Algorithms ROBERT SEDGEWICK | KEVIN WAYNE

http://algs4.cs.princeton.edu

2.1 ELEMENTARY SORTS

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

Last updated on Feb 10, 2015, 5:57 AM

http://algs4.cs.princeton.edu

2.1 ELEMENTARY SORTS

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

Sorting problem

Ex. Student records in a university.

item →

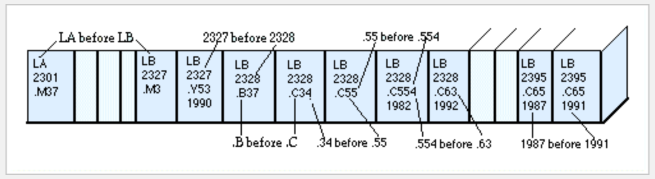
key →

Chen	3	A	(991) 878-4944	308 Blair
Rohde	2	A	(232) 343-5555	343 Forbes
Gazsi	4	B	(800) 867-5309	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 in ascending order by key.

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	(800) 867-5309	101 Brown
Kanaga	3	B	(898) 122-9643	22 Brown
Rohde	2	A	(232) 343-5555	343 Forbes

Sorting applications

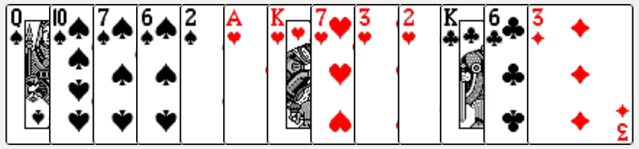


Library of Congress numbers

Groups All Contacts
Katie Abeles
Aaron Ackerman
Kelly Altick
Bryan Alvarez
David Amer
Greg Apodaca
Jane Appleseed
John Baily
Kamaldeep Bai

contacts

FedEx packages



playing cards

Hogwarts houses

Sample sort client 1

Goal. Sort any type of data.

Ex 1. Sort random real numbers in ascending order.

seems artificial (stay tuned for an application)

```
public class Experiment
{
   public static void main(String[] args)
   {
      int N = Integer.parseInt(args[0]);
      Double[] a = new Double[N];
      for (int i = 0; i < N; i++)
         a[i] = StdRandom.uniform();
      Insertion.sort(a);
      for (int i = 0; i < N; i++)
         StdOut.println(a[i]);
   }
}
```

```
% java Experiment 10
0.08614716385210452
0.09054270895414829
0.10708746304898642
0.21166190071646818
0.363292849257276
0.460954145685913
0.5340026311350087
0.7216129793703496
0.9003500354411443
0.9293994908845686
```

Sample sort client 2

Goal. Sort any type of data.

Ex 2. Sort strings in alphabetical order.

```
public class StringSorter
{
   public static void main(String[] args)
   {
      String[] a = StdIn.readAllStrings();
      Insertion.sort(a);
      for (int i = 0; i < a.length; i++)
         StdOut.println(a[i]);
   }
}
```

```
% more words3.txt
bed bug dad yet zoo ... all bad yes

% java StringSorter < words3.txt
all bad bed bug dad ... yes yet zoo
[suppressing newlines]
```

Sample sort client 3

Goal. Sort any type of data.

Ex 3. Sort the files in a given directory by filename.

```
import java.io.File;

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

```
% java FileSorter .
Insertion.class
Insertion.java
InsertionX.class
InsertionX.java
Selection.class
Selection.java
Shell.class
Shell.java
ShellX.class
ShellX.java
```

Total order

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

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

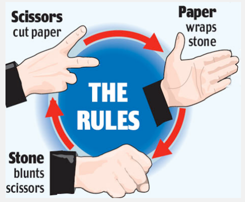
- Antisymmetry: if both $v \le w$ and $w \le v$, then $v = w$.
- Transitivity: if both $v \le w$ and $w \le x$, then $v \le x$.
- Totality: either $v \le w$ or $w \le v$ or both.

Ex.

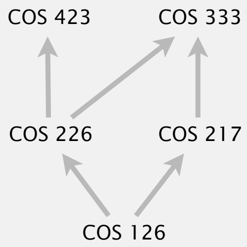
- Standard order for natural and real numbers.
- Chronological order for dates or times.
- Lexicographic order for strings.

Not transitive. Ro-sham-bo.

Not total. PU course prerequisites.



violates transitivity



violates totality

Callbacks

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

Q. How can `sort()` compare data of type `Double`, `String`, and `java.io.File` without hardwiring in type-specific information.

Callback = reference to executable code.

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

Implementing callbacks.

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

Callbacks: Java interfaces

Interface. Specifies a set of methods that a concrete class can provide.

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

contract: one method with this signature and prescribed behavior

Concrete class. Can provide the set of methods in the interface.

```
public class String implements Comparable<String>
{
   ...
   public int compareTo(String that)
   {
      ...
   }
}
```

class promises to honor the contract

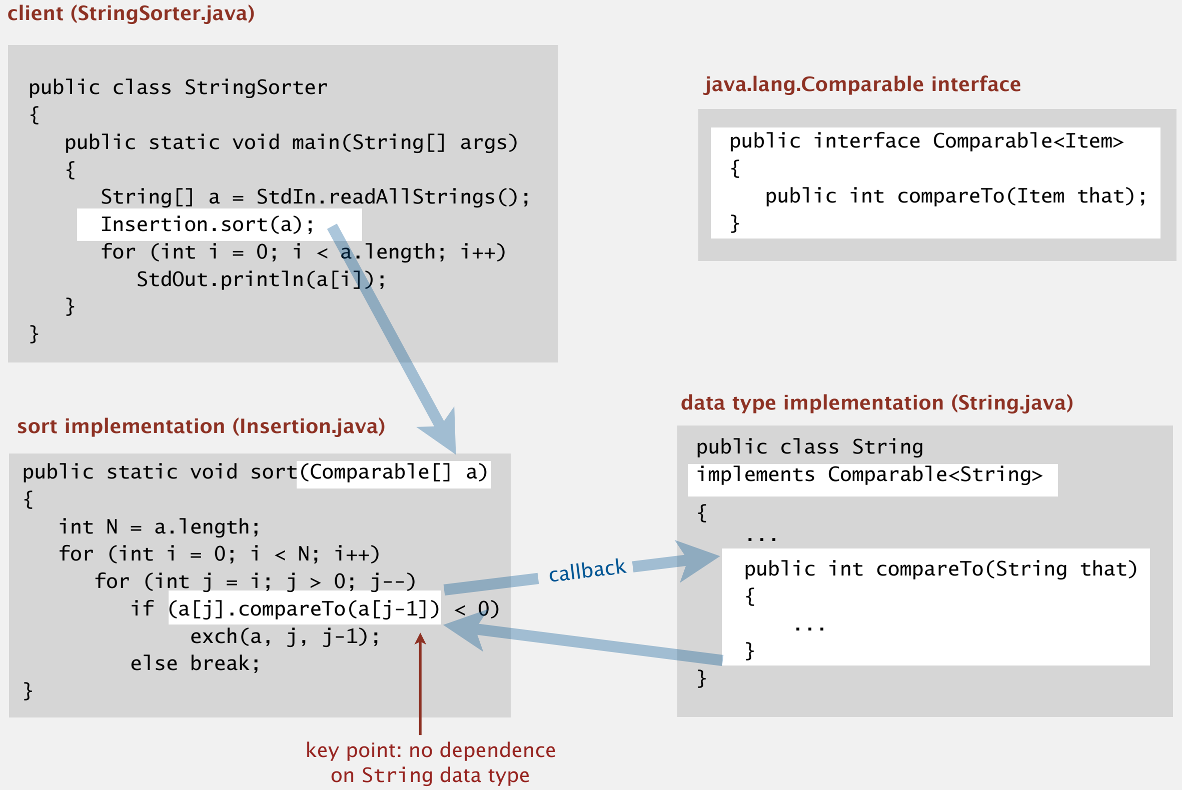
class honors the contract

Impact.

- You can treat any `String` object as an object of type `Comparable`.
- On a `Comparable` object, you can invoke (only) the `compareTo()` method.
- Enables callbacks.

"polymorphism"

Callbacks: roadmap



client (StringSorter.java)

```
public class StringSorter
{
   public static void main(String[] args)
   {
      String[] a = StdIn.readAllStrings();
      Insertion.sort(a);
      for (int i = 0; i < a.length; i++)
         StdOut.println(a[i]);
   }
}
```

java.lang.Comparable interface

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

sort implementation (Insertion.java)

```
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;
}
```

data type implementation (String.java)

```
public class String
implements Comparable<String>
{
   ...
   public int compareTo(String that)
   {
      ...
   }
}
```

key point: no dependence on String data type

Elementary sorts: quiz 1

Suppose that the Java architects leave out `implements Comparable<String>` in the class declaration for `String`. What would be the effect?

A. `String.java` won't compile.

B. `StringSorter.java` won't compile.

C. `Insertion.java` won't compile.

D. `Insertion.java` will throw a run-time exception.

E. *I don't know.*

java.lang.Comparable API

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

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

less than
(return negative integer)

equal to
(return 0)

greater than
(return positive integer)

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

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

Implementing the Comparable interface

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

```
public class Date implements Comparable<Date>
{
   private final int month, day, year;

   public Date(int m, int d, int y)
   {
      month = m;
      day   = d;
      year  = y;
   }

   public int compareTo(Date that)
   {
      if (this.year  < that.year ) return -1;
      if (this.year  > that.year ) return +1;
      if (this.month < that.month) return -1;
      if (this.month > that.month) return +1;
      if (this.day   < that.day  ) return -1;
      if (this.day   > that.day  ) return +1;
      return 0;
   }
}
```

only compare dates
to other dates

http://algs4.cs.princeton.edu/12oop/Date.java.html

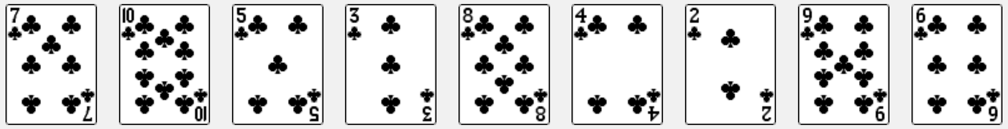
http://algs4.cs.princeton.edu

2.1 Elementary Sorts

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

Selection sort demo

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



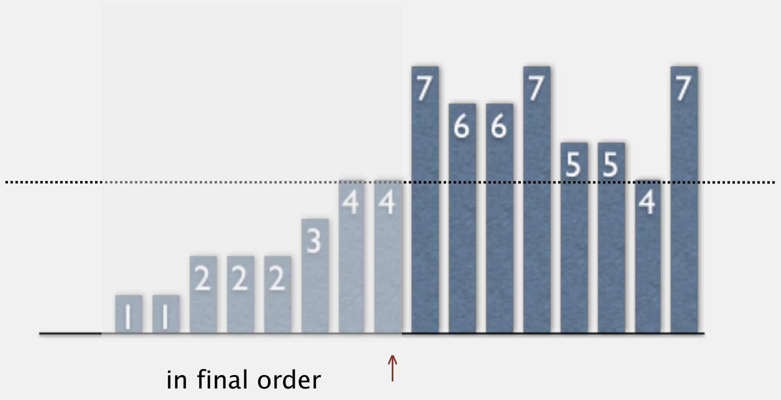
initial

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 ↑.

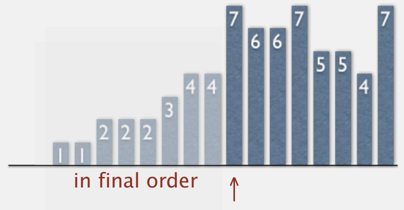


Selection sort inner loop

To maintain algorithm invariants:

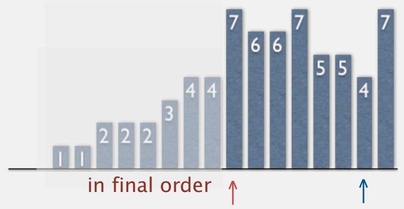
- Move the pointer to the right.

```
i++;
```



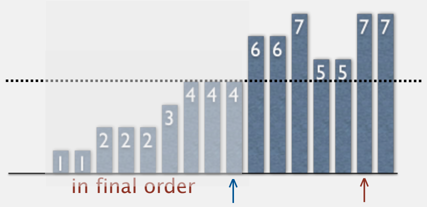
- Identify index of minimum entry on right.

```
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);
```



Two useful sorting abstractions

Helper functions. Refer to data only 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(Object[] a, int i, int j)
{
   Object swap = a[i];
   a[i] = a[j];
   a[j] = swap;
}
```

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)
   {  /* see previous slide */  }

   private static void exch(Object[] a, int i, int j)
   {  /* see previous slide */  }
}
```

http://algs4.cs.princeton.edu/21elementary/Selection.java.html

Generic methods

Oops. The compiler complains.

```
% javac Selection.java
Note: Selection.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.
```

```
 % javac -Xlint:unchecked Selection.java
 Selection.java:83: warning: [unchecked] unchecked call to
compareTo(T) as a member of the raw type java.lang.Comparable
        return (v.compareTo(w) < 0);
                           ^
1 warning
```

Q. How to fix?

Generic methods

Pedantic (type-safe) version. Compiles cleanly.

generic type variable
(type inferred from argument; must be Comparable)

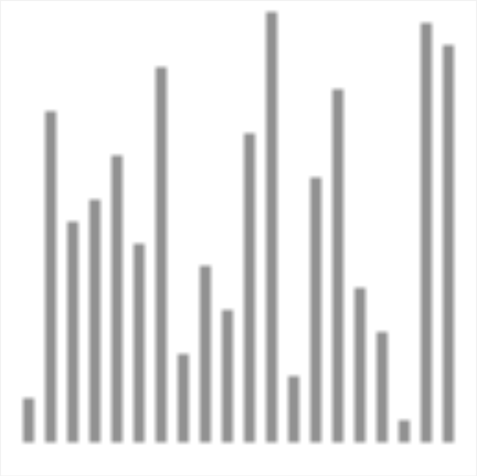
```
public class SelectionPedantic
{
   public  static <Key extends Comparable<Key>> void sort(Key[] a)
   {  /* as before */  }

   private static <Key extends Comparable<Key>> boolean less(Key v, Key w)
   {  /* as before */  }

   private static Object void exch(Object[] a, int i, int j)
   {  /* as before */  }
}
```

http://algs4.cs.princeton.edu/21elementary/SelectionPedantic.java.html

Remark. Use type-safe version in system code (but not in lecture).

Selection sort: animations

20 random items

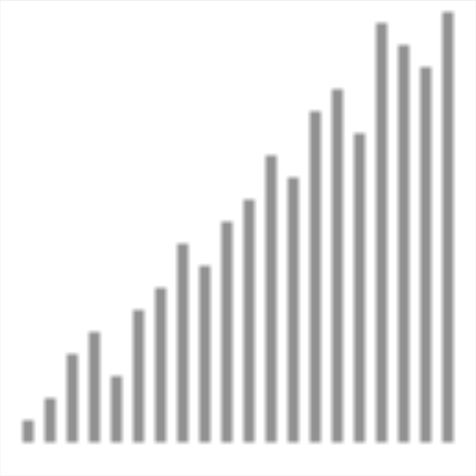


▲ algorithm position
▬ in final order
▬ not in final order

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

Elementary sorts: quiz 1

How many compares does selection sort make to sort an array of N keys?

A. $\sim N$

B. $\sim 1/4\ N^2$

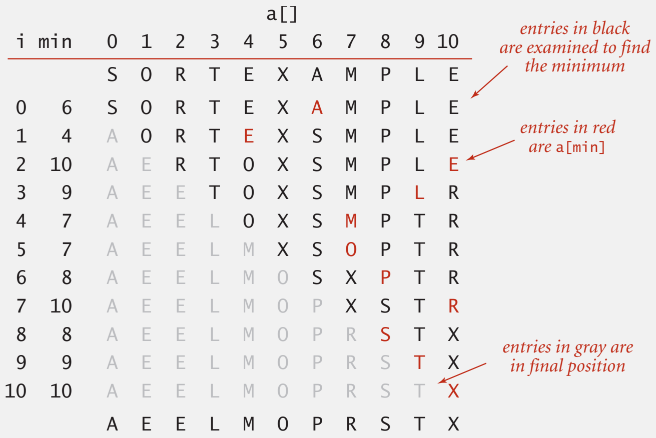
C. $\sim 1/2\ N^2$

D. $\sim N^2$

E. *I don't know.*

Selection sort: mathematical analysis

Proposition. Selection sort uses $(N-1) + (N-2) + \ldots + 1 + 0 \sim N^2 / 2$ compares and N exchanges to sort any array of N items.



		a[]										
i	min	0	1	2	3	4	5	6	7	8	9	10
		S	O	R	T	E	X	A	M	P	L	E
0	6	S	O	R	T	E	X	A	M	P	L	E
1	4	A	O	R	T	E	X	S	M	P	L	E
2	10	A	E	R	T	O	X	S	M	P	L	E
3	9	A	E	E	T	O	X	S	M	P	L	R
4	7	A	E	E	L	O	X	S	M	P	T	R
5	7	A	E	E	L	M	X	S	O	P	T	R
6	8	A	E	E	L	M	O	S	X	P	T	R
7	10	A	E	E	L	M	O	P	X	S	T	R
8	8	A	E	E	L	M	O	P	R	S	T	X
9	9	A	E	E	L	M	O	P	R	S	T	X
10	10	A	E	E	L	M	O	P	R	S	T	X
		A	E	E	L	M	O	P	R	S	T	X

entries in black are examined to find the minimum

entries in red are a[min]

entries in gray are in final position

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

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

Data movement is minimal. Linear number of exchanges.

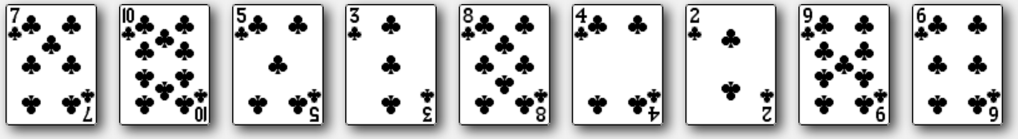
http://algs4.cs.princeton.edu

2.1 Elementary Sorts

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

Insertion sort demo

- In iteration i, swap a[i] with each larger entry to its left.

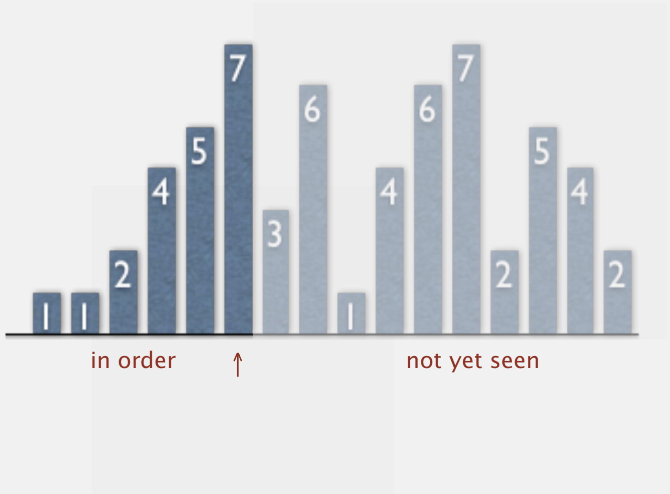


Insertion sort

Algorithm. ↑ scans from left to right.

Invariants.

- Entries to the left of ↑ (including ↑) are in ascending order.
- Entries to the right of ↑ have not yet been seen.

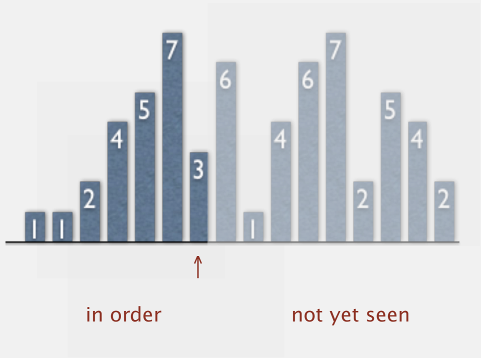


Insertion sort: inner loop

To maintain algorithm invariants:

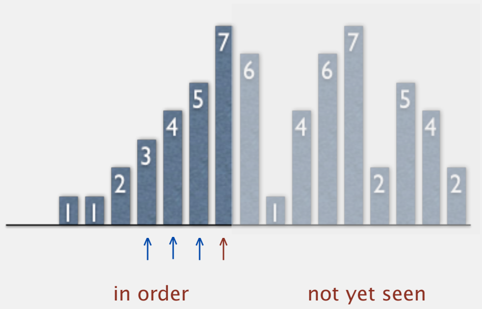
- Move the pointer to the right.

```
i++;
```



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

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



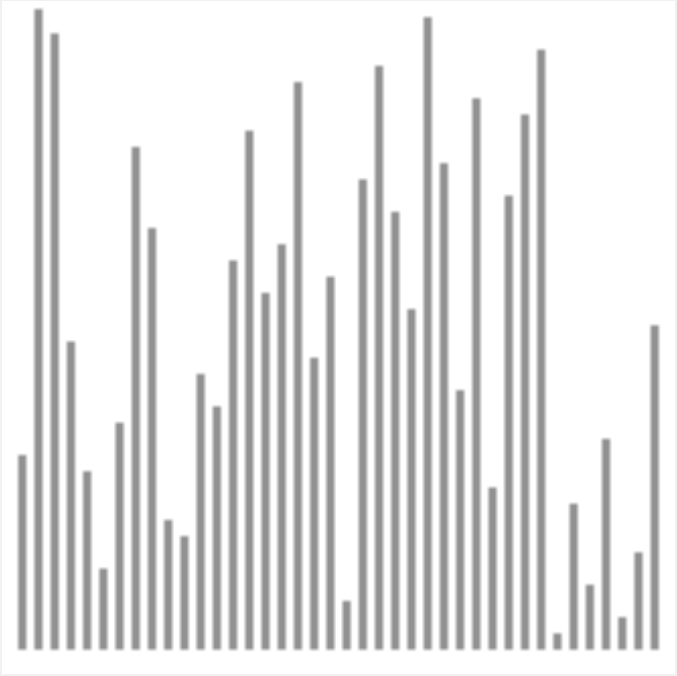
Insertion sort: Java implementation

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

   private static boolean less(Comparable v, Comparable w)
   {  /* as before */  }

   private static void exch(Object[] a, int i, int j)
   {  /* as before */  }
}
```

http://algs4.cs.princeton.edu/21elementary/Insertion.java.html

Insertion sort: animation

40 random items



▲ algorithm position

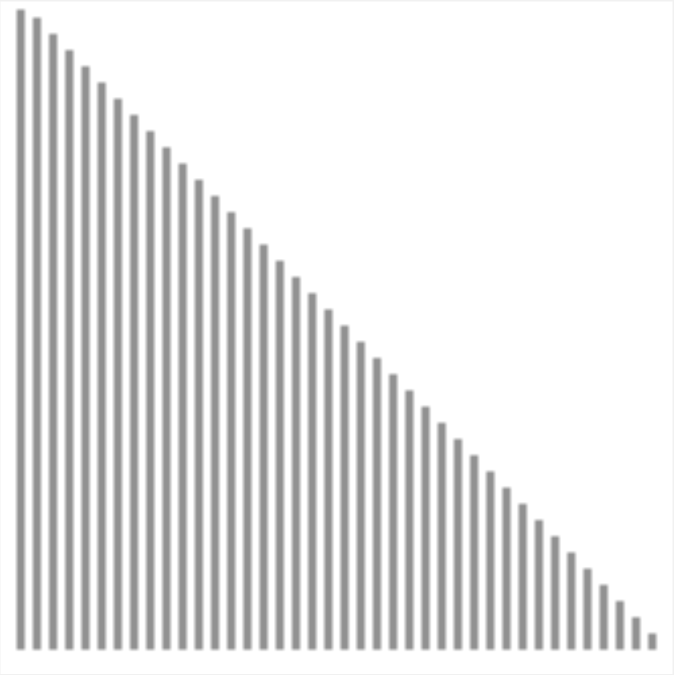
▬ in order

▬ not yet seen

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

Insertion sort: animation

40 reverse-sorted items



▲ algorithm position

▬ in order

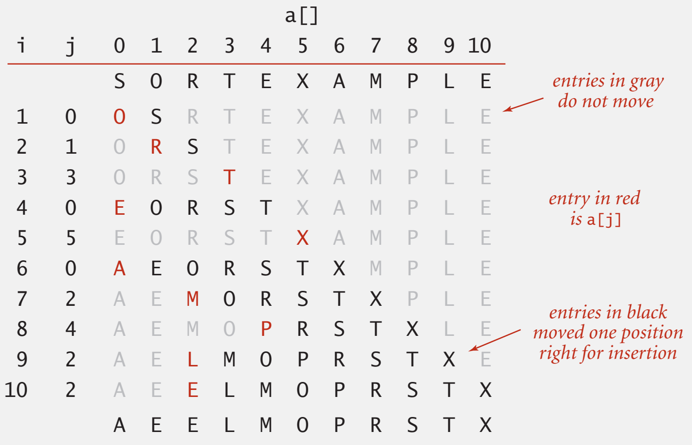
▬ not yet seen

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

Insertion sort: mathematical analysis

Proposition. To sort a randomly-ordered array with distinct keys, insertion sort uses $\sim \frac{1}{4} N^2$ compares and $\sim \frac{1}{4} N^2$ exchanges on average.

Pf. Expect each entry to move halfway back.



		a[]										
i	j	0	1	2	3	4	5	6	7	8	9	10
		S	O	R	T	E	X	A	M	P	L	E
1	0	O	S	R	T	E	X	A	M	P	L	E
2	1	O	R	S	T	E	X	A	M	P	L	E
3	3	O	R	S	T	E	X	A	M	P	L	E
4	0	E	O	R	S	T	X	A	M	P	L	E
5	5	E	O	R	S	T	X	A	M	P	L	E
6	0	A	E	O	R	S	T	X	M	P	L	E
7	2	A	E	M	O	R	S	T	X	P	L	E
8	4	A	E	M	O	P	R	S	T	X	L	E
9	2	A	E	L	M	O	P	R	S	T	X	E
10	2	A	E	E	L	M	O	P	R	S	T	X
		A	E	E	L	M	O	P	R	S	T	X

entries in gray do not move

entry in red is a[j]

entries in black moved one position right for insertion

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

Elementary sorts: quiz 2

How many compares does insertion sort make to sort an array of N distinct keys in reverse order?

A. $\sim N$

B. $\sim 1/4\ N^2$

C. $\sim 1/2\ N^2$

D. $\sim N^2$

E. *I don't know.*

Insertion sort: analysis

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

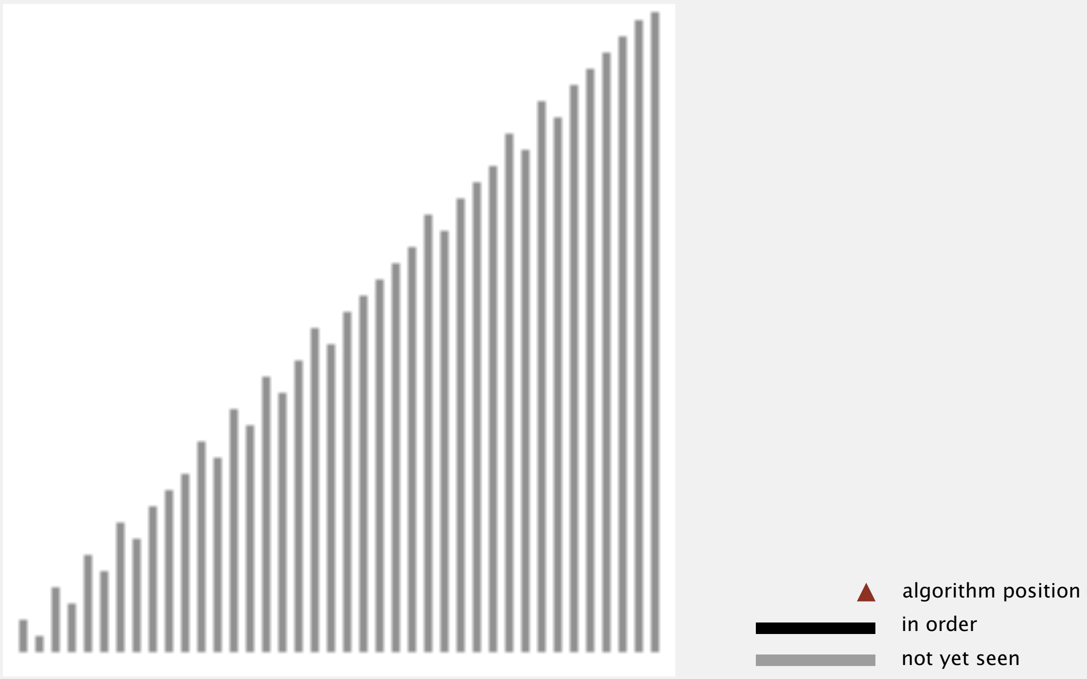
X T S R P O M L F E A

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

Insertion sort: animation

40 partially-sorted items



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 $\le c\,N$.

- Ex 1. A sorted array has 0 inversions.
- Ex 2. A subarray of size 10 appended to a sorted subarray of size N.

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

Pf. Number of exchanges equals the number of inversions.

number of compares $\le$ exchanges + (N – 1)

Insertion sort: practical improvements

Half exchanges. Shift items over (instead of exchanging).

- Eliminates unnecessary data movement.
- No longer uses only `less()` and `exch()` to access data.

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

Binary insertion sort. Use binary search to find insertion point.

- Number of compares $\sim N \lg N$.
- But still a quadratic number of array accesses.

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

binary search for first key > K

Elementary sorts: quiz 3

Which is faster in practice, selection sort or insertion sort?

A. Selection sort.

B. Insertion sort.

C. No significant difference.

D. *I don't know.*

http://algs4.cs.princeton.edu

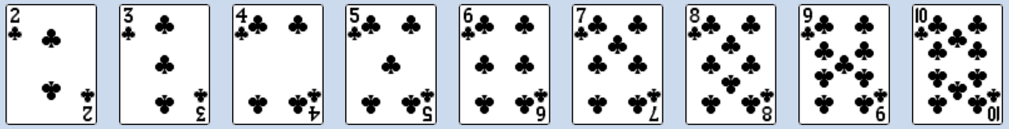
2.1 Elementary Sorts

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

Interview question: shuffle an array

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

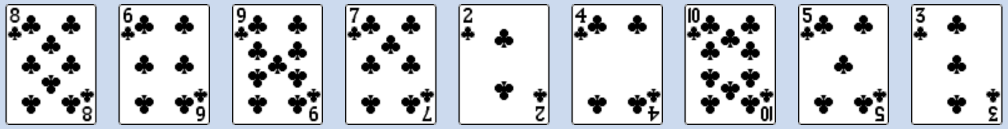
all N! permutations equally likely



Interview question: shuffle an array

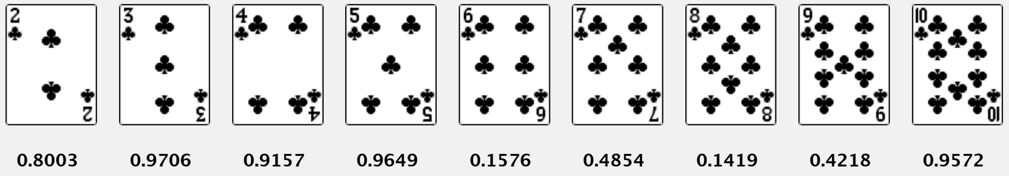
Goal. Rearrange array so that result is a uniformly random permutation.

all N! permutations equally likely



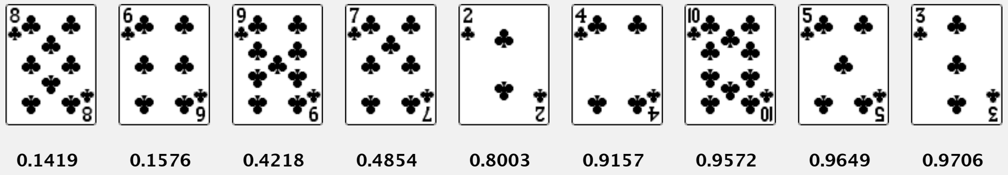
Shuffle sort

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



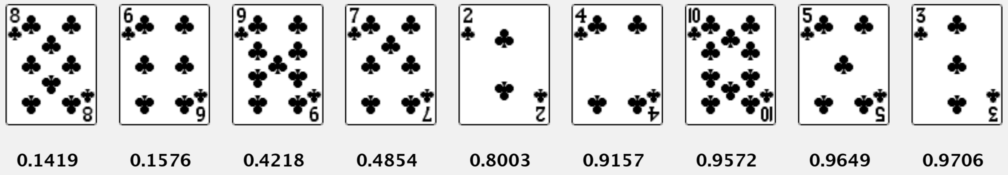
Shuffle sort

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



Shuffle sort

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



Proposition. Shuffle sort produces a uniformly random permutation.

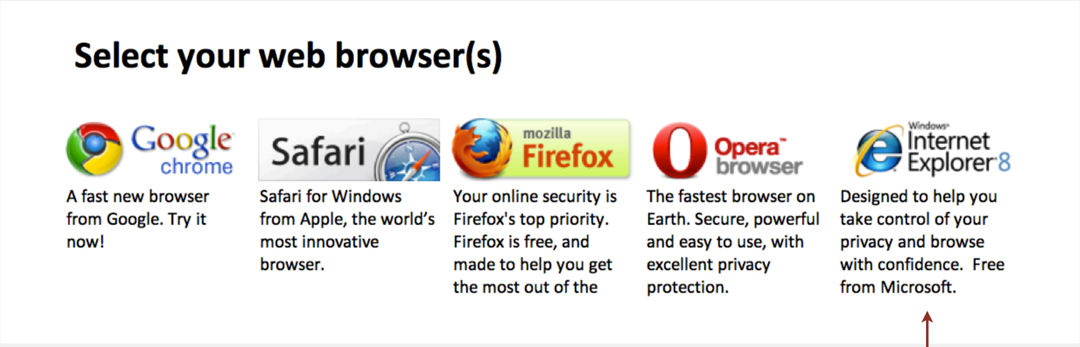
assuming real numbers are uniformly random (and no ties)

Application. Shuffle columns in a spreadsheet.

War story (Microsoft)

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

http://www.browserchoice.eu



Select your web browser(s)

A fast new browser from Google. Try it now!

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

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

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

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

appeared last 50% of the time

War story (Microsoft)

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

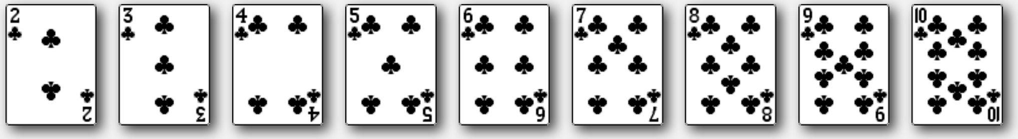
Solution? Implement shuffle sort by making comparator always return a random answer.

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

browser comparator (should implement a total order)

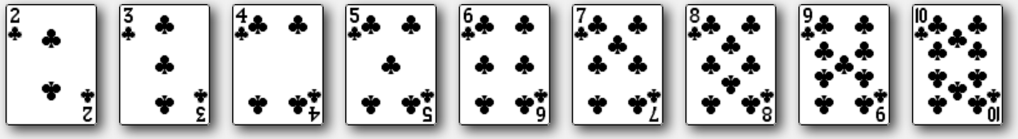
Knuth shuffle demo

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



Knuth shuffle

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



Proposition. [Fisher-Yates 1938] Knuth shuffling algorithm produces a uniformly random permutation of the input array in linear time.

assuming integers uniformly at random

Knuth shuffle

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

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

```
public class Knuth
{
   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

http://algs4.cs.princeton.edu/11model/Knuth.java.html

Broken Knuth shuffle

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

A. Not uniformly random!

instead of between 0 and i

permutation	Knuth shuffle	broken shuffle
A B C	1/6	4/27
A C B	1/6	5/27
B A C	1/6	5/27
B C A	1/6	5/27
C A B	1/6	4/27
C B A	1/6	4/27

probability of each permutation when shuffling { A, B, C }

War story (online poker)

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

mamajoe: Hey guys, Big B is in!

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

http://www.cigital.com/papers/download/developer_gambling.php

War story (online poker)

Shuffling algorithm in FAQ at www.planetpoker.com

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

between 1 and 51

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

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

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

Bug 4. Seed = milliseconds since midnight ⇒ 86.4 million shuffles.

> *“ The generation of random numbers is too important to be left to chance. ”*
>
> *— Robert R. Coveyou*

War story (online poker)

Best practices for shuffling (if your business depends on it).

- Use a hardware random-number generator that has passed both the FIPS 140-2 and the NIST statistical test suites.
- Continuously monitor statistic properties: hardware random-number generators are fragile and fail silently.
- Use an unbiased shuffling algorithm.

RANDOM.ORG

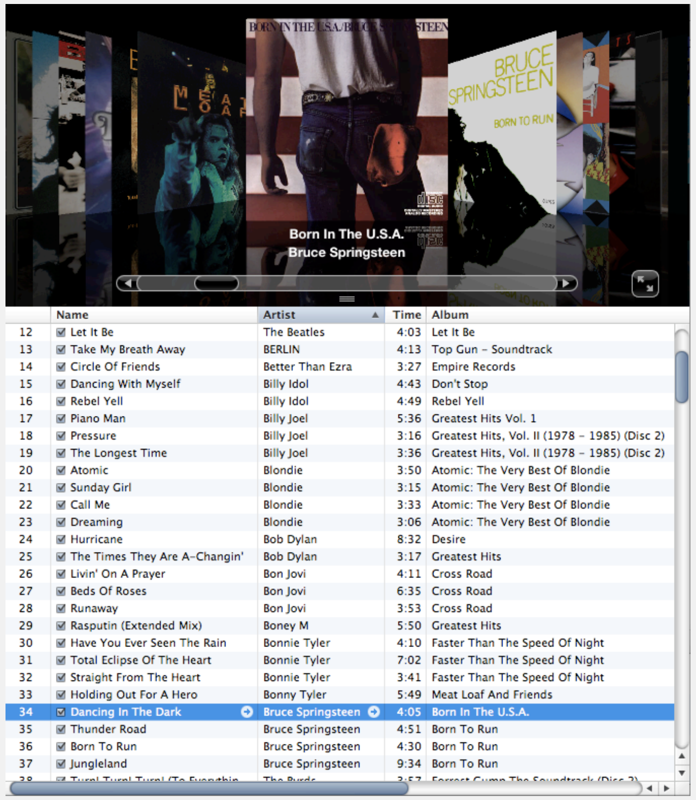
Bottom line. Shuffling a deck of cards is hard!

http://algs4.cs.princeton.edu

2.1 Elementary Sorts

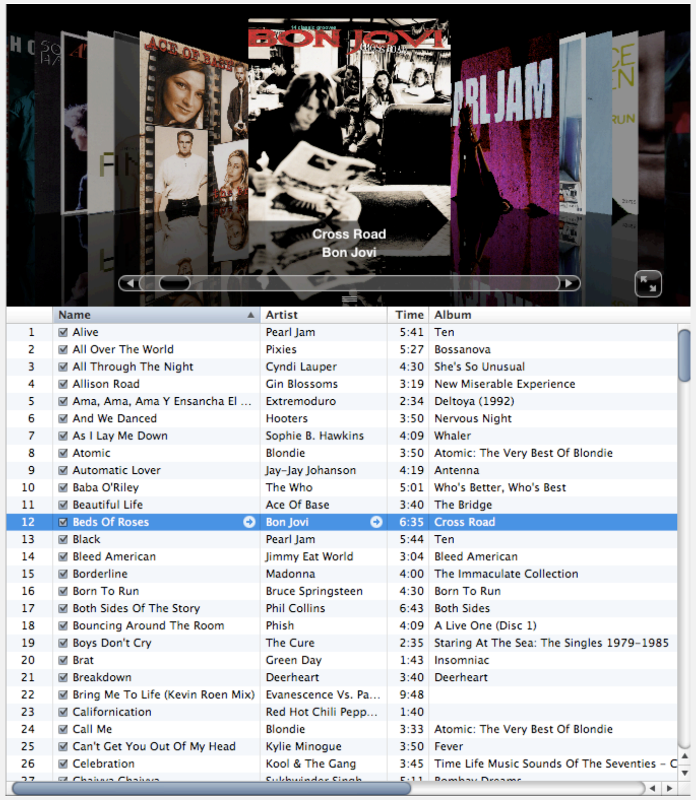
- rules of the game
- selection sort
- insertion sort
- shuffling
- *comparators*

Sort music library by artist



	Name	Artist ▲	Time	Album
12	☑ Let It Be	The Beatles	4:03	Let It Be
13	☑ Take My Breath Away	BERLIN	4:13	Top Gun – Soundtrack
14	☑ Circle Of Friends	Better Than Ezra	3:27	Empire Records
15	☑ Dancing With Myself	Billy Idol	4:43	Don't Stop
16	☑ Rebel Yell	Billy Idol	4:49	Rebel Yell
17	☑ Piano Man	Billy Joel	5:36	Greatest Hits Vol. 1
18	☑ Pressure	Billy Joel	3:16	Greatest Hits, Vol. II (1978 – 1985) (Disc 2)
19	☑ The Longest Time	Billy Joel	3:36	Greatest Hits, Vol. II (1978 – 1985) (Disc 2)
20	☑ Atomic	Blondie	3:50	Atomic: The Very Best Of Blondie
21	☑ Sunday Girl	Blondie	3:15	Atomic: The Very Best Of Blondie
22	☑ Call Me	Blondie	3:33	Atomic: The Very Best Of Blondie
23	☑ Dreaming	Blondie	3:06	Atomic: The Very Best Of Blondie
24	☑ Hurricane	Bob Dylan	8:32	Desire
25	☑ The Times They Are A-Changin'	Bob Dylan	3:17	Greatest Hits
26	☑ Livin' On A Prayer	Bon Jovi	4:11	Cross Road
27	☑ Beds Of Roses	Bon Jovi	6:35	Cross Road
28	☑ Runaway	Bon Jovi	3:53	Cross Road
29	☑ Rasputin (Extended Mix)	Boney M	5:50	Greatest Hits
30	☑ Have You Ever Seen The Rain	Bonnie Tyler	4:10	Faster Than The Speed Of Night
31	☑ Total Eclipse Of The Heart	Bonnie Tyler	7:02	Faster Than The Speed Of Night
32	☑ Straight From The Heart	Bonnie Tyler	3:41	Faster Than The Speed Of Night
33	☑ Holding Out For A Hero	Bonny Tyler	5:49	Meat Loaf And Friends
34	☑ Dancing In The Dark	Bruce Springsteen	4:05	Born In The U.S.A.
35	☑ Thunder Road	Bruce Springsteen	4:51	Born To Run
36	☑ Born To Run	Bruce Springsteen	4:30	Born To Run
37	☑ Jungleland	Bruce Springsteen	9:34	Born To Run
38	☑ Turn! Turn! Turn! (To Everythin	The Byrds	3:57	Forrest Gump The Soundtrack (Disc 2)

Sort music library by song name



	Name ▲	Artist	Time	Album
1	☑ Alive	Pearl Jam	5:41	Ten
2	☑ All Over The World	Pixies	5:27	Bossanova
3	☑ All Through The Night	Cyndi Lauper	4:30	She's So Unusual
4	☑ Allison Road	Gin Blossoms	3:19	New Miserable Experience
5	☑ Ama, Ama, Ama Y Ensancha El ...	Extremoduro	2:34	Deltoya (1992)
6	☑ And We Danced	Hooters	3:50	Nervous Night
7	☑ As I Lay Me Down	Sophie B. Hawkins	4:09	Whaler
8	☑ Atomic	Blondie	3:50	Atomic: The Very Best Of Blondie
9	☑ Automatic Lover	Jay-Jay Johanson	4:19	Antenna
10	☑ Baba O'Riley	The Who	5:01	Who's Better, Who's Best
11	☑ Beautiful Life	Ace Of Base	3:40	The Bridge
12	☑ Beds Of Roses	Bon Jovi	6:35	Cross Road
13	☑ Black	Pearl Jam	5:44	Ten
14	☑ Bleed American	Jimmy Eat World	3:04	Bleed American
15	☑ Borderline	Madonna	4:00	The Immaculate Collection
16	☑ Born To Run	Bruce Springsteen	4:30	Born To Run
17	☑ Both Sides Of The Story	Phil Collins	6:43	Both Sides
18	☑ Bouncing Around The Room	Phish	4:09	A Live One (Disc 1)
19	☑ Boys Don't Cry	The Cure	2:35	Staring At The Sea: The Singles 1979-1985
20	☑ Brat	Green Day	1:43	Insomniac
21	☑ Breakdown	Deerheart	3:40	Deerheart
22	☑ Bring Me To Life (Kevin Roen Mix)	Evanescence Vs. Pa...	9:48	
23	☑ Californication	Red Hot Chili Pepp...	1:40	
24	☑ Call Me	Blondie	3:33	Atomic: The Very Best Of Blondie
25	☑ Can't Get You Out Of My Head	Kylie Minogue	3:50	Fever
26	☑ Celebration	Kool & The Gang	3:45	Time Life Music Sounds Of The Seventies – C
27	☑ Chaiyya Chaiyya	Sukhwinder Singh	5:11	Bombay Dreams

Comparable interface: review

Comparable interface: sort using a type's natural order.

```
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;
   }
}
```

natural order

Comparator interface

Comparator interface: sort using an alternate order.

```
public interface Comparator<Item>
{
   public int compare(Item v, Item w);
}
```

Required property. Must be a total order.

string order	example
natural order	Now is the time
case insensitive	is Now the time
Spanish language	café cafetero cuarto churro nube ñoño
British phone book	McKinley Mackintosh

pre-1994 order for digraphs ch and ll and rr

Comparator interface: system sort

To use with Java system sort:

- Create `Comparator` object.
- Pass as second argument to `Arrays.sort()`.

uses natural order

uses alternate order defined by `Comparator<String>` object

```
String[] a;
...
Arrays.sort(a);
...
Arrays.sort(a, String.CASE_INSENSITIVE_ORDER);
...
Arrays.sort(a, Collator.getInstance(new Locale("es")));
...
Arrays.sort(a, new BritishPhoneBookOrder());
...
```

Bottom line. Decouples the definition of the data type from the definition of what it means to compare two objects of that type.

Comparator interface: using with our sorting libraries

To support comparators in our sort implementations:

- Pass `Comparator` to both `sort()` and `less()`, and use it in `less()`.
- Use `Object` instead of `Comparable`.

```
import java.util.Comparator;

public class Insertion
{
   ...

   public static void sort(Object[] a, Comparator comparator)
   {
      int N = a.length;
      for (int i = 0; i < N; i++)
         for (int j = i; j > 0 && less(comparator, a[j], a[j-1]); j--)
            exch(a, j, j-1);
   }

   private static boolean less(Comparator comparator, Object v, Object w)
   {  return comparator.compare(v, w) < 0;  }
}
```

http://algs4.cs.princeton.edu/21elementary/Insertion.java.html

Comparator interface: implementing

To implement a comparator:

- Define a (nested) class that implements the `Comparator` interface.
- Implement the `compare()` method.
- Provide client access to `Comparator`.

```
import java.util.Comparator;

public class Student
{
   private final String name;
   private final int section;
   ...

   public static Comparator<Student> byNameOrder()
   {  return new NameOrder();  }

   private static class NameOrder implements Comparator<Student>
   {
      public int compare(Student v, Student w)
      {  return v.name.compareTo(w.name);  }
   }
   ...
}
```

one Comparator for the class

Comparator interface: implementing

To implement a comparator:

- Define a (nested) class that implements the `Comparator` interface.
- Implement the `compare()` method.
- Provide client access to `Comparator`.

```
import java.util.Comparator;

public class Student
{
   private final String name;
   private final int section;
   ...

   public static Comparator<Student> bySectionOrder()
   {  return new SectionOrder();  }

   private static class SectionOrder implements Comparator<Student>
   {
      public int compare(Student v, Student w)
      {  return v.section - w.section;  }
   }
   ...
}
```

this trick works here since no danger of overflow

Comparator interface: implementing

To implement a comparator:

- Define a (nested) class that implements the `Comparator` interface.
- Implement the `compare()` method.
- Provide client access to `Comparator`.

`Insertion.sort(a, Student.byNameOrder());`

Andrews	3	A	(664) 480-0023	097 Little
Battle	4	C	(874) 088-1212	121 Whitman
Chen	3	A	(991) 878-4944	308 Blair
Fox	3	A	(884) 232-5341	11 Dickinson
Furia	1	A	(766) 093-9873	101 Brown
Gazsi	4	B	(800) 867-5309	101 Brown
Kanaga	3	B	(898) 122-9643	22 Brown
Rohde	2	A	(232) 343-5555	343 Forbes

`Insertion.sort(a, Student.bySectionOrder());`

Furia	1	A	(766) 093-9873	101 Brown
Rohde	2	A	(232) 343-5555	343 Forbes
Andrews	3	A	(664) 480-0023	097 Little
Chen	3	A	(991) 878-4944	308 Blair
Fox	3	A	(884) 232-5341	11 Dickinson
Kanaga	3	B	(898) 122-9643	22 Brown
Battle	4	C	(874) 088-1212	121 Whitman
Gazsi	4	B	(800) 867-5309	101 Brown

Stability

A typical application. First, sort by name; then sort by section.

`Selection.sort(a, Student.byNameOrder());`

Andrews	3	A	(664) 480-0023	097 Little
Battle	4	C	(874) 088-1212	121 Whitman
Chen	3	A	(991) 878-4944	308 Blair
Fox	3	A	(884) 232-5341	11 Dickinson
Furia	1	A	(766) 093-9873	101 Brown
Gazsi	4	B	(800) 867-5309	101 Brown
Kanaga	3	B	(898) 122-9643	22 Brown
Rohde	2	A	(232) 343-5555	343 Forbes

`Selection.sort(a, Student.bySectionOrder());`

Furia	1	A	(766) 093-9873	101 Brown
Rohde	2	A	(232) 343-5555	343 Forbes
Chen	3	A	(991) 878-4944	308 Blair
Fox	3	A	(884) 232-5341	11 Dickinson
Andrews	3	A	(664) 480-0023	097 Little
Kanaga	3	B	(898) 122-9643	22 Brown
Gazsi	4	B	(800) 867-5309	101 Brown
Battle	4	C	(874) 088-1212	121 Whitman

@#%&@! Students in section 3 no longer sorted by name.

A stable sort preserves the relative order of items with equal keys.

Elementary sorts: quiz 4

Which sorting algorithms are stable?

A. Selection sort.

B. Insertion sort.

C. Both A and B.

D. Neither A nor B.

E. *I don't know.*

Stability: insertion sort

Proposition. Insertion sort is stable.

```
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 && less(a[j], a[j-1]); j--)
            exch(a, j, j-1);
   }
}
```

i	j	0	1	2	3	4
0	0	B_1	A_1	A_2	A_3	B_2
1	0	A_1	B_1	A_2	A_3	B_2
2	1	A_1	A_2	B_1	A_3	B_2
3	2	A_1	A_2	A_3	B_1	B_2
4	4	A_1	A_2	A_3	B_1	B_2
		A_1	A_2	A_3	B_1	B_2

Pf. Equal items never move past each other.

Stability: selection sort

Proposition. Selection sort is not stable.

```
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);
      }
   }
}
```

i	min	0	1	2
0	2	B_1	B_2	A
1	1	A	B_2	B_1
2	2	A	B_2	B_1
		A	B_2	B_1

Pf by counterexample. Long-distance exchange can move one equal item past another one.