



<http://algs4.cs.princeton.edu>

2.1 ELEMENTARY SORTS

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



<http://algs4.cs.princeton.edu>

2.1 ELEMENTARY SORTS

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

Sorting problem

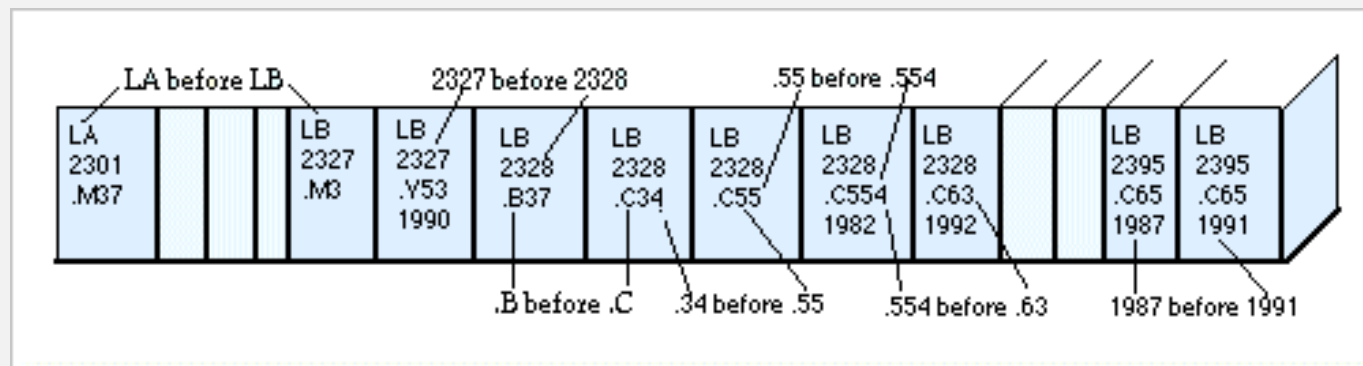
Ex. Student records in a university.

	Chen	3	A	991-878-4944	308 Blair
	Rohde	2	A	232-343-5555	343 Forbes
	Gazsi	4	B	766-093-9873	101 Brown
item →	Furia	1	A	766-093-9873	101 Brown
	Kanaga	3	B	898-122-9643	22 Brown
	Andrews	3	A	664-480-0023	097 Little
key →	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

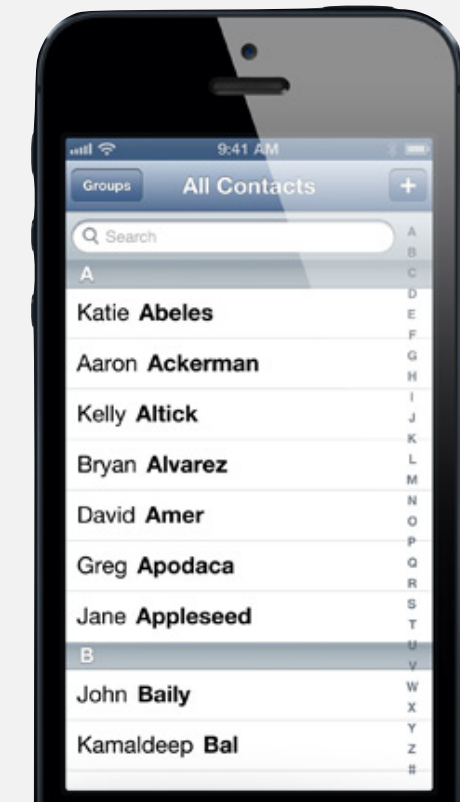
Sorting applications



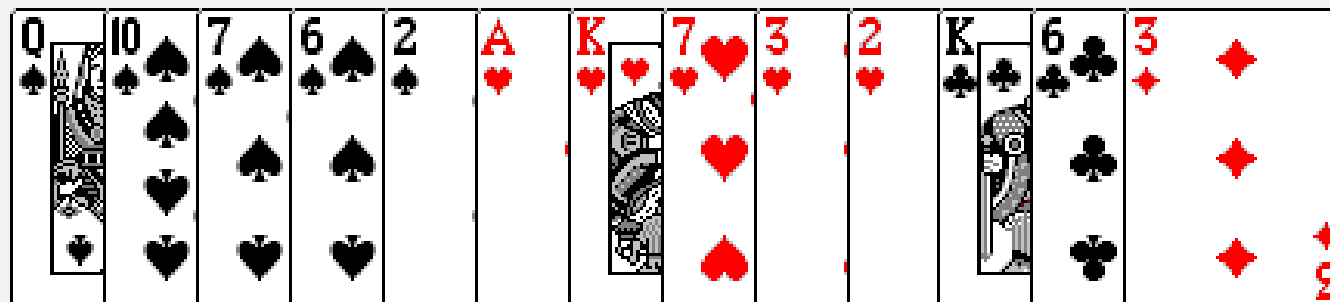
Library of Congress numbers



FedEx packages



contacts



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 \leq that satisfies:

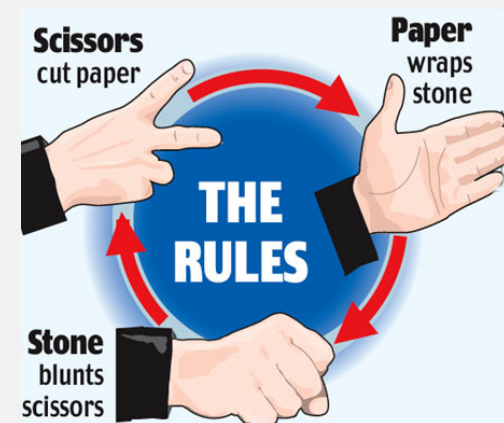
- Antisymmetry: if both $v \leq w$ and $w \leq v$, then $v = w$.
- Transitivity: if both $v \leq w$ and $w \leq x$, then $v \leq x$.
- Totality: either $v \leq w$ or $w \leq v$ or both.

Ex.

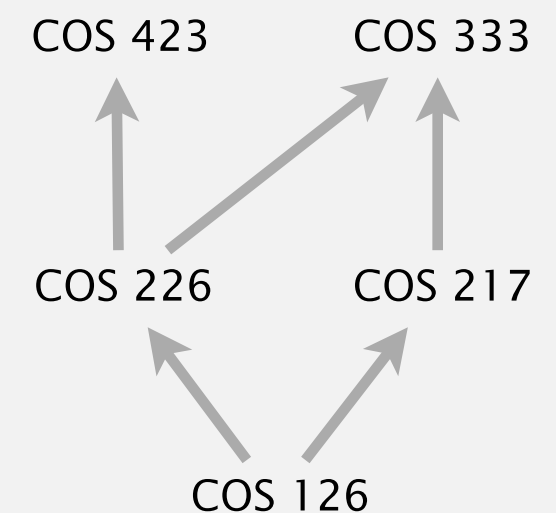
- Standard order for natural and real numbers.
- Chronological order for dates or times.
- Alphabetical order for strings.

No transitivity. Rock-paper-scissors.

No totality. PU course prerequisites.



violates transitivity



violates totality

Callbacks

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

Q. How can `sort()` know how to compare data of type `Double`, `String`, and `java.io.File` without any information about the type of an item's key?

Callback = reference to executable code.

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

Implementing callbacks.

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

Callbacks: roadmap

client

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

data-type implementation

```
public class String
implements Comparable<String>
{
    ...
    public int compareTo(String b)
    {
        ...
        return -1;
        ...
        return +1;
        ...
        return 0;
    }
}
```

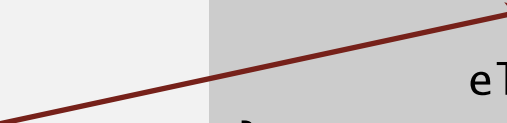
Comparable interface (built in to Java)

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

sort implementation

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

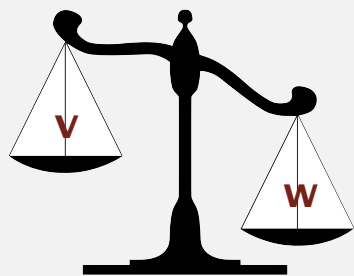
key point: no dependence
on String data type



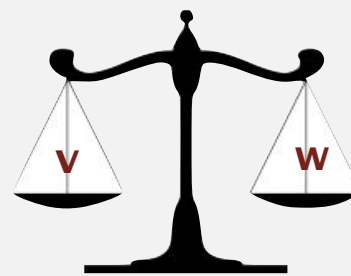
Comparable API

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

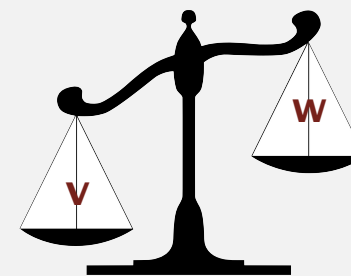
- Defines a total order.
- Returns a negative integer, zero, or positive integer if v is less than, equal to, or greater than w , respectively.
- Throws an exception if incompatible types (or either is `null`).



less than (return -1)



equal to (return 0)



greater than (return +1)

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

User-defined comparable types. Implement the Comparable interface.

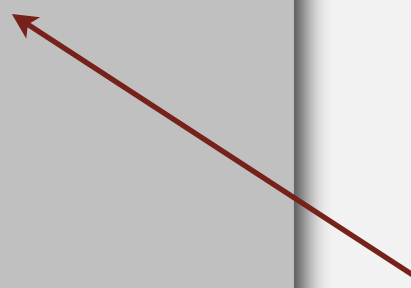
Implementing the Comparable interface

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

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

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

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



only compare dates
to other dates