2.1 Elementary Sorts

- rules of the game
- selection sort
- insertion sort
- binary search
- comparators

stability see precept
2.1 Elementary Sorts

- rules of the game
- selection sort
- insertion sort
- binary search
- comparators
- stability
**Sorting problem**

**Goal.** Rearrange array of \( n \) items in ascending order by key.

<table>
<thead>
<tr>
<th>Last</th>
<th>First</th>
<th>House</th>
<th>Year</th>
</tr>
</thead>
<tbody>
<tr>
<td>Longbottom</td>
<td>Neville</td>
<td>Gryffindor</td>
<td>1998</td>
</tr>
<tr>
<td>Weasley</td>
<td>Ron</td>
<td>Gryffindor</td>
<td>1998</td>
</tr>
<tr>
<td>Abbott</td>
<td>Hannah</td>
<td>Hufflepuff</td>
<td>1998</td>
</tr>
<tr>
<td><strong>Potter</strong></td>
<td>Harry</td>
<td>Gryffindor</td>
<td>1998</td>
</tr>
<tr>
<td>Chang</td>
<td>Cho</td>
<td>Ravenclaw</td>
<td>1997</td>
</tr>
<tr>
<td>Granger</td>
<td>Hermione</td>
<td>Gryffindor</td>
<td>1998</td>
</tr>
<tr>
<td><strong>Malfoy</strong></td>
<td>Draco</td>
<td>Slytherin</td>
<td>1998</td>
</tr>
<tr>
<td>Diggory</td>
<td>Cedric</td>
<td>Hufflepuff</td>
<td>1996</td>
</tr>
<tr>
<td>Weasley</td>
<td>Ginny</td>
<td>Gryffindor</td>
<td>1999</td>
</tr>
<tr>
<td>Parkinson</td>
<td>Pansy</td>
<td>Slytherin</td>
<td>1998</td>
</tr>
</tbody>
</table>

*sorting hat (now running JDK 11)*
Sorting problem

**Goal.** Rearrange array of $n$ items in ascending order by key.

<table>
<thead>
<tr>
<th>Last</th>
<th>First</th>
<th>House</th>
<th>Year</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abbott</td>
<td>Hannah</td>
<td>Hufflepuff</td>
<td>1998</td>
</tr>
<tr>
<td>Chang</td>
<td>Cho</td>
<td>Ravenclaw</td>
<td>1997</td>
</tr>
<tr>
<td>Granger</td>
<td>Hermione</td>
<td>Gryffindor</td>
<td>1998</td>
</tr>
<tr>
<td>Diggory</td>
<td>Cedric</td>
<td>Hufflepuff</td>
<td>1996</td>
</tr>
<tr>
<td>Longbottom</td>
<td>Neville</td>
<td>Gryffindor</td>
<td>1998</td>
</tr>
<tr>
<td>Malfoy</td>
<td>Draco</td>
<td>Slytherin</td>
<td>1998</td>
</tr>
<tr>
<td>Parkinson</td>
<td>Pansy</td>
<td>Slytherin</td>
<td>1998</td>
</tr>
<tr>
<td>Potter</td>
<td>Harry</td>
<td>Gryffindor</td>
<td>1998</td>
</tr>
<tr>
<td>Weasley</td>
<td>Ron</td>
<td>Gryffindor</td>
<td>1998</td>
</tr>
<tr>
<td>Weasley</td>
<td>Ginny</td>
<td>Gryffindor</td>
<td>1999</td>
</tr>
</tbody>
</table>

[sorting hat (now running JDK 11)]
Total preorder

Sorting is a well-defined problem if there is a total preorder.

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

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

**Examples.**

<table>
<thead>
<tr>
<th>Video name</th>
<th>Views (billions)</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Despacito&quot;[23]</td>
<td>6.96</td>
</tr>
<tr>
<td>&quot;Baby Shark Dance&quot;[28]</td>
<td>6.55</td>
</tr>
<tr>
<td>&quot;Shape of You&quot;[29]</td>
<td>4.97</td>
</tr>
<tr>
<td>&quot;See You Again&quot;[30]</td>
<td>4.72</td>
</tr>
<tr>
<td>&quot;Masha and the Bear – Recipe for&quot;</td>
<td>4.33</td>
</tr>
<tr>
<td>&quot;Uptown Funk&quot;[38]</td>
<td>3.94</td>
</tr>
</tbody>
</table>

**Numerical order (descending)**

**Chronological order**

**Lexicographic order**
Total preorder

Sorting is a well-defined problem if there is a total preorder.

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

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

Non-examples.

course prerequisites (violates totality)

Ro–sham–bo order (violates transitivity)

the $\leq$ operator for double (violates totality)
Goal. Single function that sorts any type of data (that has a total preorder).

Ex 1. Sort strings in alphabetical order.

```java
public class StringSorter {
    public static void main(String[] args) {
        String[] a = StdIn.readStringAllStrings();
        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 clients

**Goal.** Single function that sorts any type of data (that has a total preorder).

**Ex 2.** Sort real numbers in ascending order.

```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 clients

**Goal.** Single function that sorts any type of data (that has a total preorder).

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

```java
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
How can a single function sort any type of data?

**Goal.** Single function that sorts any type of data (that has a total preorder).

**Solution.** Callback = reference to executable code.

Please sort these Japanese names for me: あゆみ, アユミ, Ayumi, 歩美, ....

But I don't speak Japanese and I don't know how words are ordered.

No problem. Whenever you need to compare two words, give me a call back.

オーケー. Just make sure to use a total preorder.
Callbacks

**Goal.** Single function that sorts any type of data (that has a total preorder).

**Solution.** 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.
Java interfaces

**Interface.** A set of methods that define some behavior (partial API) for a class.

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

Class that implements interface. Must implement all interface methods.

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

**Enforcement.** Compile-time error if a class fails to define the requisite methods.
Callbacks in Java: roadmap

client (StringSorter.java)

```java
public class StringSorter {
    public static void main(String[] args) {
        String[] a = StdIn.readStringArray();
        Insertion.sort(a);
        ...
    }
}
```

sort implementation (Insertion.java)

```java
public class Insertion {
    public static void sort(Comparable[] a) {
        ...
        if (a[i].compareTo(a[j]) < 0) {
            ...
        }
    }
}
```

data type implementation (String.java)

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

key point: client code does not depend upon type of data to be sorted

java.lang.Comparable interface

```java
public interface Comparable<Item> {
    public int compareTo(Item that);
}
```
Suppose that the Java architects left 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 an exception.
Comparable API

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

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

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;
    }
}
```

(can compare Date objects only to other Date objects)

https://algs4.cs.princeton.edu/12oop/Date.java.html
2.1 Elementary Sorts

- rules of the game
- selection sort
- insertion sort
- binary search
- comparators
- stability
Selection sort demo

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

initial array
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 primitives (and a cost model)

**Helper functions.** Refer to data only through `compares` and `exchanges`.

use as our cost model for sorting

**Compare.** Is item v less than w?

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

polymorphic method call

**Exchange.** Swap array entries a[i] and a[j].

```java
private static void exch(Comparable[] a, int i, int j)
{
    Comparable swap = a[i];
    a[i] = a[j];
    a[j] = swap;
}
```
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) {
        /* see previous slide */
    }

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

https://algs4.cs.princeton.edu/21elementary/Selection.java.html
Selection sort: animations

20 random items

http://www.sorting-algorithms.com/selection-sort
Elementary sorts: quiz 2

How many compares to selection sort an array of $n$ distinct items in reverse order?

A. $\sim n$
B. $\sim \frac{1}{4} n^2$
C. $\sim \frac{1}{2} n^2$
D. $\sim n^2$
Selection sort: mathematical analysis

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

<table>
<thead>
<tr>
<th>(a_i)</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>(i)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(\min)</td>
<td>S</td>
<td>O</td>
<td>R</td>
<td>T</td>
<td>E</td>
<td>X</td>
<td>A</td>
<td>M</td>
<td>P</td>
<td>L</td>
<td>E</td>
</tr>
<tr>
<td>0</td>
<td>6</td>
<td>S</td>
<td>O</td>
<td>R</td>
<td>T</td>
<td>E</td>
<td>X</td>
<td>A</td>
<td>M</td>
<td>P</td>
<td>L</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>A</td>
<td>O</td>
<td>R</td>
<td>T</td>
<td>E</td>
<td>X</td>
<td>S</td>
<td>M</td>
<td>P</td>
<td>L</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>A</td>
<td>E</td>
<td>R</td>
<td>T</td>
<td>O</td>
<td>X</td>
<td>S</td>
<td>M</td>
<td>P</td>
<td>L</td>
</tr>
<tr>
<td>3</td>
<td>9</td>
<td>A</td>
<td>E</td>
<td>E</td>
<td>T</td>
<td>O</td>
<td>X</td>
<td>S</td>
<td>M</td>
<td>P</td>
<td>L</td>
</tr>
<tr>
<td>4</td>
<td>7</td>
<td>A</td>
<td>E</td>
<td>E</td>
<td>L</td>
<td>O</td>
<td>X</td>
<td>S</td>
<td>M</td>
<td>P</td>
<td>T</td>
</tr>
<tr>
<td>5</td>
<td>7</td>
<td>A</td>
<td>E</td>
<td>E</td>
<td>L</td>
<td>M</td>
<td>X</td>
<td>S</td>
<td>O</td>
<td>P</td>
<td>T</td>
</tr>
<tr>
<td>6</td>
<td>8</td>
<td>A</td>
<td>E</td>
<td>E</td>
<td>L</td>
<td>M</td>
<td>O</td>
<td>S</td>
<td>X</td>
<td>P</td>
<td>T</td>
</tr>
<tr>
<td>7</td>
<td>10</td>
<td>A</td>
<td>E</td>
<td>E</td>
<td>L</td>
<td>M</td>
<td>O</td>
<td>P</td>
<td>X</td>
<td>S</td>
<td>T</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>A</td>
<td>E</td>
<td>E</td>
<td>L</td>
<td>M</td>
<td>O</td>
<td>P</td>
<td>R</td>
<td>S</td>
<td>T</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>A</td>
<td>E</td>
<td>E</td>
<td>L</td>
<td>M</td>
<td>O</td>
<td>P</td>
<td>R</td>
<td>S</td>
<td>T</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>A</td>
<td>E</td>
<td>E</td>
<td>L</td>
<td>M</td>
<td>O</td>
<td>P</td>
<td>R</td>
<td>S</td>
<td>T</td>
</tr>
</tbody>
</table>

Entries in black are examined to find the minimum.
Entries in red are \(a[\min]\).
Entries in gray are in final position.

**Running time insensitive to input.** \(\Theta(n^2)\) compares, even if input is sorted.

**Data movement is minimal.** \(\Theta(n)\) exchanges.

**In place.** \(\Theta(1)\) extra space.
2.1 Elementary Sorts

- rules of the game
- selection sort
- insertion sort
- binary search
- comparators
- stability

https://algs4.cs.princeton.edu
• In iteration $i$, swap $a[i]$ with each larger entry to its left.
• In iteration $i$, swap $a[i]$ with each larger entry to its left.
Insertion sort

Algorithm. \( \uparrow \) scans from left to right.

Invariants.

- Entries to the left of \( \uparrow \) (including \( \uparrow \)) are in ascending order.
- Entries to the right of \( \uparrow \) have not yet been seen.
Insertion sort: inner loop

To maintain algorithm invariants:

- Move the pointer to the right.

```java
i++;
```

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

```java
for (int j = i; j > 0; j--)
    if (less(a[j], a[j-1]))
        exch(a, j, j-1);
    else break;
```
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 */
    }
}
How many compares to insertion 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. $\sim n^2$
Worst case. Insertion sort makes $\sim \frac{1}{2} n^2$ compares and $\sim \frac{1}{2} n^2$ exchanges to sort an array of $n$ distinct keys in reverse order.

Pf. Exactly $i$ compares and exchanges in iteration $i$. 

\[ 0 + 1 + 2 + \ldots + (n - 1) \]
**Insertion sort: analysis**

**Best case.** Insertion sort makes \( n - 1 \) compares and 0 exchanges to sort an array of \( n \) distinct keys in ascending order.

http://www.sorting-algorithms.com/insertion-sort
Good case. Insertion sort takes $\Theta(n)$ time on “partially sorted” arrays.

Q. Can we formalize what we mean by partially sorted?
A. Yes, in terms of “inversions” (see textbook).
Insertion sort: practical improvements

**Half exchanges.** Shift items over (instead of exchanging).
- Same compares but fewer array accesses.
- No longer uses only `less()` and `exch()` to access data.

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

**Binary insertion sort.** Use binary search to find insertion point.
- Now, worst-case number of compares $\sim n \log_2 n$.
- But can still make $\Theta(n^2)$ array accesses.

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

binary search for first key > K
1.4 **Analysis of Algorithms**

- rules of the game
- selection sort
- insertion sort
- binary search
- comparators
- stability
Binary search

Goal. Given a sorted array and a key, find index of the key in the array?

Binary search. Compare key against middle entry.

- Too small, go left.
- Too big, go right.
- Equal, found.

sorted array

<table>
<thead>
<tr>
<th>6</th>
<th>13</th>
<th>14</th>
<th>25</th>
<th>33</th>
<th>43</th>
<th>51</th>
<th>53</th>
<th>64</th>
<th>72</th>
<th>84</th>
<th>93</th>
<th>95</th>
<th>96</th>
<th>97</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
</tr>
<tr>
<td>lo</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>hi</td>
</tr>
</tbody>
</table>
Binary search: implementation

Trivial to implement?

- First binary search published in 1946.
- First bug-free one in 1962.
- Bug in Java's Arrays.binarySearch() discovered in 2006.

Extra, Extra - Read All About It: Nearly All Binary Searches and Mergesorts are Broken
Friday, June 02, 2006

Posted by Joshua Bloch, Software Engineer

I remember vividly Jon Bentley's first Algorithms lecture at CMU, where he asked all of us incoming Ph.D. students to write a binary search, and then dissected one of our implementations in front of the class. Of course it was broken, as were most of our implementations. This made a real impression on me, as did the treatment of this material in his wonderful Programming Pearls (Addison-Wesley, 1986, Second Edition, 2000). The key lesson was to carefully consider the invariants in your programs.

https://ai.googleblog.com/2006/06-extra-extra-read-all-about-it-nearly.html
**Invariant.** If key appears in array $a[\cdot]$, then $a[lo] \leq key \leq a[hi]$.

```java
public static int binarySearch(String[] a, String key) {
    int lo = 0, hi = a.length - 1;
    while (lo <= hi)
    {
        int mid = lo + (hi - lo) / 2;
        int compare = key.compareTo(a[mid]);
        if (compare < 0) hi = mid - 1;
        else if (compare > 0) lo = mid + 1;
        else return mid;
    }
    return -1;
}
```
Binary search: analysis

Proposition. Binary search makes at most $1 + \log_2 n$ compares to search in any sorted array of length $n$.

Pf.

- Each iteration of while loop:
  - calls `compareTo()` once
  - decreases the length of remaining subarray by at least a factor of 2

\[
\begin{align*}
n &\rightarrow n/2 \rightarrow n/4 \rightarrow n/8 \rightarrow \cdots \rightarrow 2 \rightarrow 1 \\
1 + \log_2 n \\
\end{align*}
\]

slightly better than $2x$,
due to elimination of a[mid] from subarray
(or early termination of while loop)
3-Sum. Given an array of $n$ distinct integers, find three such that $a + b + c = 0$.

Version 0. $\Theta(n^3)$ time.
Version 1. $\Theta(n^2 \log n)$ time.
Version 2. $\Theta(n^2)$ time.

Note. For full credit, use only $\Theta(1)$ extra space.

Open research problem 1. Design algorithm that takes $\Theta(n^{1.999})$ time or better.
Open research problem 2. Prove that $\Theta(n)$ time algorithm is impossible.
2.1 Elementary Sorts

- rules of the game
- selection sort
- insertion sort
- binary search
- comparators
- stability
Different orderings

Q. When might we need to define different sort orderings?
Sort music library by artist

<table>
<thead>
<tr>
<th>Name</th>
<th>Artist</th>
<th>Time</th>
<th>Album</th>
</tr>
</thead>
<tbody>
<tr>
<td>Let It Be</td>
<td>The Beatles</td>
<td>4:03</td>
<td>Let It Be</td>
</tr>
<tr>
<td>Take My Breath Away</td>
<td>Berlin</td>
<td>4:13</td>
<td>Top Gun - Soundtrack</td>
</tr>
<tr>
<td>Circle Of Friends</td>
<td>Better Than Ezra</td>
<td>3:27</td>
<td>Empire Records</td>
</tr>
<tr>
<td>Dancing With Myself</td>
<td>Billy Idol</td>
<td>4:43</td>
<td>Don't Stop</td>
</tr>
<tr>
<td>Rebel Yell</td>
<td>Billy Idol</td>
<td>4:40</td>
<td>Rebel Yell</td>
</tr>
<tr>
<td>Piano Man</td>
<td>Billy Joel</td>
<td>5:36</td>
<td>Greatest Hits Vol. 1</td>
</tr>
<tr>
<td>Atomic</td>
<td>Blondie</td>
<td>3:50</td>
<td>Atomic: The Very Best Of Blondie</td>
</tr>
<tr>
<td>Sunday Girl</td>
<td>Blondie</td>
<td>3:15</td>
<td>Atomic: The Very Best Of Blondie</td>
</tr>
<tr>
<td>Call Me</td>
<td>Blondie</td>
<td>2:33</td>
<td>Atomic: The Very Best Of Blondie</td>
</tr>
<tr>
<td>Dreaming</td>
<td>Blondie</td>
<td>3:66</td>
<td>Atomic: The Very Best Of Blondie</td>
</tr>
<tr>
<td>Hurricane</td>
<td>Bob Dylan</td>
<td>8:22</td>
<td>Desire</td>
</tr>
<tr>
<td>The Times They Are A-Changin'</td>
<td>Bob Dylan</td>
<td>3:17</td>
<td>Greatest Hits</td>
</tr>
<tr>
<td>Livin' On A Prayer</td>
<td>Bon Jovi</td>
<td>4:11</td>
<td>Cross Road</td>
</tr>
<tr>
<td>Beds Of Roses</td>
<td>Bon Jovi</td>
<td>6:35</td>
<td>Cross Road</td>
</tr>
<tr>
<td>Runaway</td>
<td>Bon Jovi</td>
<td>3:53</td>
<td>Cross Road</td>
</tr>
<tr>
<td>Rasputin (Extended Mix)</td>
<td>Bon Jovi</td>
<td>5:50</td>
<td>Greatest Hits</td>
</tr>
<tr>
<td>Have You Ever Seen The Rain</td>
<td>Bonnie Tyler</td>
<td>4:10</td>
<td>Faster Than The Speed Of Night</td>
</tr>
<tr>
<td>Total Eclipse Of The Heart</td>
<td>Bonnie Tyler</td>
<td>7:02</td>
<td>Faster Than The Speed Of Night</td>
</tr>
<tr>
<td>Straight From The Heart</td>
<td>Bonnie Tyler</td>
<td>3:41</td>
<td>Faster Than The Speed Of Night</td>
</tr>
<tr>
<td>Holding Out For A Hero</td>
<td>Bon Jovi</td>
<td>5:49</td>
<td>Meat Loaf And Friends</td>
</tr>
<tr>
<td>Thunder Road</td>
<td>Bruce Springsteen</td>
<td>4:51</td>
<td>Born To Run</td>
</tr>
<tr>
<td>Born To Run</td>
<td>Bruce Springsteen</td>
<td>4:20</td>
<td>Born To Run</td>
</tr>
<tr>
<td>Junglistand</td>
<td>Bruce Springsteen</td>
<td>9:34</td>
<td>Born To Run</td>
</tr>
<tr>
<td>A Year And A Day</td>
<td>Bruce Springsteen</td>
<td>3:40</td>
<td>Born To Run</td>
</tr>
</tbody>
</table>
Sort music library by song name

<table>
<thead>
<tr>
<th>Name</th>
<th>Artist</th>
<th>Time</th>
<th>Album</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alive</td>
<td>Pearl Jam</td>
<td>3:41</td>
<td>Ten</td>
</tr>
<tr>
<td>All Over The World</td>
<td>Pixies</td>
<td>5:27</td>
<td>Bossanova</td>
</tr>
<tr>
<td>All Through The Night</td>
<td>Cyndi Lauper</td>
<td>4:30</td>
<td>She's So Unusual</td>
</tr>
<tr>
<td>Allison Road</td>
<td>Gin Blossoms</td>
<td>3:19</td>
<td>New Miserable Experience</td>
</tr>
<tr>
<td>Anna, Anna, Anna Y Ensancha El</td>
<td>Extremoduro</td>
<td>2:34</td>
<td>Delotoy (1992)</td>
</tr>
<tr>
<td>And We Danced</td>
<td>Hooters</td>
<td>3:50</td>
<td>Nervous Night</td>
</tr>
<tr>
<td>As I Lay Me Down</td>
<td>Sophie B. Hawkins</td>
<td>4:06</td>
<td>Whaler</td>
</tr>
<tr>
<td>Atomic</td>
<td>Blondie</td>
<td>3:50</td>
<td>Atomic: The Very Best Of Blondie</td>
</tr>
<tr>
<td>Automatic Lover</td>
<td>Jay Jay Johnson</td>
<td>4:19</td>
<td>Arteria</td>
</tr>
<tr>
<td>Bob O'Reily</td>
<td>The Who</td>
<td>5:01</td>
<td>Who's Better, Who's Best</td>
</tr>
<tr>
<td>Beautiful Life</td>
<td>Ace Of Base</td>
<td>3:40</td>
<td>The Bridge</td>
</tr>
<tr>
<td>Beds Of Roses</td>
<td>Bon Jovi</td>
<td>6:35</td>
<td>Cross Road</td>
</tr>
<tr>
<td>Black</td>
<td>Pearl Jam</td>
<td>3:44</td>
<td>Ten</td>
</tr>
<tr>
<td>Blood American</td>
<td>Jimmy Eat World</td>
<td>3:04</td>
<td>Blood American</td>
</tr>
<tr>
<td>Borderline</td>
<td>Madonna</td>
<td>4:90</td>
<td>The Immaculate Collection</td>
</tr>
<tr>
<td>Born To Run</td>
<td>Bruce Springsteen</td>
<td>4:30</td>
<td>Born To Run</td>
</tr>
<tr>
<td>Both Sides Of The Story</td>
<td>Phil Collins</td>
<td>6:43</td>
<td>Both Sides</td>
</tr>
<tr>
<td>Bouncing Around The Room</td>
<td>Prince</td>
<td>4:09</td>
<td>A Live One (Disc 1)</td>
</tr>
<tr>
<td>Boys Don't Cry</td>
<td>The Cure</td>
<td>2:35</td>
<td>Staring At The Sea: The Singles 1979-1985</td>
</tr>
<tr>
<td>Brit</td>
<td>Green Day</td>
<td>1:43</td>
<td>Insomniac</td>
</tr>
<tr>
<td>Breakdown</td>
<td>Deerheart</td>
<td>3:40</td>
<td>Deerheart</td>
</tr>
<tr>
<td>Bring Me To Life (Kevin Rowe Mix)</td>
<td>Vanvessence Vs. Pao</td>
<td>9:48</td>
<td></td>
</tr>
<tr>
<td>Californiazation</td>
<td>Red Hot Chili Peppers</td>
<td>2:40</td>
<td></td>
</tr>
<tr>
<td>Call Me</td>
<td>Blondie</td>
<td>3:33</td>
<td>Atomic: The Very Best Of Blondie</td>
</tr>
<tr>
<td>Can't Get You Out Of My Head</td>
<td>Kylie Minogue</td>
<td>3:30</td>
<td>Fever</td>
</tr>
<tr>
<td>Celebration</td>
<td>Kool &amp; The Gang</td>
<td>3:45</td>
<td>Time Life Music Sounds Of The Seventies - C</td>
</tr>
<tr>
<td>Chasing Chariots</td>
<td>Kool &amp; The Gang</td>
<td>3:40</td>
<td></td>
</tr>
</tbody>
</table>
Comparable interface: review

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

```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;
    }

    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;
    }
}
```

https://algs4.cs.princeton.edu/12oop/Date.java.html
Comparator interface

Comparator interface: sort using an alternate order.

java.util.Comparator interface

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

Required property. Induces a total preorder.

<table>
<thead>
<tr>
<th>string order</th>
<th>example</th>
</tr>
</thead>
<tbody>
<tr>
<td>natural</td>
<td>Now is the time</td>
</tr>
<tr>
<td>case insensitive</td>
<td>is Now the time</td>
</tr>
<tr>
<td>Spanish (modern)</td>
<td>café cafetero churro cuarto nube ñoño ocasión</td>
</tr>
<tr>
<td>diacritic insensitive</td>
<td>Aaron Ådne Ævarr Ágnes Álke Ayşegül</td>
</tr>
</tbody>
</table>
Comparator interface: system sort

To use with Java system sort:

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

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

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: implementing

To implement a comparator:

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

```java
google
import java.util.Comparator;

public class Student
{
    private final String name;
    private final int section;
    ...
    static = one per class (not per instance of class)

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

```java
import java.util.Comparator;

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

    private static class SectionOrder implements Comparator<Student> {
        public int compare(Student v, Student w) {
            return Integer.compare(v.section, w.section);
        }
        public static Comparator<Student> bySectionOrder() {
            return new SectionOrder();
        }
    }
}
Comparator interface: using lambda expressions

Compact alternative. Use a lambda expression to implement the compare method.

```java
import java.util.Comparator;

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

    public static Comparator<Student> byNameOrder()
    { return (v, w) -> v.name.compareTo(w.name); }

    public static Comparator<Student> bySectionOrder()
    { return (v, w) -> Integer.compare(v.section, w.section); }
}
```

use a lambda expression to create a Comparator<Student>
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.

Arrays.sort(a, Student.byNameOrder());

<table>
<thead>
<tr>
<th>Name</th>
<th>Grade</th>
<th>Phone</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>Andrews</td>
<td>3</td>
<td>(664) 480–0023</td>
<td>097 Little</td>
</tr>
<tr>
<td>Battle</td>
<td>4</td>
<td>(874) 088–1212</td>
<td>121 Whitman</td>
</tr>
<tr>
<td>Chen</td>
<td>3</td>
<td>(991) 878–4944</td>
<td>308 Blair</td>
</tr>
<tr>
<td>Fox</td>
<td>3</td>
<td>(884) 232–5341</td>
<td>11 Dickinson</td>
</tr>
<tr>
<td>Furia</td>
<td>1</td>
<td>(766) 093–9873</td>
<td>101 Brown</td>
</tr>
<tr>
<td>Gazi</td>
<td>4</td>
<td>(800) 867–5309</td>
<td>101 Brown</td>
</tr>
<tr>
<td>Kanaga</td>
<td>3</td>
<td>(898) 122–9643</td>
<td>22 Brown</td>
</tr>
<tr>
<td>Rohde</td>
<td>2</td>
<td>(232) 343–5555</td>
<td>343 Forbes</td>
</tr>
</tbody>
</table>

Arrays.sort(a, Student.bySectionOrder());

<table>
<thead>
<tr>
<th>Name</th>
<th>Grade</th>
<th>Phone</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>Furia</td>
<td>1</td>
<td>(766) 093–9873</td>
<td>101 Brown</td>
</tr>
<tr>
<td>Rohde</td>
<td>2</td>
<td>(232) 343–5555</td>
<td>343 Forbes</td>
</tr>
<tr>
<td>Andrews</td>
<td>3</td>
<td>(664) 480–0023</td>
<td>097 Little</td>
</tr>
<tr>
<td>Chen</td>
<td>3</td>
<td>(991) 878–4944</td>
<td>308 Blair</td>
</tr>
<tr>
<td>Fox</td>
<td>3</td>
<td>(884) 232–5341</td>
<td>11 Dickinson</td>
</tr>
<tr>
<td>Kanaga</td>
<td>3</td>
<td>(898) 122–9643</td>
<td>22 Brown</td>
</tr>
<tr>
<td>Battle</td>
<td>4</td>
<td>(874) 088–1212</td>
<td>121 Whitman</td>
</tr>
<tr>
<td>Gazi</td>
<td>4</td>
<td>(800) 867–5309</td>
<td>101 Brown</td>
</tr>
</tbody>
</table>
Summary

Java framework.
- Use Comparable interface to define natural order.
- Use Comparator interface to define alternative orders.

Elementary sorting algorithms.
- Selection sort.
- Insertion sort.
- Takes $\Theta(n^2)$ time in worst case $\Rightarrow$ too slow!

Ahead. $\Theta(n \log n)$ time algorithms.
2.1 **Elementary Sorts**

- rules of the game
- selection sort
- insertion sort
- binary search
- comparators
- stability

*skipped in lecture (see precept)*
A typical application. First, sort by name; then sort by section.

Stability

A stable sort preserves the relative order of items with equal keys.
Which sorting algorithm(s) are stable?

A. Selection sort.
B. Insertion sort.
C. Both A and B.
D. Neither A nor B.
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);
            }
        }
    }
}
```

<table>
<thead>
<tr>
<th>i</th>
<th>j</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>B₁</td>
<td>A₁</td>
<td>A₂</td>
<td>A₃</td>
<td>B₂</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>A₁</td>
<td>B₁</td>
<td>A₂</td>
<td>A₃</td>
<td>B₂</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>A₁</td>
<td>A₂</td>
<td>B₁</td>
<td>A₃</td>
<td>B₂</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>A₁</td>
<td>A₂</td>
<td>A₃</td>
<td>B₁</td>
<td>B₂</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>A₁</td>
<td>A₂</td>
<td>A₃</td>
<td>B₁</td>
<td>B₂</td>
</tr>
</tbody>
</table>

**Pf.** Equal items never move past each other.
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);
        }
    }
}
```

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