2.1 ELEMENTARY SORTS

Algorithms
FOURTH EDITION

ROBERT SEDGEWICK | KEVIN WAYNE

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

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

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 ⟶ Furia
key ⟶ Battle

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

Goal. Sort any type of data.

Ex 1. Sort random real numbers in ascending order.

seems artificial, but stay tuned for an application

```java
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 file in alphabetical order.

```
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

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 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.
- 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 dependence on File data type

Comparable API

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

- 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)

Required property: Must be 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

Date data type. Simplified version of `java.util.Date`.

```java
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;
    }
```

only compare dates
to other dates

```java
    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;
    }
}
```

Two useful sorting abstractions

Helper functions. Refer to data through compares and exchanges.

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

```java
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`.

```java
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.

```java
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.

13

Selection sort demo

14

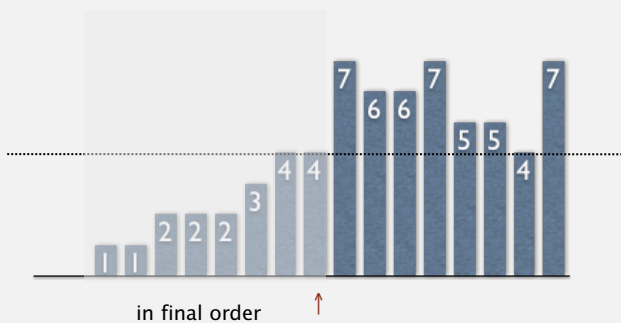## Selection sort

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 ↑.

in final order

15

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);
```


in final order

16

Selection sort: Java implementation

```java
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     entries in black
      1   4   A  O  R  T  E  X  S  M  P  L  E     are examined to find
      2  10   A  E  R  T  O  X  S  M  P  L  E     the minimum
      3   9   A  E  E  T  O  X  S  M  P  L  R     entries in red
      4   7   A  E  E  L  O  X  S  M  P  T  R     are a[min]
      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     entries in gray are
      9   9   A  E  E  L  M  O  P  R  S  T  X     in final position
     10  10   A  E  E  L  M  O  P  R  S  T  X
              A  E  E  L  M  O  P  R  S  T  X
```

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 items

▲ algorithm position
▬ in final order
▬ not in final order

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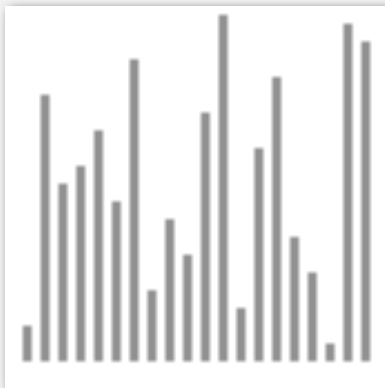
Selection sort: animations

20 partially-sorted items

▲ algorithm position
▬ in final order
▬ not in final order

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

Selection sort: Gypsy folk dance
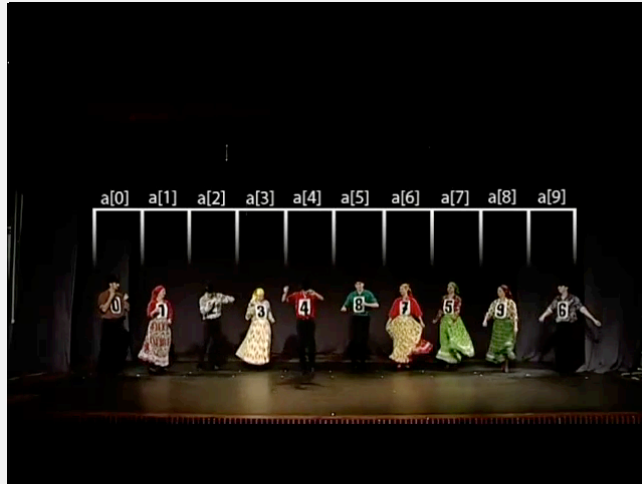
Insertion sort demo

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.

in order   ↑   not yet seen

Insertion sort inner loop

To maintain algorithm invariants:

- Move the pointer to the right.

```
i++;
```


in order   not yet seen

- 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;
```


in order   not yet seen

Insertion sort: Java implementation

```java
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

Insertion sort: animation

40 random items

▲ algorithm position

在 order (in order)

not yet seen

http://www.sorting-algorithms.com/insertion-sort
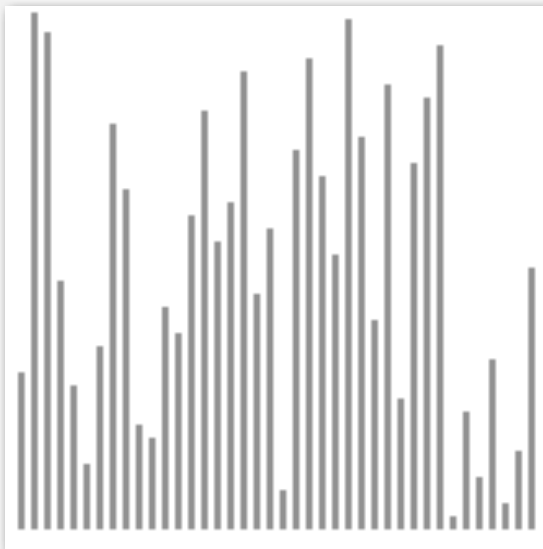
Insertion sort: best and worst case

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

$$\texttt{A E E L M O P R S T X}$$

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.

$$\texttt{X T S R P O M L E E A}$$

Insertion sort: animation

40 reverse-sorted items

▲ algorithm position

in order

not yet seen

http://www.sorting-algorithms.com/insertion-sort
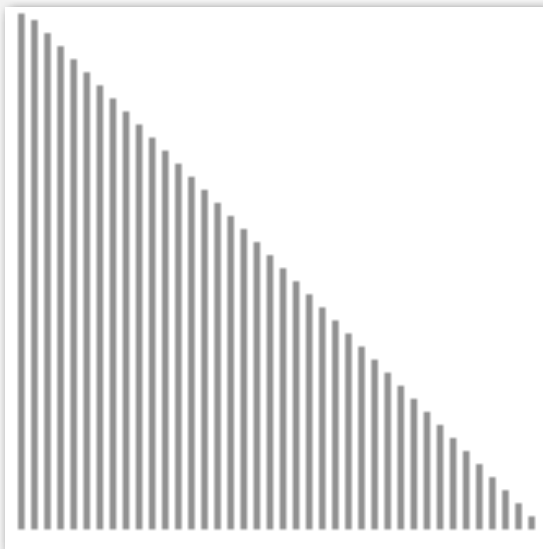
Insertion sort: partially-sorted arrays

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

$$\texttt{A E E L M O T R X P S}$$

$$\texttt{T-R T-P T-S R-P X-P X-S}$$
(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.

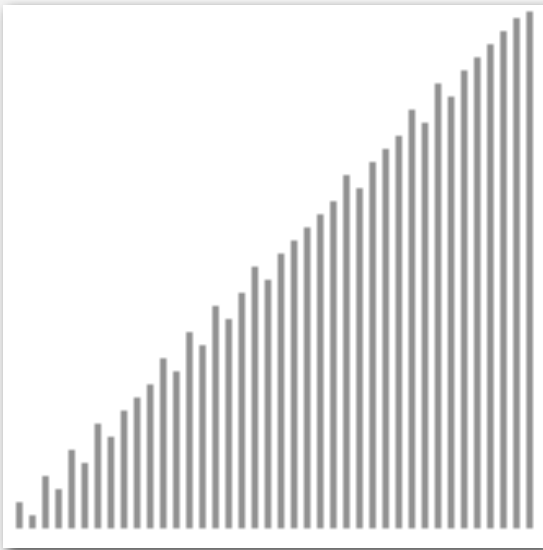**Proposition.** For partially-sorted arrays, insertion sort runs in linear time.

Pf. Number of exchanges equals the number of inversions.

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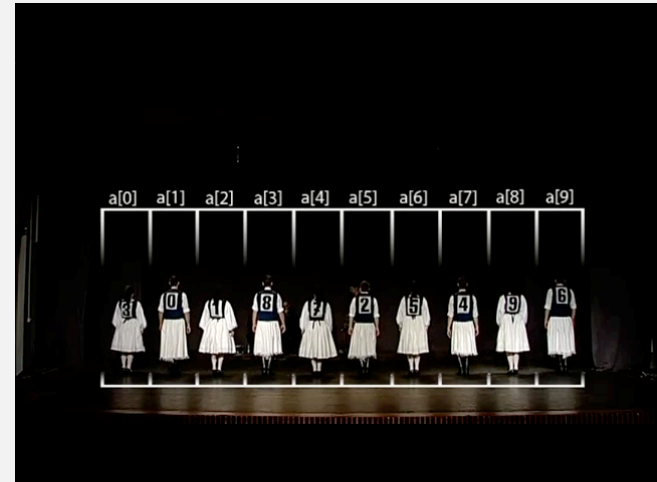number of compares = exchanges + (N-1)

40 partially-sorted items

▲ algorithm position
in order
not yet seen

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

33

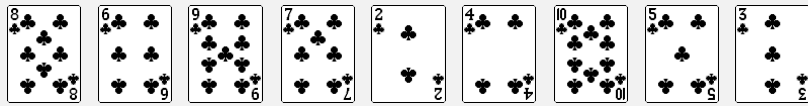a[0] a[1] a[2] a[3] a[4] a[5] a[6] a[7] a[8] a[9]

34

35

How to shuffle an array

Shuffling. Rearrange an array so that result is a uniformly random permutation.

36

How to shuffle an array
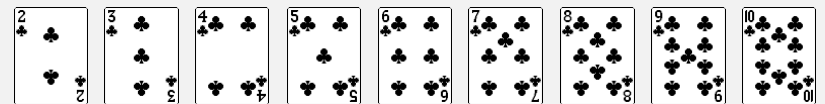
Shuffling. Rearrange an array so that result is a uniformly random permutation.

Knuth shuffle demo

Knuth shuffle

Knuth shuffle. [Fisher-Yates 1938]

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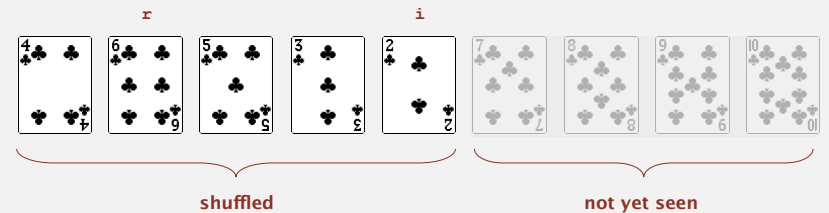

shuffled   not yet seen
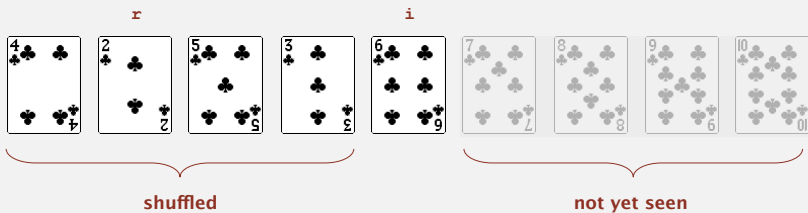
Proposition. Knuth shuffling algorithm produces a uniformly random permutation of the input array in linear time.

assuming integers uniformly at random

---

shuffled   not yet seen

Knuth shuffle

Knuth shuffle. [Fisher-Yates 1938]

- 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

```java
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);
      }
   }
}
```
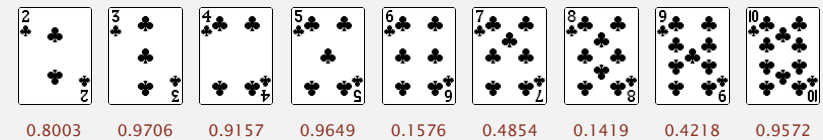
between 0 and i

41

Shuffle sort

Shuffle sort.

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

useful for shuffling columns in a spreadsheet

0.8003 0.9706 0.9157 0.9649 0.1576 0.4854 0.1419 0.4218 0.9572

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

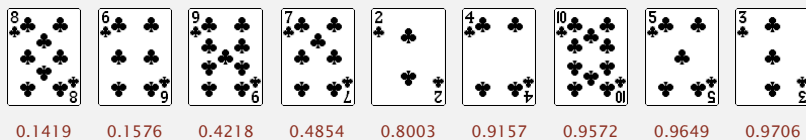*assuming real numbers uniformly at random*

42

Shuffle sort

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

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

assuming real numbers uniformly at random

43

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

44

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)

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

Bug 1. Random number `r` never 52 $\Rightarrow$ 52nd card can't end up in 52nd place.

Bug 2. Shuffle not uniform (should be between i and 51).

Bug 3. `random()` uses 32-bit seed $\Rightarrow$ 2^{32} possible shuffles.

Bug 4. Seed = milliseconds since midnight $\Rightarrow$ 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 both the FIPS 140-2 and the NIST statistical test suites.
- Continuously monitor statistic properties:
  hardware random-number generators are fragile and fail silently.
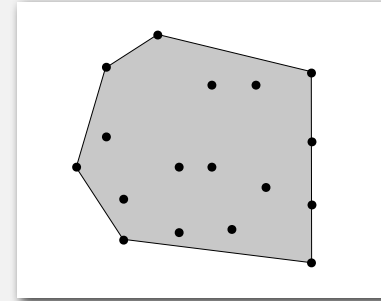- Use an unbiased shuffling algorithm.

Bottom line. Shuffling a deck of cards is hard!

49

Convex hull

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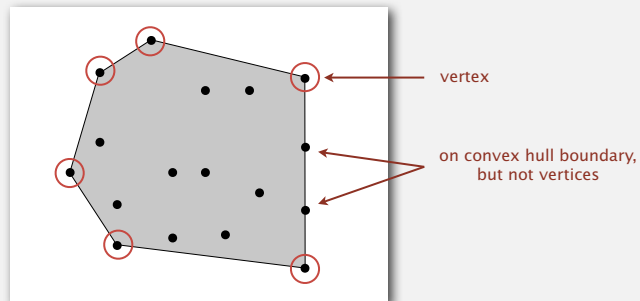

Equivalent definitions.

- Smallest convex set containing all the points.
- Smallest area convex polygon enclosing the points.
- Convex polygon enclosing the points, whose vertices are points in the set.
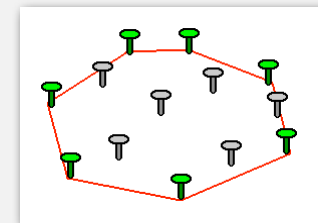
50

Convex hull

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

vertex

on convex hull boundary, but not vertices

Convex hull output. Sequence of vertices in counterclockwise order.

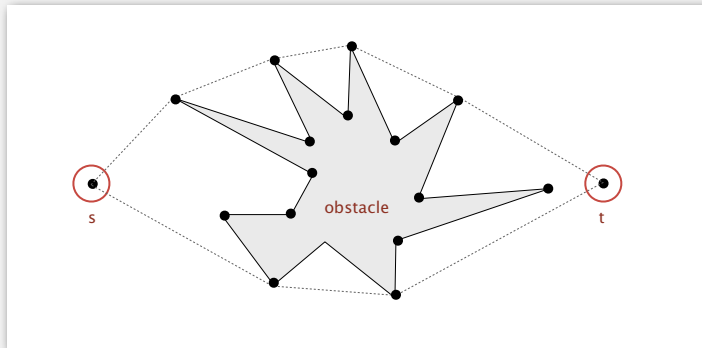51

Convex hull: mechanical algorithm

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

http://www.dfanning.com/math_tips/convexhull_1.gif

52

Convex hull application: motion planning

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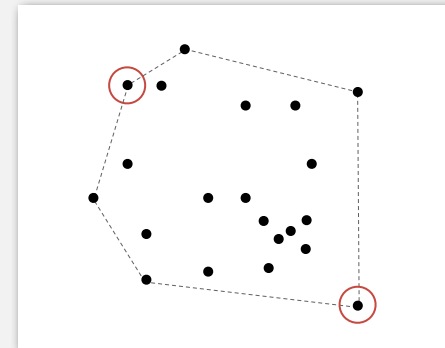

Fact. Shortest path is either straight line from s to t or it is one of two polygonal chains of convex hull.

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.
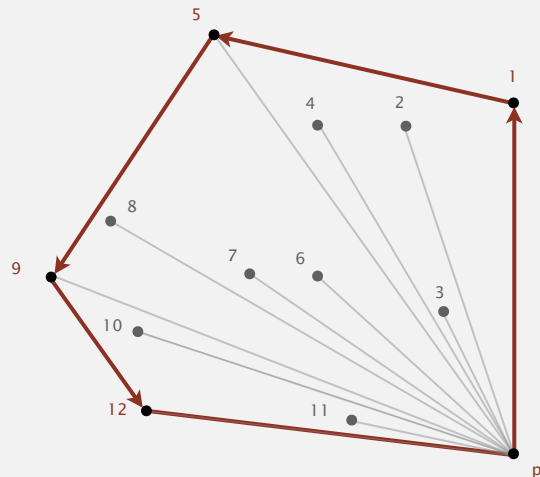

Fact. Farthest pair of points are extreme points on convex hull.

Convex hull: geometric properties

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

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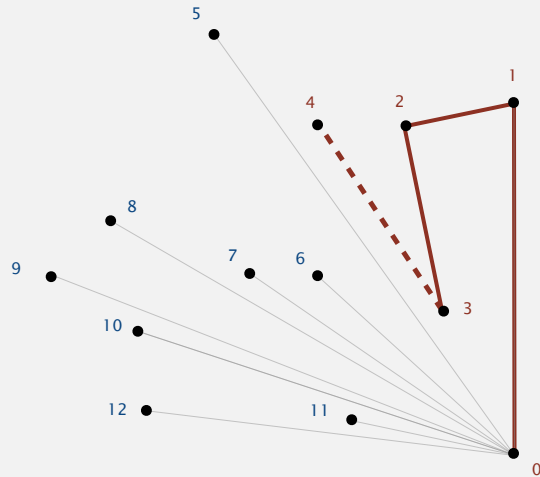
Graham scan demo

Convex hull: Graham scan

- Choose point p with smallest y-coordinate.
- Sort points by polar angle with p.
- Consider points in order, and discard unless that would create a ccw turn.

## Graham scan: implementation challenges

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

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

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 $p_1 \to p_2 \to p_3$ is a counterclockwise turn?

A. Computational geometry. [next two slides]

Q. How to sort efficiently?

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

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 $a \to b \to c$ a counterclockwise turn?

is c to the left of the ray a→b

yes   no   yes (∞–slope)   no (collinear)   no (collinear)   no (collinear)
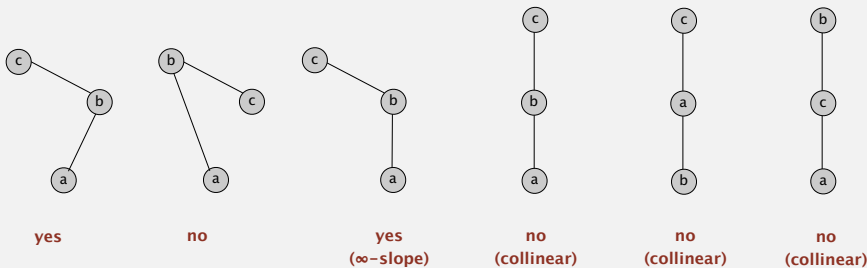
Lesson. Geometric primitives are tricky to implement.

- Dealing with degenerate cases.
- Coping with floating-point precision.

Implementing ccw

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

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

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

$(b - a) \times (c - a)$

- 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
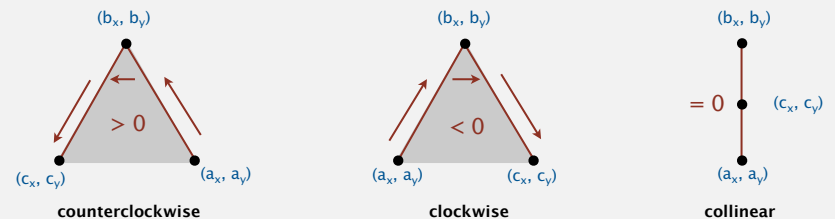
Immutable point data type

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

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

   ...

      public static int ccw(Point a, Point b, Point c)
      {
         int 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