Sortiert werden soll eine **Liste** mit **Elementen** nach einem bestimmten **Schlüssel**
Wenn die Elemente anhand des Schlüssels in einer **Reihenfolge** sind, ist die Liste **sortiert**.

Aubine	Boice	4	aboice3@si.edu	Female	168.71.9.183
Ariela	Dearle	12	adearleb@eventbrite.com	Female	173.134.1.151
Adeline	McLeish	18	amcleishh@zimbio.com	Female	175.230.40.94
Bernice	Andrzejczak	13	bandrzejczakc@squidoo.com	Female	98.126.168.21
Burty	Deinhardt	5	bdeinhardt4@pcworld.com	Male	126.152.85.239
Carter	Donet	2	cdonet1@unc.edu	Genderqueer	56.47.22.73
Derrick	Kropp	14	dkroppd@icq.com	Male	172.189.31.27
Ellary	Pask	8	epask7@ebay.com	Male	212.220.123.92
Hollie	Elijah	9	helijah8@indiegogo.com	Female	4.105.153.54

Adeline	Boice	4	aboice3@si.edu	Female	168.71.9.183
Ariela	Bartlett	3	lbartlett2@technorati.com	Female	208.201.22.15
Aubine	Andrzejczak	13	bandrzejczakc@squidoo.com	Female	98.126.168.21
Bernice	Curtoys	15	lcurtoyse@desdev.cn	Female	92.18.63.145
Burty	Dearle	12	adearleb@eventbrite.com	Female	173.134.1.151
Carter	Deinhardt	5	bdeinhardt4@pcworld.com	Male	126.152.85.239
Derrick	Donet	2	cdonet1@unc.edu	Genderqueer	56.47.22.73
Ellary	Elijah	9	helijah8@indiegogo.com	Female	4.105.153.54
Hollie	Howes	7	khowes6@mozilla.org	Female	77.185.37.165
Jojo	Kropp	14	dkroppd@icq.com	Male	172.189.31.27
Kai	McReice	19	lmcreicei@sciencedaily.com	Female	114.35.75.115

Verschiedene Schlüssel → verschiedene Reihenfolgen

Adeline	Boice	4	aboice3@si.edu	Female	168.71.9.183
Burty	Dearle	12	adearleb@eventbrite.com	Female	173.134.1.151
Karla	McLeish	18	amcleishh@zimbio.com	Female	175.230.40.94
Aubine	Andrzejczak	13	bandrzejczakc@squidoo.com	Female	98.126.168.21
Carter	Deinhardt	5	bdeinhardt4@pcworld.com	Male	126.152.85.239
Derrick	Donet	2	cdonet1@unc.edu	Genderqueer	56.47.22.73
Jojo	Kropp	14	dkroppd@icq.com	Male	172.189.31.27
Lavina	Pask	8	epask7@ebay.com	Male	212.220.123.92
Ellary	Elijah	9	helijah8@indiegogo.com	Female	4.105.153.54
Ronda	Shakespear	11	jshakespeara@google.ru	Female	225.177.127.18
Hollie	Howes	7	khowes6@mozilla.org	Female	77.185.37.165

Linnet	Pettigrew	16	rpettigrewf@usnews.com	Female	109.148.204.18
Kai	McPeice	19	lmcpeicei@sciencedaily.com	Female	114.35.75.115
Carter	Deinhardt	5	bdeinhardt4@pcworld.com	Male	126.152.85.239
Lindie	Pettie	6	mpettie5@blog.com	Female	161.132.43.23
Adeline	Boice	4	aboice3@si.edu	Female	168.71.9.183
Jojo	Kropp	14	dkroppd@icq.com	Male	172.189.31.27
Burty	Dearle	12	adearleb@eventbrite.com	Female	173.134.1.151
Karla	McLeish	18	amcleishh@zimbio.com	Female	175.230.40.94
Ariela	Bartlett	3	lbartlett2@technorati.com	Female	208.201.22.15
Lavina	Pask	8	epask7@ebay.com	Male	212.220.123.92
Ronda	Shakespear	11	jshakespeara@google.ru	Female	225.177.127.18

Sortierproblem ist universell

```
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
```

```
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
```

```
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
```

Alle Listen wurden mit derselben Methode sortiert….

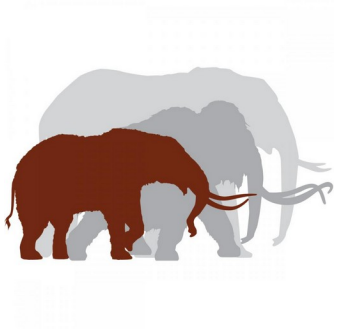
`Insertion.sort(array)`

Comparable Interface

In Java „eingebaut“

```
public interface Comparable<T> {
    public int compareTo(T vergleichsobjekt)
}
```

```
public class Ruesseltier implements Comparable<Ruesseltier>
{
...
   public int compareTo(Ruesseltier r){
   ...
    Return -1; // wenn r > als das aktuelle Objekt
   ...
    Return +1; // wenn r < als das aktuelle Objekt
   ...
    Return 0;  // wenn r gleich wie das aktuelle Objekt
   }
}
```



`Elefant.compareTo(Mastodon)`

→ Gibt -1 zurück, wenn Elefant kleiner als Mastodon

→ Gibt +1 zurück, wenn Elefant größer als Mastodon.

→ Gibt 0 zurück, wenn beide gleich groß sind.

Regeln… → Hilfsfunktionen

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

```
private void exch(Comparable[] a, int i, int j)
{
    Comparable swap = a[i];
    a[i] = a[j];
    a[j] = swap;
}
```

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