
}
```
Testing

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.
2.1 Elementary Sorts

- rules of the game
- selection sort
- insertion sort
- shellsort
- shuffling
- convex hull
2.1 **Elementary Sorts**

- rules of the game
- selection sort
- insertion sort
- shellsort
- shuffling
- convex hull
Selection sort demo

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

![Initial cards](image)
Selection sort

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

Invariants.
- Entries the left of \( \uparrow \) (including \( \uparrow \)) fixed and in ascending order.
- No entry to right of \( \uparrow \) is smaller than any entry to the left of \( \uparrow \).
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

```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) { /* 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.

<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>a[]</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>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>R</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>

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.
Selection sort: animations

20 random items

algorithm position

in final order

not in final order

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

20 partially-sorted items

http://www.sorting-algorithms.com/selection-sort
2.1 Elementary Sorts

- rules of the game
- selection sort
- insertion sort
- shellsort
- shuffling
- convex hull
2.1 ELEMENTARY SORTS

- rules of the game
- selection sort
- insertion sort
- shellsort
- shuffling
- convex hull
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.

\[ i++; \]

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

\[
\begin{align*}
\text{for} \ (\text{int} \ j = i; \ j > 0; \ j--) & \\
\quad \text{if} \ (\text{less}(a[j], \ a[j-1])) & \\
\quad \quad \text{exch}(a, \ j, \ j-1); & \\
\quad \text{else} \ \text{break}; & 
\end{align*}
\]
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></td>
<td></td>
<td><strong>S</strong></td>
<td><strong>O</strong></td>
<td><strong>R</strong></td>
<td><strong>T</strong></td>
<td><strong>E</strong></td>
<td><strong>X</strong></td>
<td><strong>A</strong></td>
<td><strong>M</strong></td>
<td><strong>P</strong></td>
<td><strong>L</strong></td>
<td><strong>E</strong></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>O</td>
<td>S</td>
<td>R</td>
<td>T</td>
<td>X</td>
<td>A</td>
<td>M</td>
<td>P</td>
<td>L</td>
<td>E</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1</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></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></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>A</td>
<td>M</td>
<td>P</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>A</td>
<td>M</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>A</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)*
Insertion sort: trace
Insertion sort: animation

40 random items

algorithm position

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.

```
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 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
2.1 Elementary Sorts

- rules of the game
- selection sort
- insertion sort
- shellsort
- shuffling
- convex hull
2.1 Elementary Sorts

- rules of the game
- selection sort
- insertion sort
- shellsort
- shuffling
- convex hull
Shellsort overview

**Idea.** Move entries more than one position at a time by \( h \)-sorting the array.

An \( h \)-sorted array is \( h \) interleaved sorted subsequences

\[
\begin{align*}
\text{h = 4} & \\
L & \ E \ E \ A \ M \ H \ L \ E \ P \ S \ O \ L \ T \ S \ X \ R \\
& \text{L} \quad \text{M} \quad \text{P} \quad \text{T} \\
& \text{E} \quad \text{H} \quad \text{S} \quad \text{S} \\
& \text{E} \quad \text{L} \quad \text{O} \quad \text{X} \\
& \text{A} \quad \text{E} \quad \text{L} \quad \text{R}
\end{align*}
\]

**Shellsort.** [Shell 1959] \( h \)-sort array for decreasing sequence of values of \( h \).

**input**

\[
\begin{align*}
\text{SHELLLSORTEXAMPLE} \\
\text{13-sort} & \quad \text{PHELLSORTEXAMPLE} \\
\text{4-sort} & \quad \text{LEEMHLEPSOLTSXR} \\
\text{1-sort} & \quad \text{AEEEEHLLLMOPRSSTX}
\end{align*}
\]
h-sorting

How to $h$-sort an array? Insertion sort, with stride length $h$.

3-sorting an array

<table>
<thead>
<tr>
<th>M</th>
<th>O</th>
<th>L</th>
<th>E</th>
<th>E</th>
<th>X</th>
<th>A</th>
<th>S</th>
<th>P</th>
<th>R</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>E</td>
<td>O</td>
<td>L</td>
<td>M</td>
<td>E</td>
<td>X</td>
<td>A</td>
<td>S</td>
<td>P</td>
<td>R</td>
<td>T</td>
</tr>
<tr>
<td>E</td>
<td>E</td>
<td>L</td>
<td>M</td>
<td>O</td>
<td>X</td>
<td>A</td>
<td>S</td>
<td>P</td>
<td>R</td>
<td>T</td>
</tr>
<tr>
<td>A</td>
<td>E</td>
<td>L</td>
<td>E</td>
<td>O</td>
<td>X</td>
<td>M</td>
<td>S</td>
<td>P</td>
<td>R</td>
<td>T</td>
</tr>
<tr>
<td>A</td>
<td>E</td>
<td>L</td>
<td>E</td>
<td>O</td>
<td>X</td>
<td>M</td>
<td>S</td>
<td>P</td>
<td>R</td>
<td>T</td>
</tr>
<tr>
<td>A</td>
<td>E</td>
<td>L</td>
<td>E</td>
<td>O</td>
<td>P</td>
<td>M</td>
<td>S</td>
<td>X</td>
<td>R</td>
<td>T</td>
</tr>
<tr>
<td>A</td>
<td>E</td>
<td>L</td>
<td>E</td>
<td>O</td>
<td>P</td>
<td>M</td>
<td>S</td>
<td>X</td>
<td>R</td>
<td>T</td>
</tr>
<tr>
<td>A</td>
<td>E</td>
<td>L</td>
<td>E</td>
<td>O</td>
<td>P</td>
<td>M</td>
<td>S</td>
<td>X</td>
<td>R</td>
<td>T</td>
</tr>
<tr>
<td>A</td>
<td>E</td>
<td>L</td>
<td>E</td>
<td>O</td>
<td>P</td>
<td>M</td>
<td>S</td>
<td>X</td>
<td>R</td>
<td>T</td>
</tr>
</tbody>
</table>

Why insertion sort?

- Big increments $\implies$ small subarray.
- Small increments $\implies$ nearly in order.  [stay tuned]
Shell sort example: increments 7, 3, 1

**input**

SORTEXAMPLE

**7-sort**

SORTEXAMPLE
MORTEXASPLE
MORTEXASPLE
MOLTEXASPRE
MOLEXASPR

**3-sort**

MOLEXASPR
EOLVEXSPT
EELMXASPRT
AELEXMSPRT
AELXMSPT
AELOPSXR
AELOPSMXR
AELOPMSXR

**1-sort**

AELEOPMSXR
AELEOPMSXR
AELELOPMSXR
AELELOPMSXR
AELELOPMSXR
AELELOPMSXR

**result**

AEELMOPRSXT
Shellsort: intuition

**Proposition.** A $g$-sorted array remains $g$-sorted after $h$-sorting it.

<table>
<thead>
<tr>
<th>7-sort</th>
<th>3-sort</th>
</tr>
</thead>
<tbody>
<tr>
<td>S O R T E X A M P L E</td>
<td>M O L E E X A S P R T</td>
</tr>
<tr>
<td>M O R T E X A S P L E</td>
<td>E O L M E X A S P R T</td>
</tr>
<tr>
<td>M O R T E X A S P L E</td>
<td>E E L M O X A S P R T</td>
</tr>
<tr>
<td>M O L T E X A S P R E</td>
<td>E E L M O X A S P R T</td>
</tr>
<tr>
<td>M O L E E X A S P R T</td>
<td>A E L E O X M S P R T</td>
</tr>
<tr>
<td></td>
<td>A E L E O X M S P R T</td>
</tr>
<tr>
<td></td>
<td>A E L E O P M S X R T</td>
</tr>
<tr>
<td></td>
<td>A E L E O P M S X R T</td>
</tr>
<tr>
<td></td>
<td>A E L E O P M S X R T</td>
</tr>
</tbody>
</table>

**Challenge.** Prove this fact—it's more subtle than you'd think!
Shellsort: which increment sequence to use?

Powers of two. 1, 2, 4, 8, 16, 32, ...
No.

Powers of two minus one. 1, 3, 7, 15, 31, 63, ...
Maybe.

→ $3x + 1$. 1, 4, 13, 40, 121, 364, ...
OK. Easy to compute.

Sedgewick. 1, 5, 19, 41, 109, 209, 505, 929, 2161, 3905, ...
Good. Tough to beat in empirical studies.

merging of $(9 \times 4^i) - (9 \times 2^i) + 1$
and $4^i - (3 \times 2^i) + 1$
### Shellsort: Java implementation

```java
public class Shell {
    public static void sort(Comparable[] a) {
        int N = a.length;

        int h = 1;
        while (h < N/3) h = 3*h + 1; // 1, 4, 13, 40, 121, 364, ...

        while (h >= 1) {
            // h-sort the array.
            for (int i = h; i < N; i++) {
                for (int j = i; j >= h && less(a[j], a[j-h]); j -= h)
                    exch(a, j, j-h);
            }
            h = h/3;
        }
    }

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

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

**Shellsort**

- **Insertion sort**: The inner loop of the h-sort algorithm performs an insertion sort on each subsequence, moving elements to their correct positions.
- **3x+1 increment sequence**: The initial increment values follow the sequence 1, 4, 13, 40, 121, 364, ..., where each increment is given by \(3^k + 1\) for some integer \(k\).
- **Move to next increment**: The increment is halved after each pass, allowing smaller subarrays to be sorted more efficiently.

---

43
Shellsort: visual trace

input

40-sorted

13-sorted

4-sorted

result
Shellsort: animation

50 random items

http://www.sorting-algorithms.com/shell-sort
Shellsort: animation

50 partially–sorted items

http://www.sorting-algorithms.com/shell-sort
Shellsort: analysis

**Proposition.** The worst-case number of compares used by shellsort with the 3x+1 increments is $O(N^{3/2})$.

**Property.** Number of compares used by shellsort with the 3x+1 increments is at most by a small multiple of $N$ times the # of increments used.

<table>
<thead>
<tr>
<th>N</th>
<th>compares</th>
<th>$N^{1.289}$</th>
<th>$2.5 \times N \log N$</th>
</tr>
</thead>
<tbody>
<tr>
<td>5,000</td>
<td>93</td>
<td>58</td>
<td>106</td>
</tr>
<tr>
<td>10,000</td>
<td>209</td>
<td>143</td>
<td>230</td>
</tr>
<tr>
<td>20,000</td>
<td>467</td>
<td>349</td>
<td>495</td>
</tr>
<tr>
<td>40,000</td>
<td>1022</td>
<td>855</td>
<td>1059</td>
</tr>
<tr>
<td>80,000</td>
<td>2266</td>
<td>2089</td>
<td>2257</td>
</tr>
</tbody>
</table>

measured in thousands

**Remark.** Accurate model has not yet been discovered (!)
Why are we interested in shellsort?

Example of simple idea leading to substantial performance gains.

Useful in practice.
- Fast unless array size is huge (used for small subarrays).
- Tiny, fixed footprint for code (used in some embedded systems).
- Hardware sort prototype.

Simple algorithm, nontrivial performance, interesting questions.
- Asymptotic growth rate?
- Best sequence of increments?
- Average-case performance?

Lesson. Some good algorithms are still waiting discovery.
2.1 **Elementary Sorts**

- rules of the game
- selection sort
- insertion sort
- shellsort
- shuffling
- convex hull
2.1 Elementary Sorts

- rules of the game
- selection sort
- insertion sort
- shellsort
- shuffling
- convex hull
How to shuffle an array

Goal. Rearrange array so that result is a uniformly random permutation.
How to shuffle an array

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

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

useful for shuffling columns in a spreadsheet

2️⃣ 3️⃣ 4️⃣ 5️⃣ 6️⃣ 7️⃣ 8️⃣ 9️⃣ 1️⃣️

0.8003 0.9706 0.9157 0.9649 0.1576 0.4854 0.1419 0.4218 0.9572
Shuffle sort

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

useful for shuffling columns in a spreadsheet
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
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))
Knuth shuffle demo

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

common bug: between 0 and \( N - 1 \)
correct variant: between \( i \) and \( N - 1 \)
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://www.datamation.com/entdev/article.php/616221
War story (online poker)

Shuffling algorithm in FAQ at www.planetpoker.com

```
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. \texttt{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
- shellsort
- shuffling
- convex hull
2.1 ELEMENTARY SORTS

- 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 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.idlcoyote.com/math_tips/convexhull.html
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.
Convex hull application: farthest pair

**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.
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.

Convex hull: geometric properties
Graham scan demo

- Choose point $p$ with smallest $y$-coordinate.
- Sort points by polar angle with $p$.
- Consider points in order; discard unless it create a ccw turn.
Graham scan demo

- Choose point $p$ with smallest $y$-coordinate.
- Sort points by polar angle with $p$.
- Consider points in order; discard unless it create a ccw turn.
Graham scan: implementation challenges

Q. How to find point \( p \) with smallest \( y \)-coordinate?
A. Define a total order, comparing by \( y \)-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?

![Diagram showing various configurations of points and their ccw status]

- **yes**
- **no**
- **yes** (\(\infty\)-slope)
- **no** (collinear)
- **no** (collinear)
- **no** (collinear)

**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 $2x$ 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 Point2D(double x, double y) {
        this.x = x;
        this.y = y;
    }

    ...

    public static int ccw(Point2D a, Point2D b, Point2D c) {
        double 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
    }
}
```

danger of floating-point roundoff error
2.1 Elementary Sorts

- rules of the game
- selection sort
- insertion sort
- shellsort
- shuffling
- convex hull
2.1 Elementary Sorts

- rules of the game
- selection sort
- insertion sort
- shellsort
- shuffling
- convex hull