http://algs4.cs.princeton.edu

2.1 ELEMENTARY SORTS

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

http://algs4.cs.princeton.edu

2.1 Elementary Sorts

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

Sorting problem

Ex. Student records in a university.

item →

key →

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 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

Sample sort client 1

Goal. Sort any type of data.

Ex 1. Sort random real numbers in ascending order.

seems artificial, but 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 from file in alphabetical order. 字母

```
public class StringSorter
{
   public static void main(String[] args)
   {
      String[] a = In.readStrings(args[0]);
      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
```

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
```

Callbacks 回调

Goal. Sort any type of data.

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 back object's `compareTo()` method as needed.

Implementing callbacks. 有很多实现回调函数的方法，和具体的编程语言有关，不同的语言有不同的机制。核心思想是将函数作为实参传递给其他函数。

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

Callbacks: roadmap 路线图

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;
   }
}
```

在函数声明的时候，要求参数必须是 Comparable 类型数组，这意味着数组中的对象需要实现 Comparable 接口，或者说对象有 compareTo 方法。

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;
}
```

关键在于排序实现与数据类型无关，具体的比较由 Comparable 接口处理，不同类型的 Comparable 数组最终会以相同的方式排序，依赖于接口机制，回调实际的被排序对象类型的 compareTo 代码

key point: no dependence on File data type

Total order 全序

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

反对称性

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

总体性：要么 v≤w 要么 w≤v 要么两者兼而有之

Ex.

- Standard order for natural and real numbers.

按发生时间顺序排列的

- Chronological order for dates or times.
- Alphabetical order for strings.
- ...

an intransitive relation

非传递性关系示例：石头剪刀布

violates totality: `(Double.NaN <= Double.NaN)` is `false`

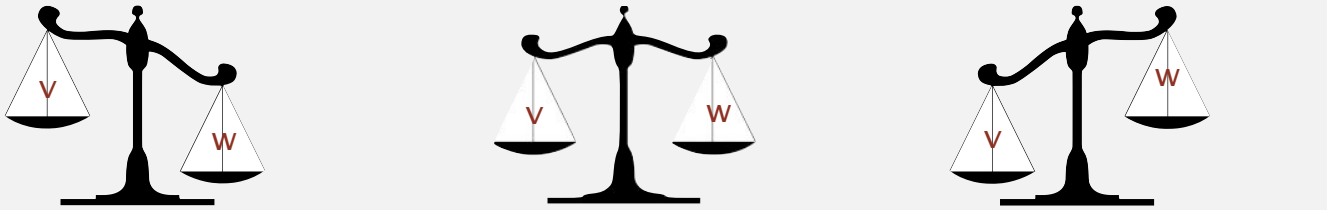
Surprising but true. The `<=` operator for `double` is not a total order. (!)

Comparable API

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

- Is 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`).

不兼容



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

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;
}
```

Testing

Goal. 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 the array?

A.

http://algs4.cs.princeton.edu

2.1 Elementary Sorts

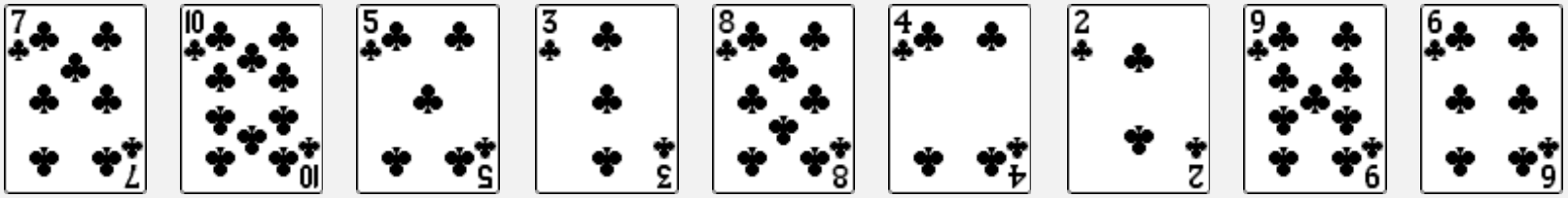
- *rules of the game*
- *selection sort*
- *insertion sort*
- *shellsort*
- *shuffling*
- *convex hull*

2.1 Elementary Sorts

- *rules of the game*
- *selection sort* 选择排序
- *insertion sort*
- *shellsort*
- *shuffling*
- *convex hull*

Selection sort demo

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



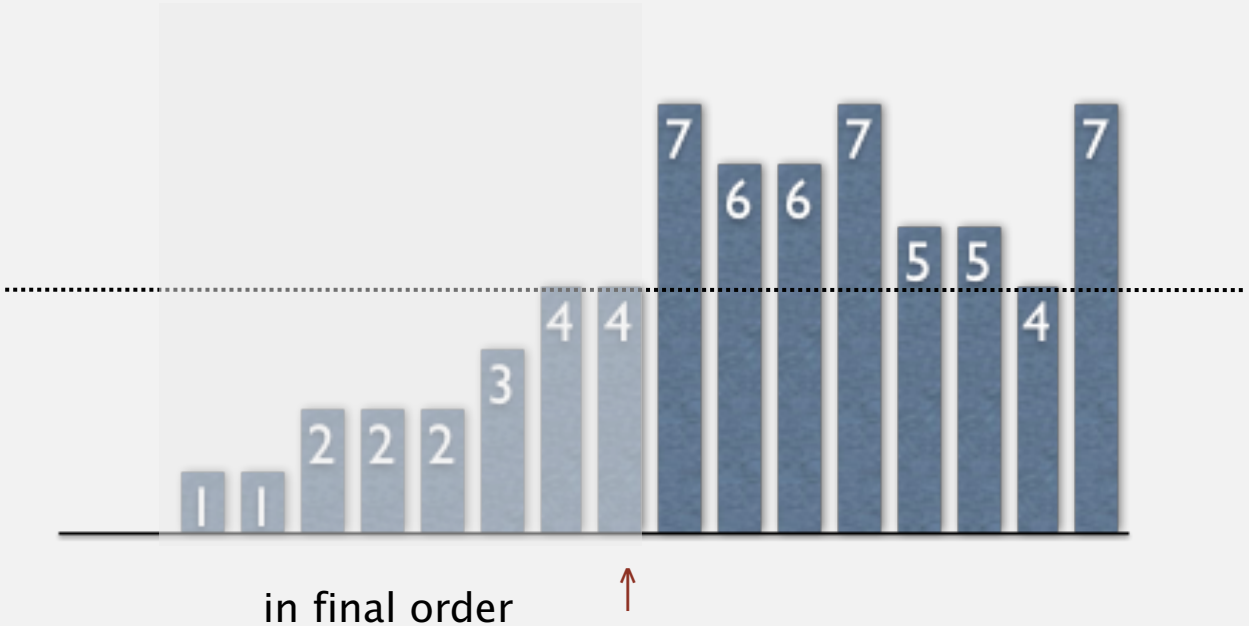
initial

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 ↑.



Selection sort inner loop

To maintain algorithm invariants:

- Move the pointer to the right.

```
i++;
```

移动之后，不变式 1 依然正确，
但不变式 2 可能被破坏了。

- 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: mathematical analysis

Proposition. Selection sort uses $(N-1)+(N-2)+\ \ldots\ +1+0\ \sim\ N^2/2$ 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)

不敏感

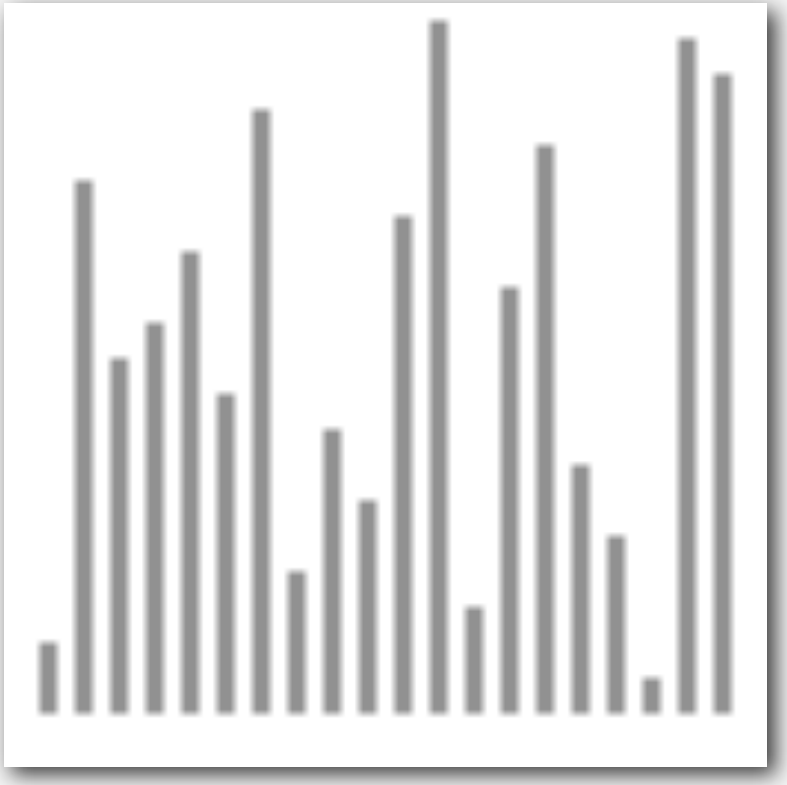
Running time insensitive to input. Quadratic time, even if input is sorted.

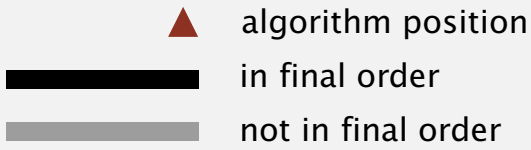
Data movement is minimal. Linear number of exchanges.

数据移动最少

Selection sort: animations

20 random items

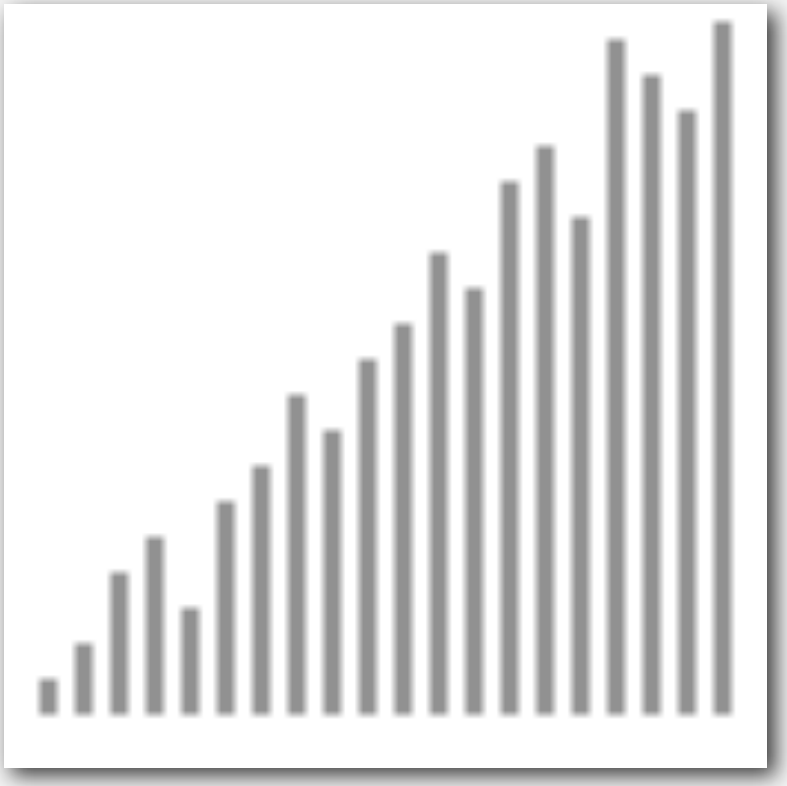


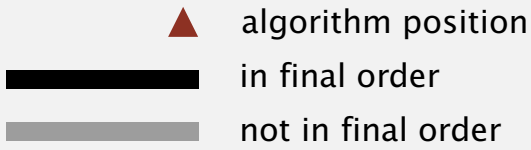


http://www.sorting-algorithms.com/selection-sort

Selection sort: animations

20 partially-sorted items





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

http://algs4.cs.princeton.edu

2.1 Elementary Sorts

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

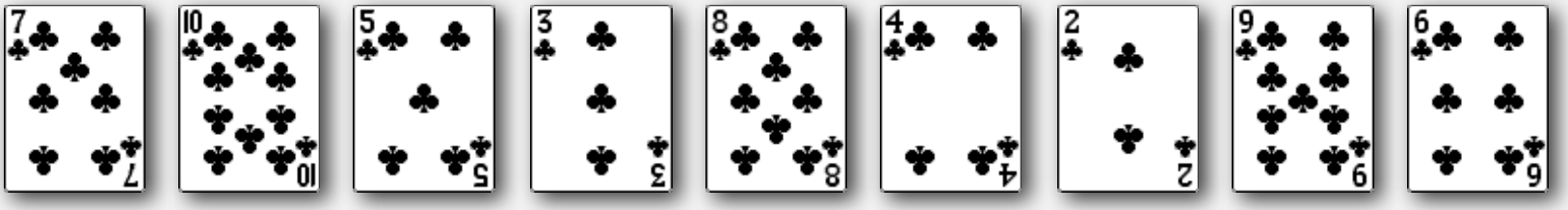
http://algs4.cs.princeton.edu

2.1 Elementary Sorts

- *rules of the game*
- *selection sort*
- *insertion sort* 插入排序
- *shellsort*
- *shuffling*
- *convex hull*

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++;
```

这里是不变式 1 被破坏，
不变式 2 依然正确。

- 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 */  }
}
```

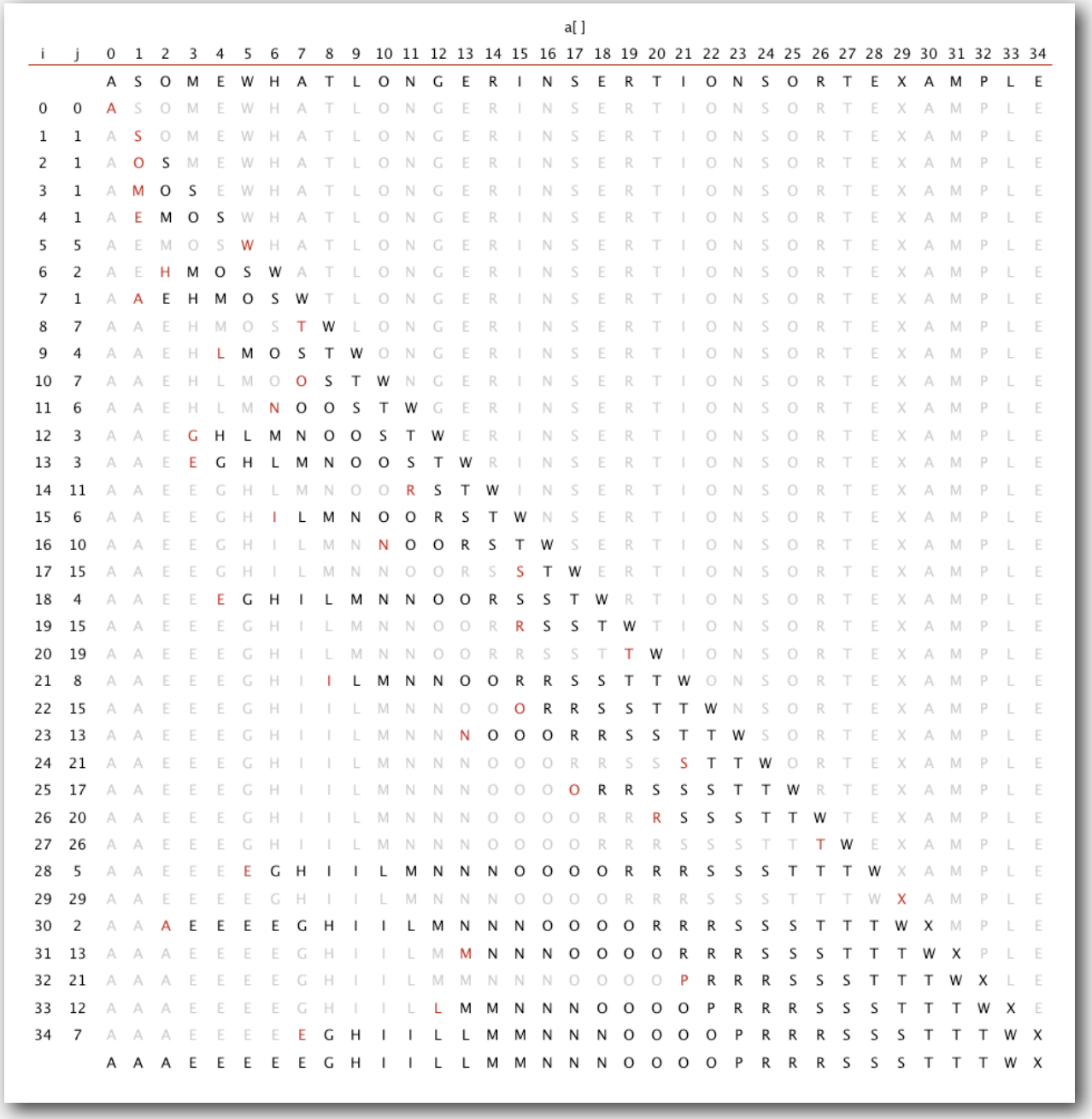
Insertion sort: mathematical analysis

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.

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

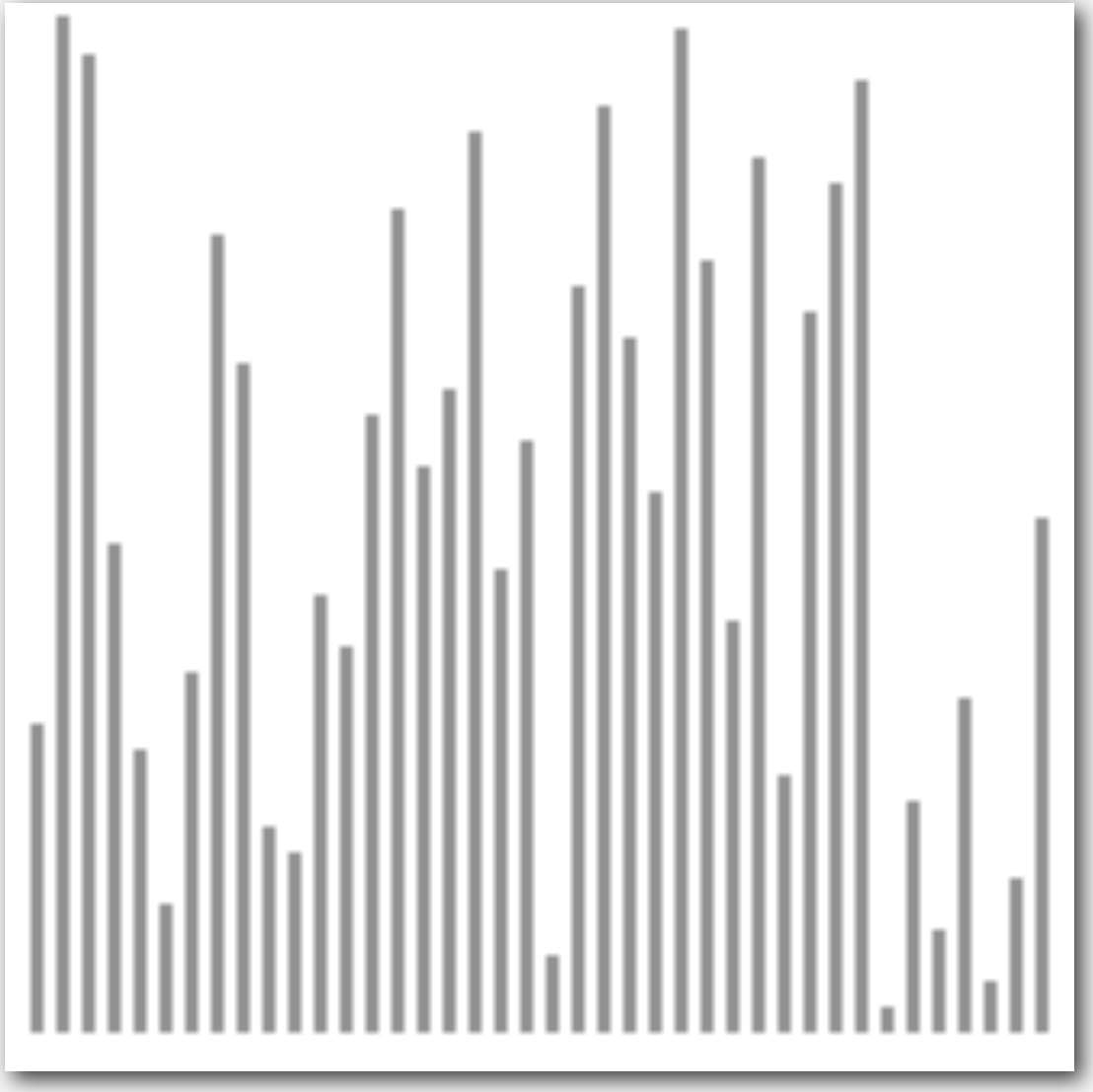
Insertion sort: trace



		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	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 items



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

algorithm position

in order

not yet seen

Insertion sort: best and worst case

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

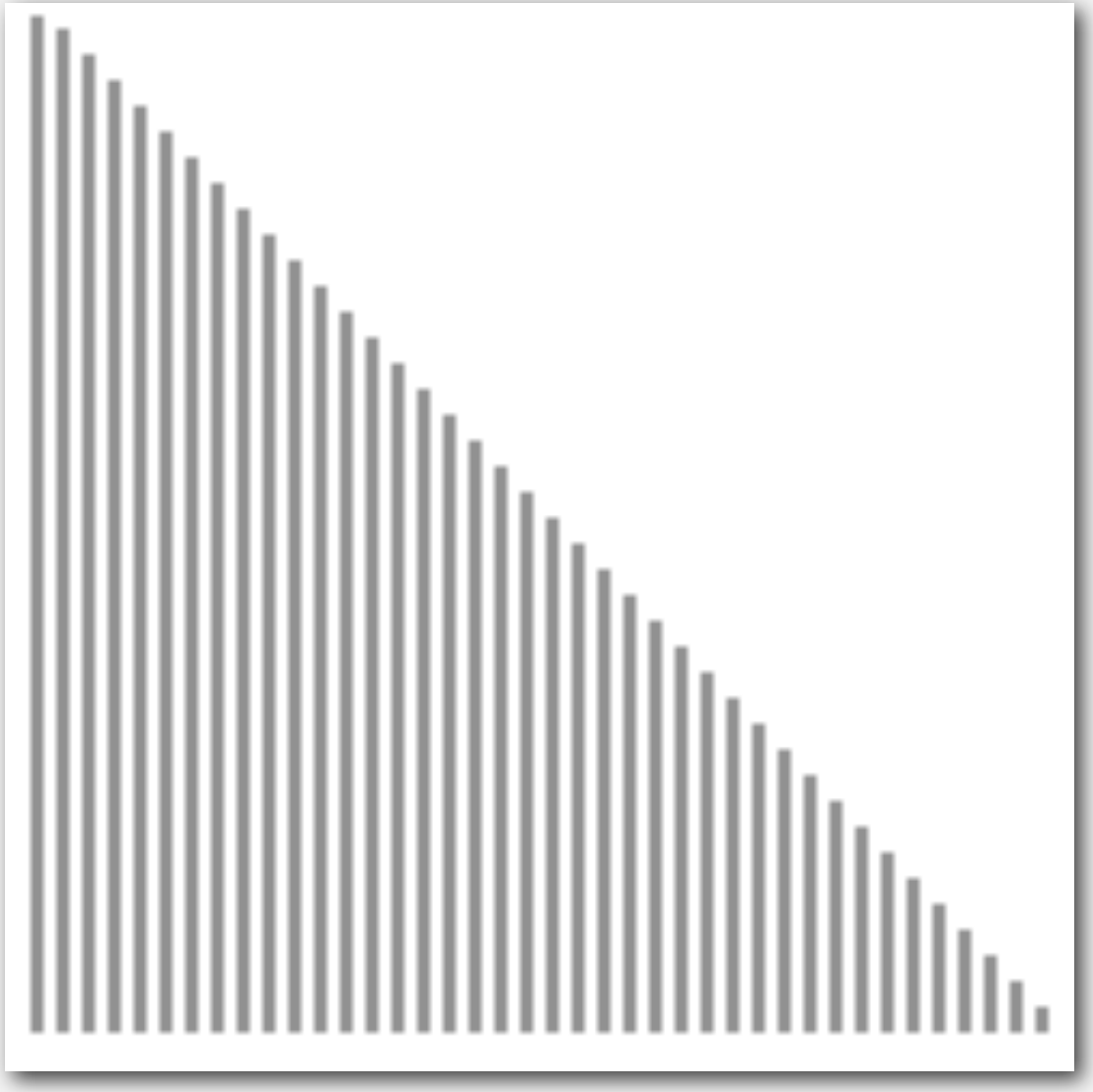
```
A E E L M O P R S T X
```

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.

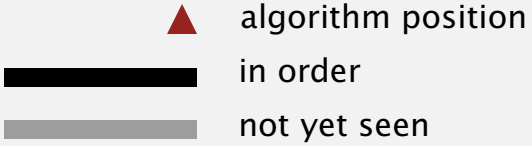
```
X T S R P O M L E E A
```

Insertion sort: animation

40 reverse-sorted items



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



部分有序

Insertion sort: partially-sorted arrays

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

逆序

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 $\leq c\,N$.

- Ex 1. A subarray of size 10 appended to a sorted subarray of size N.
- Ex 2. An array of size N with only 10 entries out of place.

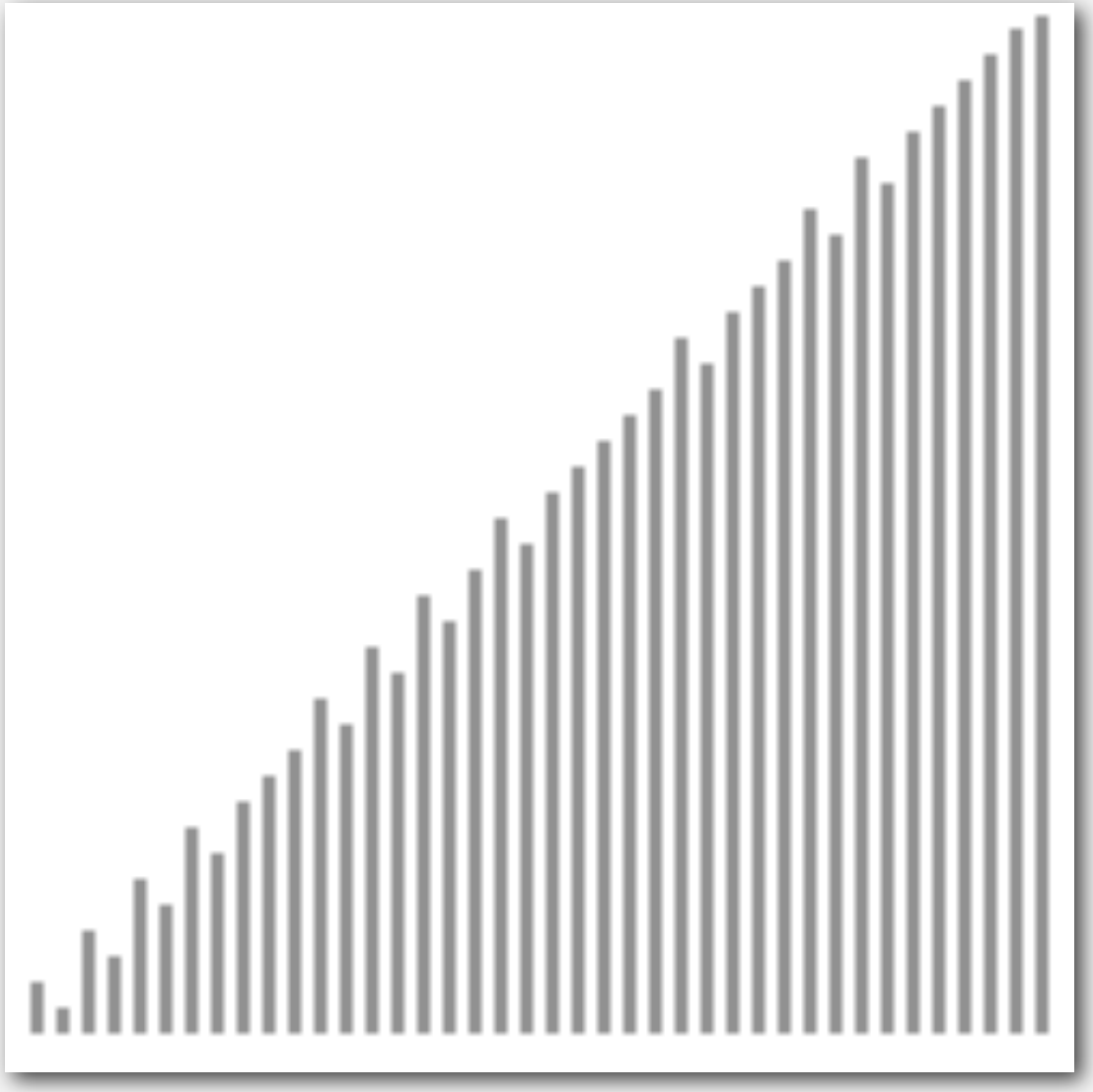
Proposition. For partially-sorted arrays, insertion sort runs in linear time.

Pf. Number of exchanges equals the number of inversions.

number of compares = exchanges + (N – 1) 有序情况下也要比较 N-1 次

Insertion sort: animation

40 partially-sorted items



algorithm position

in order

not yet seen

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

http://algs4.cs.princeton.edu

2.1 Elementary Sorts

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

http://algs4.cs.princeton.edu

2.1 Elementary Sorts

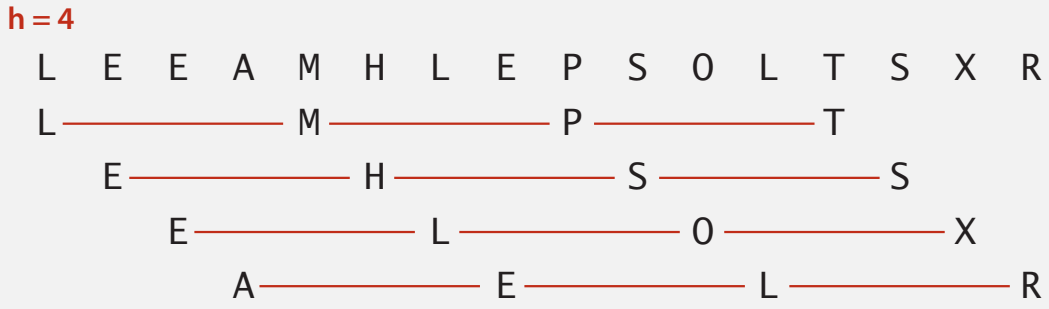
- *rules of the game*
- *selection sort*
- *insertion sort*
- *shellsort* 希尔排序
- *shuffling*
- *convex hull*

Shellsort overview

对数组进行 h-排序：每次要移动数组中条目时，移动跨越多个位置

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

一个完成 h-排序的数组可以分为 h 个交叉的已排序好的子序列
an h-sorted array is h interleaved sorted subsequences



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

希尔排序：不断地用递减的 h 值对数组进行 h-排序

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

希尔排序算法的思想在于，数组每次排序都基于前面进行过的排序，虽然看起来排序了多次，但每次排序只需要进行少数几次比较和交换

h-sorting

如何进行 h-排序：使用步长为 h 的插入排序！

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
```

为什么选择插入排序进行 h-排序呢？这基于我们对插入排序的原理的理解。
1.如果增量 h 很大，那么进行排序的子数组长度就很小，包括插入排序在内的任何排序方法都会有很好的性能
2.如果增量 h 很小，那么因为之前我们已经用更大的 h 值进行了 h-排序，数组是部分有序的，所以插入排序会很快
使用选择排序进行 h-排序就不行，因为无论序列是什么，选择排序总需要平方时间

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 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: intuition

g-有序的数组经过 h-排序以后依然是 g-有序的

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

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

巧妙

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

增量序列

Shellsort: which increment sequence to use?

Powers of two. 1, 2, 4, 8, 16, 32, ...

No. 2的幂并不是一个好的选择，因为这样的增量序列在进行 1-排序前不会将偶数位置的元素和奇数位置的元素进行比较，也即至少有一半的元素（奇数位置元素）是保持原封不动的，这意味着性能就会很差

Powers of two minus one. 1, 3, 7, 15, 31, 63, ...

Maybe. 希尔自己的想法是使用2的幂减1序列，这是行得通的，比如 8-3=5,5-3=2，8,5,2，这样的序列显然有进行奇数位置和偶数位置的元素比较

→ 3x + 1. 1, 4, 13, 40, 121, 364, ...

OK. Easy to compute. Knuth提出的一种增量序列

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

Good. Tough to beat in empirical studies.

在实证研究中很难击败

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

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

我们使用希尔排序的时候，我们首先找到小于待排序数组长度最大的增量值，然后依照递减的增量值进行排序

Shellsort: Java implementation

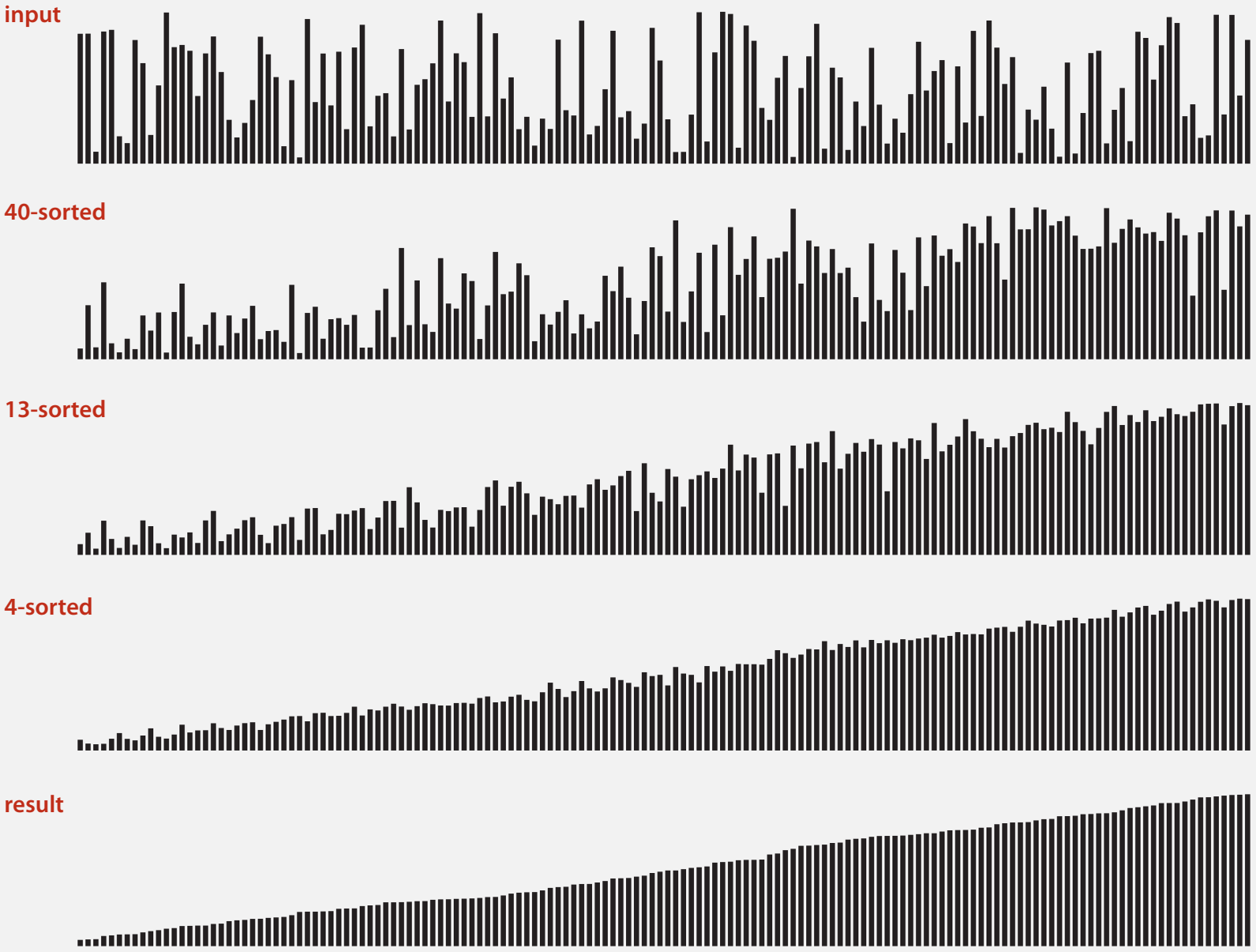
```
public class Shell
{
   public static void sort(Comparable[] a)
   {
      int N = a.length;

      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

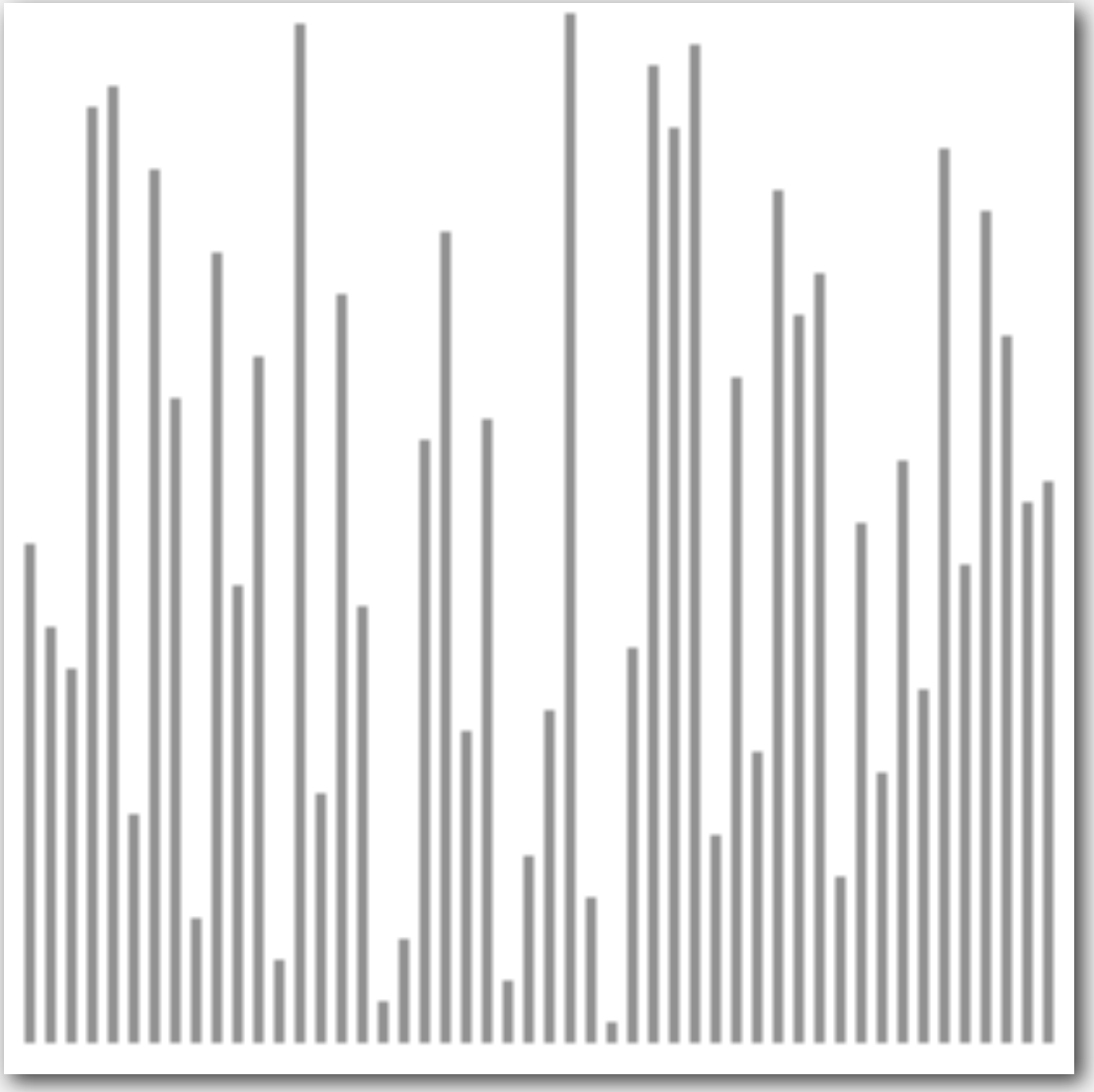
move to next increment

Shellsort: visual trace

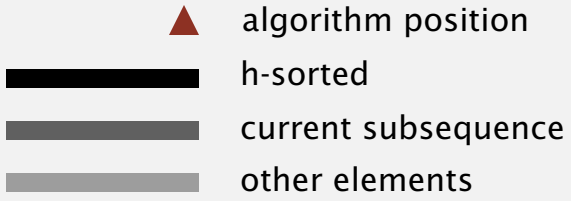


Shellsort: animation

50 random items

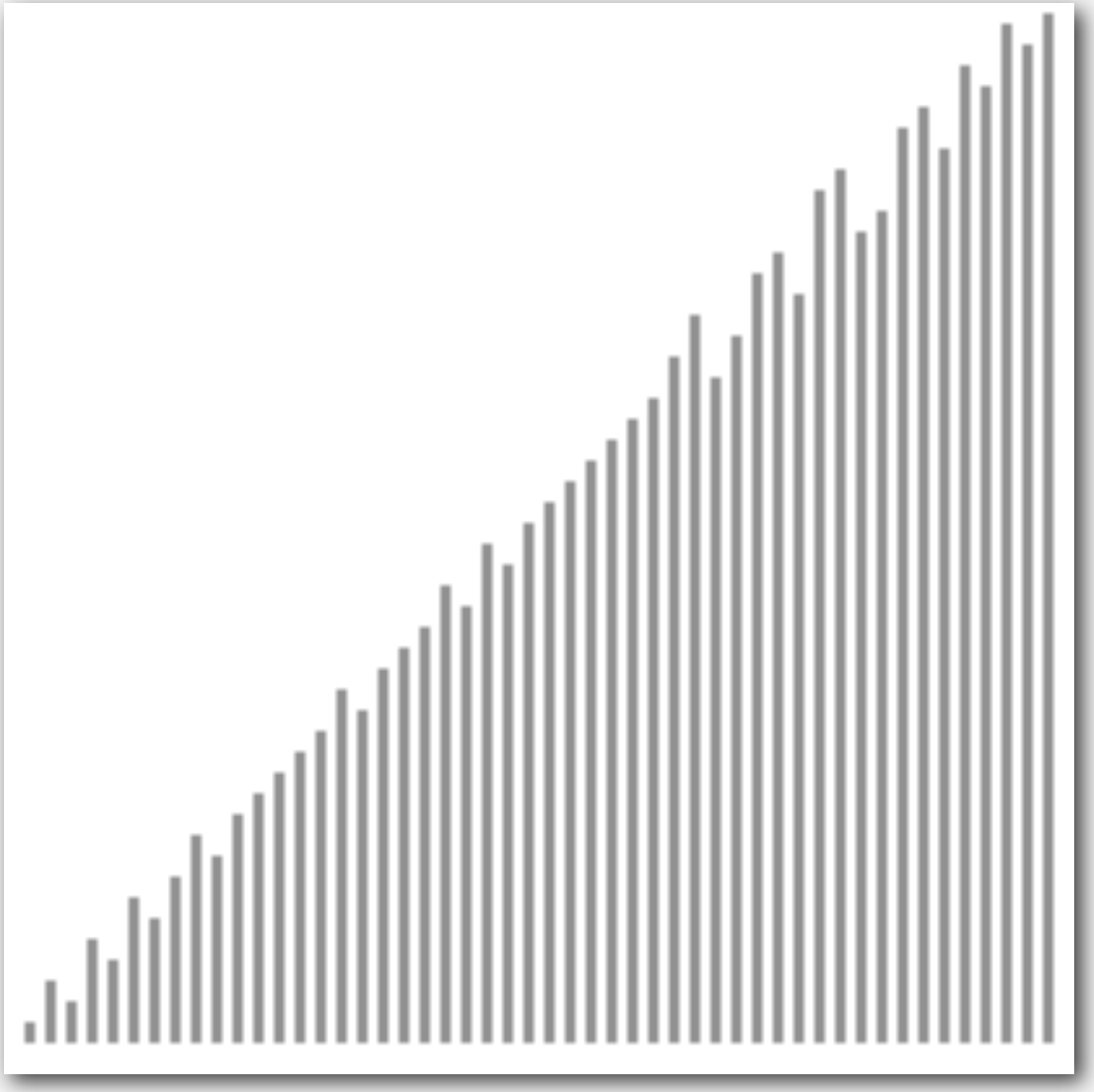


http://www.sorting-algorithms.com/shell-sort

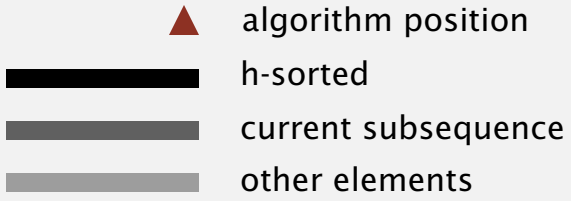


Shellsort: animation

50 partially-sorted items 从 50 个随机项到 50 个部分有序项



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



Shellsort: analysis

Proposition. The worst-case number of compares used by shellsort with the 3x+1 increments is $O(N^{3/2})$. 实际应用中要比 O(N^1.5)要小得多

Property. 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

问题是，没有精确的模型能够描述使用任何一种有效的增量序列的希尔排序需要进行比较的次数。
看起来似乎应该是N乘以增量次数的若干倍，近似地，也即NlogN的若干倍，但是没人能够构建准确的模型对使用有效的增量序列的希尔排序去证明这一点

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

Why are we interested in shellsort?

大量的/可观的

Example of simple idea leading to substantial performance gains.

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). 占用的空间
- Hardware sort prototype. 硬件排序原型

uClibc

非凡的

Simple algorithm, nontrivial performance, interesting questions.

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

Lesson. Some good algorithms are still waiting discovery.

http://algs4.cs.princeton.edu

2.1 Elementary Sorts

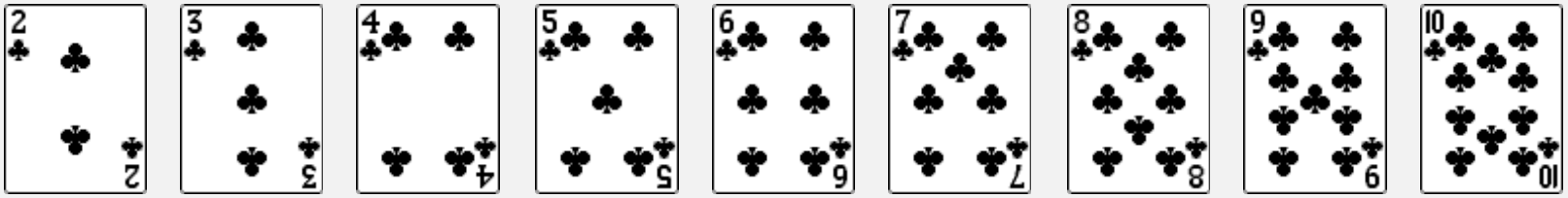
- *rules of the game*
- *selection sort*
- *insertion sort*
- *shellsort*
- *shuffling*
- *convex hull*

2.1 Elementary Sorts

- *rules of the game*
- *selection sort*
- *insertion sort*
- *shellsort*
- *shuffling* 洗牌
- *convex hull*

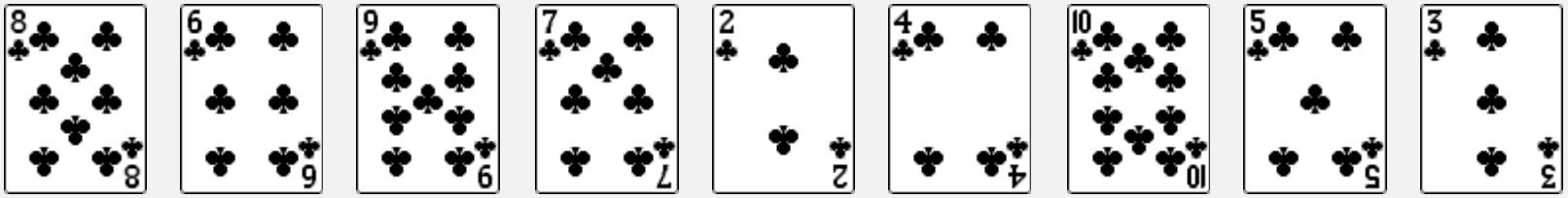
How to shuffle an array

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



How to shuffle an array

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

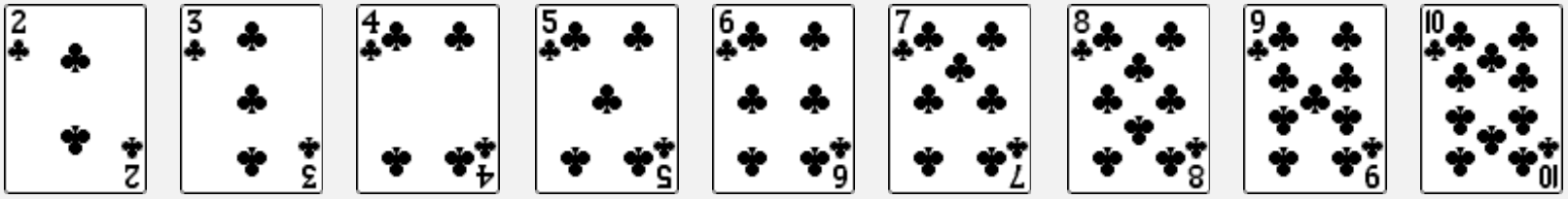


Shuffle sort 虽然看起来排序似乎是洗牌的反过程，但我们确实可以利用排序来进行洗牌

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

useful for shuffling columns in a spreadsheet

可用于在电子表格中混洗列

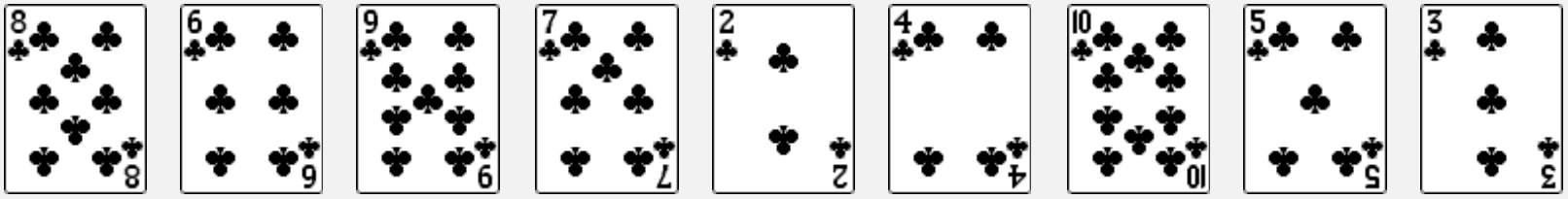


0.8003 0.9706 0.9157 0.9649 0.1576 0.4854 0.1419 0.4218 0.9572

Shuffle sort

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

useful for shuffling columns in a spreadsheet



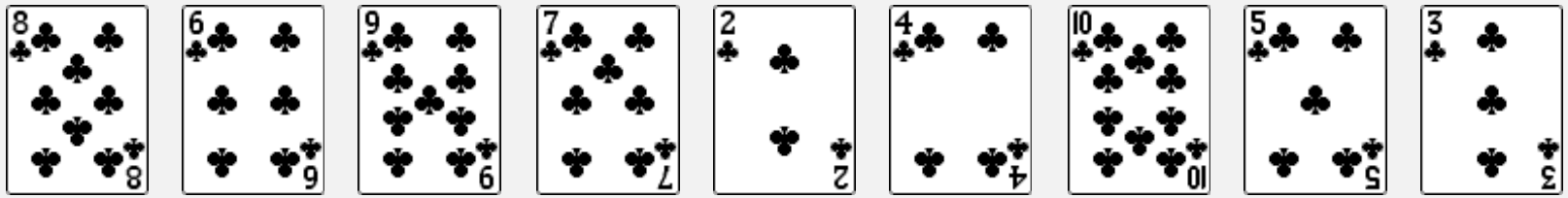
0.1419 0.1576 0.4218 0.4854 0.8003 0.9157 0.9572 0.9649 0.9706

Shuffle sort

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

useful for shuffling columns in a spreadsheet

如果每种可能的扑克牌排列方式都有相同的出现概率，那就说明洗牌方法是正确的！
正确固然好，但这种方法需要进行一次排序，似乎排序对洗牌问题来说有些累赘，我们能否做得更好呢？
我们能找到一种更快的洗牌方法吗？我们真的需要付出一次完整排序的代价吗？这个问题的答案是，不。



0.1419 0.1576 0.4218 0.4854 0.8003 0.9157 0.9572 0.9649 0.9706

在输入中没有重复值且假设产生均匀随机实数的情况下，shuffle sort 确实能够产生一个均匀的随机排列
PS：虽然我个人觉得输入中有重复值也没关系...

Proposition. Shuffle sort produces a uniformly random permutation of the input array, provided no duplicate values.

assuming real numbers uniformly at random

War story (Microsoft)

欧盟的微软反垄断调查

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

投票

http://www.browserchoice.eu

appeared last
50% of the time

IE有50%的时间都出现在
这个随机投票屏幕的最后

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.

为何会如此？通过让 compareTo 方法总是随机地返回结果来实现 shuffle sort（实际上不正确）

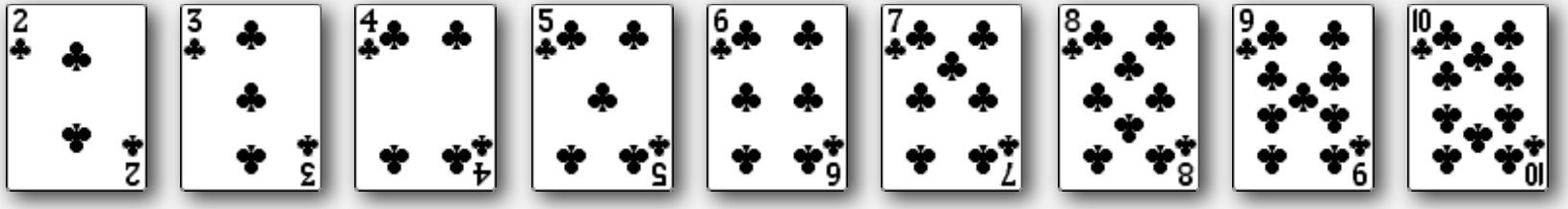
```
public int compareTo(Browser that)
{
   double r = Math.random();
   if (r < 0.5) return -1;
   if (r > 0.5) return +1;
   return 0;
}
```

browser comparator
(should implement a total order)

Knuth shuffle demo

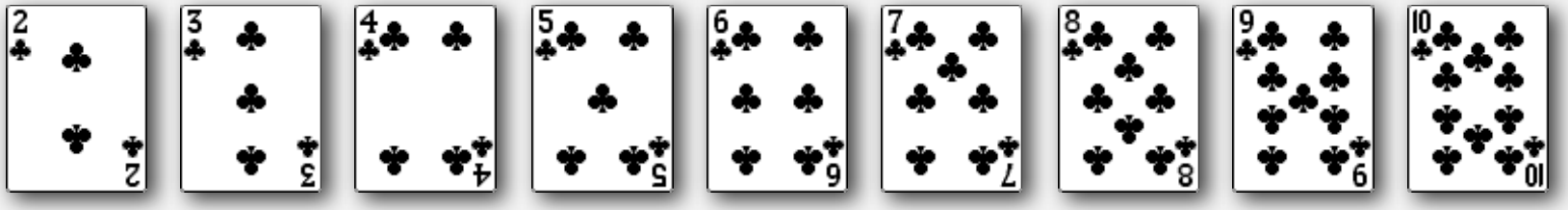
- In iteration `i`, pick integer `r` between `0` and `i` uniformly at random.
- Swap `a[i]` and `a[r]`.

虽然 demo 中我们从一个已经有序的数组开始洗牌，但实际上，数组的初始情况并不影响洗牌的最终结果



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

0到i是可以的，i到N-1也是可以的

```
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

War story (online poker)

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

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

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

War story (online poker)

Shuffling algorithm in FAQ at www.planetpoker.com

```
for i := 1 to 52 do begin
   r := random(51) + 1;
   swap := card[r];
   card[r] := card[i];
   card[i] := swap;
end;
```

between 1 and 51

洗牌后原来的第52张牌一定会被洗去其他地方！

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

应该是在[1,i]或者[i+1,52]之间

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

使用一个 32 位数字生成随机数，不能涵盖全部可能的洗牌方式，如果共有 52 张牌，可能的洗牌方法一共有 52！那么多种

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

如果使用从午夜到现在的毫秒数作为种子，那么可能的洗牌方式就更少了

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: 硬件随机数生成器是脆弱的，并且无声地失败。
  hardware random-number generators are fragile and fail silently.
- Use an unbiased shuffling algorithm.
  无偏

Bottom line. Shuffling a deck of cards is hard!

http://algs4.cs.princeton.edu

2.1 Elementary Sorts

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

http://algs4.cs.princeton.edu

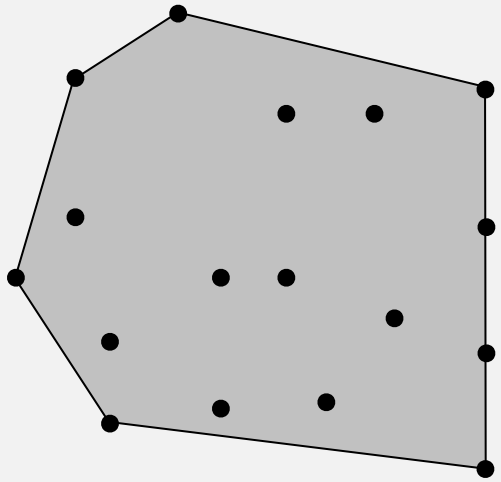
2.1 Elementary Sorts

- *rules of the game*
- *selection sort*
- *insertion sort*
- *shellsort*
- *shuffling*
- *convex hull* 凸包

Convex hull

周边围栏

The convex hull of a set of N points is the smallest perimeter fence enclosing the points.

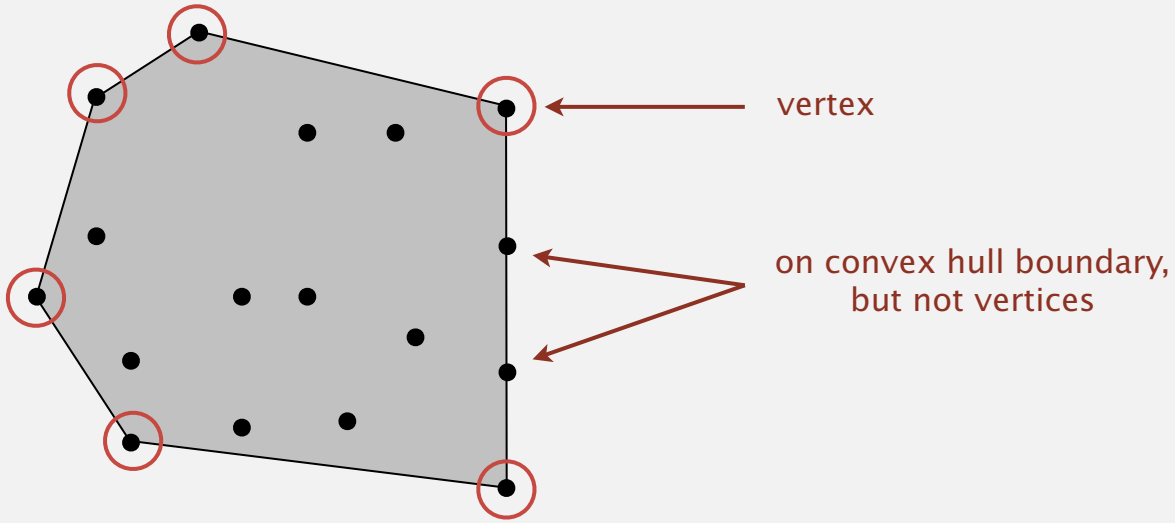


Equivalent definitions.

- Smallest convex set containing all the points.
- Smallest area convex polygon enclosing the points.
- Convex polygon enclosing the points, whose vertices are points in set.

Convex hull

The convex hull of a set of N points is the smallest perimeter fence enclosing the points.



我们要做的就是编写一个程序，对于给定的点集生成它的凸包，为了程序的结果更加明确易用，这个程序逆时针输出这个凸包的顶点序列

Convex hull output. Sequence of vertices in counterclockwise order.

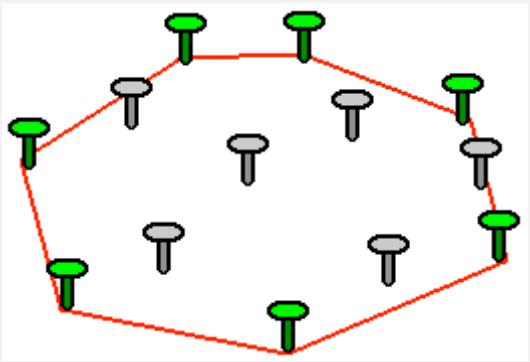
逆时针方向

Convex hull: mechanical algorithm

锤击 钉子 垂直于 拉紧 弹性

Mechanical algorithm. Hammer nails perpendicular to plane; stretch elastic rubber band around points.

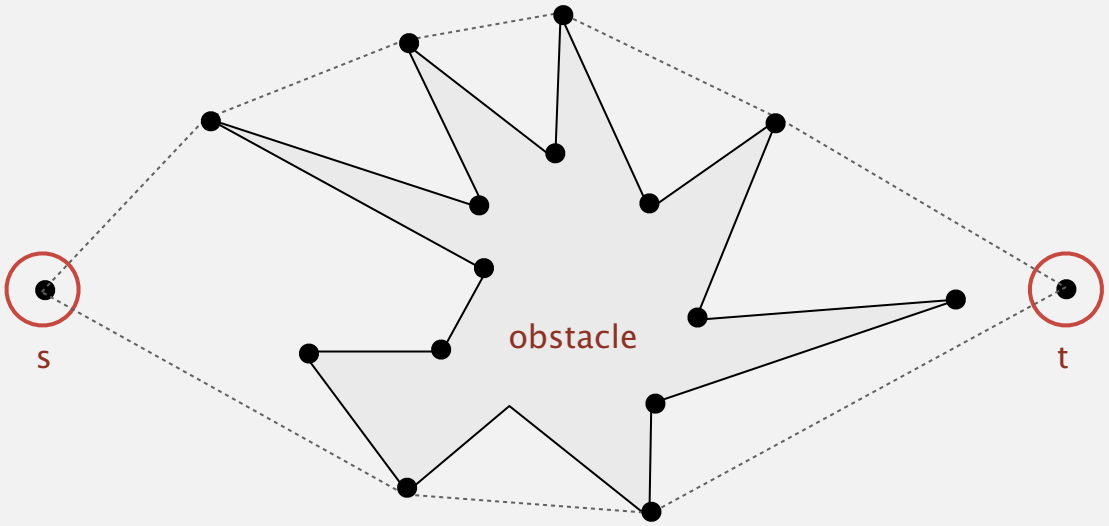
橡胶



http://www.idlcoyote.com/math_tips/convexhull.html

Convex hull application: motion planning 运动规划

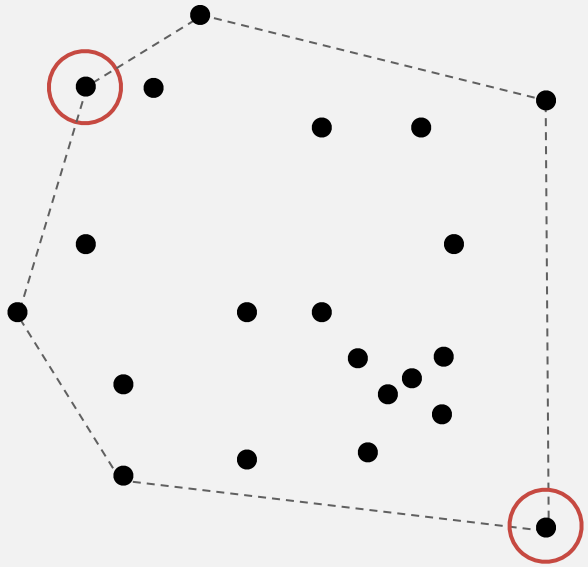
Robot motion planning. Find shortest path in the plane from s to t that avoids a polygonal obstacle.



Fact. Shortest path is either straight line from s to t or it is one of two polygonal chains of convex hull.

Convex hull application: farthest pair

Farthest pair problem. Given N points in the plane, find a pair of points with the largest Euclidean distance between them.



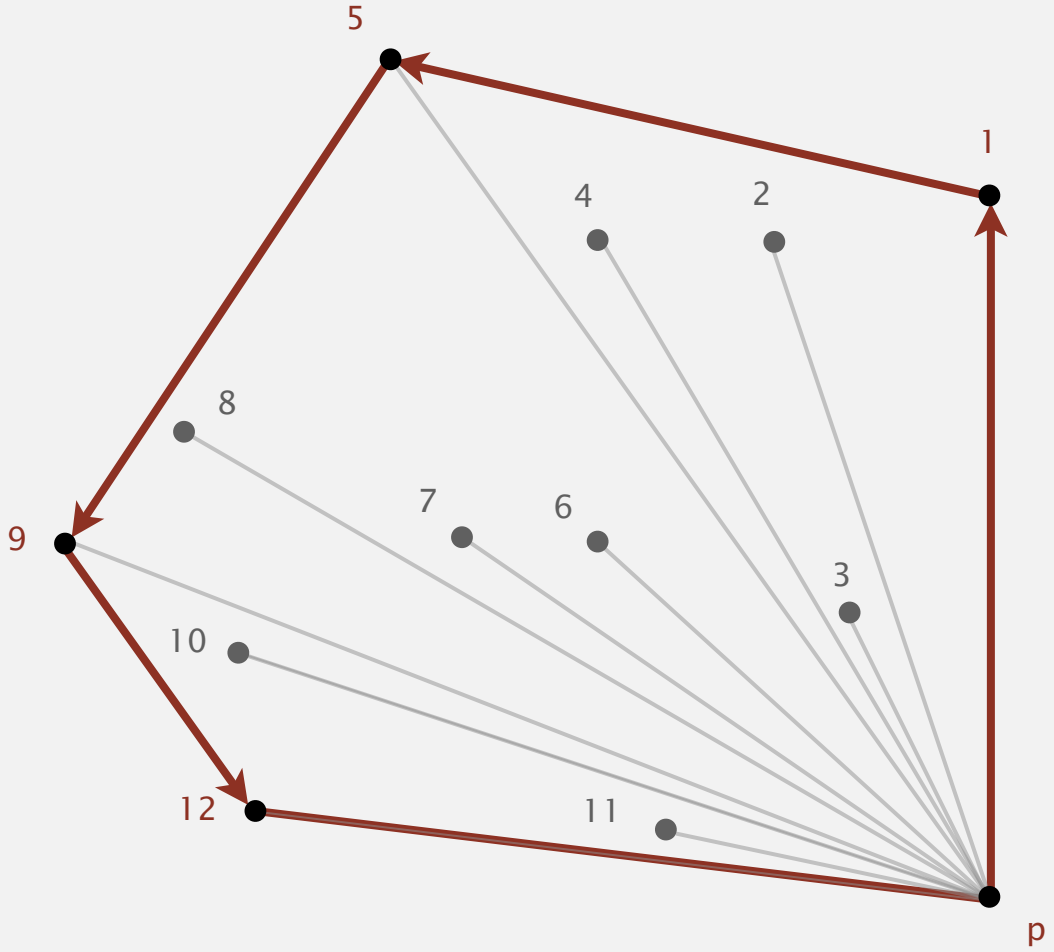
Fact. Farthest pair of points are extreme points on convex hull.

最远的一对点实际上就是凸包上的极端点（意即我们可以只在凸包的顶点上进行查找）

Convex hull: geometric properties 几何性质

Fact. Can traverse the convex hull by making only counterclockwise turns.

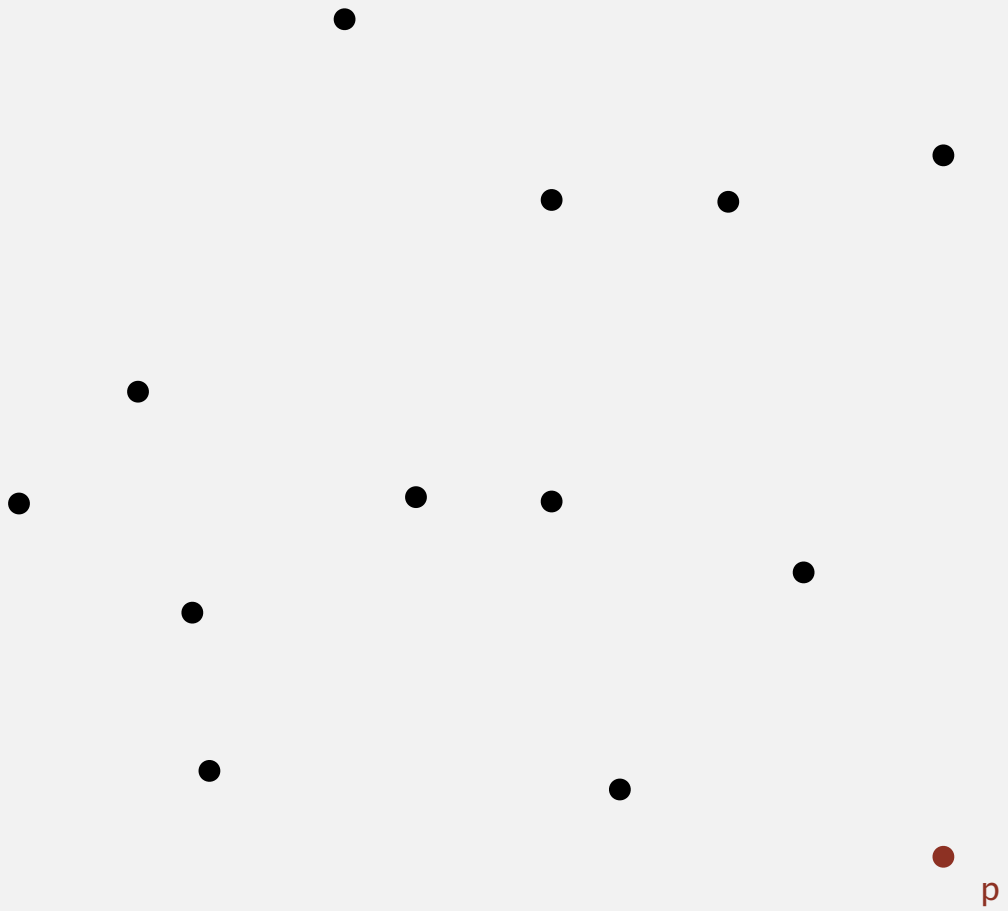
Fact. The vertices of convex hull appear in increasing order of polar angle 极角 with respect to point p with lowest y-coordinate.



Graham scan demo

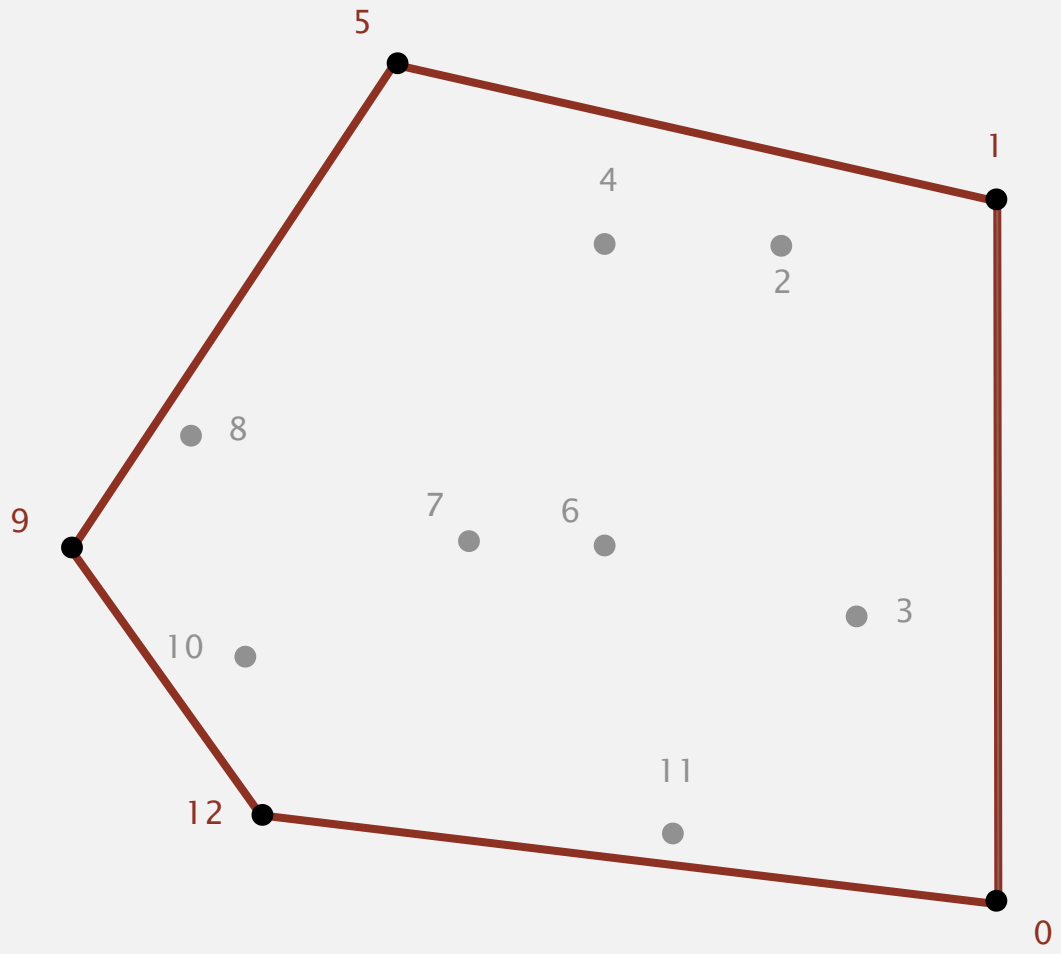
- Choose point p with smallest y-coordinate.
- Sort points by polar angle with p.
- Consider points in order; discard unless it create a ccw turn.

逐个考虑排序后的点，丢弃那些无法产生逆时针旋转（左转）的点



Graham scan demo

- Choose point p with smallest y-coordinate.
- Sort points by polar angle with p.
- Consider points in order; discard unless it create a ccw turn.



Graham scan: implementation challenges

Q. How to find point p with smallest y-coordinate?

A. Define a total order, comparing by y-coordinate. [next lecture]

Graham 扫描法是一个完美的例子，我们不仅仅要学会如何排序，如何定义排序和比较，还要能对同样的对象进行不同方式的排序

Q. How to sort points by polar angle with respect to p ?

A. Define a total order for each point p. [next lecture]

Q. How to determine whether $p_1 \rightarrow p_2 \rightarrow p_3$ is a counterclockwise turn?

A. Computational geometry. [next two slides]

计算几何的一些小知识点

Q. How to sort efficiently? 高效的排序算法让凸包算法也更高效！计算凸包最主要的部分就是排序。

A. Mergesort sorts in $N \log N$ time. [next lecture]

简并

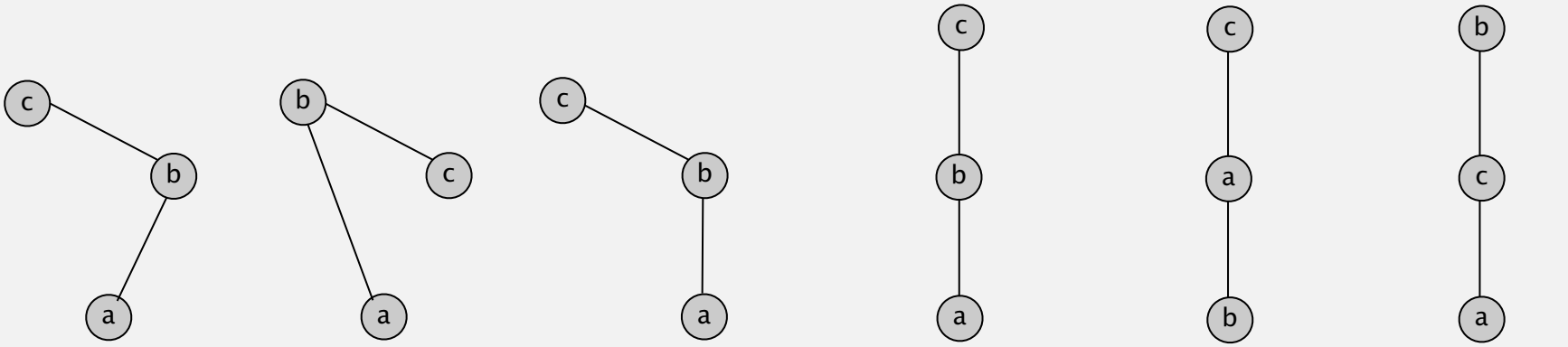
Q. How to handle degeneracies (three or more points on a line)?

A. Requires some care, but not hard. [see booksite]

Implementing ccw

CCW. Given three points a, b, and c, is $a \rightarrow b \rightarrow c$ a counterclockwise turn?

is c to the left of the ray a→b



几何基元往往是很难实现的！

Lesson. Geometric primitives are tricky to implement.

- Dealing with degenerate cases.
- Coping with floating-point precision.
  应对浮点精度

Implementing ccw

CCW. Given three points a, b, and c, is $a \to b \to c$ a counterclockwise turn?

- Determinant (or cross product) gives 2x signed area of planar triangle.

行列式（或叉乘）给出了平面的三角形的2倍的带符号面积（联系线性代数）

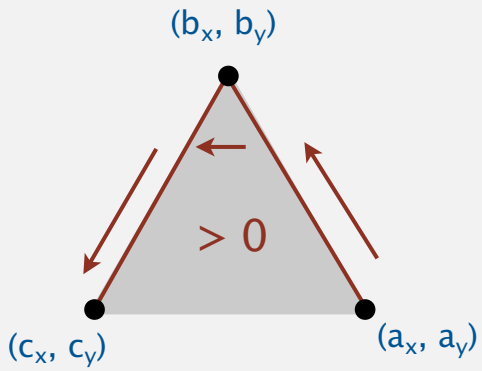
$$2 \times Area(a,b,c) = \begin{vmatrix} a_x & a_y & 1 \\ b_x & b_y & 1 \\ c_x & c_y & 1 \end{vmatrix} = (b_x - a_x)(c_y - a_y) - (b_y - a_y)(c_x - a_x)$$

因为最后一列都是 1，所以求的是面积

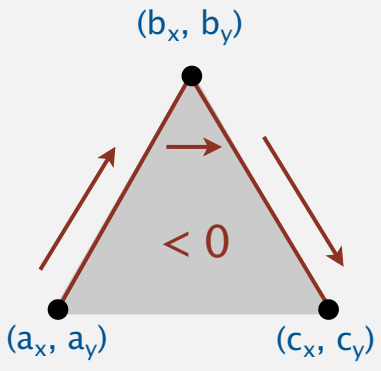
(b − a) × (c − a)

a->b射线与a->c射线的叉乘

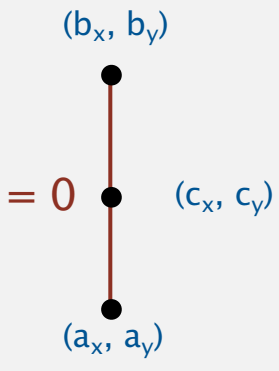
- If signed area > 0, then $a \to b \to c$ is counterclockwise.
- If signed area < 0, then $a \to b \to c$ is clockwise.
- If signed area $= 0$, then $a \to b \to c$ are collinear.



counterclockwise



clockwise



collinear

Immutable point data type

```
public class Point2D
{
   private final double x;
   private final double y;

   public Point2D(double x, double y)
   {
      this.x = x;
      this.y = y;
   }

   ...

   public static int ccw(Point2D a, Point2D b, Point2D c)
   {
      double area2 = (b.x-a.x)*(c.y-a.y) - (b.y-a.y)*(c.x-a.x);
      if      (area2 < 0) return -1;  // clockwise
      else if (area2 > 0) return +1;  // counter-clockwise
      else                return  0;  // collinear
   }
}
```

danger of floating-point roundoff error

http://algs4.cs.princeton.edu

2.1 Elementary Sorts

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

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

http://algs4.cs.princeton.edu

2.1 ELEMENTARY SORTS

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