2.1 ELEMENTARY Sorts

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

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

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

<table>
<thead>
<tr>
<th>Item</th>
<th>Key</th>
<th>Key Type</th>
<th>Phone</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chen</td>
<td>3</td>
<td>A</td>
<td>991-878-4944</td>
<td>308 Blair</td>
</tr>
<tr>
<td>Rohde</td>
<td>2</td>
<td>A</td>
<td>232-343-5555</td>
<td>343 Forbes</td>
</tr>
<tr>
<td>Gazsi</td>
<td>4</td>
<td>B</td>
<td>766-093-9873</td>
<td>101 Brown</td>
</tr>
<tr>
<td><strong>Furia</strong></td>
<td>1</td>
<td>A</td>
<td>766-093-9873</td>
<td>101 Brown</td>
</tr>
<tr>
<td>Kanaga</td>
<td>3</td>
<td>B</td>
<td>898-122-9643</td>
<td>22 Brown</td>
</tr>
<tr>
<td>Andrews</td>
<td>3</td>
<td>A</td>
<td>664-480-0023</td>
<td>097 Little</td>
</tr>
<tr>
<td><strong>Battle</strong></td>
<td>4</td>
<td>C</td>
<td>874-088-1212</td>
<td>121 Whitman</td>
</tr>
</tbody>
</table>

**Sort.** Rearrange array of \( N \) items into ascending order.

<table>
<thead>
<tr>
<th>Item</th>
<th>Key</th>
<th>Key Type</th>
<th>Phone</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>Andrews</td>
<td>3</td>
<td>A</td>
<td>664-480-0023</td>
<td>097 Little</td>
</tr>
<tr>
<td><strong>Battle</strong></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>766-093-9873</td>
<td>101 Brown</td>
</tr>
<tr>
<td>Kanaga</td>
<td>3</td>
<td>B</td>
<td>898-122-9643</td>
<td>22 Brown</td>
</tr>
<tr>
<td>Rohde</td>
<td>2</td>
<td>A</td>
<td>232-343-5555</td>
<td>343 Forbes</td>
</tr>
</tbody>
</table>
Sorting applications

Library of Congress numbers

FedEx packages

playing cards

Hogwarts houses

contacts
Sample sort 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());
    }
}
```
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.
- Alphabetical order for strings.

No transitivity. Rock-paper-scissors.
No totality. PU course prerequisites.
Callbacks

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

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 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
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]);
    }
}
```

data-type implementation

```java
public class String implements Comparable<String>
{
    public int compareTo(String 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 String data type
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;
    }
}
```

Only compare dates to other dates.
2.1 Elementary Sorts

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

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

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 ↑.
**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;
}
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: 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

algorithm position

— in final order

— not in final order

http://www.sorting-algorithms.com/selection-sort
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>(a[#] )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{min} )</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>S</td>
</tr>
<tr>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>10</td>
</tr>
<tr>
<td>4</td>
<td>9</td>
</tr>
<tr>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>8</td>
<td>10</td>
</tr>
<tr>
<td>9</td>
<td>8</td>
</tr>
<tr>
<td>10</td>
<td>9</td>
</tr>
</tbody>
</table>

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

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

**Running time insensitive to input.** Quadratic time, even if input is sorted.

**Data movement is minimal.** Linear number of exchanges.
2.1 ELEMENTARY SORTS

- rules of the game
- selection sort
- insertion sort
- shellsort
- shuffling
Insertion sort demo

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

```java
i++; // Move the pointer to the right.
```

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

```java
for (int j = i; j > 0; j--)
    if (less(a[j], a[j-1]))
        exch(a, j, j-1);
    else break;
```

---

![Diagram showing insertion sort inner loop](image-url)
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: 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: animation

40 partially-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>E</td>
<td>X</td>
<td>A</td>
<td>M</td>
<td>P</td>
<td>L</td>
<td>E</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>E</td>
<td>O</td>
<td>R</td>
<td>S</td>
<td>T</td>
<td>X</td>
<td>A</td>
<td>M</td>
<td>P</td>
<td>L</td>
<td>E</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>E</td>
<td>O</td>
<td>R</td>
<td>S</td>
<td>T</td>
<td>X</td>
<td>A</td>
<td>M</td>
<td>P</td>
<td>L</td>
<td>E</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>A</td>
<td>E</td>
<td>O</td>
<td>R</td>
<td>S</td>
<td>T</td>
<td>X</td>
<td>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>

*Trace of insertion sort (array contents just after each insertion)*
## Insertion sort: trace

| i | 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 |
Insertion sort: analysis

**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 F E A
```
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.

\[
\text{number of compares} = \text{exchanges} + (N - 1)
\]
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 $> K$
2.1 ELEMENTARY Sorts

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

Idea. Move entries more than one position at a time by \textit{h-sorting} the array.

an h–sorted array is h interleaved sorted subsequences

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

\textbf{Shellsort. [Shell 1959]} \textit{h-sort} array for decreasing sequence of values of \textit{h}.

\begin{verbatim}
input \quad SHELLSORTEXAMPLE \\
13-sort \quad PHELLSORTEXAMSL \\
4-sort \quad LEAMHLEPSOLTSXR \\
1-sort \quad AEHELLLMOPRSSTX \\
\end{verbatim}
h-sorting demo

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

How to \( h \)-sort an array? Insertion sort, with stride length \( h \).

3-sorting an array

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

Why insertion sort?

- Big increments ⇒ small subarray.
- Small increments ⇒ nearly in order. [stay tuned]
Shellsort example: increments 7, 3, 1

input

S O R T E X A M P L E

7-sort

S O R T E X A M P L E
M O R T E X A S P L E
M O R T E X A S P L E
M O L T E X A S P R E
M O L E E X A S P R T

3-sort

M O L E E X A S P R T
E O L M E X A S P R T
E E L M O X A S P R T
E E L M O X A S P R T
A E L E O X M S P R T
A E L E O X M S P R T
A E L E O P M S X R T
A E L E O P M S X R T
A E L E O P M S X R T

1-sort

A E L E O P M S X R T
A E L E O P M S X R T
A E L E O P M S X R T
A E L E O P M S X R T
A E L E O P M S X R T
A E L E O P M S X R T
A E L E O P M S X R T
A E L E O P M S X R T
A E L E O P M S X R T
A E L E O P M S X R T

result

A E E L M O P R S T X
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: 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: 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.
**Shellsort: intuition**

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

**Challenge.** Prove this fact—it's more subtle than you'd think!
**Shellsort: analysis**

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

**Property.** The expected number of compares to shellsort a randomly-ordered array using $3x+1$ increments is....

<table>
<thead>
<tr>
<th>N</th>
<th>compares</th>
<th>$2.5 \times N \ln N$</th>
<th>$0.25 \times N \ln^2 N$</th>
<th>$N^{1.3}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>5,000</td>
<td>93K</td>
<td>106K</td>
<td>91K</td>
<td>64K</td>
</tr>
<tr>
<td>10,000</td>
<td>209K</td>
<td>230K</td>
<td>213K</td>
<td>158K</td>
</tr>
<tr>
<td>20,000</td>
<td>467K</td>
<td>495K</td>
<td>490K</td>
<td>390K</td>
</tr>
<tr>
<td>40,000</td>
<td>1022K</td>
<td>1059K</td>
<td>1122K</td>
<td>960K</td>
</tr>
<tr>
<td>80,000</td>
<td>2266K</td>
<td>2258K</td>
<td>2549K</td>
<td>2366K</td>
</tr>
</tbody>
</table>

**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? open problem: find a better increment sequence
- Average-case performance?

Lesson. Some good algorithms are still waiting discovery.
Elementary sorts summary

**Today.** Elementary sorting algorithms.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Best</th>
<th>Average</th>
<th>Worst</th>
</tr>
</thead>
<tbody>
<tr>
<td>selection sort</td>
<td>$N^2$</td>
<td>$N^2$</td>
<td>$N^2$</td>
</tr>
<tr>
<td>insertion sort</td>
<td>$N$</td>
<td>$N^2$</td>
<td>$N^2$</td>
</tr>
<tr>
<td>Shellsort (3x+1)</td>
<td>$N \log N$</td>
<td>?</td>
<td>$N^{3/2}$</td>
</tr>
<tr>
<td>goal</td>
<td>$N$</td>
<td>$N \log N$</td>
<td>$N \log N$</td>
</tr>
</tbody>
</table>

Order of growth of running time to sort an array of $N$ items

**Next week.** $N \log N$ sorting algorithms (in worst case).
2.1 ELEMENTARY SORTS

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

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

all permutations equally likely
How to shuffle an array

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

all permutations equally likely
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.

useful for shuffling columns in a spreadsheet

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

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

Proposition. Shuffle sort produces a uniformly random permutation.
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
War story (Microsoft)

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

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

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

browser comparator (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);
        }
    }
}
```
**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 result 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
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\(^{nd} \) card can't end up in 52\(^{nd} \) place.

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

**Bug 3.** \( \text{random}() \) 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!