# ID1020:Elementary Sorting

Dr. Per Brand
pbrand@kth.se

kap 2.1


Slides adapted from Algorithms 4th Edition, Sedgewick.

# Contents

- A brief diversion
- Introduction
- Selection sort
- Insertion sort
- Shell sort
- Shuffle
- Some case studies

# What is this?

Diagram for the computation by the Engine of the Numbers of Bernoulli. See Note G. (page 722 *et seq.*)

| Number of Operation. | Nature of Operation. | Variables acted upon. | Variables receiving results. | Indication of change in the value on any Variable. | Statement of Results. | Data. | | | Working Variables. | | | | | | | | | | Results. |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | $^1V_1$ ○ 0 0 1 [1] | $^1V_2$ ○ 0 0 2 [2] | $^1V_3$ ○ 0 0 4 [$n$] | $^0V_4$ ○ 0 0 0 [ ] | $^0V_5$ ○ 0 0 0 [ ] | $^0V_6$ ○ 0 0 0 [ ] | $^0V_7$ ○ 0 0 0 [ ] | $^0V_8$ ○ 0 0 0 [ ] | $^0V_9$ ○ 0 0 0 [ ] | $^0V_{10}$ ○ 0 0 0 [ ] | $^0V_{11}$ ○ 0 0 0 [ ] | $^0V_{12}$ ○ 0 0 0 [ ] | $^0V_{13}$ ○ 0 0 0 [ ] ..... | $^1V_{21}$ $B_1$ in a decimal fraction. ○ [$B_1$] |
| 1 | × | $^1V_2 \times {}^1V_3$ | $^1V_4, {}^1V_5, {}^1V_6$ | $\left\{\begin{matrix} {}^1V_2 = {}^1V_2 \\ {}^1V_3 = {}^1V_3 \end{matrix}\right\}$ | $= 2n$ ..... | ... | 2 | $n$ | $2n$ | $2n$ | $2n$ | | | | | | | | | |
| 2 | − | $^1V_4 - {}^1V_1$ | $^2V_4$ ..... | $\left\{\begin{matrix} {}^1V_4 = {}^2V_4 \\ {}^1V_1 = {}^1V_1 \end{matrix}\right\}$ | $= 2n-1$ ..... | 1 | ... | ... | $2n-1$ | | | | | | | | | | | |
| 3 | + | $^1V_5 + {}^1V_1$ | $^2V_5$ ..... | $\left\{\begin{matrix} {}^1V_5 = {}^2V_5 \\ {}^1V_1 = {}^1V_1 \end{matrix}\right\}$ | $= 2n+1$ ..... | 1 | ... | ... | ... | $2n+1$ | | | | | | | | | | |
| 4 | ÷ | $^2V_5 \div {}^2V_4$ | $^1V_{11}$ ..... | $\left\{\begin{matrix} {}^2V_5 = {}^0V_5 \\ {}^2V_4 = {}^0V_4 \end{matrix}\right\}$ | $= \frac{2n-1}{2n+1}$ ..... | ... | ... | ... | 0 | 0 | ... | ... | ... | ... | ... | $\frac{2n-1}{2n+1}$ | | | | |
| 5 | ÷ | $^1V_{11} \div {}^1V_2$ | $^2V_{11}$ ..... | $\left\{\begin{matrix} {}^1V_{11} = {}^2V_{11} \\ {}^1V_2 = {}^1V_2 \end{matrix}\right\}$ | $= \frac{1}{2} \cdot \frac{2n-1}{2n+1}$ ..... | ... | 2 | ... | ... | ... | ... | ... | ... | ... | ... | $\frac{1}{2} \cdot \frac{2n-1}{2n+1}$ | | | | |
| 6 | − | $^0V_{13} - {}^2V_{11}$ | $^1V_{13}$ ..... | $\left\{\begin{matrix} {}^2V_{11} = {}^0V_{11} \\ {}^0V_{13} = {}^1V_{13} \end{matrix}\right\}$ | $= -\frac{1}{2} \cdot \frac{2n-1}{2n+1} = A_0$ ..... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | 0 | ..... | $-\frac{1}{2} \cdot \frac{2n-1}{2n+1} = A_0$ | | |
| 7 | − | $^1V_3 - {}^1V_1$ | $^1V_{10}$ ..... | $\left\{\begin{matrix} {}^1V_3 = {}^1V_3 \\ {}^1V_1 = {}^1V_1 \end{matrix}\right\}$ | $= n-1\,(=3)$ ..... | 1 | ... | $n$ | ... | ... | ... | ... | ... | ... | $n-1$ | | | | | |
| 8 | + | $^1V_2 + {}^0V_7$ | $^1V_7$ ..... | $\left\{\begin{matrix} {}^1V_2 = {}^1V_2 \\ {}^0V_7 = {}^1V_7 \end{matrix}\right\}$ | $= 2+0=2$ ..... | ... | 2 | ... | ... | ... | ... | 2 | | | | | | | | |
| 9 | ÷ | $^1V_6 \div {}^1V_7$ | $^3V_{11}$ ..... | $\left\{\begin{matrix} {}^1V_6 = {}^1V_6 \\ {}^0V_{11} = {}^3V_{11} \end{matrix}\right\}$ | $= \frac{2n}{2} = A_1$ ..... | ... | ... | ... | ... | ... | $2n$ | 2 | ... | ... | ... | $\frac{2n}{2} = A_1$ | | | | |
| 10 | × | $^1V_{21} \times {}^3V_{11}$ | $^1V_{12}$ ..... | $\left\{\begin{matrix} {}^1V_{21} = {}^1V_{21} \\ {}^3V_{11} = {}^3V_{11} \end{matrix}\right\}$ | $= B_1 \cdot \frac{2n}{2} = B_1 A_1$ ..... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | $\frac{2n}{2} = A_1$ | $B_1 \cdot \frac{2n}{2} = B_1 A_1$ | ..... | | $B_1$ |
| 11 | + | $^1V_{12} + {}^1V_{13}$ | $^2V_{13}$ ..... | $\left\{\begin{matrix} {}^1V_{12} = {}^0V_{12} \\ {}^1V_{13} = {}^2V_{13} \end{matrix}\right\}$ | $= -\frac{1}{2} \cdot \frac{2n-1}{2n+1} + B_1 \cdot \frac{2n}{2}$ ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ..... | 0 | $\left\{-\frac{1}{2} \cdot \frac{2n-1}{2n+1} + B_1 \cdot \frac{2n}{2}\right\}$ | | |
| 12 | − | $^1V_{10} - {}^1V_1$ | $^2V_{10}$ ..... | $\left\{\begin{matrix} {}^1V_{10} = {}^2V_{10} \\ {}^1V_1 = {}^1V_1 \end{matrix}\right\}$ | $= n-2\,(=2)$ ..... | 1 | ... | ... | ... | ... | ... | ... | ... | ... | $n-2$ | | | | | |
| 13 | − | $^1V_6 - {}^1V_1$ | $^2V_6$ ..... | $\left\{\begin{matrix} {}^1V_6 = {}^2V_6 \\ {}^1V_1 = {}^1V_1 \end{matrix}\right\}$ | $= 2n-1$ ..... | 1 | ... | ... | ... | ... | $2n-1$ | | | | | | | | | |
| 14 | + | $^1V_1 + {}^1V_7$ | $^2V_7$ ..... | $\left\{\begin{matrix} {}^1V_1 = {}^1V_1 \\ {}^1V_7 = {}^2V_7 \end{matrix}\right\}$ | $= 2+1=3$ ..... | 1 | ... | ... | ... | ... | ... | 3 | | | | | | | | |
| 15 | ÷ | $^2V_6 \div {}^2V_7$ | $^1V_8$ ..... | $\left\{\begin{matrix} {}^2V_6 = {}^2V_6 \\ {}^2V_7 = {}^2V_7 \end{matrix}\right\}$ | $= \frac{2n-1}{3}$ ..... | ... | ... | ... | ... | ... | $2n-1$ | 3 | $\frac{2n-1}{3}$ | | | | | | | |
| 16 | × | $^1V_8 \times {}^3V_{11}$ | $^4V_{11}$ ..... | $\left\{\begin{matrix} {}^1V_8 = {}^0V_8 \\ {}^3V_{11} = {}^4V_{11} \end{matrix}\right\}$ | $= \frac{2n}{2} \cdot \frac{2n-1}{3}$ ..... | ... | ... | ... | ... | ... | ... | ... | 0 | ... | ... | $\frac{2n}{2} \cdot \frac{2n-1}{3}$ | | | | |
| 17 | − | $^2V_6 - {}^1V_1$ | $^3V_6$ ..... | $\left\{\begin{matrix} {}^2V_6 = {}^3V_6 \\ {}^1V_1 = {}^1V_1 \end{matrix}\right\}$ | $= 2n-2$ ..... | 1 | ... | ... | ... | ... | $2n-2$ | | | | | | | | | |
| 18 | + | $^1V_1 + {}^2V_7$ | $^3V_7$ ..... | $\left\{\begin{matrix} {}^2V_7 = {}^3V_7 \\ {}^1V_1 = {}^1V_1 \end{matrix}\right\}$ | $= 3+1=4$ ..... | 1 | ... | ... | ... | ... | ... | 4 | | | | | | | | |
| 19 | ÷ | $^3V_6 \div {}^3V_7$ | $^1V_9$ ..... | $\left\{\begin{matrix} {}^3V_6 = {}^3V_6 \\ {}^3V_7 = {}^3V_7 \end{matrix}\right\}$ | $= \frac{2n-2}{4}$ ..... | ... | ... | ... | ... | ... | $2n-2$ | 4 | ... | $\frac{2n-2}{4}$ | ... | $\left\{\frac{2n}{2} \cdot \frac{2n-1}{3} \cdot \frac{2n-2}{3}\right\} = A_3$ | | | | |
| 20 | × | $^1V_9 \times {}^4V_{11}$ | $^5V_{11}$ ..... | $\left\{\begin{matrix} {}^1V_9 = {}^0V_9 \\ {}^4V_{11} = {}^5V_{11} \end{matrix}\right\}$ | $= \frac{2n}{2} \cdot \frac{2n-1}{3} \cdot \frac{2n-2}{4} = A_3$ | ... | ... | ... | ... | ... | ... | ... | ... | 0 | | | | | | |
| | | | | $\left\{\begin{matrix} {}^1V_{22} = {}^1V_{22} \\ \quad = 2n \\ {}^3V_{11} = {}^3V_{11} \end{matrix}\right\}$ | $= 2n \cdot \frac{2n-1}{2n+1} \cdot \frac{2n-2}{3} \cdot \frac{2n-3}{4}$ | | | | | | | | | | | | | | | |

# The world's first program– who wrote it?

- First program already in 1843
- Caculates Bernoulli numbers
- Written for Babbages 'Analytical Engine'
  - which was never completed

- The Analytical Engine] might act upon other things besides number, were objects found whose mutual fundamental relations could be expressed by those of the abstract science of operations, and which should be also susceptible of adaptations to the action of the operating notation and mechanism of the engine...
- Supposing, for instance, that the fundamental relations of pitched sounds in the science of harmony and of musical composition were susceptible of such expression and adaptations, the engine might compose elaborate and scientific pieces of music of any degree of complexity or extent

# Ada Lovelace

# A sorting problem

- Example: Student records.

post →

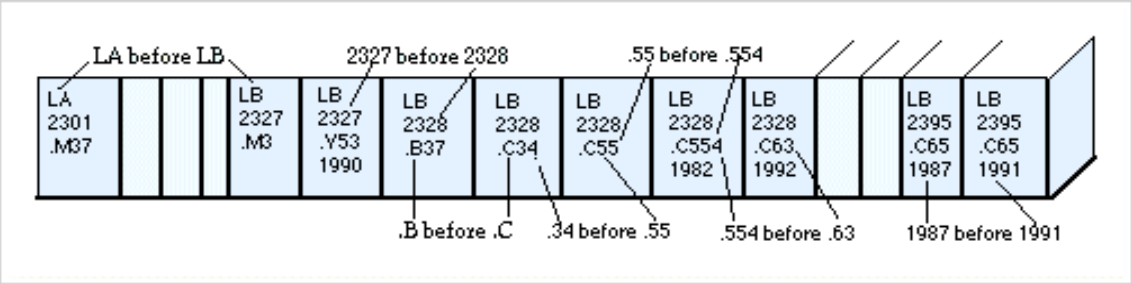
nyckel →

| | | | | |
|---|---|---|---|---|
| Chen | 3 | A | 991-878-4944 | 308 Blair |
| Rohde | 2 | A | 232-343-5555 | 343 Forbes |
| Gazsi | 4 | B | 766-093-9873 | 101 Brown |
| Furia | 1 | A | 766-093-9873 | 101 Brown |
| Kanaga | 3 | B | 898-122-9643 | 22 Brown |
| Andrews | 3 | A | 664-480-0023 | 097 Little |
| Battle | 4 | C | 874-088-1212 | 121 Whitman |

- Sort. Rearrange array of N items in ascending key order.

| | | | | |
|---|---|---|---|---|
| Andrews | 3 | A | 664-480-0023 | 097 Little |
| Battle | 4 | C | 874-088-1212 | 121 Whitman |
| Chen | 3 | A | 991-878-4944 | 308 Blair |
| Furia | 1 | A | 766-093-9873 | 101 Brown |
| Gazsi | 4 | B | 766-093-9873 | 101 Brown |
| Kanaga | 3 | B | 898-122-9643 | 22 Brown |
| Rohde | 2 | A | 232-343-5555 | 343 Forbes |

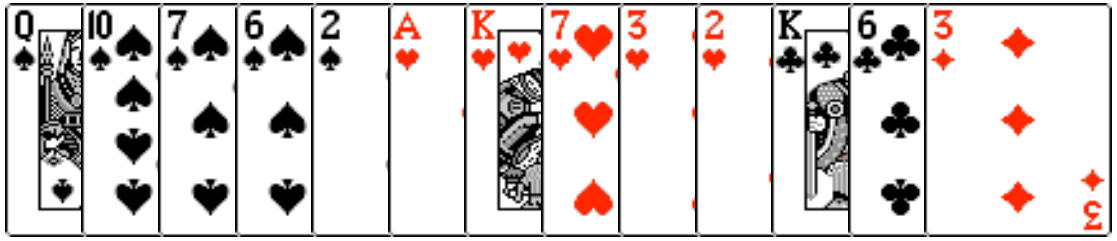
# Sorting applications




Library of Congress numbers

FedEx packages


contacts



playing cards

# Example: sorting client (1)

- Mål. Sort any type of data
- Ex. Sort random real numbers in ascending order.

```
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
0.2116619007164681 8
0.363292849257276
0.460954145685913
0.5340026311350087
0.7216129793703496
0.9003500354411443
0.9293994908845686
```

# Example: sorting client (2)

- Mål. Sort any type of data
- Ex. Sort strings from a file in alphabetical order.

```
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]
```

# Example: sorting client (3)

- Mål. Sort any type of data
- Ex. Sort files in a given directory by filename.
- .

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

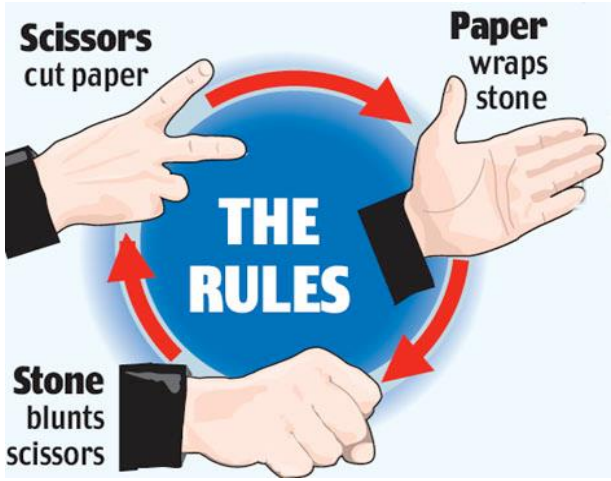
```
% java FileSorter .
Insertion.class
Insertion.java
InsertionX.class
InsertionX.java
Selection.class
Selection.java
Shell.class
Shell.java
ShellX.class
ShellX.java
```

# Total order

- Goal. Sort all types of data (where sorting is well-defined).
- A total order is a binary relation $\leq$ that satisfies:
  - Non-symmerty: IF ($v \leq w$ och $w \leq v$) THEN $v = w$.
  - transitive: IF ($v \leq w$ och $w \leq x$) THEN $v \leq x$.
  - totality: either $v \leq w$ or $w \leq v$ or both.

- Ex.
  - Usual orrder for integers and reals.
  - Chronological order for dates and times.
  - Alphabetical order for names.

- Ex. Non-transitive. rock-paper-scissors.



# Callbacks

- Goal. Sort all types of data (where sorting is well-defined).

- How can `sort()` know how to compare item of data types, like `Double`, `String`, `java.io.File`, or user-defined datatypes without having information on the datatype and the key?

- Callback = a reference to executable code.
  - Client gives an array of objects to the `sort()` method.
  - `sort()` will upon need call the `compareTo()` method on objects.

- To implement callbacks.
  - Java: interfaces.
  - C: function pointers.
  - C++: class-type functors.
  - C#: delegates.
  - Python, Perl, ML, Javascript: first-class functions.

# Callbacks in Java

client

```
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]);
   }
}
```

datatype implementation

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

Comparable interface (built into Java)

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

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;
}
```

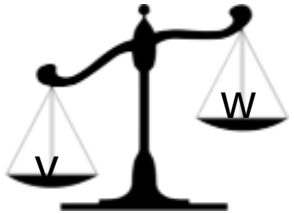
**No dependence on string datatype**

# Comparable API

- Implement `compareTo()` so that `v.compareTo(w)`:
  - Defines a total order;
  - Returns a negative integer, zero, or a positive integer IF `v` is respectively less than, equal to, or larger than `w`.
  - Will throw and exception if the types are incompatable (or i any of them are null)


less than (returns -1)


equal to (returns 0)




larger than (returns +1)

- Built-in comparable types. `Integer`, `Double`, `String`, `Date`, `File`, ...
- User-defined comparable types. Implement the `Comparable` interface

# Implementing the Comparable interface

- Date datatype. Simplified version of `java.util.Date`.

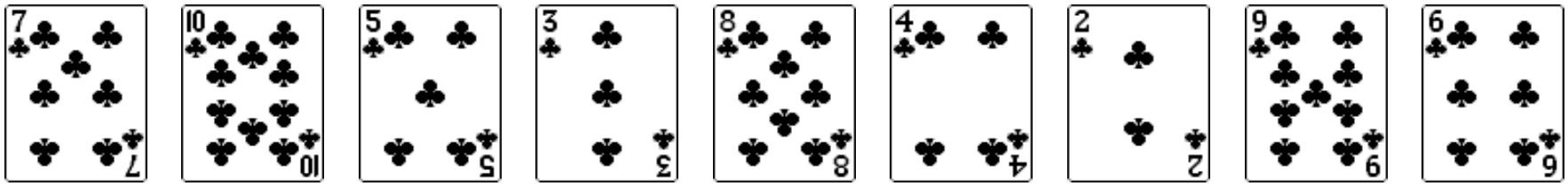
```
public class Date implements Comparable<Date>{
   private final int month, day, year;

   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 Date objects with other Date objects

# Elementary sorting

# Selection sort demo

- During iteration `i`, find the index `min` of the smallest remaining element
- Exchange `a[i]` with `a[min]`.

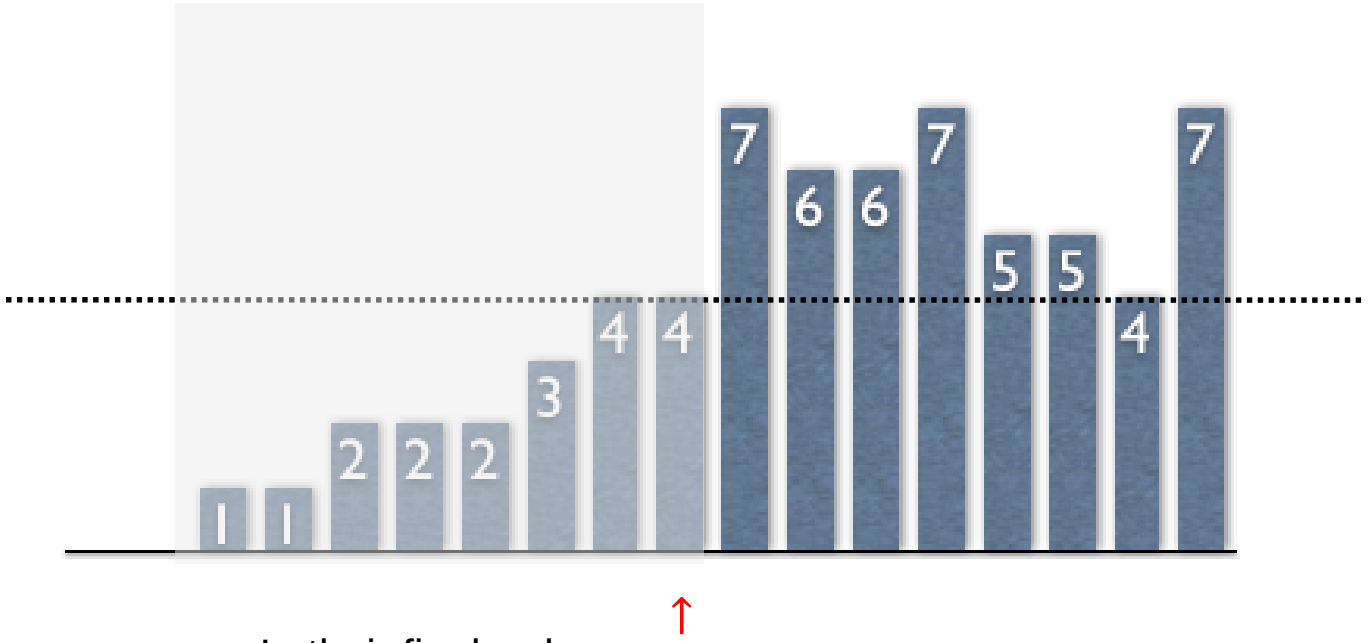


initial

# Selection sort algorithm

- Algorithm. ↑ Scans from left to right.

- Invariants.
  - Elements to the left of and including ↑ are sorted.
  - No element to right of ↑ is less than any element to the left of ↑.



# Two useful helper functions

- Helper functions. Compare and exchange operations on the elements

- Less. Is `v` less than `w` ?

```
private static boolean less(Comparable v, Comparable w) {
        return v.compareTo(w) < 0;
}
```

- exch. Exchange the element at index j with the element at index k.

```
private static void exch(Comparable[] a, int j, int k)
{
   Comparable swap = a[j];
   a[j] = a[k];
   a[k] = swap;
}
```

# Selection sort – the inner loop

- To maintain the algorithms invariants:

  - Move the scaning pointer to the right.

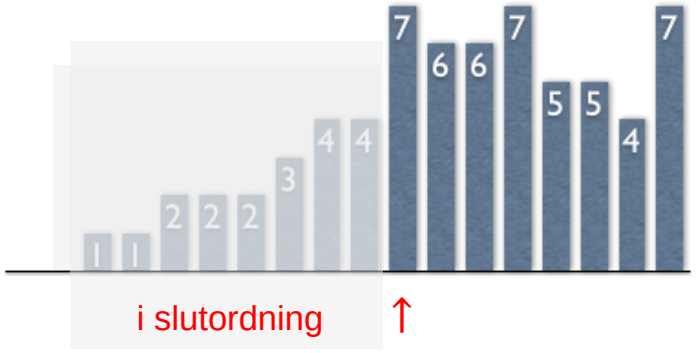
```
i++;
```

  - Identify the index of the smallest element to the right of the pointer

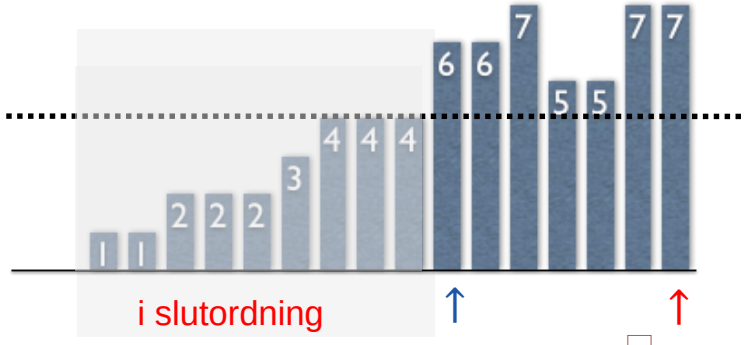
```
int min = i;
for (int j = i+1; j < N; j++) {
   if (less(a[j], a[min])) {
      min = j;
   }
}
```

  - Exchange the two elements.

```
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 analys

- Theorem. Selection sort uses $(N-1) + (N-2) + \ldots + 1 + 0 \sim N^2/2$ comparisons och $N$ exchanges.

| | | a[] | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| i | min | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| | | S | O | R | T | E | X | A | M | P | L | E |
| 0 | 6 | S | O | R | T | E | X | A | M | P | L | E |
| 1 | 4 | A | O | R | T | E | X | S | M | P | L | E |
| 2 | 10 | A | E | R | T | O | X | S | M | P | L | E |
| 3 | 9 | A | E | E | T | O | X | S | M | P | L | R |
| 4 | 7 | A | E | E | L | O | X | S | M | P | T | R |
| 5 | 7 | A | E | E | L | M | X | S | O | P | T | R |
| 6 | 8 | A | E | E | L | M | O | S | X | P | T | R |
| 7 | 10 | A | E | E | L | M | O | P | X | S | T | R |
| 8 | 8 | A | E | E | L | M | O | P | R | S | T | X |
| 9 | 9 | A | E | E | L | M | O | P | R | S | T | X |
| 10 | 10 | A | E | E | L | M | O | P | R | S | T | X |
| | | A | E | E | L | M | O | P | R | S | T | X |

*entries in black are examined to find the minimum*

*entries in red are a[min]*

*entries in gray are in final position*

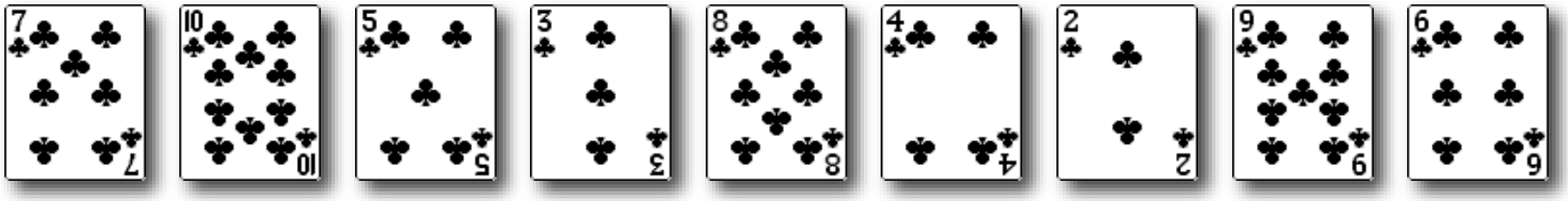
Trace of selection sort (array contents just after each exchange)

- Running time insensitive to input. Quadratic time, even if input is sorted.
- Data movement minimal. Linear number of exchanges.

# Insertion Sort

# Insertion sort demo

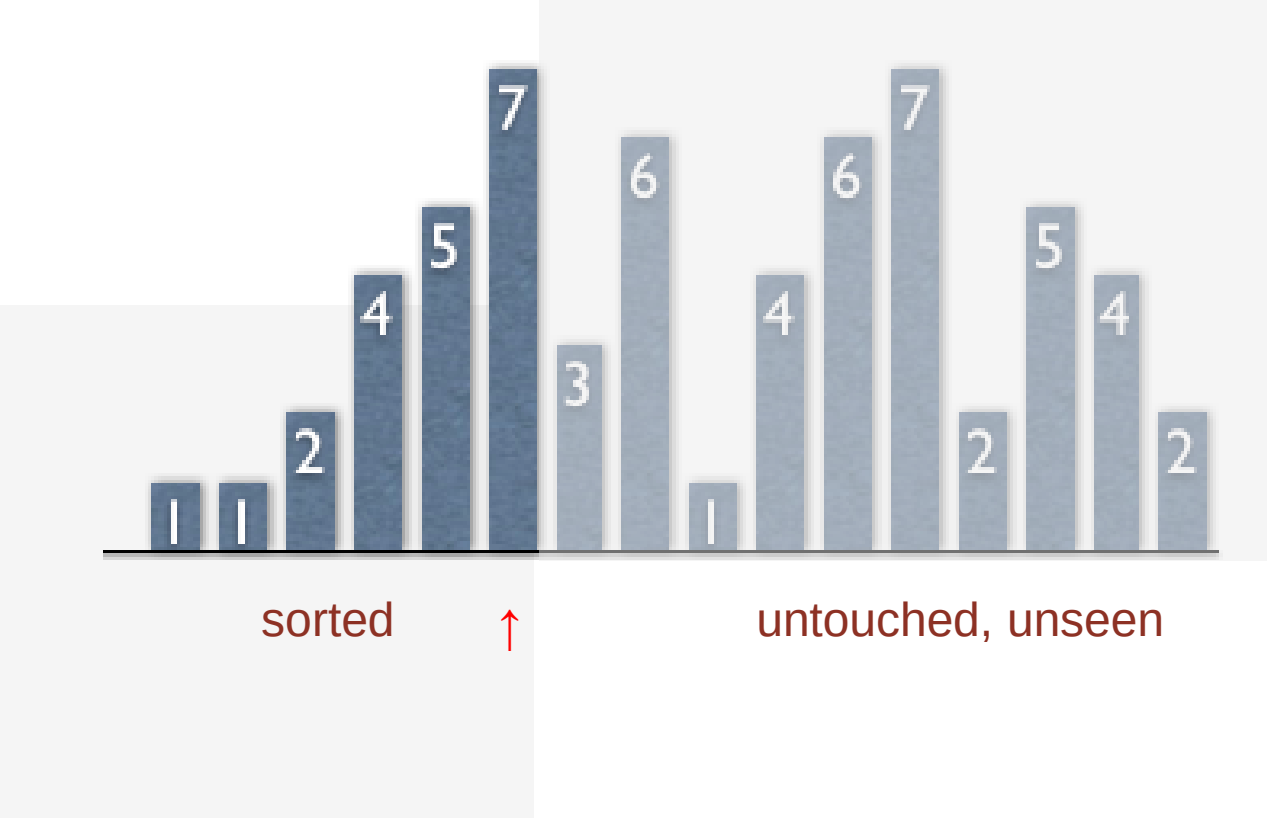
- During iteration `i`, `a[i] will do an exchange` with each element that is greater to the left.



# Insertion sort

- Algorithm. ↑ scans from left to right.

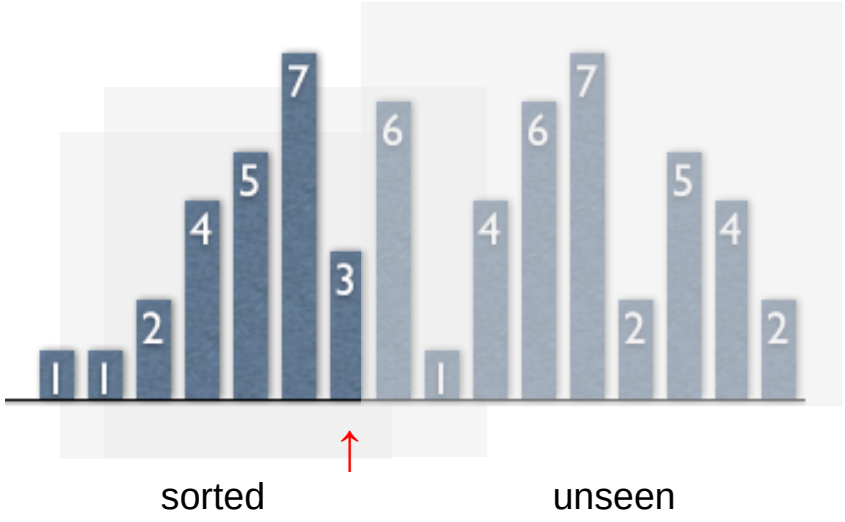
- Invariants.
  - Elements to the left of and including ↑ are in ascending order.
  - Elements to the right of ↑ have not yet been seen.



# Insertion sort inner loop

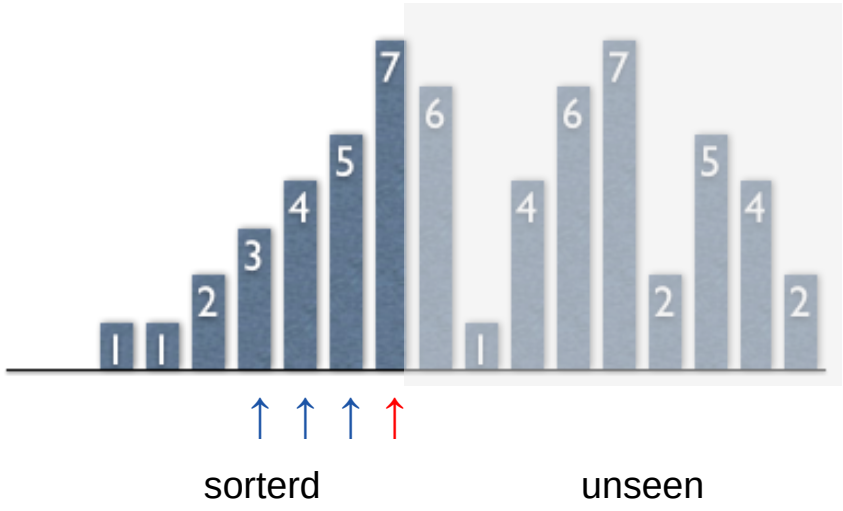
- Maintain the invariants:
  - Move the scanning pointer one step to right.

```
i++;
```




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

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

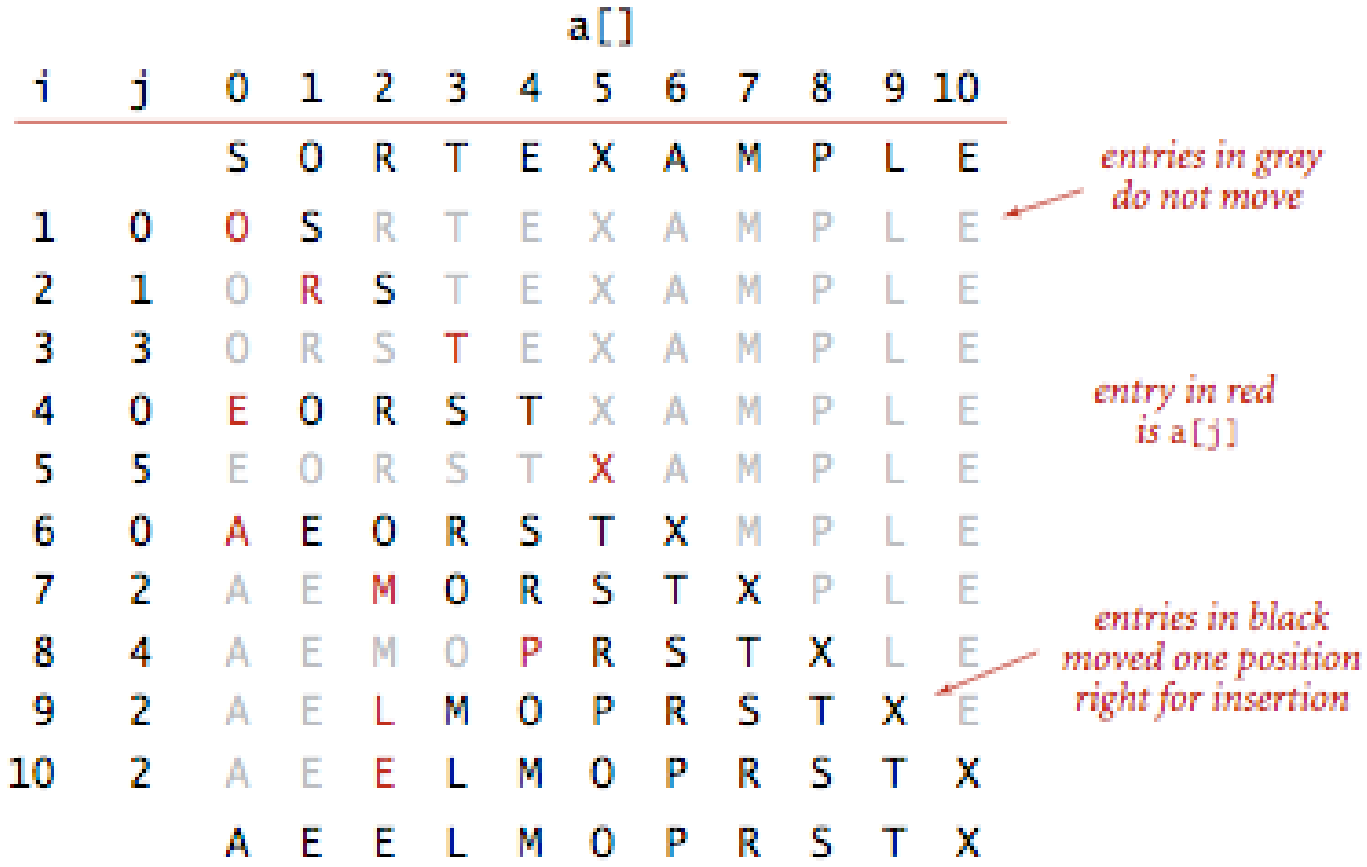


# Insertion sort: Java implementation

```
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

- Theorem. To sort a random array with distinct keys, insertion sort compares $\sim \frac{1}{4} N^2$ times and exchanges $\sim \frac{1}{4} N^2$ times on average.
- Proof. Expect each entry to move halfway back.




Trace of insertion sort (array contents just after each insertion)

# Insertion sort: trace

| | | a[ ] | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 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 |
| | | A | S | O | M | E | W | H | A | T | L | O | N | G | E | R | I | N | S | E | R | T | I | O | N | S | O | R | T | E | X | A | M | P | L | E |
| 0 | 0 | A | S | O | M | E | W | H | A | T | L | O | N | G | E | R | I | N | S | E | R | T | I | O | N | S | O | R | T | E | X | A | M | P | L | E |
| 1 | 1 | A | S | O | M | E | W | H | A | T | L | O | N | G | E | R | I | N | S | E | R | T | I | O | N | S | O | R | T | E | X | A | M | P | L | E |
| 2 | 1 | A | O | S | M | E | W | H | A | T | L | O | N | G | E | R | I | N | S | E | R | T | I | O | N | S | O | R | T | E | X | A | M | P | L | E |
| 3 | 1 | A | M | O | S | E | W | H | A | T | L | O | N | G | E | R | I | N | S | E | R | T | I | O | N | S | O | R | T | E | X | A | M | P | L | E |
| 4 | 1 | A | E | M | O | S | W | H | A | T | L | O | N | G | E | R | I | N | S | E | R | T | I | O | N | S | O | R | T | E | X | A | M | P | L | E |
| 5 | 5 | A | E | M | O | S | W | H | A | T | L | O | N | G | E | R | I | N | S | E | R | T | I | O | N | S | O | R | T | E | X | A | M | P | L | E |
| 6 | 2 | A | E | H | M | O | S | W | A | T | L | O | N | G | E | R | I | N | S | E | R | T | I | O | N | S | O | R | T | E | X | A | M | P | L | E |
| 7 | 1 | A | A | E | H | M | O | S | W | T | L | O | N | G | E | R | I | N | S | E | R | T | I | O | N | S | O | R | T | E | X | A | M | P | L | E |
| 8 | 7 | A | A | E | H | M | O | S | T | W | L | O | N | G | E | R | I | N | S | E | R | T | I | O | N | S | O | R | T | E | X | A | M | P | L | E |
| 9 | 4 | A | A | E | H | L | M | O | S | T | W | O | N | G | E | R | I | N | S | E | R | T | I | O | N | S | O | R | T | E | X | A | M | P | L | E |
| 10 | 7 | A | A | E | H | L | M | O | O | S | T | W | N | G | E | R | I | N | S | E | R | T | I | O | N | S | O | R | T | E | X | A | M | P | L | E |
| 11 | 6 | A | A | E | H | L | M | N | O | O | S | T | W | G | E | R | I | N | S | E | R | T | I | O | N | S | O | R | T | E | X | A | M | P | L | E |
| 12 | 3 | A | A | E | G | H | L | M | N | O | O | S | T | W | E | R | I | N | S | E | R | T | I | O | N | S | O | R | T | E | X | A | M | P | L | E |
| 13 | 3 | A | A | E | E | G | H | L | M | N | O | O | S | T | W | R | I | N | S | E | R | T | I | O | N | S | O | R | T | E | X | A | M | P | L | E |
| 14 | 11 | A | A | E | E | G | H | L | M | N | O | O | R | S | T | W | I | N | S | E | R | T | I | O | N | S | O | R | T | E | X | A | M | P | L | E |
| 15 | 6 | A | A | E | E | G | H | I | L | M | N | O | O | R | S | T | W | N | S | E | R | T | I | O | N | S | O | R | T | E | X | A | M | P | L | E |
| 16 | 10 | A | A | E | E | G | H | I | L | M | N | N | O | O | R | S | T | W | S | E | R | T | I | O | N | S | O | R | T | E | X | A | M | P | L | E |
| 17 | 15 | A | A | E | E | G | H | I | L | M | N | N | O | O | R | S | S | T | W | E | R | T | I | O | N | S | O | R | T | E | X | A | M | P | L | E |
| 18 | 4 | A | A | E | E | E | G | H | I | L | M | N | N | O | O | R | S | S | T | W | R | T | I | O | N | S | O | R | T | E | X | A | M | P | L | E |
| 19 | 15 | A | A | E | E | E | G | H | I | L | M | N | N | O | O | R | R | S | S | T | W | T | I | O | N | S | O | R | T | E | X | A | M | P | L | E |
| 20 | 19 | A | A | E | E | E | G | H | I | L | M | N | N | O | O | R | R | S | S | T | T | W | I | O | N | S | O | R | T | E | X | A | M | P | L | E |
| 21 | 8 | A | A | E | E | E | G | H | I | I | L | M | N | N | O | O | R | R | S | S | T | T | W | O | N | S | O | R | T | E | X | A | M | P | L | E |
| 22 | 15 | A | A | E | E | E | G | H | I | I | L | M | N | N | O | O | O | R | R | S | S | T | T | W | N | S | O | R | T | E | X | A | M | P | L | E |
| 23 | 13 | A | A | E | E | E | G | H | I | I | L | M | N | N | N | O | O | O | R | R | S | S | T | T | W | S | O | R | T | E | X | A | M | P | L | E |
| 24 | 21 | A | A | E | E | E | G | H | I | I | L | M | N | N | N | O | O | O | R | R | S | S | S | T | T | W | O | R | T | E | X | A | M | P | L | E |
| 25 | 17 | A | A | E | E | E | G | H | I | I | L | M | N | N | N | O | O | O | O | R | R | S | S | S | T | T | W | R | T | E | X | A | M | P | L | E |
| 26 | 20 | A | A | E | E | E | G | H | I | I | L | M | N | N | N | O | O | O | O | O | R | R | R | S | S | S | T | T | W | T | E | X | A | M | P | L | E |
| 27 | 26 | A | A | E | E | E | G | H | I | I | L | M | N | N | N | O | O | O | O | O | R | R | R | S | S | S | T | T | T | W | E | X | A | M | P | L | E |
| 28 | 5 | A | A | E | E | E | E | G | H | I | I | L | M | N | N | N | O | O | O | O | O | R | R | R | S | S | S | T | T | T | W | X | A | M | P | L | E |
| 29 | 29 | A | A | E | E | E | E | G | H | I | I | L | M | N | N | N | O | O | O | O | O | R | R | R | S | S | S | T | T | T | W | X | A | M | P | L | E |
| 30 | 2 | A | A | A | E | E | E | E | G | H | I | I | L | M | N | N | N | O | O | O | O | R | R | R | S | S | S | T | T | T | W | X | M | P | L | E |
| 31 | 13 | A | A | A | E | E | E | E | G | H | I | I | L | M | M | N | N | N | O | O | O | O | R | R | R | S | S | S | T | T | T | W | X | P | L | E |
| 32 | 21 | A | A | A | E | E | E | E | G | H | I | I | L | M | M | N | N | N | O | O | O | O | P | R | R | R | S | S | S | T | T | T | W | X | L | E |
| 33 | 12 | A | A | A | E | E | E | E | G | H | I | I | L | L | M | M | N | N | N | O | O | O | O | P | R | R | R | S | S | S | T | T | T | W | X | E |
| 34 | 7 | A | A | A | E | E | E | E | E | G | H | I | I | L | L | M | M | N | N | N | O | O | O | O | P | R | R | R | S | S | S | T | T | T | W | X |
| | | A | A | A | E | E | E | E | E | G | H | I | I | L | L | M | M | N | N | N | O | O | O | O | P | R | R | R | S | S | S | T | T | T | W | X |

# Insertion sort: analysis

- Best case. If the array is already in order, insertion sort will compare $N-1$ times and do $0$ exchanges.

A E E L M O P R S T X

- Worst case. If the array is in reverse order, insertion sort will do $\sim \frac{1}{2} N^2$ comparisons 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.

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 $\leq c\,N$.
  - T.ex. 1. A sorted array has $0$ inversions.
  - T.ex. 2. Append a subarray of size $10$ to a sorted array of length $N$.
    - How many inversions?
    - How many inversions in the worst case input?

- Theorem. For partially sorted arrays, insertion sort runs in linear time.
- Proof. Number of exchanges is equal to the number of inversions.

number comparisons= number of exchanges + (N – 1)

# Insertion sort: improvements

- Binary insertion sort. Use binary search to find the "insertion point".
  - Comparisons are now $\sim N \lg N$.
  - Unfortunately, still quadratic in the number of exchanges.

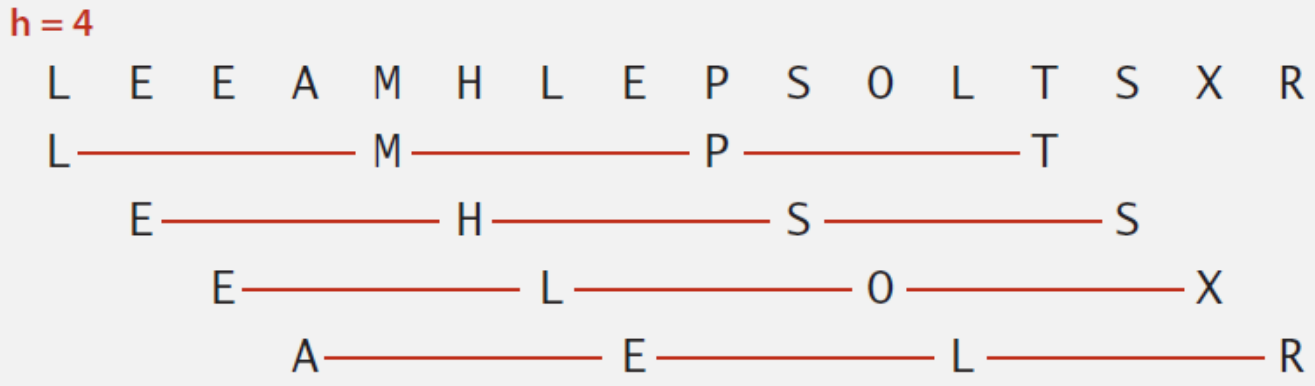

Binary search for the first key > K

# Shellsort

# Shellsort overview

- Idea. Move elements in bigger steps to the left by executing `h-sorting` on the array

an h-sorted array is h interleaved sorted subsequences

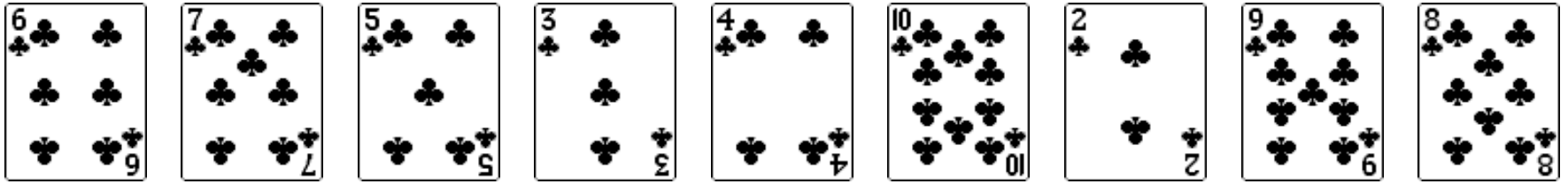



- Shellsort. [Shell 1959] `h-sort` array for a decreasing sequence of values for `h`.

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| input | S | H | E | L | L | S | O | R | T | E | X | A | M | P | L | E |
| 13-sort | P | H | E | L | L | S | O | R | T | E | X | A | M | S | L | E |
| 4-sort | L | E | E | A | M | H | L | E | P | S | O | L | T | S | X | R |
| 1-sort | A | E | E | E | H | L | L | L | M | O | P | R | S | S | T | X |

# h-sorting demo

- At iteration i, exchange `a[i]` vid elements that are `h` positions to the left



# h-sorting

- How to `h-sort` on an array?
  Run insertion sort, but with *stride length* `h`.

3-sorting an array

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

- Why insertion sort?
  - Large increments ⇒small subarray.
  - Small increments ⇒ almost already sorted.

# Shellsort exempel: increment by 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 E L O P M S X R T
A E E L O P M S X R T
A E E L O P M S X R T
A E E L M O P S X R T
A E E L M O P S X R T
A E E L M O P R S X T
A E E L M O P R S T X

resultat

A E E L M O P R S T X

# Shellsort: Java implementation

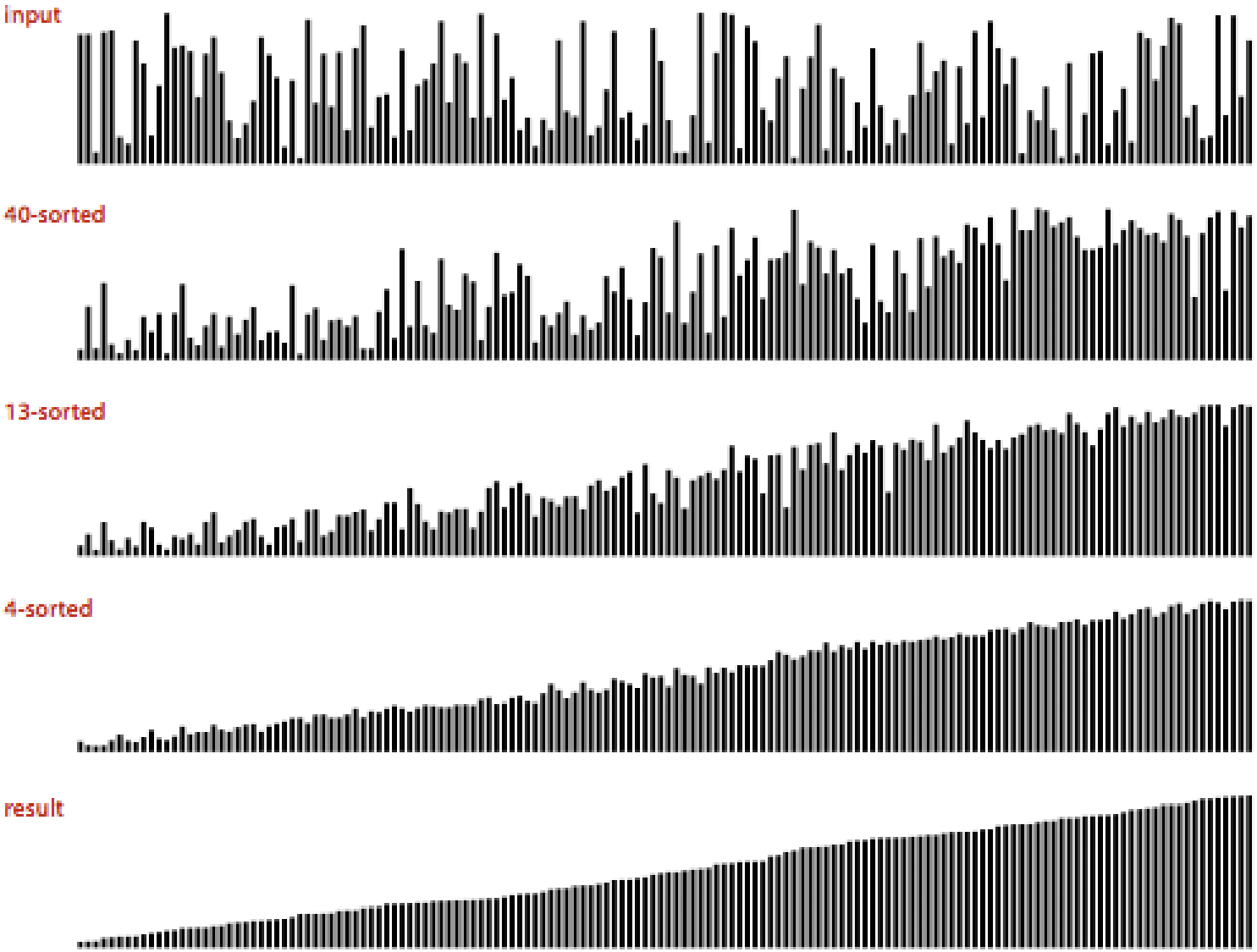
```
public class Shell {
   public static void sort(Comparable[] a)
   {
      int N = a.length;
      int h = 1;
      while (h < N/3) h = 3*h + 1; // 1, 4, 13, 40, 121, 364, ...
      while (h >= 1) {  // h-sort the array.
         for (int i = h; i < N; i++)  {
            for (int j = i; j >= h && 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 */ }
}
```

3x+1 inkrement sekvens

insertion sort

nästa inkrement

# Shellsort: visualization




Visual trace of shellsort

# Shellsort: what increment sequence should we 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 calculate

- Other sequences. Slightly better than above. Best increment sequence is unknown.

# Shellsort: intuition

- Theorem. A h-sorted array remains h-sorted after having done g-sort on the array.

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
A E L E O P M S X R T

still 7-sorted

Challenge. Prove this; more difficult than it seems!

# Shellsort: analysis

- Proposition. With the increment of 3x+1, the number of compares is $\sim N^{3/2}$.

- Some statistics

| N | compares | 2.5 N ln N | 0.25 N ln $^2$ N | N $^{1.3}$ |
|---|---|---|---|---|
| 5,000 | 93K | 106K | 91K | 64K |
| 10,000 | 209K | 230K | 213K | 158K |
| 20,000 | 467K | 495K | 490K | 390K |
| 40,000 | 1022K | 1059K | 1122K | 960K |
| 80,000 | 2266K | 2258K | 2549K | 2366K |

- Remark. Accurate model not yet discovered(!)
- Remark 2. Other sequences can bring it down to $\sim N^{6/5}$

# Why is shellsort interesting?

- A simple idea can give significant performance gains.

- Useful in practice.
  - Fast if array is not gigantic.
  - Tiny footprint for code (used in some embedded systems).

R, bzip2, /linux/kernel/groups.c

uClibc

- Simple algorithm, nontrivial performance, interesting open questions.
  - Asymptotical time complexity?
  - Best sequence of increment?
  - Average-case performance?

öppet problem: hitta en bättre inkrement sekvens

- Lesson. Many good algorithm not yet discovered.

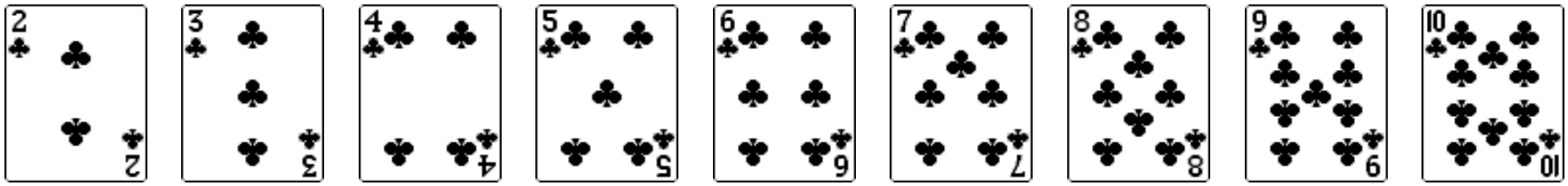
# Elementary sortering summary

| algoritm | best | average | worst |
|---|---|---|---|
| **selection sort** | $N^2$ | $N^2$ | $N^2$ |
| **insertion sort** | $N$ | $N^2$ | $N^2$ |
| **Shellsort (3x+1)** | $N \log N$ | ? | $N^{3/2}$ |
| **Goal** | $N$ | $N \log N$ | $N \log N$ |

# Shuffling

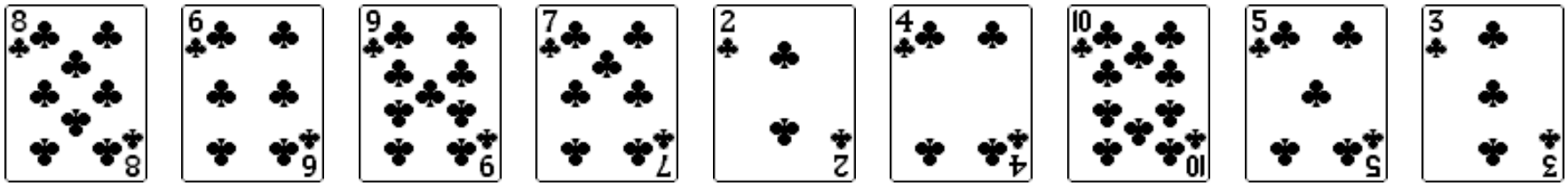
# How to shuffle an array?

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



# How to shuffle an array?

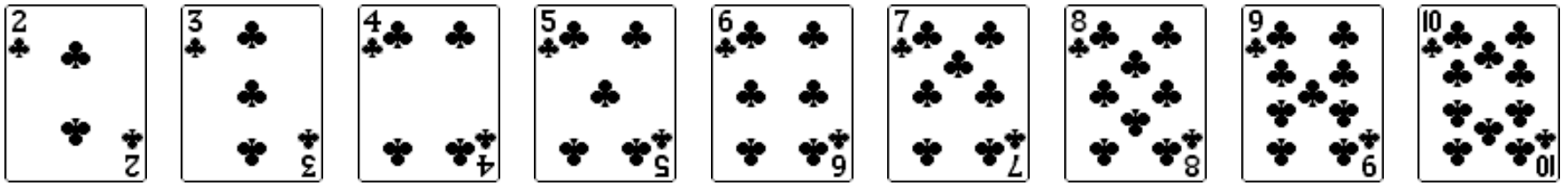
- Goal. Rearrange an array so that the result is a uniformly random permutation.



# Shuffle sort

- Generate a real number for each element in the array.
- Sort the array.

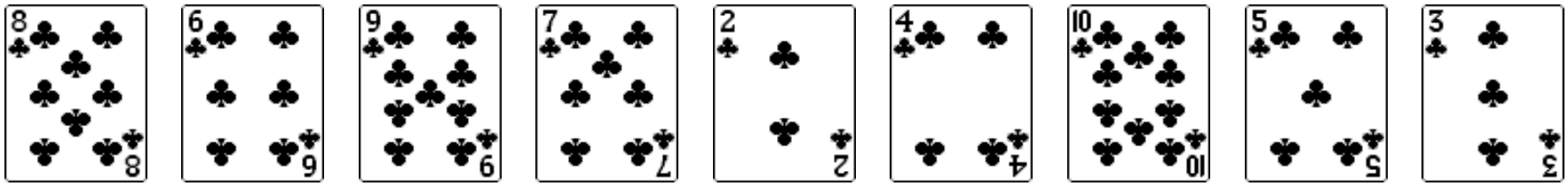
Useful for shuffling columns in a spreadsheet



0.8003 0.9706 0.9157 0.9649 0.1576 0.4854 0.1419 0.4218 0.9572

# Shuffle sort

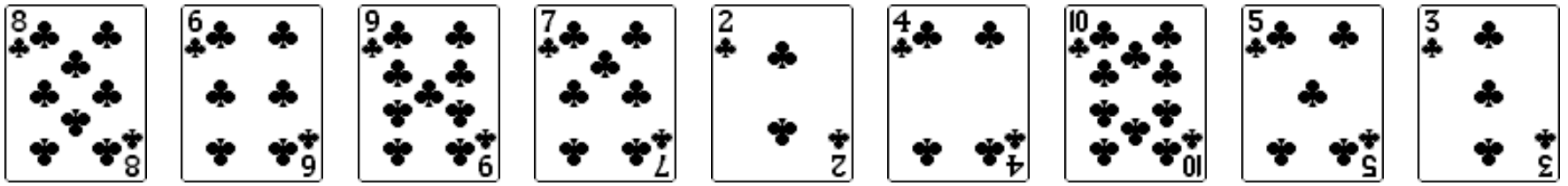
- Generate a real number for each element in the array.
- Sort the array in the order given by the real numbers.



0.1419 0.1576 0.4218 0.4854 0.8003 0.9157 0.9572 0.9649 0.9706

# Shuffle sort

- Generate a real number for each element in the array.
- Sort the array in the order given by the real numbers.



0.1419 0.1576 0.4218 0.4854 0.8003 0.9157 0.9572 0.9649 0.9706

- Theorem. Shuffle sort produces a uniformly random permutation.

# The Microsoft Shuffle

- Microsoft antitrust probe by the EU. Microsoft agreed to show a randomly generated screen so that users could choose their browser in Windows 7.
- Was this a coding error?

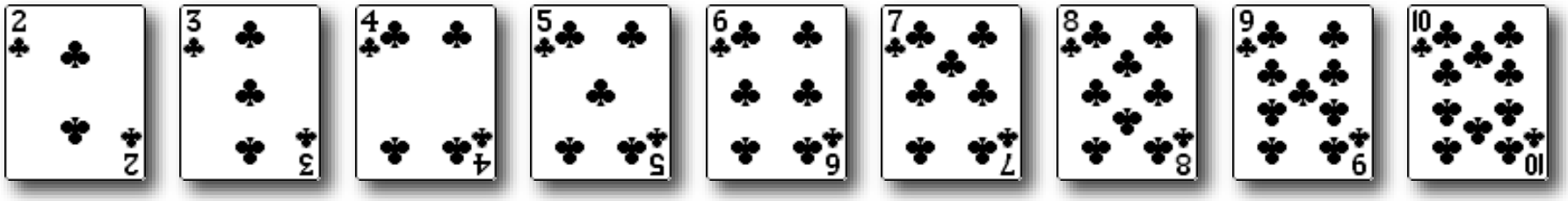
http://www.browserchoice.eu


Was last over

50% of the time

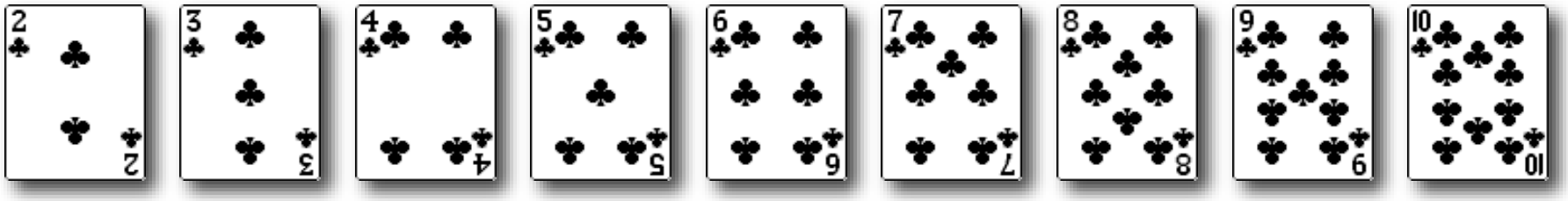
# Knuth shuffle demo

- During iteration `i`, pick a number between `0` and `i` uniformly at random.
- Swap `a[i]` and `a[r]`.



# Knuth shuffle

- During iteration `i`, pick a number between `0` and `i` uniformly at random.
- Swap `a[i]` and `a[r]`.



- Theorem. [Fisher-Yates 1938] Knuth shuffle sort produces a uniformly random permutation of the input array in linear time.

# Knuth shuffle

- During iteration `i`, pick a number between `0` and `i` uniformly at random.
- Swap `a[i]` and `a[r]`.

Common error: beteen 0 and N – 1

Should be between i and N – 1

```
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);
      }
   }
}
```

Between 0 and i

# Broken Knuth shuffle

- What happens if we choose between `0` and `N-1` ?
- No longer uniformly random!

Instead of between

0 och i

| | | |
|---|---|---|
| A B C | 1/6 | 4/27 |
| A C B | 1/6 | 5/27 |
| B A C | 1/6 | 5/27 |
| B C A | 1/6 | 5/27 |
| C A B | 1/6 | 4/27 |
| C B A | 1/6 | 4/27 |

Probability when shuffling { A, B, C }

# Online Poker

- Texas hold'em poker. Shuffling electronic cards.


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

**http://www.datamation.com/entdev/article.php/616221**

# Online poker anecdote

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;
```

between 1 and 51

- Bug 1. Number `r` is never 52 => 52:de card can never end up in place 52.
- Bug 2. Shuffle not uniform (should be between 1 och i).
- Bug 3. `random()` only uses a 32-bit seed => $2^{32}$ possible shuffles.
- Bug 4. Seed = milliseconds since midnight=> 86.4 million shuffles.

- Exploit. After seeing 5 cards and synchronizing with the server clock, can predict future cards in realtime.

*" The generation of random numbers is too important to be left to chance. "*

*— Robert R. Coveyou*

# Online poker: best practice

- Best practice for shuffling.
  - Use a hardware solution that has passed both FIPS 140-2 och NIST statistical test suites.
    - But, hardware random number generators are fragile and can ”fail silently".
  - Use an unbiased shuffling algorithm.


RANDOM.ORG

# Summary

- Elementary algorithms for sorting arrays
  - Selection sort
  - Insertion sort
  - Shell sort

- Shuffling
  - Shuffling is difficult!