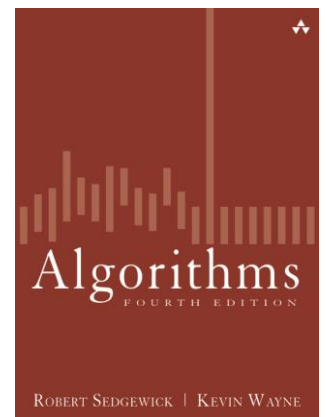


# ID1020:elementary sorts

Ch 2.1



Slides adapted from Algorithms 4<sup>th</sup> Edition, Sedgewick.

# A sorting problem

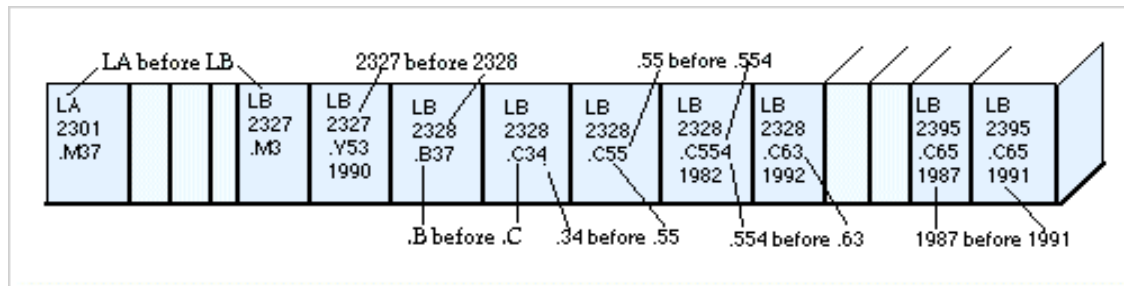
- **Eg.** Student records at a university.

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

- **Sorting.** Order an array of N recods in ascending 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

# Sort applications



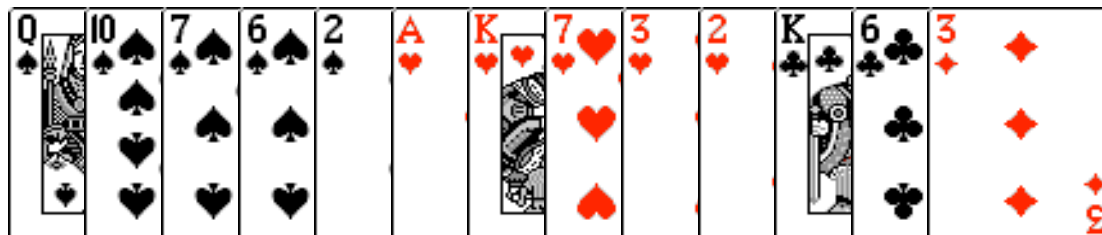
Library of Congress numbers



FedEx packages



contacts



playing cards

# Exemple: sorting client (1)

- **Target.** Sort any possible types of data.
- **Eg.1.** Sort random 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
0.21166190071646818
0.363292849257276
0.460954145685913
0.5340026311350087
0.7216129793703496
0.9003500354411443
0.9293994908845686
```

# Example: sorting client (2)

- Target. Sort any possible data types.
- Eg. 2. Sort all strings alphabetically.

```
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]
```

# Exempel: sorting client (3)

- **Target.** Sort any possible types of data.
- **Eg. 3.** Sort all files in a directory alphabetically by file name.

```
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 possible data types (when sorting is well-defined).
- A **total order** is a binary relation  $\leq$  which satisfy:
  - antisymmetric: IF  $(v \leq w \text{ and } w \leq v)$  THEN  $v = w$ .
  - transitive: IF  $(v \leq w \text{ and } w \leq x)$  THEN  $v \leq x$ .
  - total: either  $v \leq w$  or  $w \leq v$  or both.
- **Eg.**
  - Common order for integer and real numbers.
  - Chronological order for dates and times.
  - Alphabetic order for strings.
- **Non-transitive.** Stone-scissors-paper.



# Callbacks

- **Goal.** Sort all possible data types (when sorting is well-defined).
- How can `sort()` know how to compare data of different types, such as `Double`, `String`, and `java.io.File`, without any knowledge of the data types keys?
- **Callback = a reference to executable code.**
  - The client passes an array of objects as argument to the `sort()` function.
  - The `sort()` function calls the `compareTo()` method of the object when needed.
- **Implementing 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]);
    }
}
```

## Data type implementation

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

## Comparable interface (built-in of Java)

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

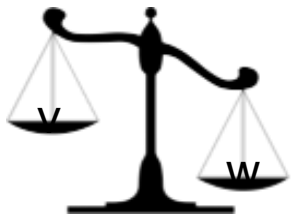
## sort implementation

```
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 dependencies  
of the String data type**

# Comparable API

- Implement `compareTo()` so that `v.compareTo(w)` :
  - It defines a total order;
  - returns a negative integer, zero, or a positive integer  
IF `v` is respectively less than, equal to, or greater than `w`.
  - Throws an exception if the types are incompatible or if either is `null`.



Less than (return -1)



Equal (return 0)



Greater than (return +1)

- **Built-in comparable types.** `Integer`, `Double`, `String`, `Date`, `File`, ...
- **User-defined comparable types.** Implement the `Comparable` interface - instances of the class (object) could be compared by the `compareTo` method.

# Why should compareTo() return negative, 0, positive

- Comparisons are often effectively implemented by subtraction
- Eg: String comparisons are common
- Simple and fast implementation:
  - Subtract the ASCII-values of each character until the difference is non-zero or the end of one or both strings are reached.

- Eg.

```
    A l b e r t
-   A l i c e
-----
    0 0 -7
```


# Implement the Comparable interface

- **Date data type.** 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;
    }
}
```



Compare Date objects only  
to other Date objects

# A non-transitive order

- Why does not `compareTo()` implement a total order?

```
public class Double implements Comparable<Double> {  
    private double x;  
  
    ...  
  
    public int compareTo(Double that) {  
        if      (this.x < that.x) return -1;  
        else if (this.x > that.x) return +1;  
        else                               return 0;  
    }  
}
```

`compareTo()` is non-transitiv!

It does not handle: `-0.0` vs. `0.0` och `Double.NaN`

# Elementary sorting

# Two useful operations

- **Operations.** Comparison and swap operations on elements (objects).
- **Less-than.** Is the element (object)  $v$  less than  $w$  ?

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

- **Swap (exchange).** Swap the element in the array  $a[]$  at index  $j$  to 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;  
}
```

# Mathematical analysis

- We use the following **cost model** for the elementary sorting algorithms:
  1. Number of comparisons
  2. Number of exchanges (swaps)
- If the algorithm does not swap elements, then we count the number of array accesses.

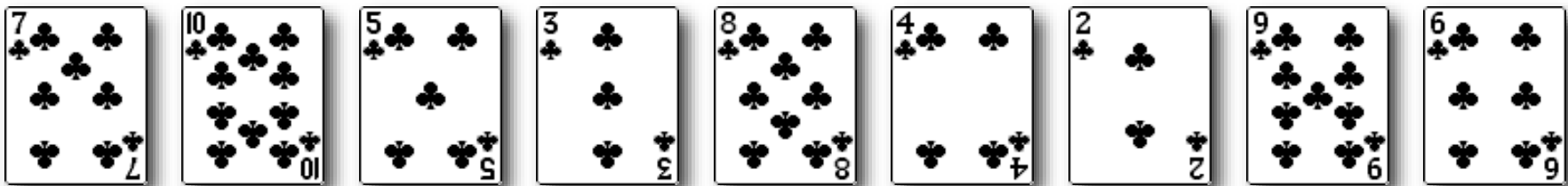




# Insertion Sort (Insättningssortering)

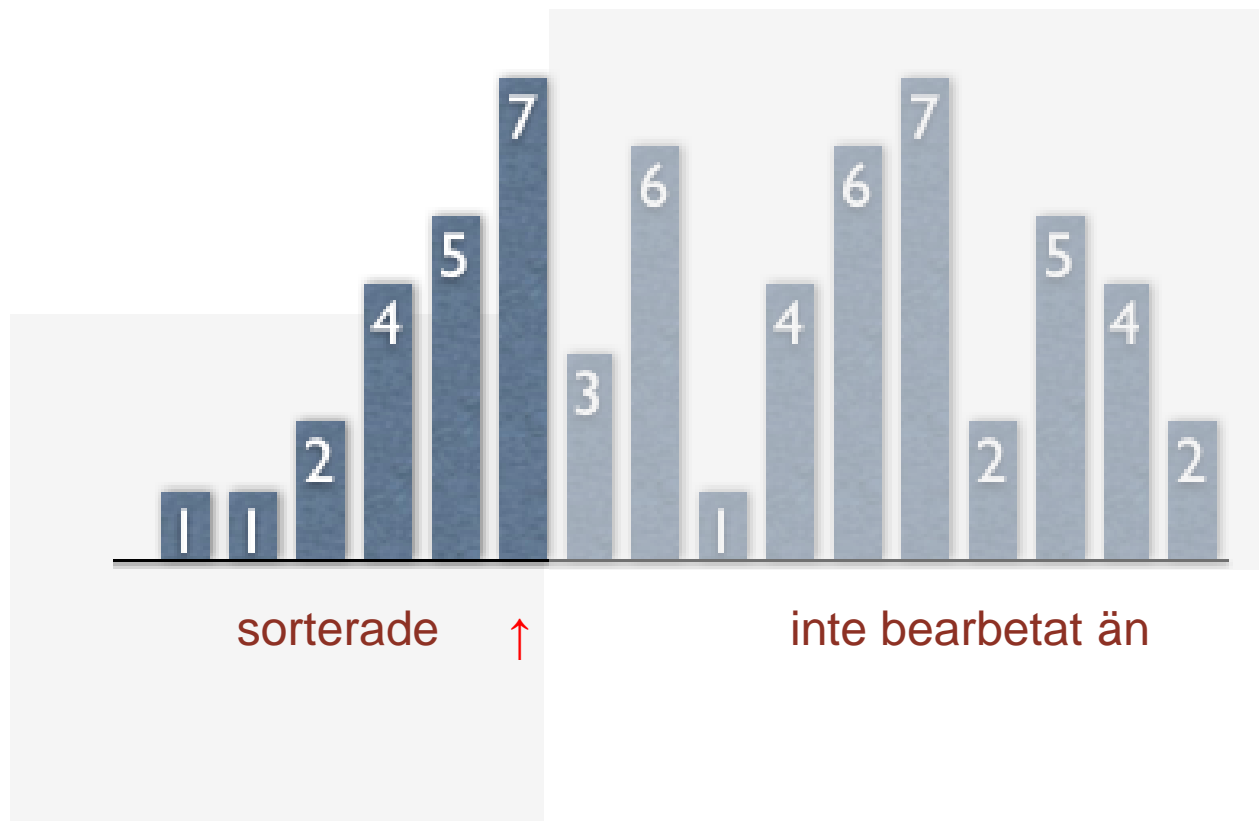
# Insertion sort

- During iteration  $i$ ,  $a[i]$  swaps with every element that is larger on the left hand side.



# Insertion sort

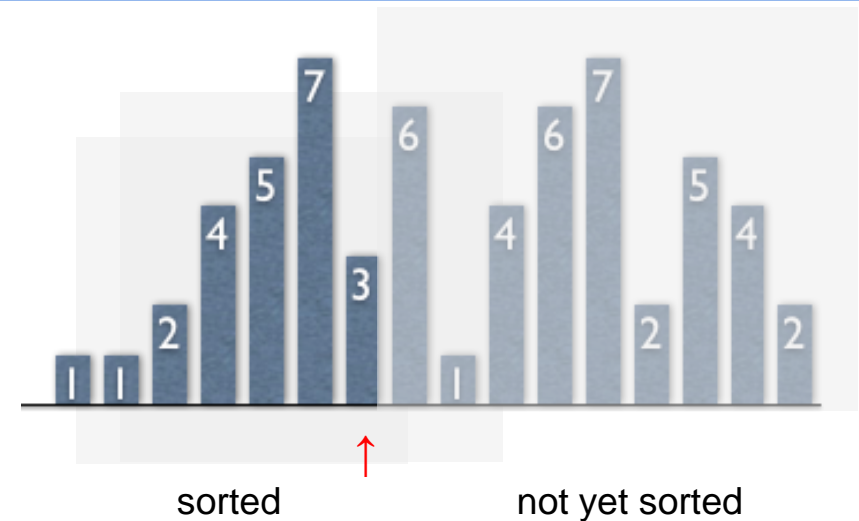
- Algorithm. ↑ scans from left to right.
- Invariants.
  - Elements to the left of ↑ (including ↑) are sorted in ascending order.
  - Elements to the right of ↑ has not been inspected (sorted) yet.



# Insertion sort **inner loop**

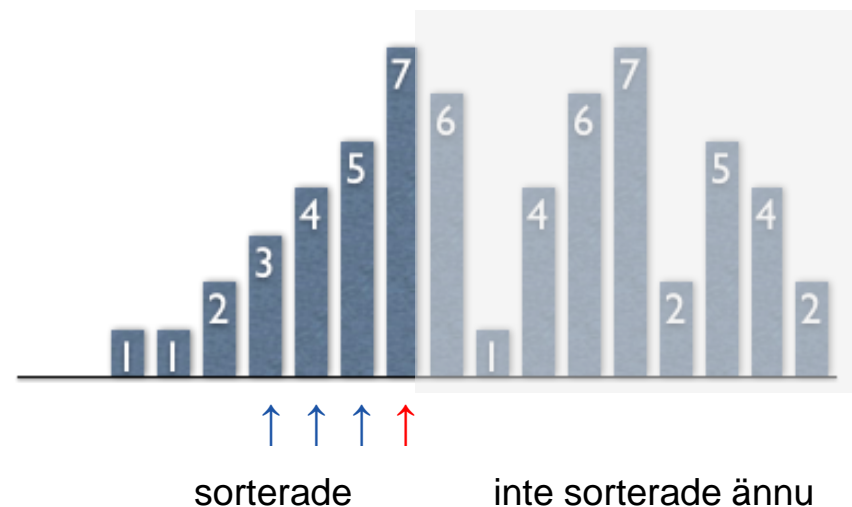
- Preserve the invariants:
  - Move the pointer (index) to the right.

```
i++;
```



- As an index is moved from right to left swap  $a[i]$  with every element on the left hand side which is larger.

```
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

- [https://en.wikipedia.org/wiki/Insertion\\_sort](https://en.wikipedia.org/wiki/Insertion_sort)

# Insertion sort: mathematical analysis

- **Proposition.** To sort an array of randomly ordered elements with unique keys, insertion sort do  $\sim \frac{1}{4} N^2$  comparisons and swaps  $\sim \frac{1}{4} N^2$  elements on average.
- **Proof.** Each of the  $N$  elements is compared to  $N/4$  elements and is moved  $N/4$  of steps on average.

		a[]											
i	j	0	1	2	3	4	5	6	7	8	9	10	
		S	O	R	T	E	X	A	M	P	L	E	
1	0	O	S	R	T	E	X	A	M	P	L	E	entries in gray do not move
2	1	O	R	S	T	E	X	A	M	P	L	E	
3	3	O	R	S	T	E	X	A	M	P	L	E	
4	0	E	O	R	S	T	X	A	M	P	L	E	entry in red is a[j]
5	5	E	O	R	S	T	X	A	M	P	L	E	
6	0	A	E	O	R	S	T	X	M	P	L	E	
7	2	A	E	M	O	R	S	T	X	P	L	E	
8	4	A	E	M	O	P	R	S	T	X	L	E	
9	2	A	E	L	M	O	P	R	S	T	X	E	entries in black moved one position right for insertion
10	2	A	E	E	L	M	O	P	R	S	T	X	
		A	E	E	L	M	O	P	R	S	T	X	

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	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		
11	6	A	A	E	H	L	M	N	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	
12	3	A	A	E	G	H	L	M	N	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
13	3	A	A	E	E	G	H	L	M	N	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	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	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	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	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 elements of the array are ordered in ascending order, then insertion sort will perform  $N-1$  comparisons and swap 0 elements.

A E E L M O P R S T X

- **Worst case.** If the elements of the array are ordered in descending order (unique keys), then insertion sort will perform  $\sim \frac{1}{2} N^2$  comparisons and swap  $\sim \frac{1}{2} N^2$  elements.

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 not ordered (in wrong 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 are  $\leq c N$ .
  - Ex. 1. A fully sorted array has 0 inversions.
  - Ex. 2. A sub-array of length 10 appended to a sorted array of length  $N$ .
- **Proposition.** Insertion sort sorts any partially sorted array in linear time ( $O(N)$ ).
- **Proof.** The number of swaps equals the number of inversions.

Number of comparisons = number of swaps +  $(N - 1)$

# Insertion sort: improvements

- **Binary insertion sort.** One can use binary search to find the “insertion point” in the sorted part of the array.
  - Number of comparisons  $\sim N \lg N$ .
  - Still  $\sim N^2$  array accesses.

A C H H I M N N P Q X Y **K** B I N A R Y



Binary search for the first key  $> K$

# Elementary sort summary

algorithm	best	average	worst
<b>selection sort</b>	$N^2$	$N^2$	$N^2$
<b>insertion sort</b>	$N$	$N^2$	$N^2$
<b>Shellsort (3x+1)</b>	$N \log N$	?	$N^{3/2}$
<b>Goal</b>	$N$	$N \log N$	$N \log N$

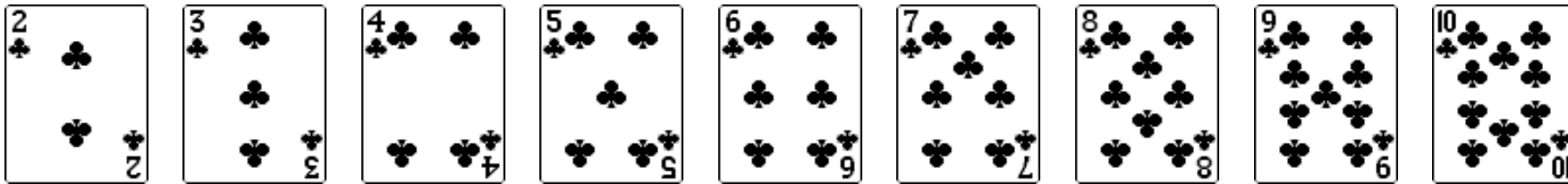
Time complexity to sort an array of  $N$  elements

# Shuffling (Blandning)

# How do you shuffle the elements of an array?

- **Target.** Shuffle the elements of an array so that the result is a random permutation of the elements.

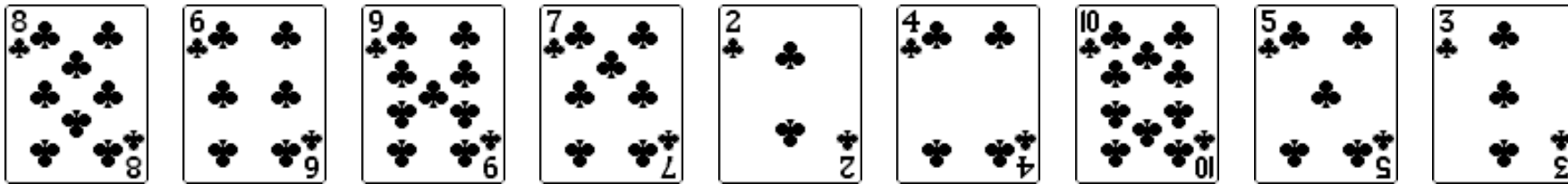
all permutations should be equally probable



# How do you shuffle the elements of an array?

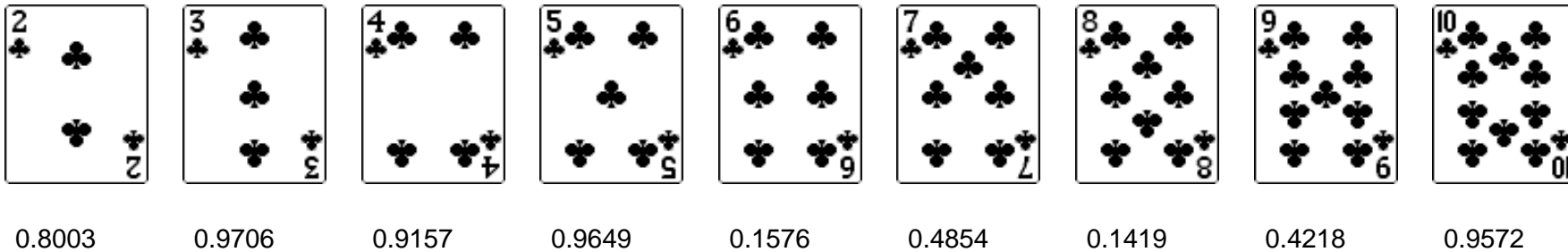
- **Target.** Shuffle the elements of an array so that the result is a random permutation of the elements.

all permutations should be equally probable



# Shuffle sort

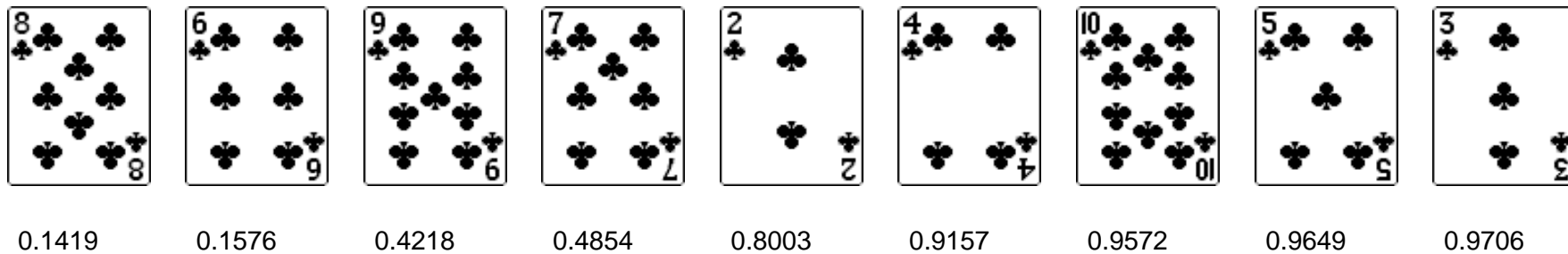
- Generate a random number associated with each element in the array.
- Sort the array.





# Shuffle sort

- Generate a random number associated with each element in the array.
- Sort the array.



- **Proposition.** Shuffle sort results in a uniformly random permutation (depending on the quality of the random number generator).

# Random number generation

- What we normally produce is a ***deterministic sequence of numbers exhibiting statistical properties of a random sequence*** (*white noise*)
- This is called ***pseudo-random number generators***
- The sequence is determined by an initial value called ***seed***
- Running the same random number generator several times with the same seed will generate the exact same sequences of numbers
- The best statistical properties are often generated if the seed is chosen as a ***large prime number***
- That the sequences can be repeated if the seeds are known is helpful for making experiments (simulations) reproducible
- The quality of the algorithm used by different random number generators may vary substantially

# The Microsoft Shuffle

- **Microsoft antitrust probe by EU.** Microsoft agreed to show a random sequence of the available browsers on the screen in Windows 7.

<http://www.browserchoice.eu>

## Select your web browser(s)



A fast new browser from Google. Try it now!



Safari for Windows from Apple, the world's most innovative browser.



Your online security is Firefox's top priority. Firefox is free, and made to help you get the most out of the



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



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



Was placed last  
50% of the time

# The Microsoft Shuffle

- **The Microsoft solution?** Implemented a shuffle sort by implementing a *comparator* which should generate a random answer.

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

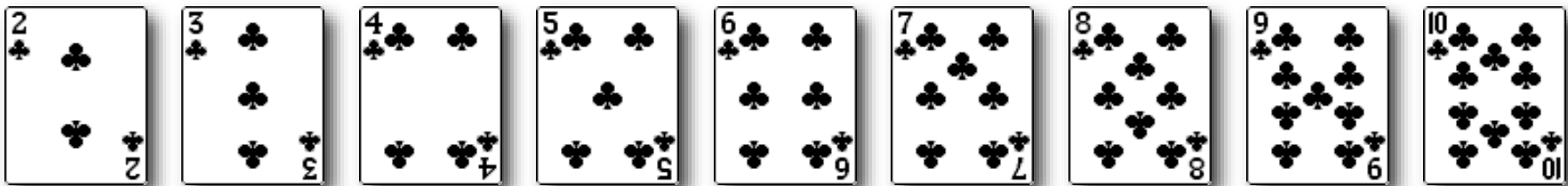
← How is the seed chosen?

← browser comparator  
(should implement a total order)

- Not reflexive, not antisymmetric, not transitive. If IE was the last in the non-permuted array and the sorting is insertion sort it would end up last in 50% of the shuffles

# Knuth shuffle

- In iteration  $i$ , generate a random integer  $r$  in the interval 0 to  $i$ .
- Swap  $a[i]$  and  $a[r]$ .



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

# Knuth shuffle

- In iteration  $i$ , generate a random integer  $r$  in the interval 0 to  $i$ .
- Swap  $a[i]$  and  $a[r]$ .

common bug :  $r$  between 0 and  $N - 1$

correct:  $r$  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 if  $r$  is drawn from the interval  $0$  to  $N-1$  ?
- No longer uniform distribution (random)!

Instead of  $0$  and  $i$

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 of each permutation of (each shuffle of)  $\{A, B, C\}$

# Online Poker

- Texas hold'em poker. The program should shuffle a deck of 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](http://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.** Random number  $r$  will never be 52  $\Rightarrow$  52<sup>nd</sup> card can never remain in the 52<sup>nd</sup> position.
- **Bug 2.** The shuffle is not uniform (should be between 1 and  $i$ ).
- **Bug 3.** `random()` used a small 32-bit seed  $\Rightarrow$   $2^{32}$  possible shuffles.
- **Bug 4.** Seed = milli seconds since midnight  $\Rightarrow$  86.4 million shuffles.
- **Exploit.** After having examined 5 cards and synchronized with the clock of the server, one can predict all future cards in real time.

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

— *Robert R. Coveyou*

# Online poker: best praxis

- Best praxis for shuffle.
  - Use a hardware solution to generate random numbers that has passed both FIPS 140-2 and NIST statistical tests.
    - But hardware random number generators may be fragile and could fail silently.
  - Use an independent generator such as random.org which use atmospheric noise to generate the random number sequence.



RANDOM.ORG

# Summary

- Simple algorithms to sort arrays
  - Selection sort
  - **Insertion sort**
  - Shell sort
- Shuffle
  - To properly shuffle a deck of cards is hard!