2.1 Elementary Sorts

- rules of the game
- selection sort
- insertion sort
- sorting challenges
- shellsort

Sorting problem

Ex. Student record in a university.

file →	Fox	1	A	243-456-9091	101 Brown
	Quilici	1	C	343-987-5642	32 McCosh
	Chen	2	A	884-232-5341	11 Dickinson
	Furia	3	A	766-093-9873	22 Brown
	Kanaga	3	B	898-122-9643	343 Forbes
record →	Andrews	3	A	874-088-1212	121 Whitman
	Rohde	3	A	232-343-5555	115 Holder
	Battle	4	C	991-878-4944	308 Blair
key →	Aaron	4	A	664-480-0023	097 Little
	Gazsi	4	B	665-303-0266	113 Walker

Sort. Rearrange array of N objects into ascending order.

Aaron	4	A	664-480-0023	097 Little
Andrews	3	A	874-088-1212	121 Whitman
Battle	4	C	991-878-4944	308 Blair
Chen	2	A	884-232-5341	11 Dickinson
Fox	1	A	243-456-9091	101 Brown
Furia	3	A	766-093-9873	22 Brown
Gazsi	4	B	665-303-0266	113 Walker
Kanaga	3	B	898-122-9643	343 Forbes
Rohde	3	A	232-343-5555	115 Holder
Quilici	1	C	343-987-5642	32 McCosh

Sample sort client

Goal. Sort any type of data.

Ex 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
```

Sample sort client

Goal. Sort any type of data.

Ex 2. Sort strings from standard input in alphabetical order.

```
public class StringSorter
{
   public static void main(String[] args)
   {
      String[] a = StdIn.readAll().split("\\s+");
      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 < words.txt
all bad bed bug dad ... yes yet zoo
```

Sample sort client

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
```

Callbacks

Goal. Sort any type of data.

Q. How can `sort()` know to compare data of type `String`, `Double`, and `File` without any information about the type of a key?

Callbacks = reference to executable code.

- Client passes array of objects to `sort()` function.
- The `sort()` function calls back object's `compareTo()` method as needed.

Implementing callbacks.

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

Callbacks: roadmap

client

```
import java.io.File;
public class FileSorter
{
   public static void main(String[] args)
   {
      File directory = new File(args[0]);
      File[] files = directory.listFiles();
      Insertion.sort(files);
      for (int i = 0; i < files.length; i++)
         StdOut.println(files[i].getName());
   }
}
```

object implementation

```
public class File
implements Comparable<File>
{
   ...
   public int compareTo(File 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 reference to `File`

Comparable API

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

- Returns a negative integer if `v` is less than `w`.
- Returns a positive integer if `v` is greater than `w`.
- Returns zero if `v` is equal to `w`.
- Throw an exception if incompatible types or either is `null`.

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

Required properties. Must ensure a total order.

- Reflexive: $(v = v)$.
- Antisymmetric: if $(v < w)$ then $(w > v)$; if $(v = w)$ then $(w = v)$.
- Transitive: if $(v \leq w)$ and $(w \leq x)$ then $(v \leq x)$.

Built-in comparable types. `String`, `Double`, `Integer`, `Date`, `File`, ...

User-defined comparable types. Implement the `Comparable` interface.

Implementing the Comparable interface: example 1

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

Implementing the Comparable interface: example 2

Domain names.

- Subdomain: `bolle.cs.princeton.edu`.
- Reverse subdomain: `edu.princeton.cs.bolle`.
- Sort by reverse subdomain to group by category.

```
public class Domain implements Comparable<Domain>
{
   private final String[] fields;
   private final int N;

   public Domain(String name)
   {
      fields = name.split("\\.");
      N = fields.length;
   }

   public int compareTo(Domain that)
   {
      for (int i = 0; i < Math.min(this.N, that.N); i++)
      {
         String s = fields[this.N - i - 1];
         String t = fields[that.N - i - 1];
         int cmp = s.compareTo(t);
         if      (cmp < 0) return -1;
         else if (cmp > 0) return +1;
      }
      return this.N - that.N;
   }
}
```

only use this trick when no danger of overflow

subdomains

```
ee.princeton.edu
cs.princeton.edu
princeton.edu
cnn.com
google.com
apple.com
www.cs.princeton.edu
bolle.cs.princeton.edu
```

reverse-sorted subdomains

```
com.apple
com.cnn
com.google
edu.princeton
edu.princeton.cs
edu.princeton.cs.bolle
edu.princeton.cs.www
edu.princeton.ee
```

Two useful sorting abstractions

Helper functions. Refer to data through compares and exchanges.

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

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

Exchange. Swap object 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;
}
```

Testing

Q. How to test if an array is sorted?

```
private static boolean isSorted(Comparable[] a)
{
   for (int i = 1; i < a.length; i++)
      if (less(a[i], a[i-1])) return false;
   return true;
}
```

Q. If the sorting algorithm passes the test, did it correctly sort its input?

A. Yes, if data accessed only through `exch()` and `less()`.

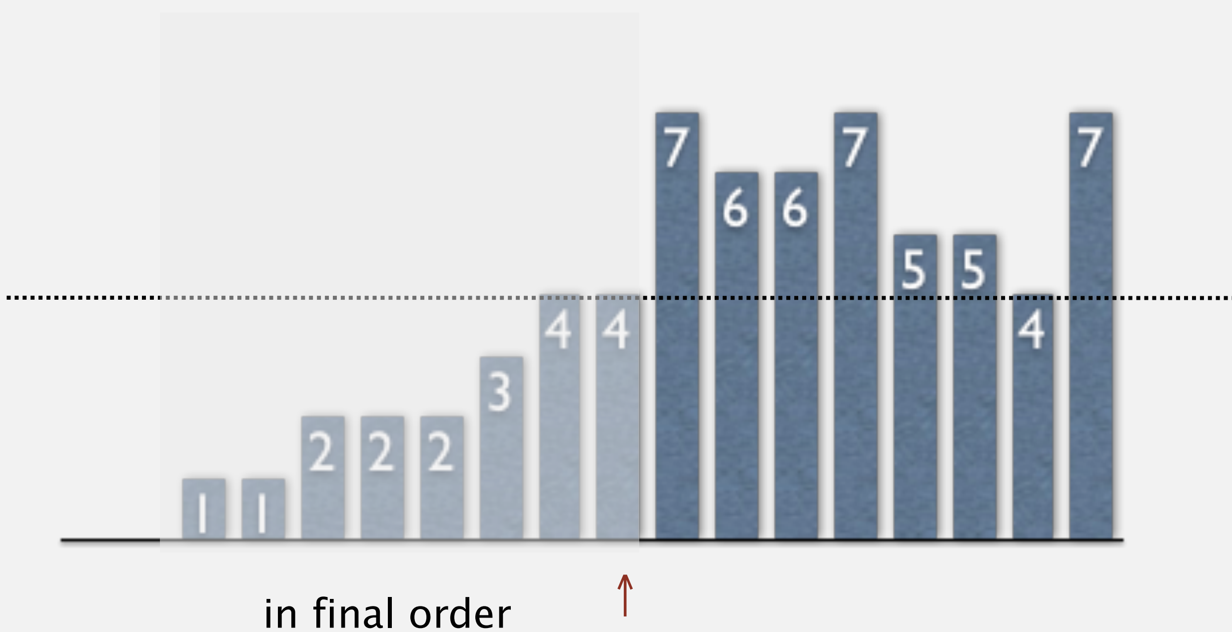
‣ rules of the game
‣ **selection sort**
‣ insertion sort
‣ sorting challenges
‣ shellsort

Selection sort

Algorithm. ↑ scans from left to right.

Invariants.

- Elements to the left of ↑ (including ↑) fixed and in ascending order.
- No element to right of ↑ is smaller than any element to its left.



Selection sort inner loop

To maintain algorithm invariants:

- Move the pointer to the right.

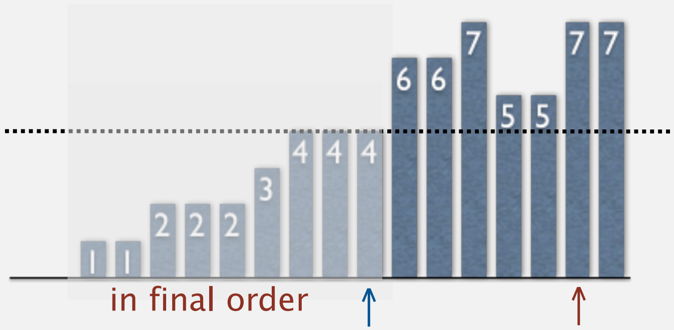
```
i++;
```

- Identify index of minimum item 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: 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

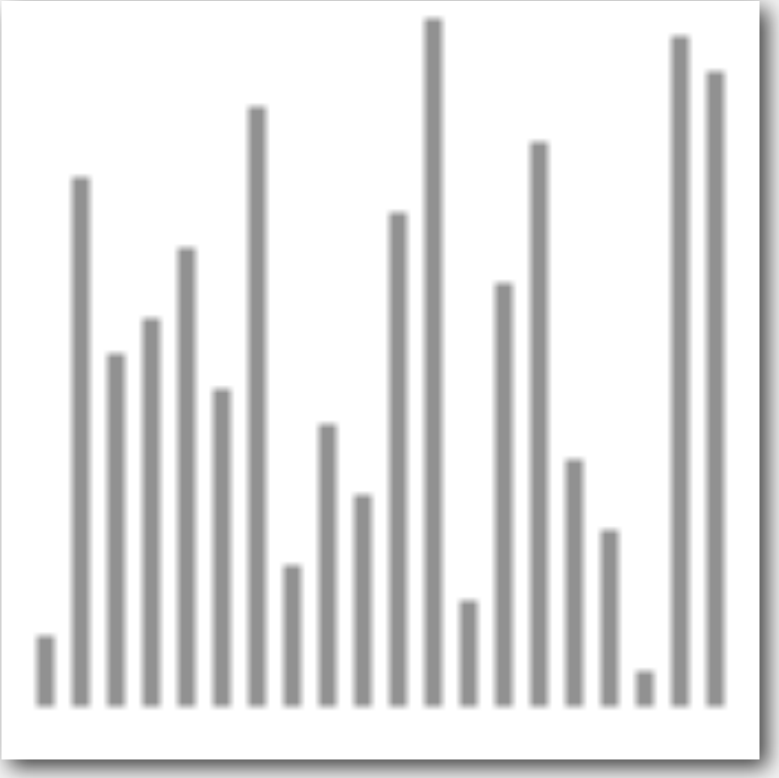
Trace of selection sort (array contents just after each exchange)

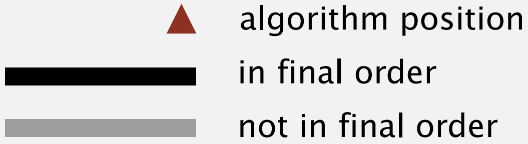
Running time insensitive to input. Quadratic time, even if input array is sorted.

Data movement is minimal. Linear number of exchanges.

Selection sort animations

20 random elements

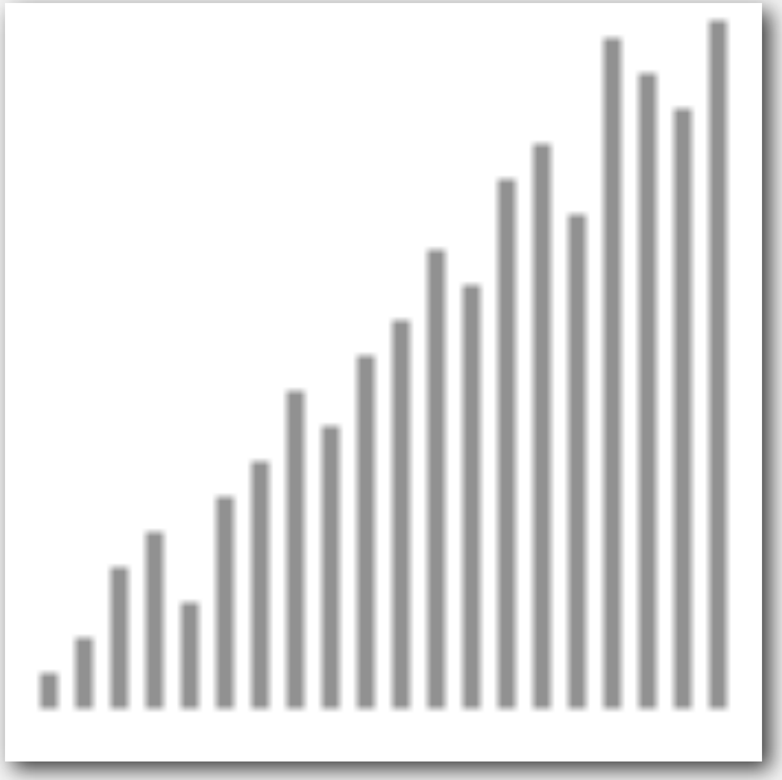




http://www.sorting-algorithms.com/selection-sort

Selection sort animations

20 partially-sorted elements



▲ algorithm position

in final order

not in final order

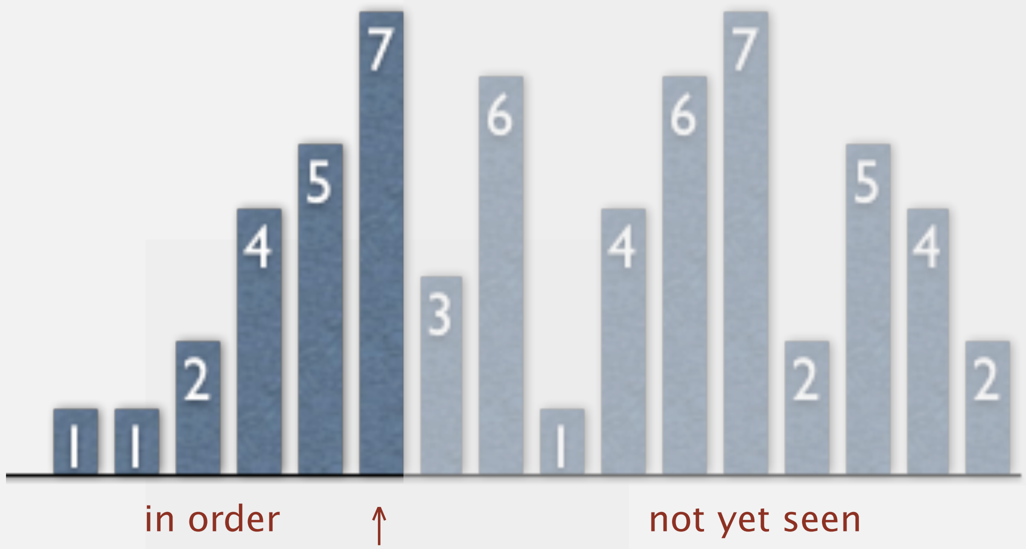
http://www.sorting-algorithms.com/selection-sort

- rules of the game
- selection sort
- **insertion sort**
- sorting challenges
- shellsort

Algorithm. ↑ scans from left to right.

Invariants.

- Elements to the left of ↑ (including ↑) are in ascending order.
- Elements 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 element 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

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 element 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)

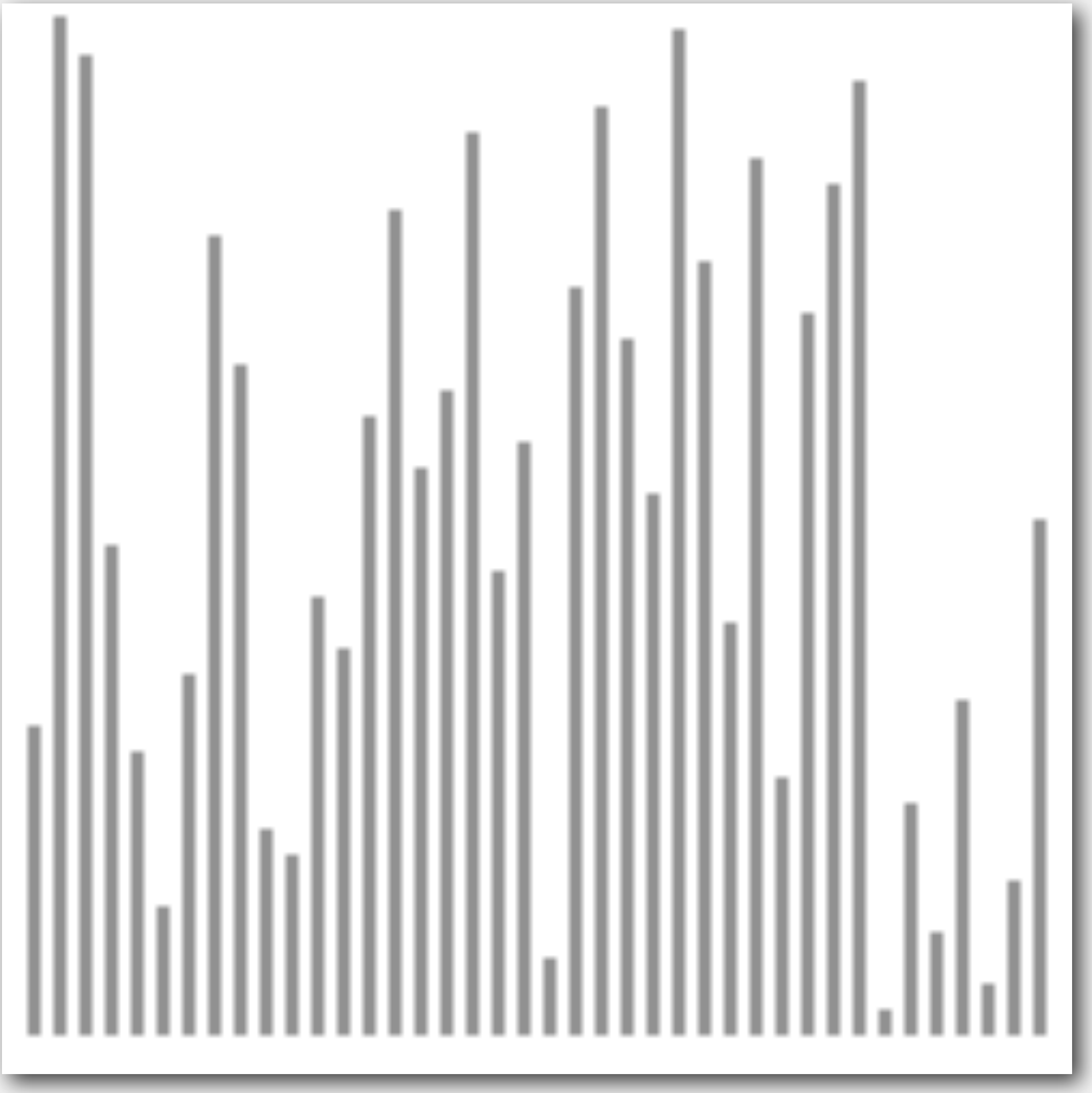
Insertion sort: trace

a[ ]

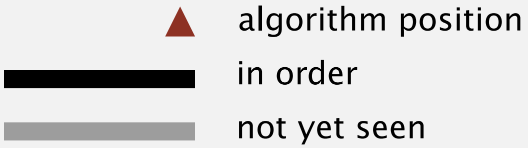
i	j	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34
		A	S	O	M	E	W	H	A	T	L	O	N	G	E	R	I	N	S	E	R	T	I	O	N	S	O	R	T	E	X	A	M	P	L	E
0	0	A	S	O	M	E	W	H	A	T	L	O	N	G	E	R	I	N	S	E	R	T	I	O	N	S	O	R	T	E	X	A	M	P	L	E
1	1	A	S	O	M	E	W	H	A	T	L	O	N	G	E	R	I	N	S	E	R	T	I	O	N	S	O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

40 random elements



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



Insertion sort: best and worst case

Best case. If the array is in ascending order, insertion sort makes $N-1$ compares and 0 exchanges.

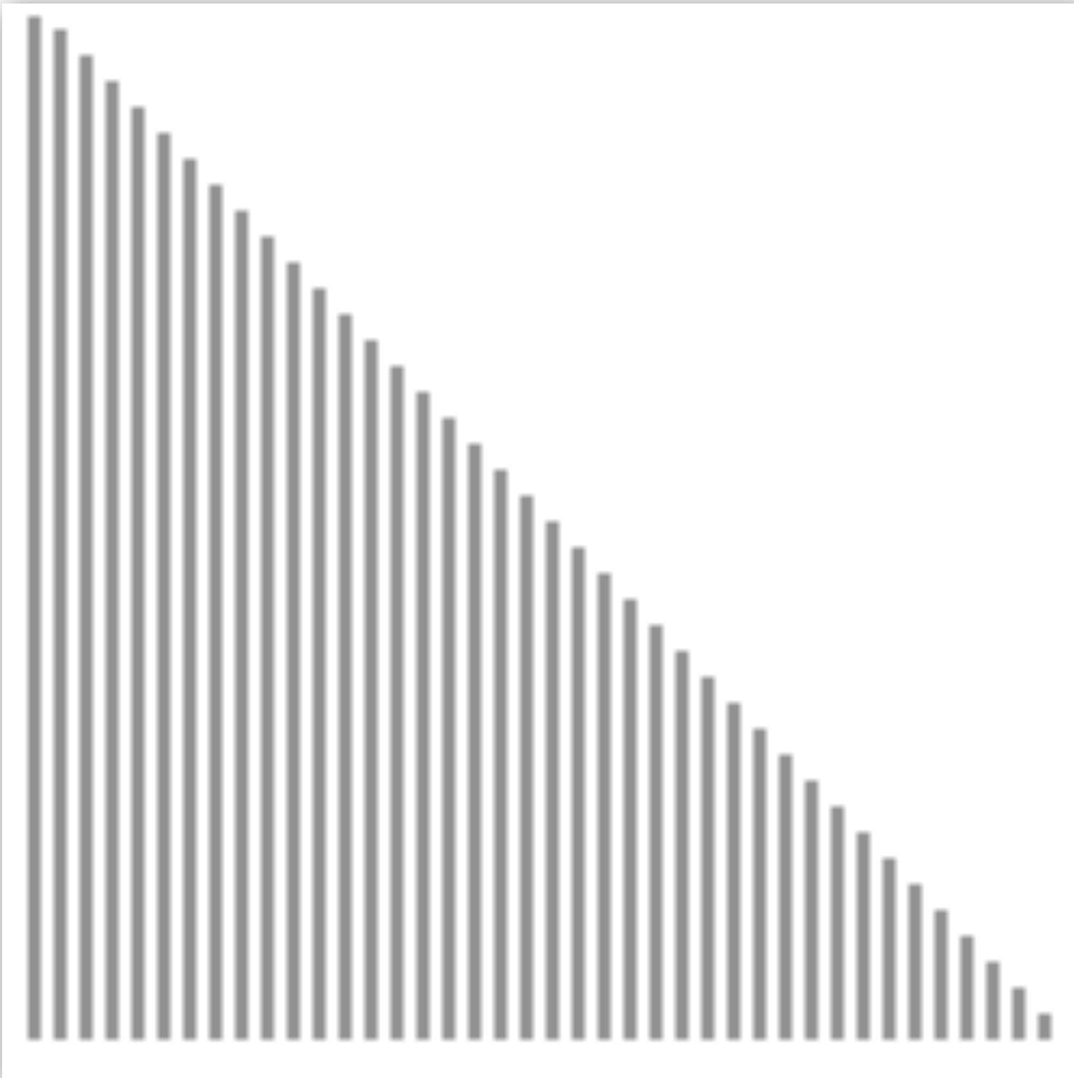
```
A E E L M O P R S T X
```

Worst case. If the array is in descending order (and no duplicates), insertion sort makes $\sim \frac{1}{2} N^2$ compares and $\sim \frac{1}{2} N^2$ exchanges.

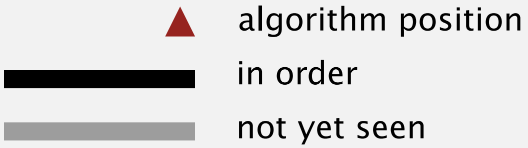
```
X T S R P O M L E E A
```

Insertion sort animation

40 reverse-sorted elements



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



Insertion sort: partially-sorted arrays

Def. An inversion is a pair of keys that are out of order.

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

T-R T-P T-S R-P X-P X-S

(6 inversions)

Def. An array is partially sorted if the number of inversions is $O(N)$.

- Ex 1. A small array appended to a large sorted array.
- Ex 2. An array with only a few elements out of place.

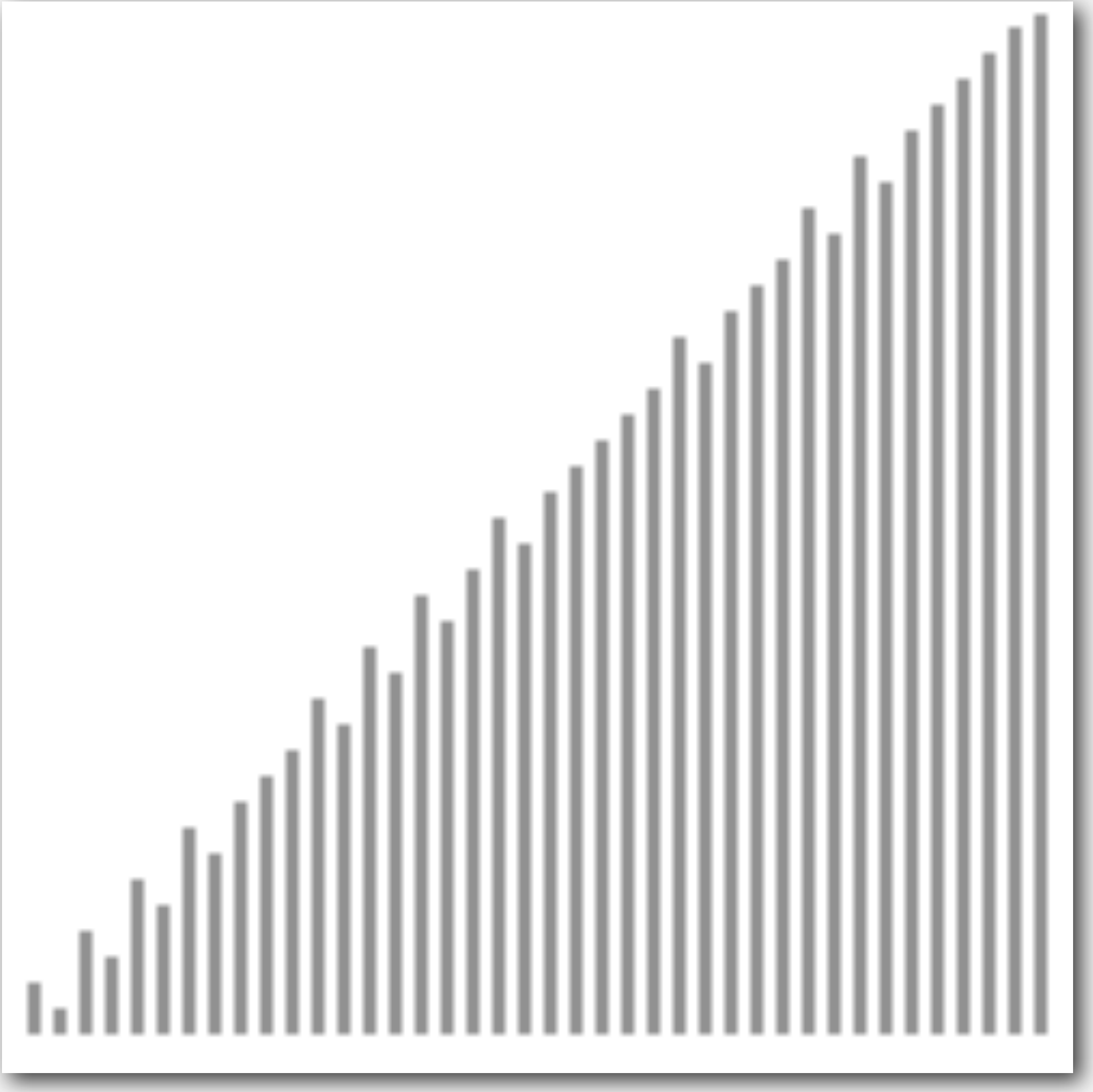
Proposition C. For partially-sorted arrays, insertion sort runs in linear time.

Pf. Number of exchanges equals the number of inversions.

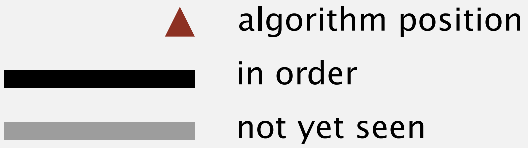
↑ number of compares = exchanges + (N-1)

Insertion sort animation

40 partially-sorted elements



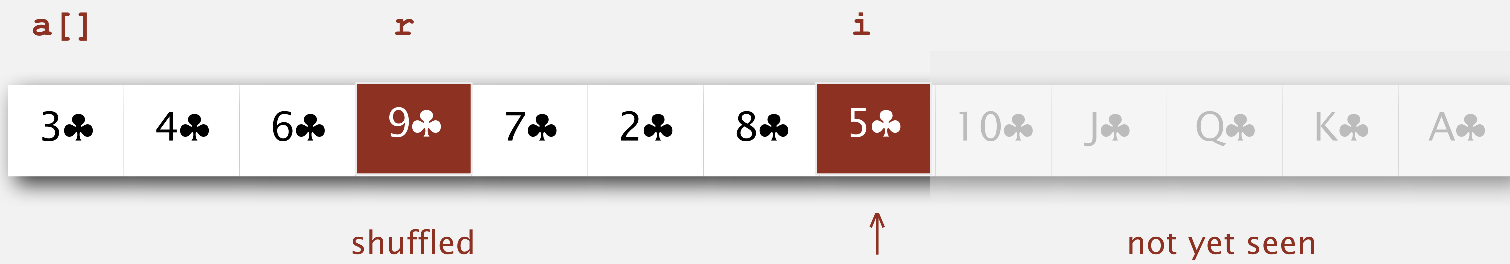
http://www.sorting-algorithms.com/insertion-sort



- rules of the game
- selection sort
- insertion sort
- **sorting challenges**
- shellsort

Knuth shuffle. [Fisher-Yates 1938]

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



Invariants.

- Elements to the left of ↑ (including ↑) are shuffled.
- Elements to the right of ↑ have not yet been seen.

Proposition. Knuth shuffling algorithm produces a uniformly random permutation of the input array in linear time.

assuming integers uniformly at random

Diversion: how to shuffle an array

Knuth shuffle. [Fisher-Yates 1938]

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

```
public class StdRandom
{
   ...
   public static void shuffle(Object[] a)
   {
      int N = a.length;
      for (int i = 0; i < N; i++) {
         int r = i + StdRandom.uniform(1 + i);
         exch(a, i, r);
      }
   }
}
```

between 0 and i

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

appeared last 50% of the time

War story (Microsoft)

Shuffling algorithm by sorting. Assign a random value to each card; sort.

- Uniformly random shuffle, provided no duplicate values.
- Useful in spreadsheets.

Browser	Value
Firefox	0.406782
Chrome	0.134853
Opera	0.590623
Safari	0.343267
IE 8	0.876543

Browser	Value
Chrome	0.134853
Safari	0.343267
Firefox	0.406782
Opera	0.590623
IE 8	0.876543

Microsoft's implementation in Javascript

```
function RandomSort (a,b)
{
   return (0.5 - Math.random());
}
```

browser comparator (should implement a total order)

War story (online poker)

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

How We Learned to Cheat at Online Poker: A Study in Software Security
http://itmanagement.earthweb.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 (← `r := random(51) + 1;`)

Bug 1. Random number `r` never 52 ⇒ 52^{nd} card can't end up in 52^{nd} place.

Bug 2. Shuffle not uniform.

Bug 3. `random()` uses 32-bit seed ⇒ 2^{32} billion possible shuffles.

Bug 4. Seed = milliseconds since midnight ⇒ 86.4 million possible 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
- FIPS 140-2 and the NIST statistical test suite.
- Continuously monitor statistic properties because hardware random-number generators are fragile and fail silently.
- Use an unbiased shuffling algorithm.

Bottom line. Shuffling a deck of cards is hard!

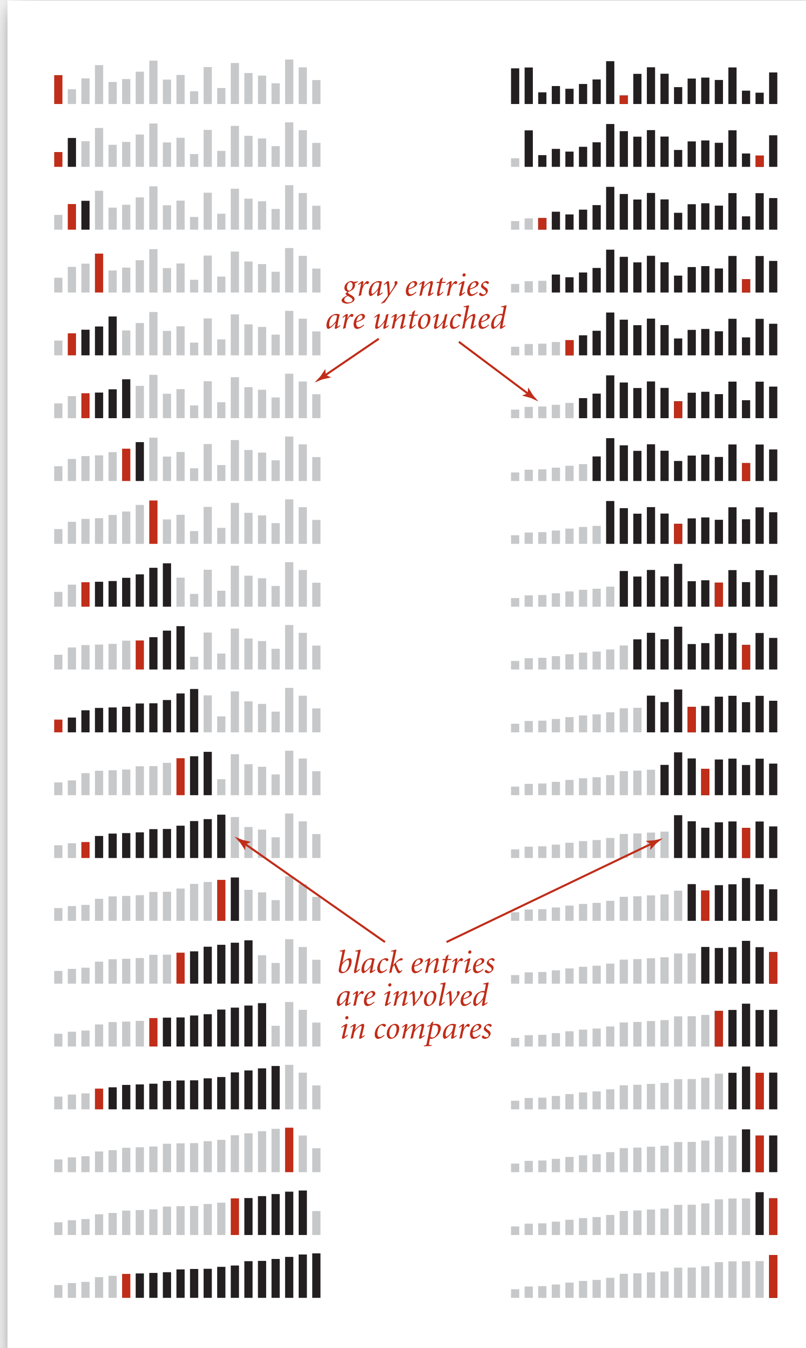
Sorting challenge 0

Input. Array of doubles.

Plot. Data proportional to length.

Name the sorting method.

- Insertion sort.
- Selection sort.



Sorting challenge 1

Problem. Sort a file of huge records with tiny keys.

Ex. Reorganize your MP3 files.

Which sorting method to use?

- System sort.
- Insertion sort.
- Selection sort.

file → record → key →

Fox	1	A	243-456-9091	101 Brown
Quilici	1	C	343-987-5642	32 McCosh
Chen	2	A	884-232-5341	11 Dickinson
Furia	3	A	766-093-9873	22 Brown
Kanaga	3	B	898-122-9643	343 Forbes
Andrews	3	A	874-088-1212	121 Whitman
Rohde	3	A	232-343-5555	115 Holder
Battle	4	C	991-878-4944	308 Blair
Aaron	4	A	664-480-0023	097 Little
Gazsi	4	B	665-303-0266	113 Walker

Sorting challenge 2

Problem. Sort a huge randomly-ordered array of small records.

Ex. Process transaction records for a phone company.

Which sorting method to use?

- System sort.
- Insertion sort.
- Selection sort.

file → record → key →

Fox	1	A	243-456-9091	101 Brown
Quilici	1	C	343-987-5642	32 McCosh
Chen	2	A	884-232-5341	11 Dickinson
Furia	3	A	766-093-9873	22 Brown
Kanaga	3	B	898-122-9643	343 Forbes
Andrews	3	A	874-088-1212	121 Whitman
Rohde	3	A	232-343-5555	115 Holder
Battle	4	C	991-878-4944	308 Blair
Aaron	4	A	664-480-0023	097 Little
Gazsi	4	B	665-303-0266	113 Walker

Sorting challenge 3

Problem. Sort a huge number of tiny arrays (each file is independent).

Ex. Daily customer transaction records.

Which sorting method to use?

- System sort.
- Insertion sort.
- Selection sort.

file →

record →

key →

Fox	1	A	243-456-9091	101 Brown
Quilici	1	C	343-987-5642	32 McCosh
Chen	2	A	884-232-5341	11 Dickinson
Furia	3	A	766-093-9873	22 Brown
Kanaga	3	B	898-122-9643	343 Forbes
Andrews	3	A	874-088-1212	121 Whitman
Rohde	3	A	232-343-5555	115 Holder
Battle	4	C	991-878-4944	308 Blair
Aaron	4	A	664-480-0023	097 Little
Gazsi	4	B	665-303-0266	113 Walker

Sorting challenge 4

Problem. Sort a huge array that is already almost in order.

Ex. Resort a huge sorted database after a few changes.

Which sorting method to use?

- System sort.
- Insertion sort.
- Selection sort.

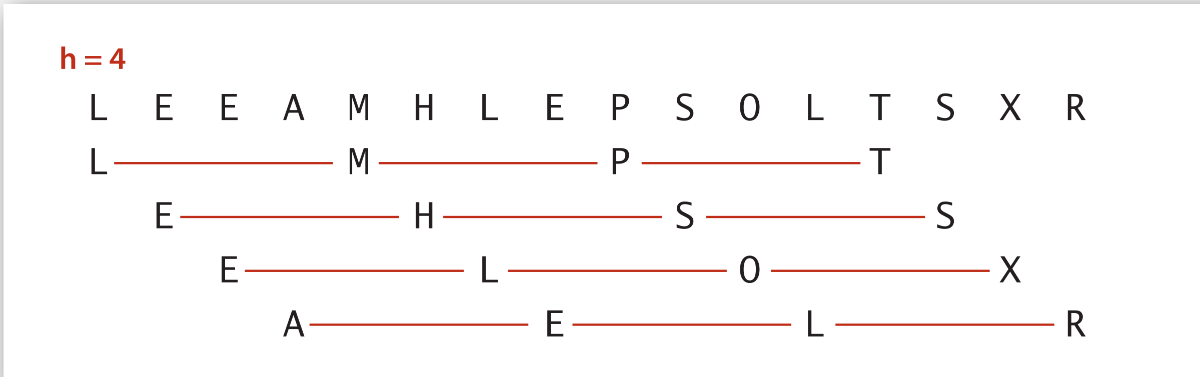
file →	Fox	1	A	243-456-9091	101 Brown
	Quilici	1	C	343-987-5642	32 McCosh
	Chen	2	A	884-232-5341	11 Dickinson
	Furia	3	A	766-093-9873	22 Brown
	Kanaga	3	B	898-122-9643	343 Forbes
record →	Andrews	3	A	874-088-1212	121 Whitman
	Rohde	3	A	232-343-5555	115 Holder
	Battle	4	C	991-878-4944	308 Blair
key →	Aaron	4	A	664-480-0023	097 Little
	Gazsi	4	B	665-303-0266	113 Walker

‣ rules of the game
‣ selection sort
‣ insertion sort
‣ animations
‣ **shellsort**

Shellsort overview

Idea. Move elements more than one position at a time by *h-sorting* the array.

an h-sorted array is h interleaved sorted subsequences



Shellsort. [Shell 1959] *h-sort* the 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

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 $\Rightarrow$ small subarray.
- Small increments $\Rightarrow$ 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

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

7-sort

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
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 for the bored. Proof this fact—it's more subtle than you'd think!

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.

merging of $(9 \times 4^i) - (9 \times 2^i) + 1$ and $4^i - (3 \times 2^i) + 1$

Sedgewick. 1, 5, 19, 41, 109, 209, 505, 929, 2161, 3905, ...

Good. Tough to beat in empirical studies.

Interested in learning more?

- See Section 6.8 of Algs, 3rd edition or Volume 3 of Knuth for details.
- Do a JP on the topic.

Shellsort: Java implementation

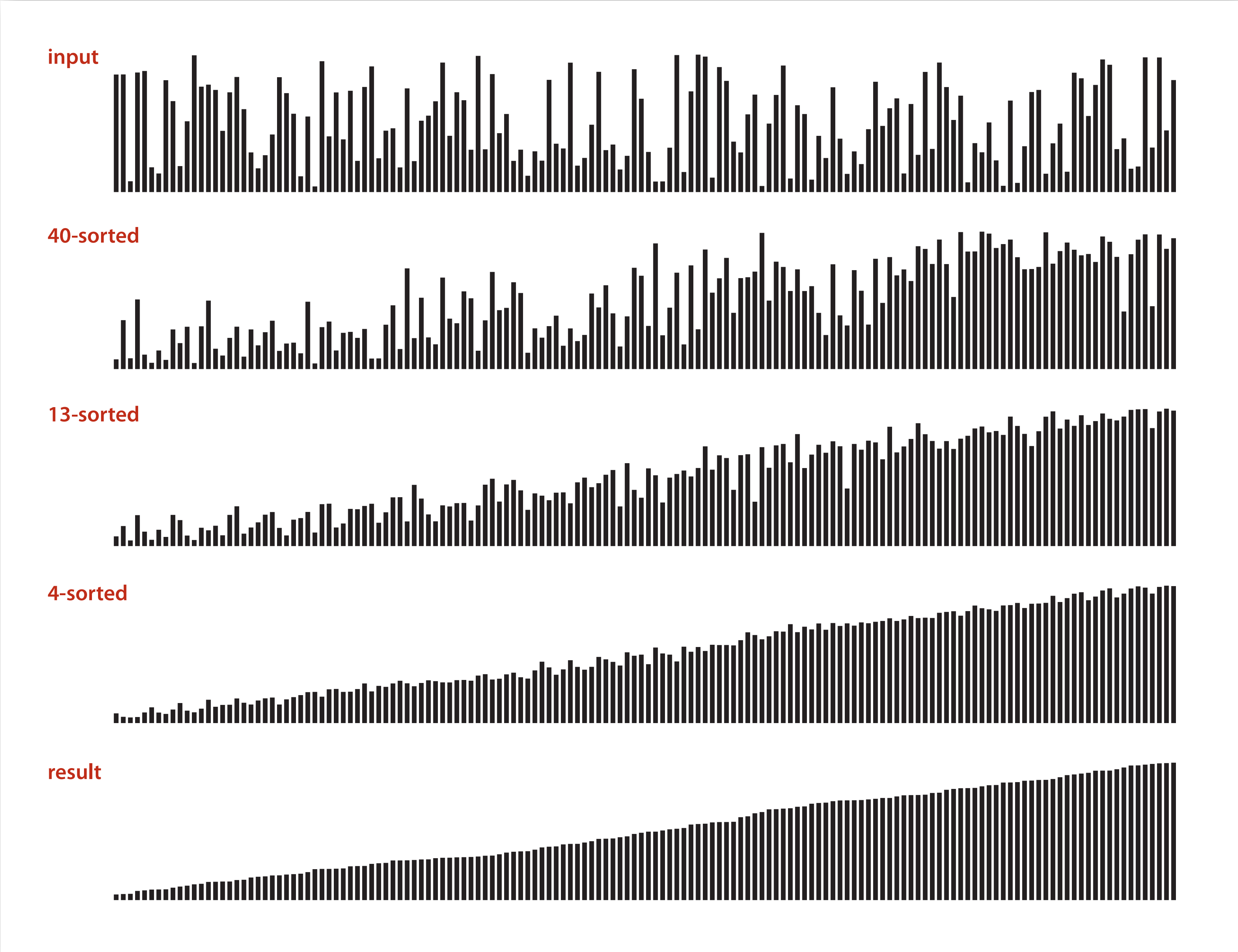
```
public class Shell
{
   public static void sort(Comparable[] a)
   {
      int N = a.length;

      int h = 1;
      while (h < N/3) h = 3*h + 1; // 1, 4, 13, 40, 121, 364, 1093, ...

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

3x+1 increment sequence

insertion sort

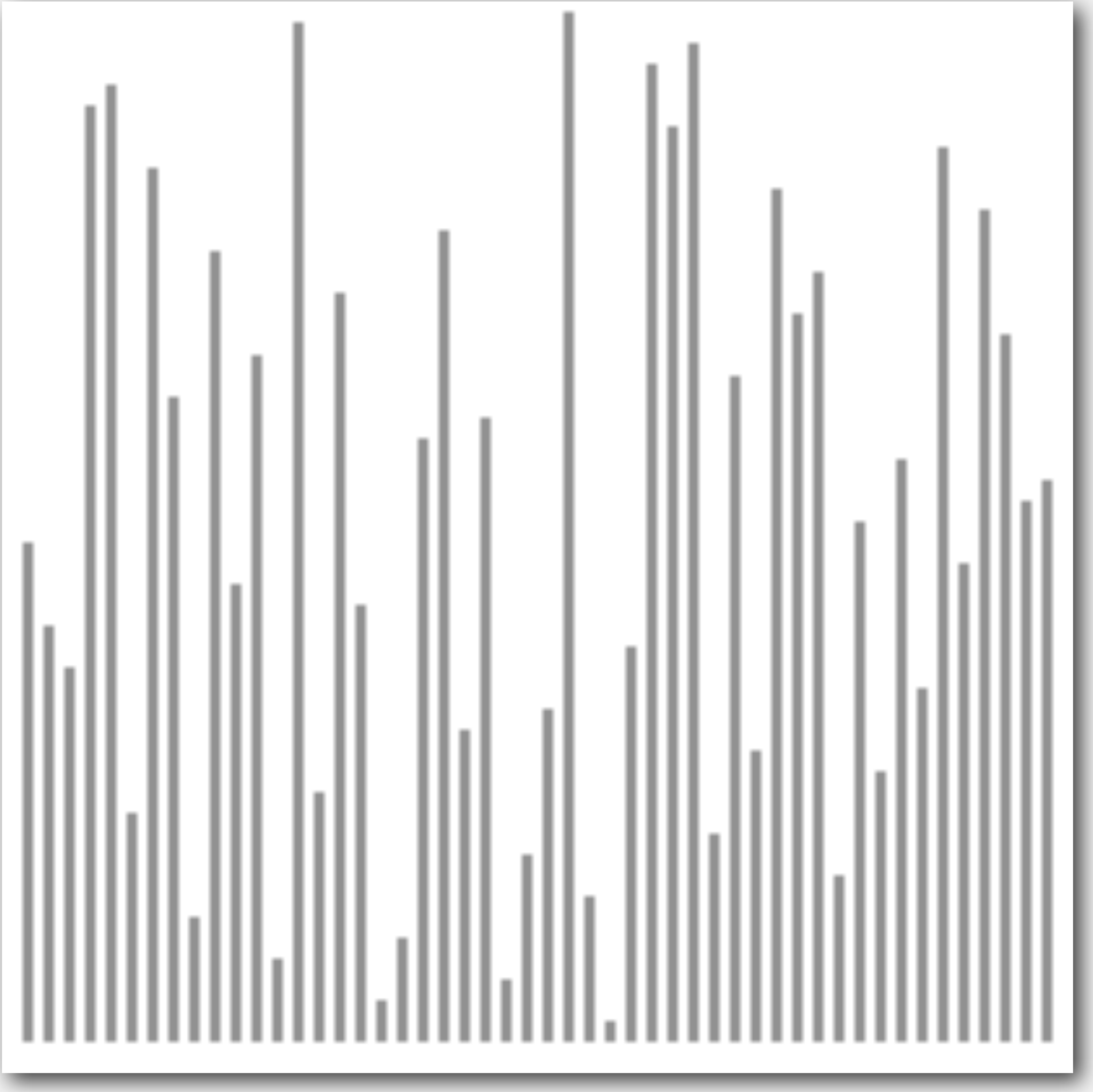
move to next increment

Visual trace of shellsort

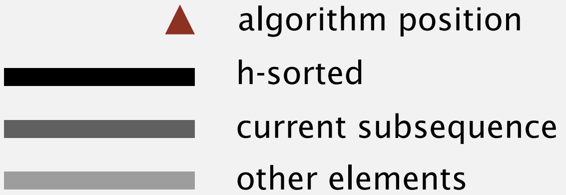


Shellsort animation

50 random elements

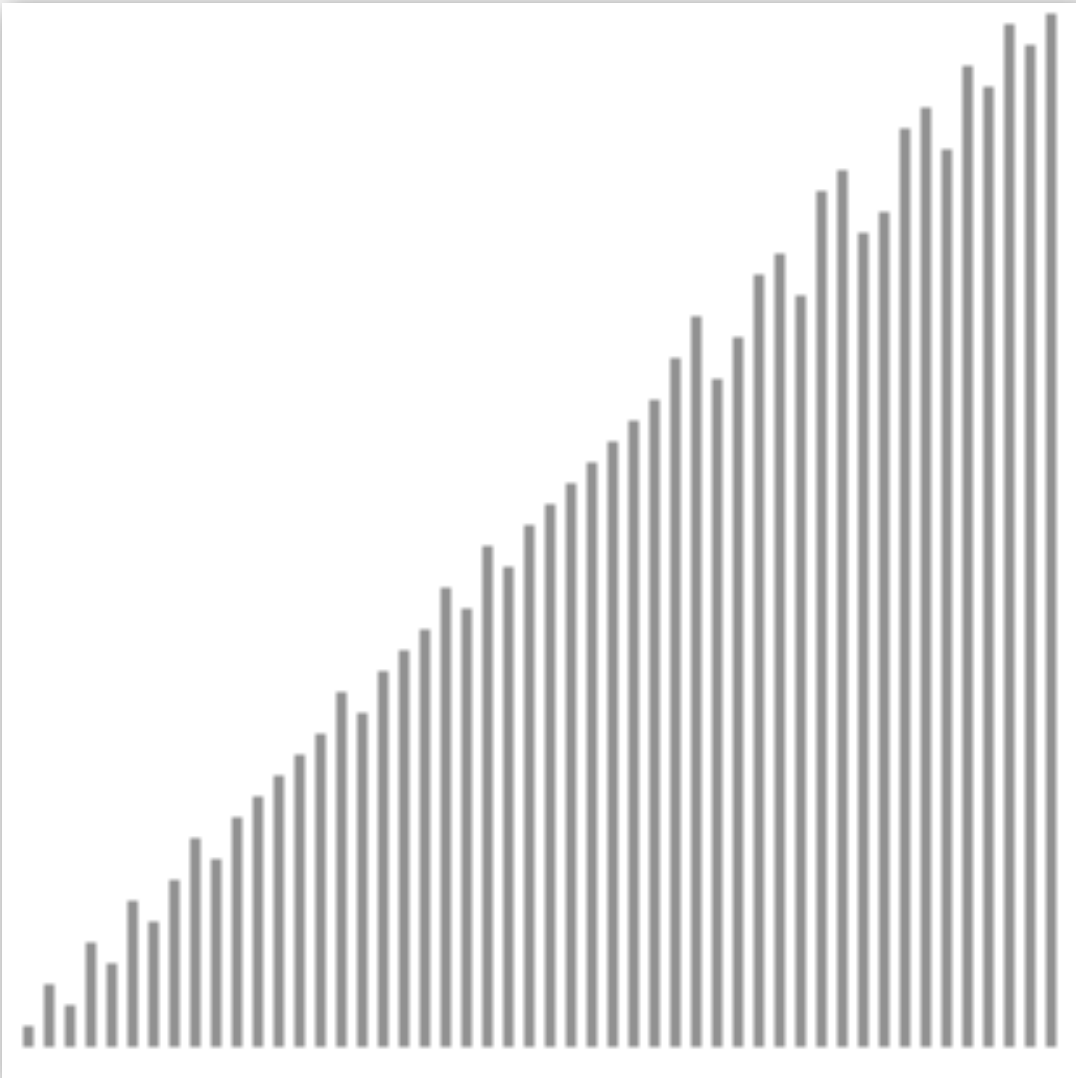


http://www.sorting-algorithms.com/shell-sort



Shellsort animation

50 partially-sorted elements



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

▲ algorithm position

h-sorted

current subsequence

other elements

Shellsort: analysis

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

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

N	compares	$N^{1.289}$	2.5 N lg N
5,000	93	58	106
10,000	209	143	230
20,000	467	349	495
40,000	1022	855	1059
80,000	2266	2089	2257

measured in thousands

Remark. Accurate model has not yet been discovered (!)

Why are we interested in shellsort?

Example of simple idea leading to substantial performance gains.

Useful in practice.

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

Simple algorithm, nontrivial performance, interesting questions.

- Asymptotic growth rate?
- Best sequence of increments? ← open problem: find a better increment sequence
- Average-case performance?

Lesson. Some good algorithms are still waiting discovery.