Algorithms ROBERT SEDGEWICK | KEVIN WAYNE

http://algs4.cs.princeton.edu

2.1 ELEMENTARY SORTS

- *rules of the game*
- *selection sort*
- *insertion sort*
- *shellsort*
- *shuffling*

http://algs4.cs.princeton.edu

2.1 ELEMENTARY SORTS

- *rules of the game*
- *selection sort*
- *insertion sort*
- *shellsort*
- *shuffling*

Sorting problem

Ex. Student records in 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
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

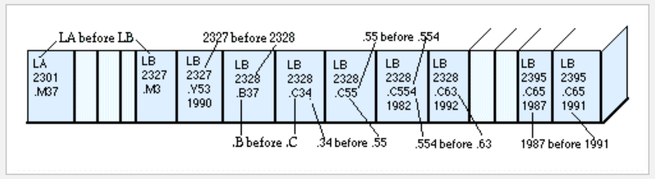
item →

key →

Sort. Rearrange array of N items into 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

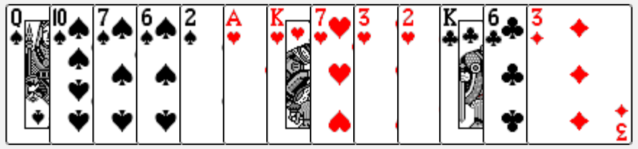
Sorting applications



Library of Congress numbers

contacts

FedEx packages



playing cards

Hogwarts houses

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)

```
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.

```
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.

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

A total order is a binary relation $\le$ that satisfies:

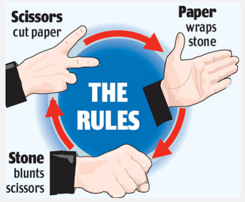
- Antisymmetry: if both $v \le w$ and $w \le v$, then $v = w$.
- Transitivity: if both $v \le w$ and $w \le x$, then $v \le x$.
- Totality: either $v \le w$ or $w \le v$ or both.

Ex.

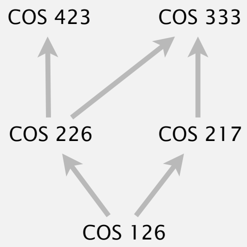
- Standard order for natural and real numbers.
- Chronological order for dates or times.
- Alphabetical order for strings.

No transitivity. Rock-paper-scissors.

No totality. PU course prerequisites.



violates transitivity



violates totality

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

```
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 to Java)

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

sort implementation

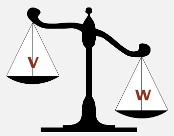
```
public static void sort(Comparable[] a)
{
   int N = a.length;
   for (int i = 0; i < N; i++)
      for (int j = i; j > 0; j--)
         if (a[j].compareTo(a[j-1]) < 0)
            exch(a, j, j-1);
         else break;
}
```

key point: no dependence on String data type

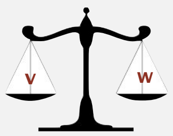
Comparable API

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

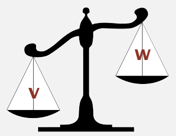
- Defines a total order.
- Returns a negative integer, zero, or positive integer if *v* is less than, equal to, or greater than *w*, respectively.
- Throws an exception if incompatible types (or either is `null`).



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`.

```
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

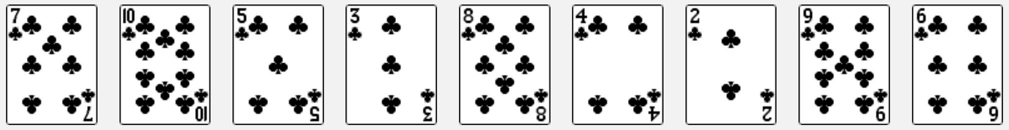
http://algs4.cs.princeton.edu

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]`.



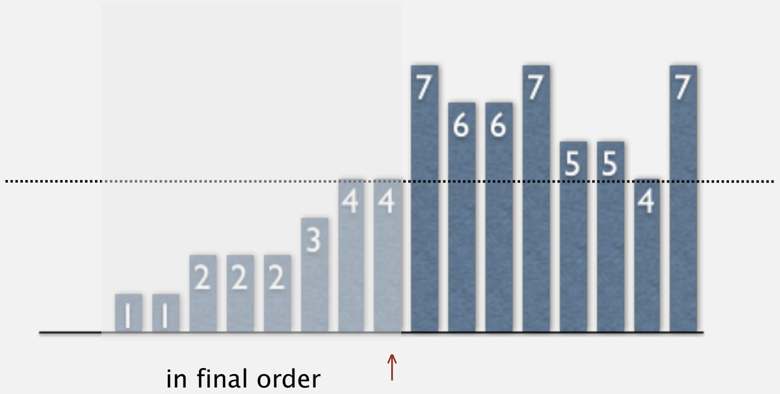
initial

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` ?

```
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`.

```
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

```
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

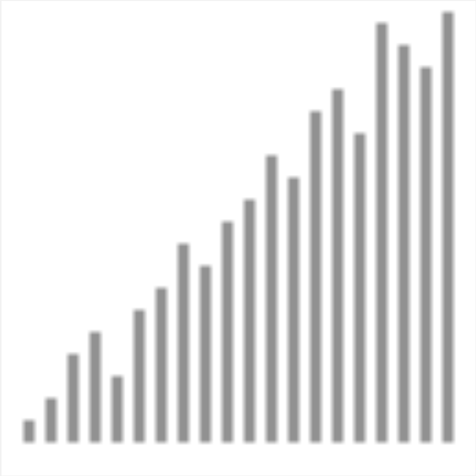
▬ in final order

▬ 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 $(N-1)+(N-2)+\ ...\ +1+0\ \sim\ N^2/2$ compares and 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 is minimal. Linear number of exchanges.

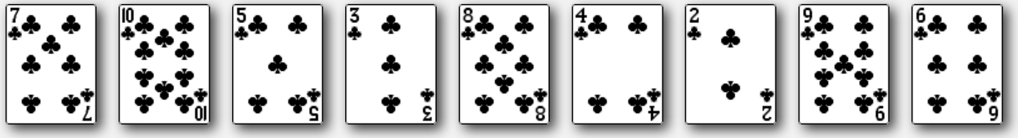
http://algs4.cs.princeton.edu

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.

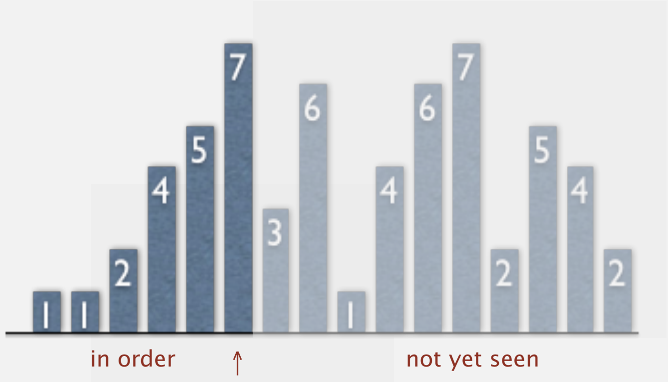


Insertion sort

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.

```
i++;
```

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

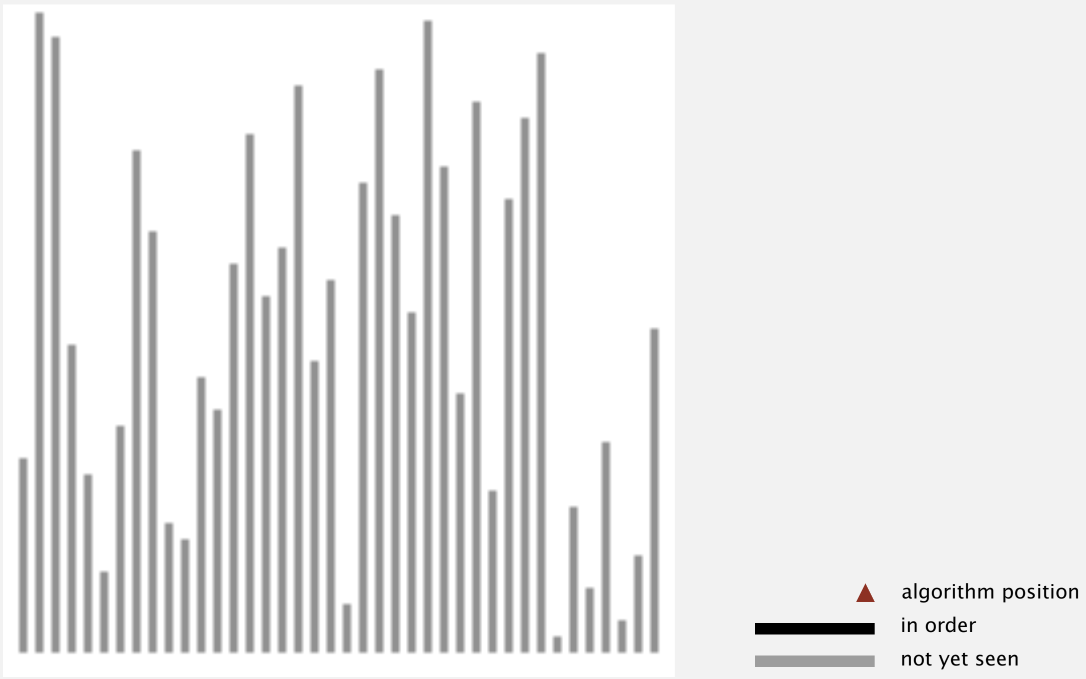
```
for (int j = i; j > 0; j--)
   if (less(a[j], a[j-1]))
        exch(a, j, j-1);
   else break;
```

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: animation

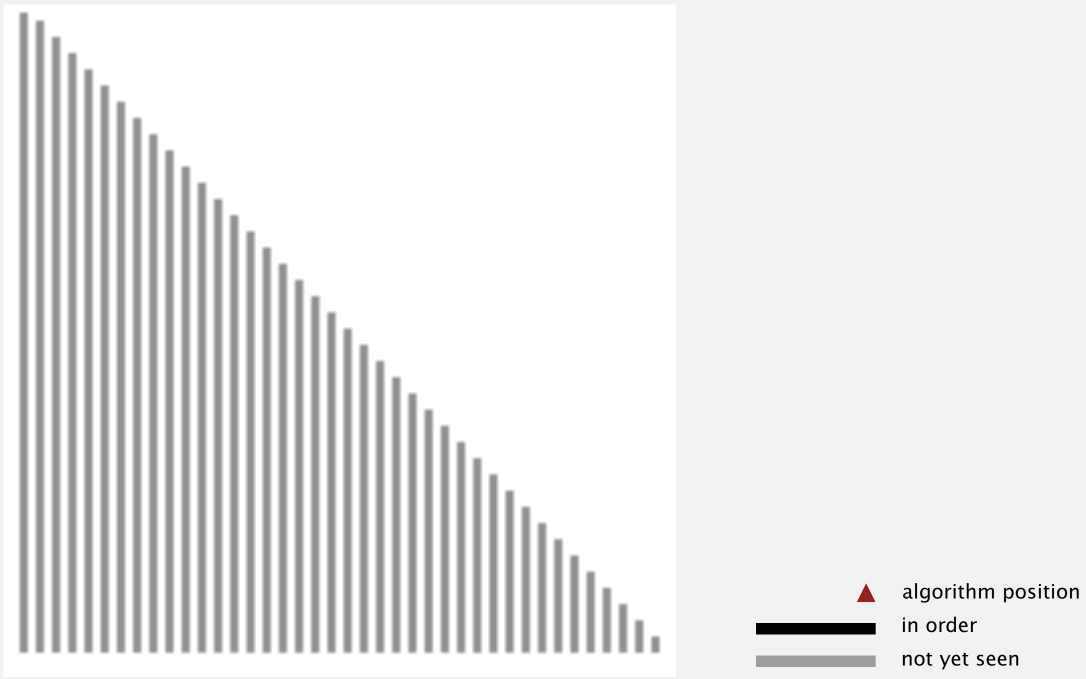
40 random items



http://www.sorting-algorithms.com/insertion-sort

Insertion sort: animation

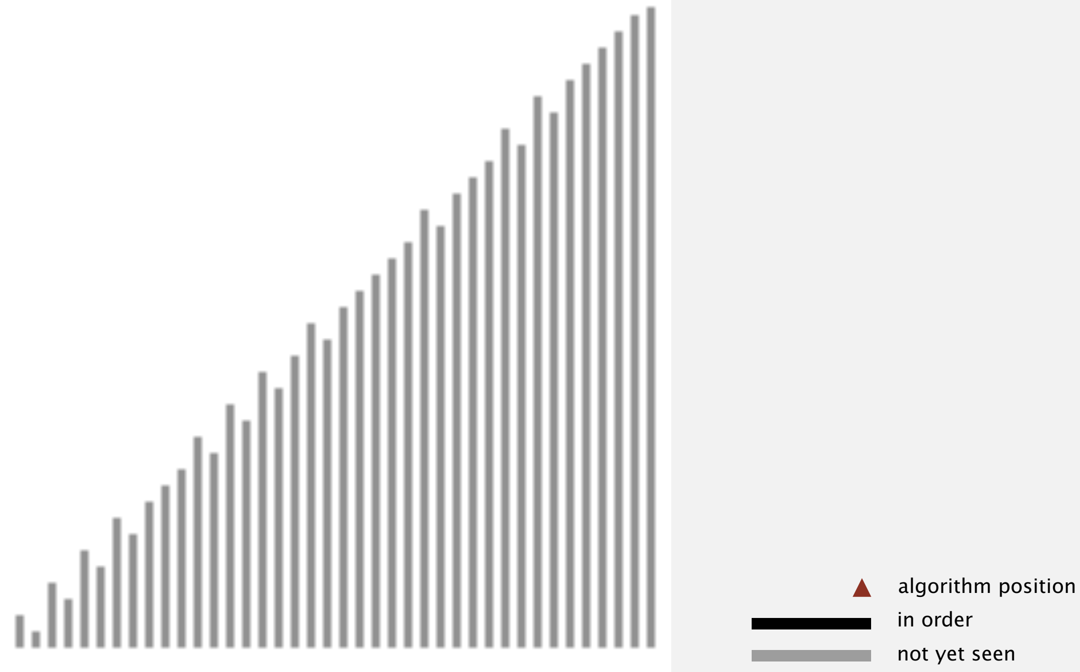
40 reverse-sorted items



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.

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
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
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
10	2	A	E	E	L	M	O	P	R	S	T	X
		A	E	E	L	M	O	P	R	S	T	X

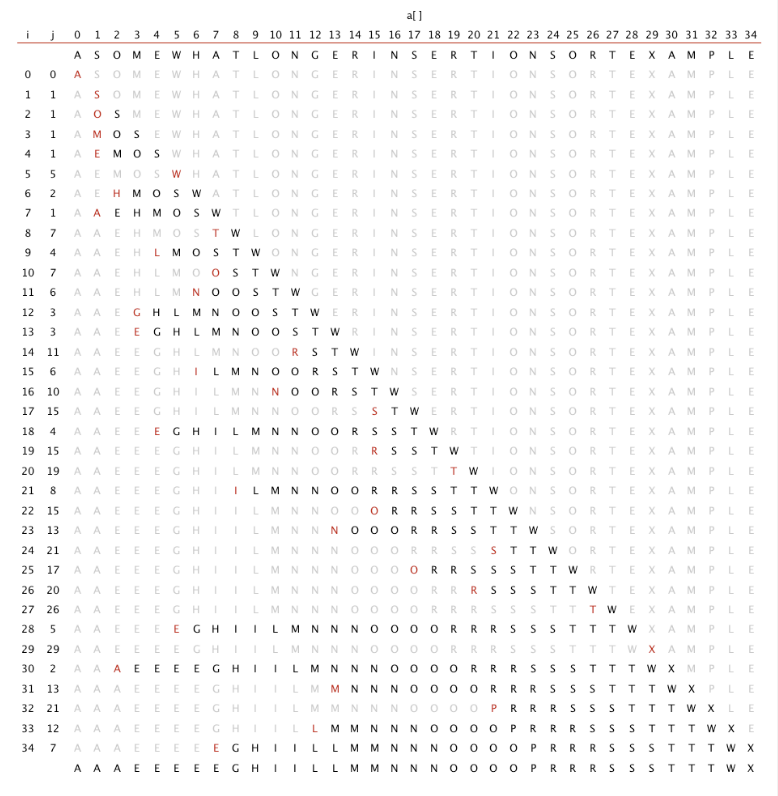
entries in gray do not move

entry in red is `a[j]`

entries in black moved one position right for insertion

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	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 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.

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 ≤ $c\,N$.

- 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.

number of compares = 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

http://algs4.cs.princeton.edu

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 h-sorting the array.

an h-sorted array is h interleaved sorted subsequences

h = 4

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

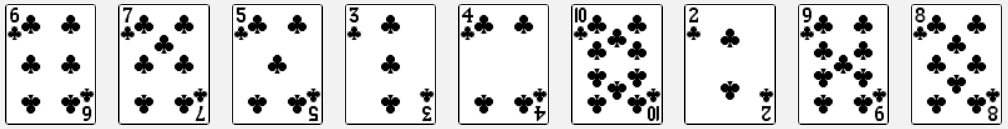
A — E — L — R

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

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

```
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?

- 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 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 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
```

result

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

Shellsort: Java implementation

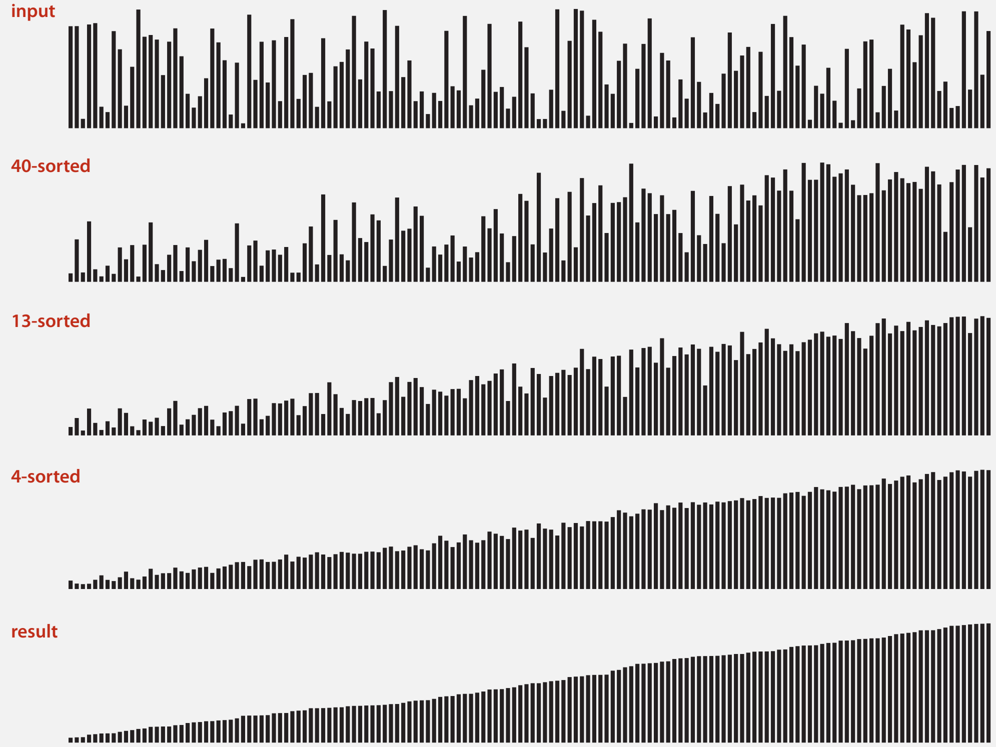
```
public class Shell
{
   public static void sort(Comparable[] a)
   {
      int N = a.length;

      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 increment sequence

insertion sort

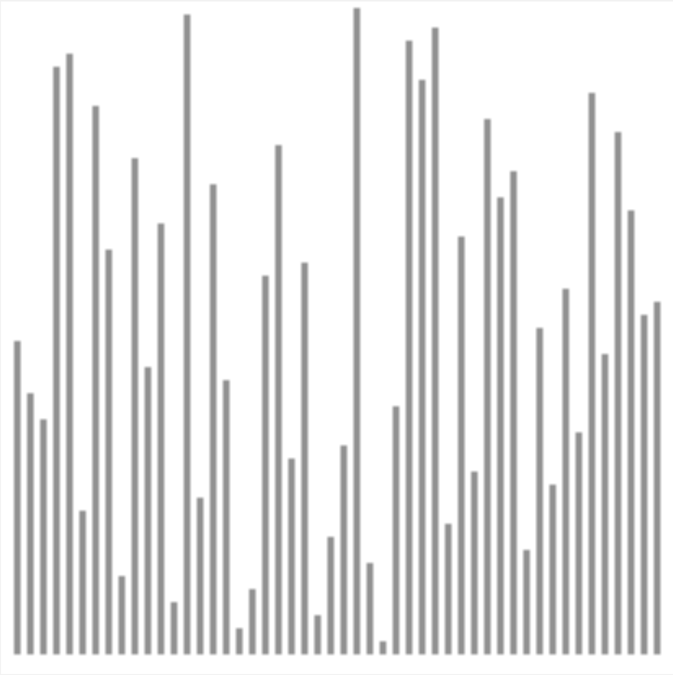
move to next increment

Shellsort: visual trace



Shellsort: animation

50 random items

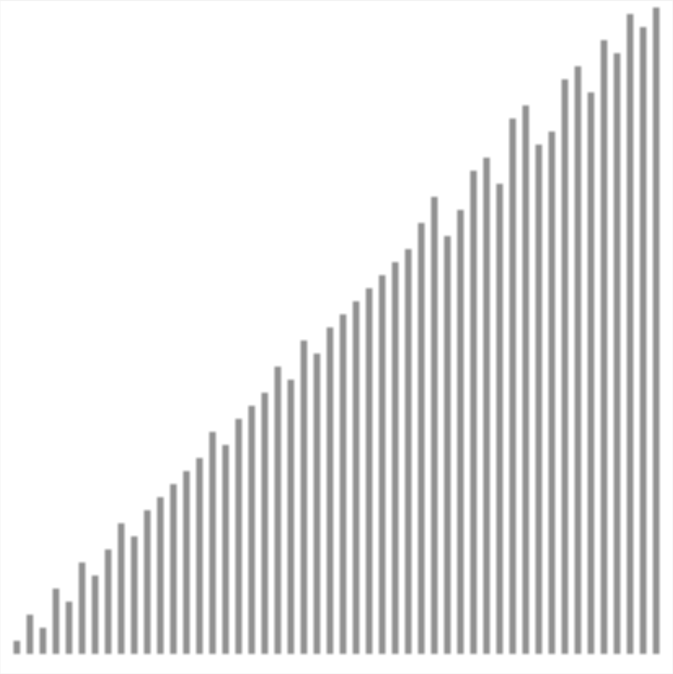


http://www.sorting-algorithms.com/shell-sort

▲ algorithm position
▬ h-sorted
▬ current subsequence
▬ other elements

Shellsort: animation

50 partially-sorted items



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

▲ algorithm position
▬ h-sorted
▬ current subsequence
▬ other elements

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.

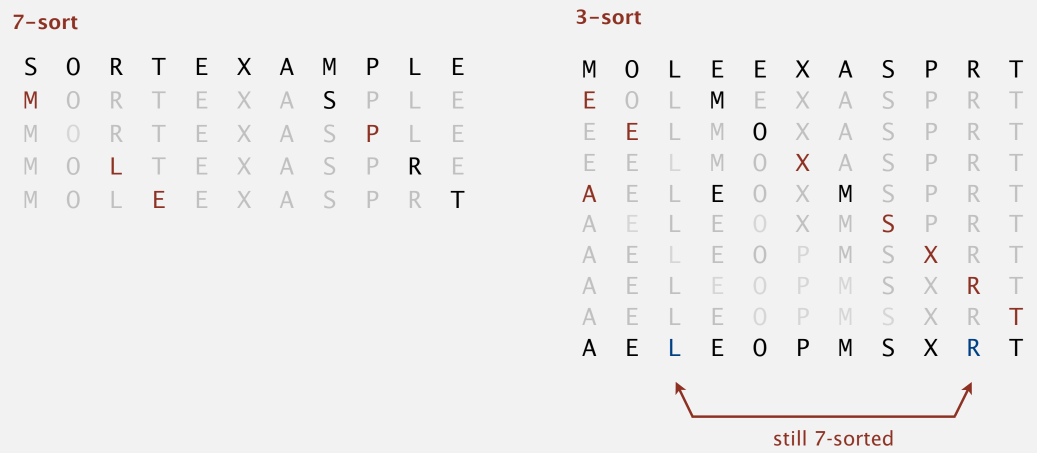
➔ 3x + 1. 1, 4, 13, 40, 121, 364, ...
OK. Easy to compute.

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: 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....

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 has not yet been discovered (!)

Why are we interested in shellsort?

Example of simple idea leading to substantial performance gains.

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

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.

uClibc

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.

algorithm	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	$N \log N$	$N \log N$

order of growth of running time to sort an array of N items

Next week. $N \log N$ sorting algorithms (in worst case).

http://algs4.cs.princeton.edu

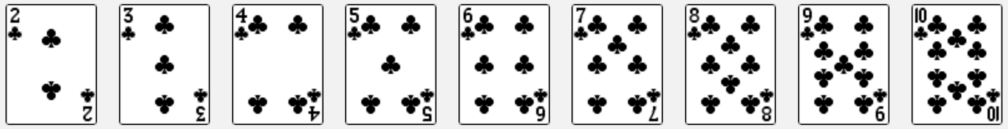
2.1 Elementary Sorts

- 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.

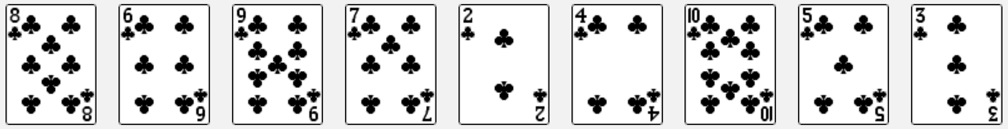
all permutations
equally likely



How to shuffle an array

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

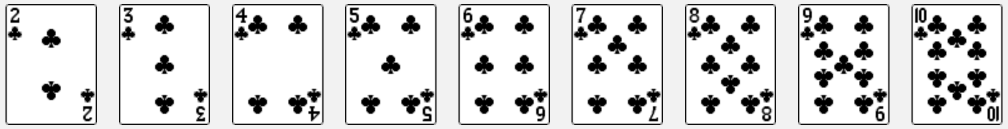
all permutations
equally likely



Shuffle sort

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

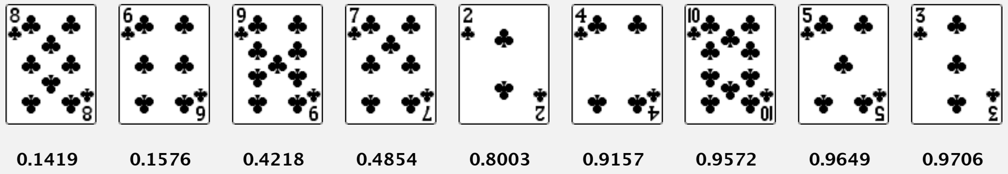
useful for shuffling
columns in a spreadsheet



Shuffle sort

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

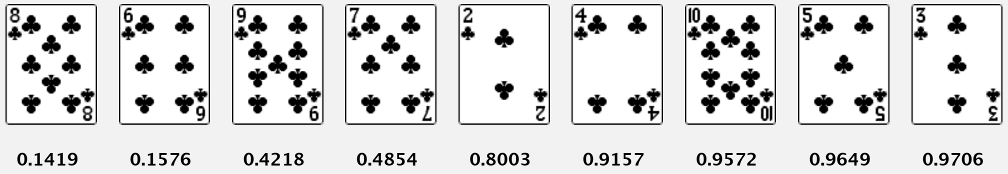
useful for shuffling columns in a spreadsheet



Shuffle sort

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

useful for shuffling columns in a spreadsheet



Proposition. Shuffle sort produces a uniformly random permutation.

assuming real numbers uniformly at random (and no ties)

War story (Microsoft)

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)

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.

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.

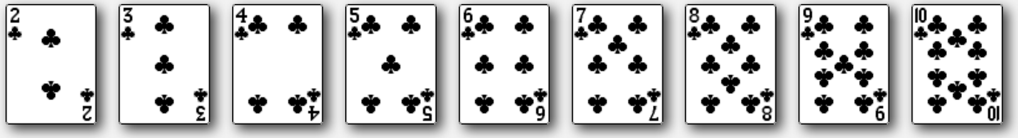
Solution? Implement shuffle sort by making comparator always return a random answer.

```
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)

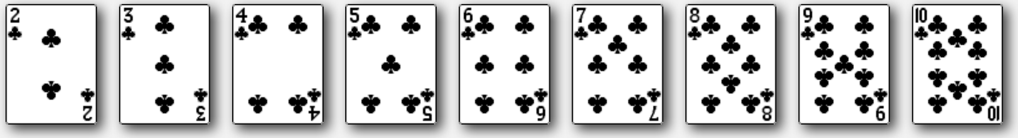
Knuth shuffle demo

- 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.

assuming integers uniformly at random

Knuth shuffle

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

common bug: between 0 and N – 1
correct variant: 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

Q. What happens if integer is chosen between `0` and `N-1` ?

A. Not uniformly random!

instead of 0 and i

permutation	Knuth shuffle	broken shuffle
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 result when shuffling { A, B, C }

War story (online poker)

Texas hold'em poker. Software must shuffle electronic cards.

mamajoe: Hey guys, Big B is in!

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;
```

between 1 and 51

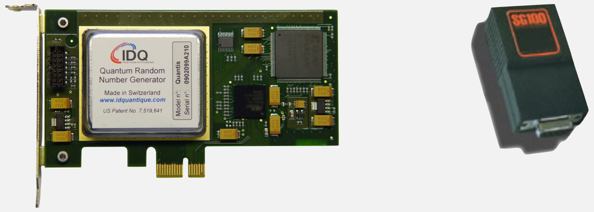
Bug 1. Random number r never 52 ⇒ 52^{nd} card can't end up in 52^{nd} place.
Bug 2. Shuffle not uniform (should be between 1 and i).
Bug 3. `random()` uses 32-bit seed ⇒ 2^{32} possible shuffles.
Bug 4. Seed = milliseconds since midnight ⇒ 86.4 million shuffles.

> *" 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.



RANDOM.ORG

Bottom line. Shuffling a deck of cards is hard!