
https://algs4.cs.princeton.edu

# 2.1 Elementary Sorts

- *rules of the game*
- *selection sort*
- *insertion sort*
- *binary search*
- *comparators*
- *stability* ← see precept


https://algs4.cs.princeton.edu

# 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.

| Last ▾ | First | House | Year |
|---|---|---|---|
| **Longbottom** | Neville | Gryffindor | 1998 |
| **Weasley** | Ron | Gryffindor | 1998 |
| **Abbott** | Hannah | Hufflepuff | 1998 |
| **Potter** | Harry | Gryffindor | 1998 |
| **Chang** | Cho | Ravenclaw | 1997 |
| **Granger** | Hermione | Gryffindor | 1998 |
| **Malfoy** | Draco | Slytherin | 1998 |
| **Diggory** | Cedric | Hufflepuff | 1996 |
| **Weasley** | Ginny | Gryffindor | 1999 |
| **Parkinson** | Pansy | Slytherin | 1998 |

item →

key →

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

# Sorting problem

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

| Last ▾ | First | House | Year |
|---|---|---|---|
| **Abbott** | Hannah | Hufflepuff | 1998 |
| **Chang** | Cho | Ravenclaw | 1997 |
| **Granger** | Hermione | Gryffindor | 1998 |
| **Diggory** | Cedric | Hufflepuff | 1996 |
| **Longbottom** | Neville | Gryffindor | 1998 |
| **Malfoy** | Draco | Slytherin | 1998 |
| **Parkinson** | Pansy | Slytherin | 1998 |
| **Potter** | Harry | Gryffindor | 1998 |
| **Weasley** | Ron | Gryffindor | 1998 |
| **Weasley** | Ginny | Gryffindor | 1999 |

key →

item →

↑ sorted by key

**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 $\le$ that satisfies:

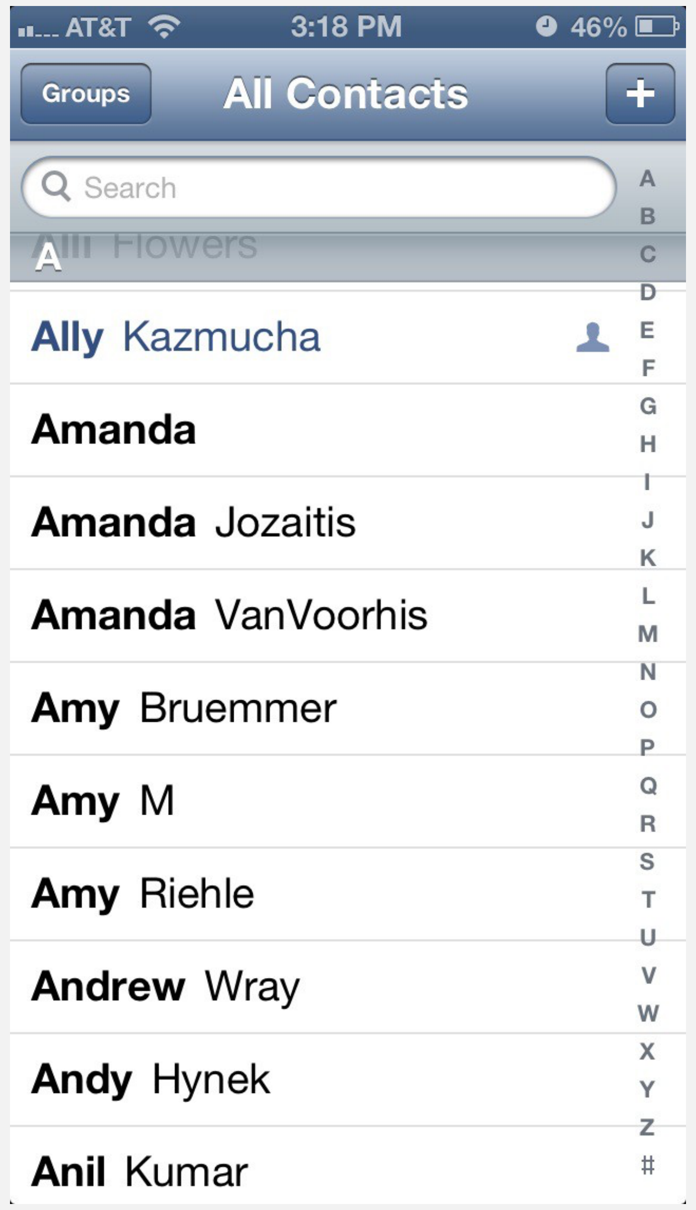
- Totality: either $v \le w$ or $w \le v$ or both.
- Transitivity: if both $v \le w$ and $w \le x$, then $v \le x$.

Examples.

| Video name | Views (billions) |
| --- | --- |
| "Despacito"[23] | 6.96 |
| "Baby Shark Dance"[28] | 6.55 |
| "Shape of You"[29] | 4.97 |
| "See You Again"[30] | 4.72 |
| "*Masha and the Bear* – Recipe for | 4.33 |
| "Uptown Funk"[38] | 3.94 |

numerical order (descending)


chronological order




lexicographic order

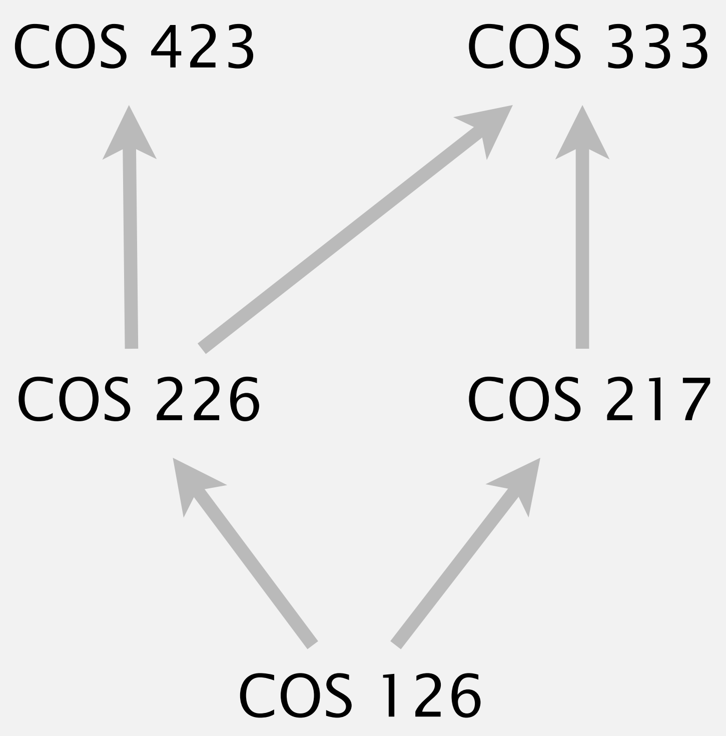
# Total preorder

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

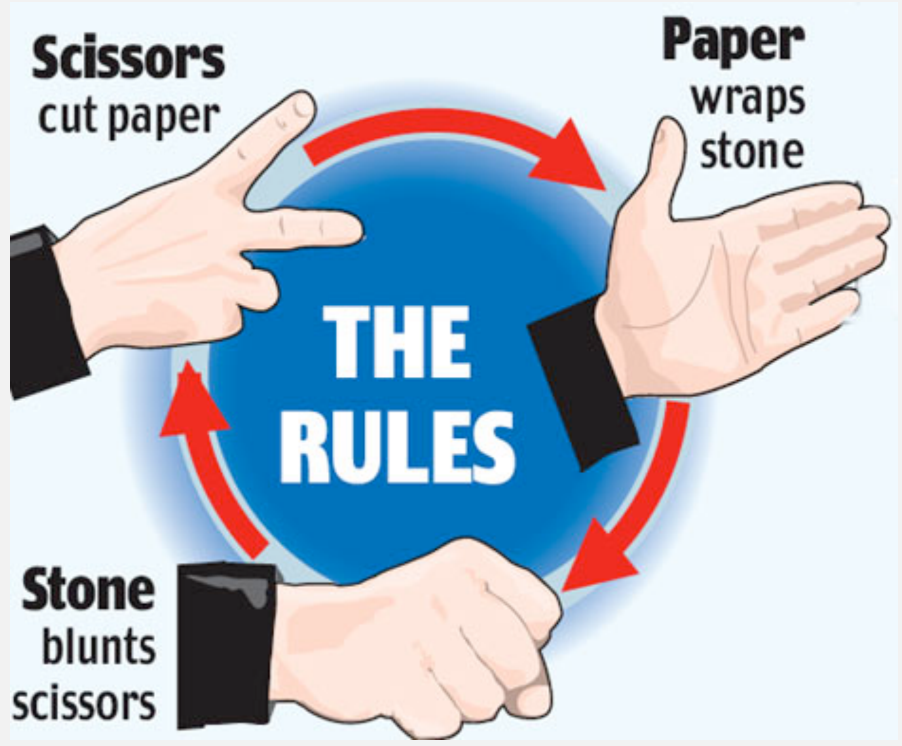
A total preorder is a binary relation $\leq$ that satisfies:

- 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)**

```
~/Desktop/21elemenary> jshell
Math.sqrt(-1.0) <= Math.sqrt(-1.0)
false
```

**the <= operator for double**
**(violates totality)**

# Sample sort clients

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

Ex 1. 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 clients

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

Ex 2. Sort real numbers in ascending order.

```
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.

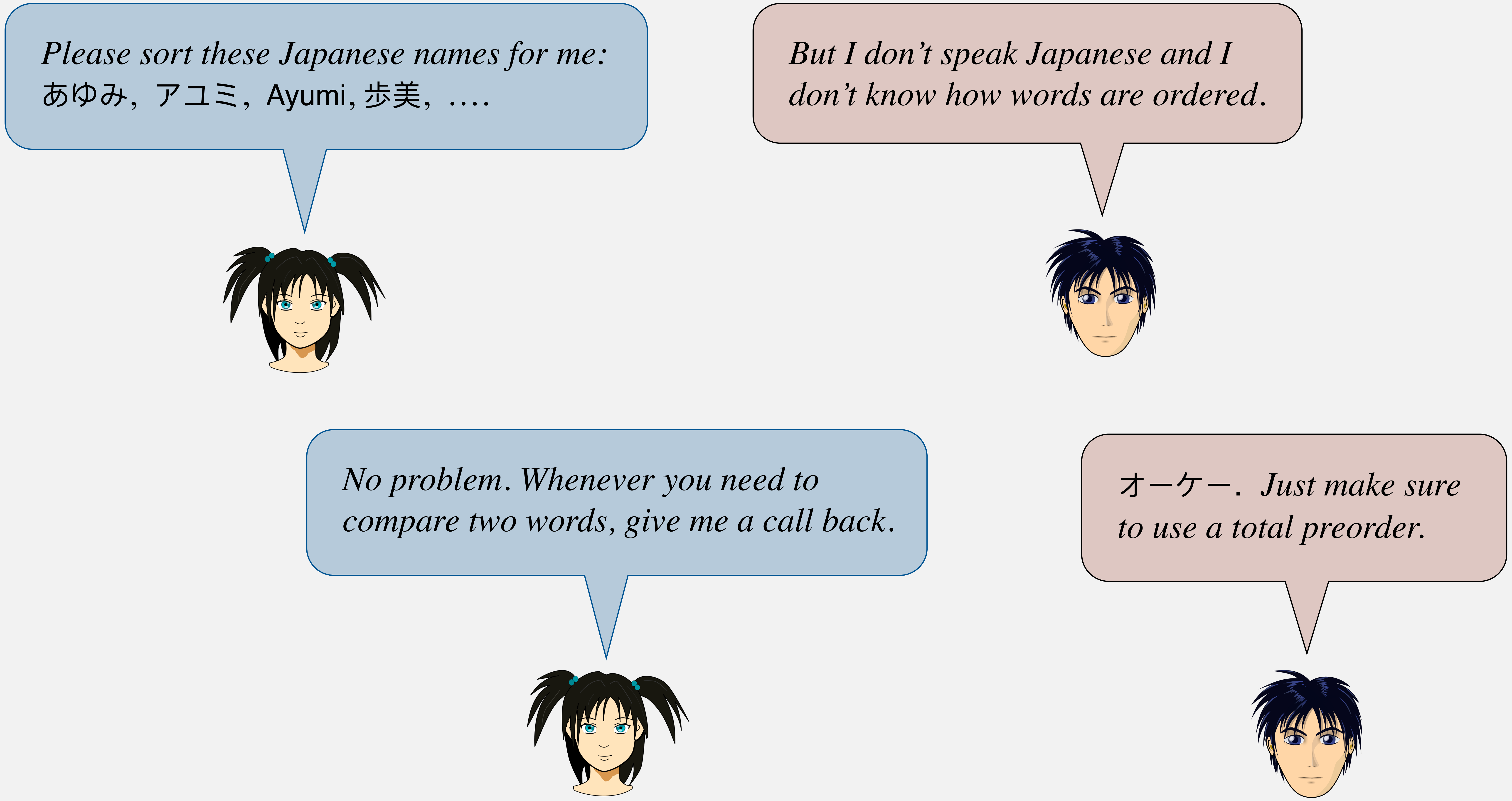
```
import java.io.File;

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.



# 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.

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

contract: method with this signature (and prescribed behavior)

Class that implements interface. Must implement all interface methods.

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

class promises to honor the contract

class abides by the contract

Enforcement. Compile-time error if a class fails to define the requisite methods.

# Callbacks in Java: roadmap

**client (StringSorter.java)**

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

**java.lang.Comparable interface**

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

**sort implementation (Insertion.java)**

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

callback

**data type implementation (String.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

# Elementary sorts: quiz 1

**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.

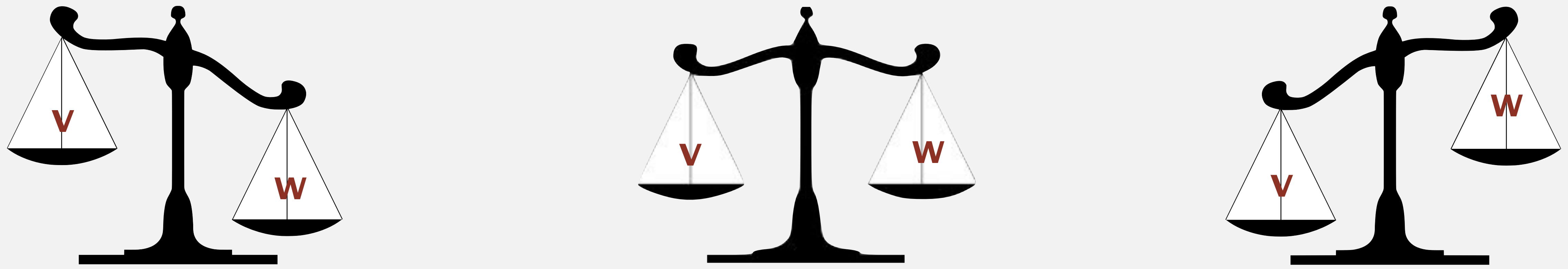
**D.** `Insertion.java` will throw an exception.

# 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`).

`v.compareTo(w) <= 0`
means v is less than or equal to w




v is less than w (return negative integer) | v is equal to w (return 0) | v is greater than w (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;
   }
}
```

can compare Date objects
only to other Date objects

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

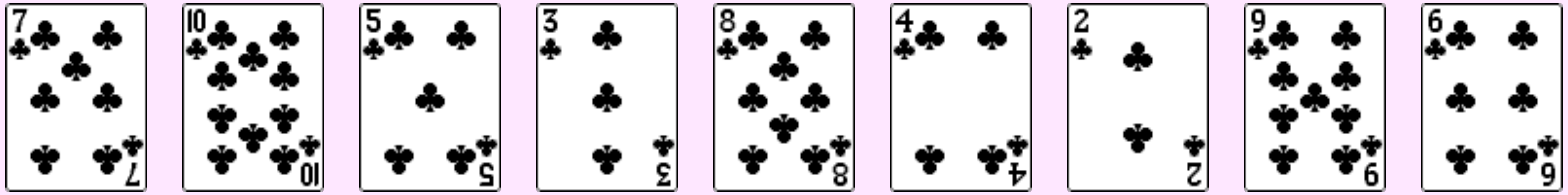

https://algs4.cs.princeton.edu

# 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 ?

```
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].

```
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

▲ algorithm position

in final order

not in final order

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.** $\sim 1/4\, n^2$

**C.** $\sim 1/2\, n^2$

**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.

| 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 |

a[]

*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.


https://algs4.cs.princeton.edu

# 2.1 Elementary Sorts

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

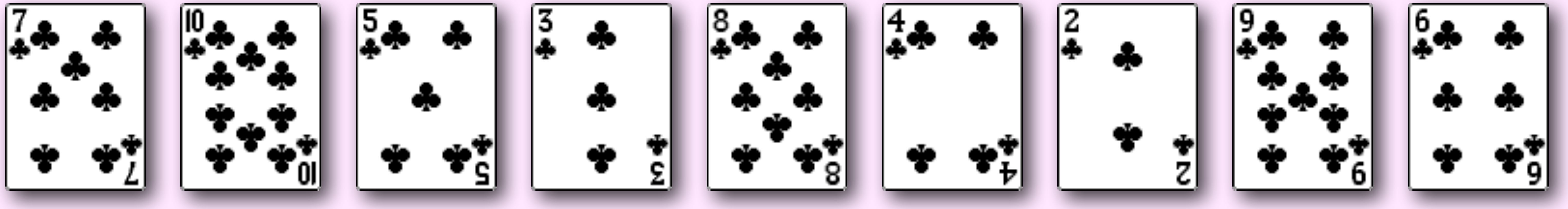
# Insertion sort demo

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


https://www.youtube.com/watch?v=ROalU379l3U

# Insertion sort demo

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



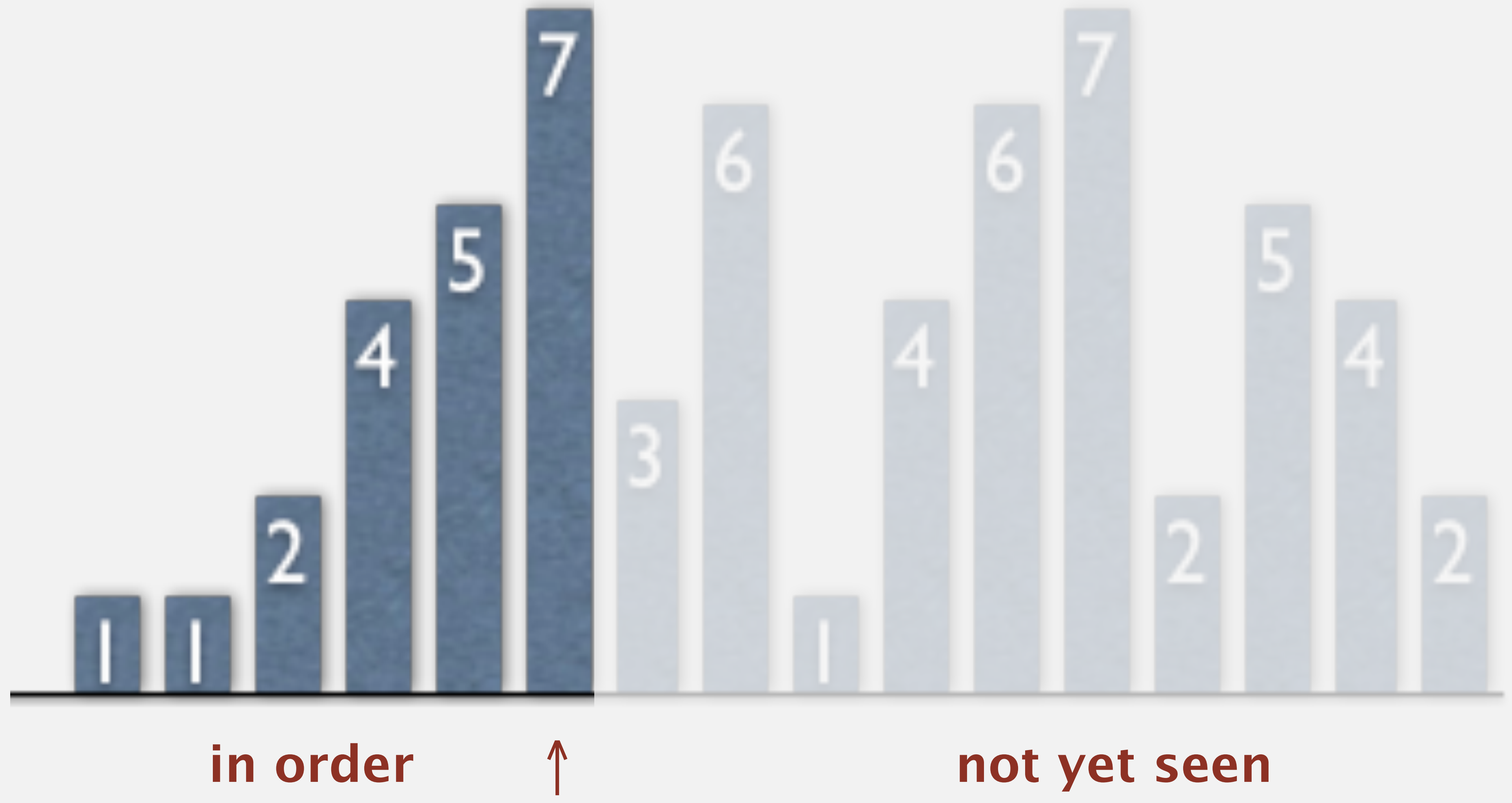
initial array

# 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.

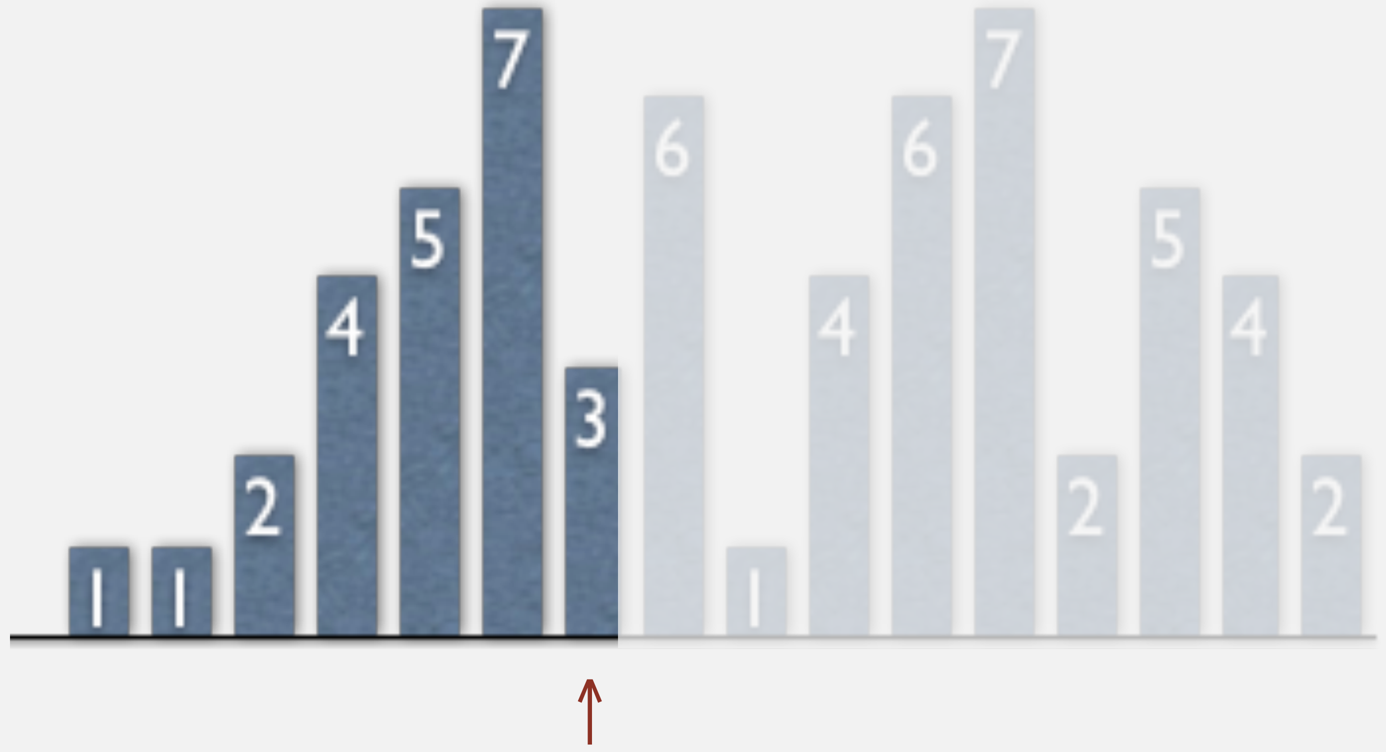


# Insertion sort: inner loop

To maintain algorithm invariants:

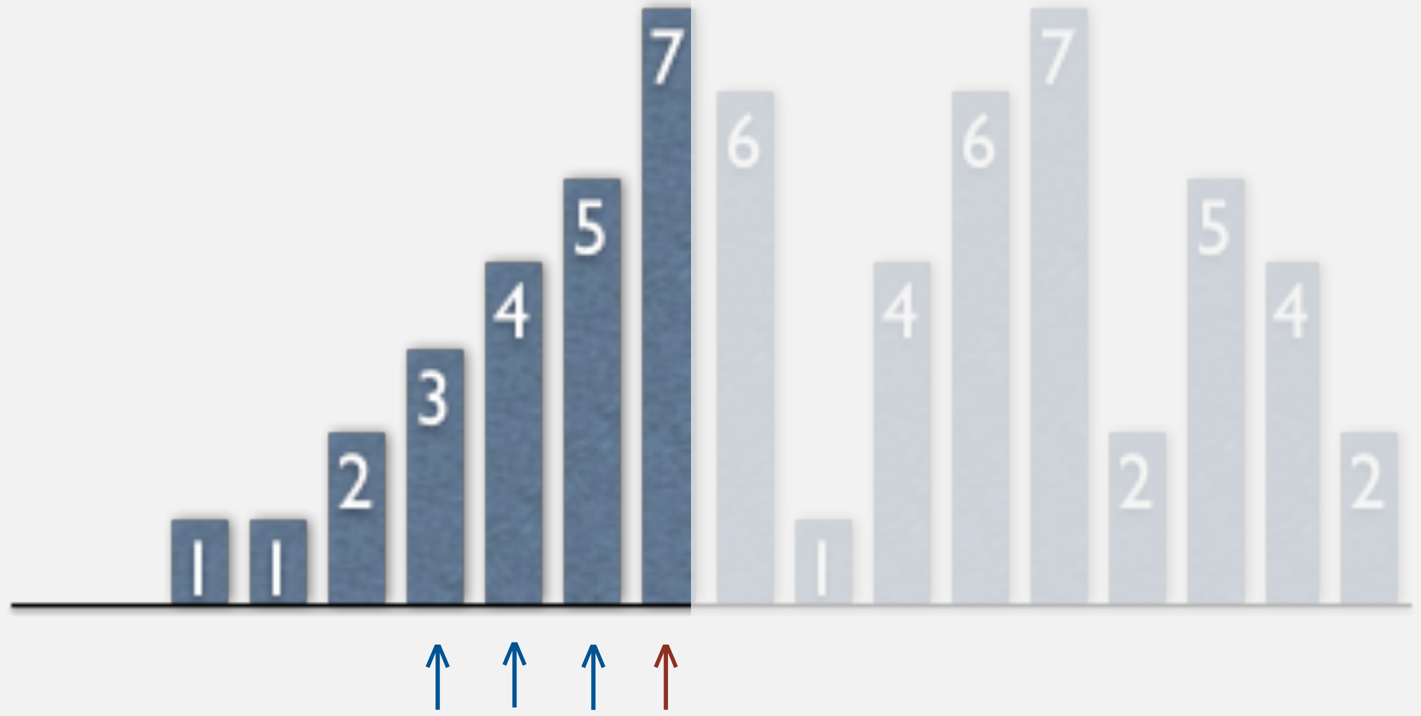
- Move the pointer to the right.

```
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

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

}
```

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

**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.** $\sim 1/2\ n^2$

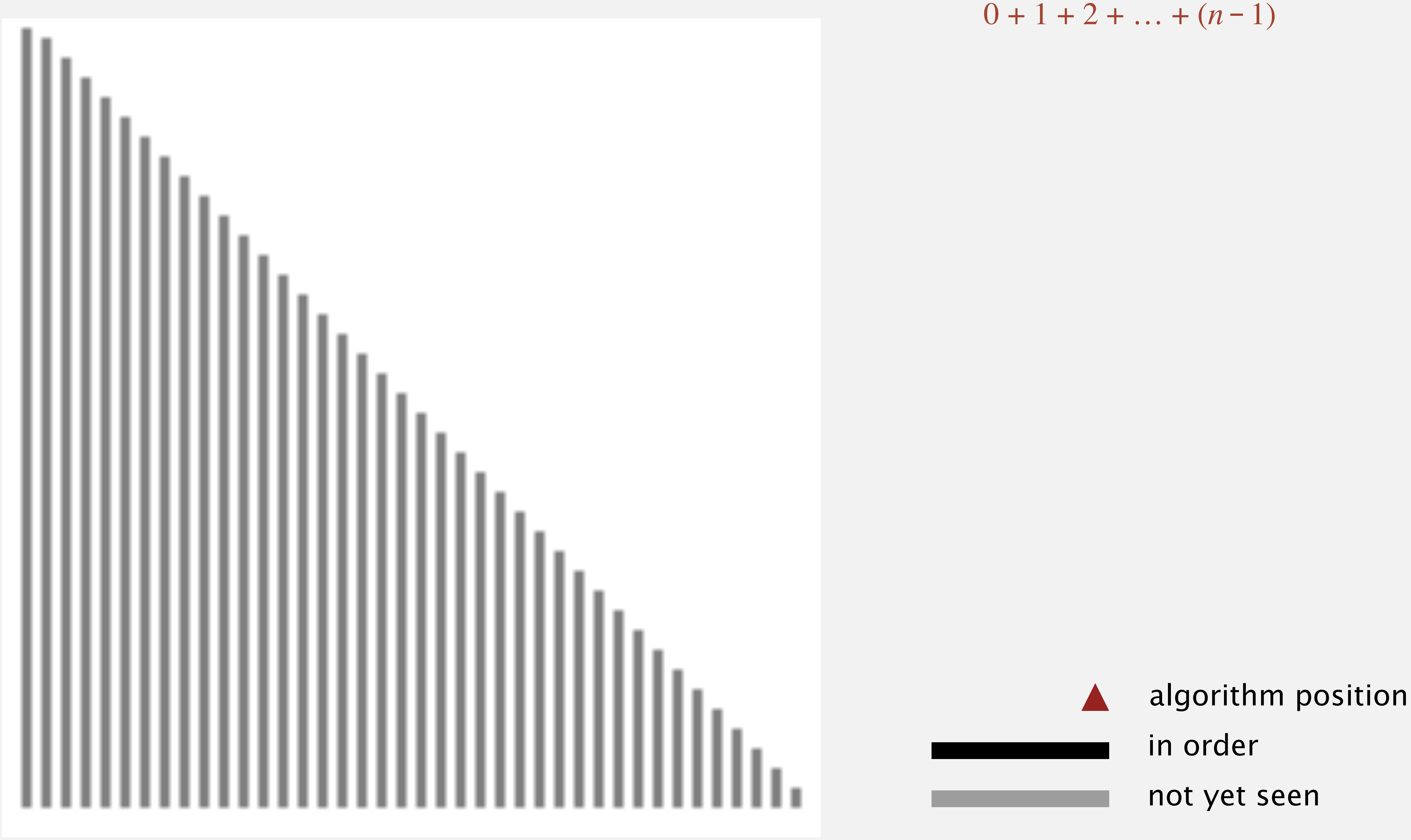
**D.** $\sim n^2$

# Insertion sort: analysis

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 $i$.

$0 + 1 + 2 + \ldots + (n-1)$

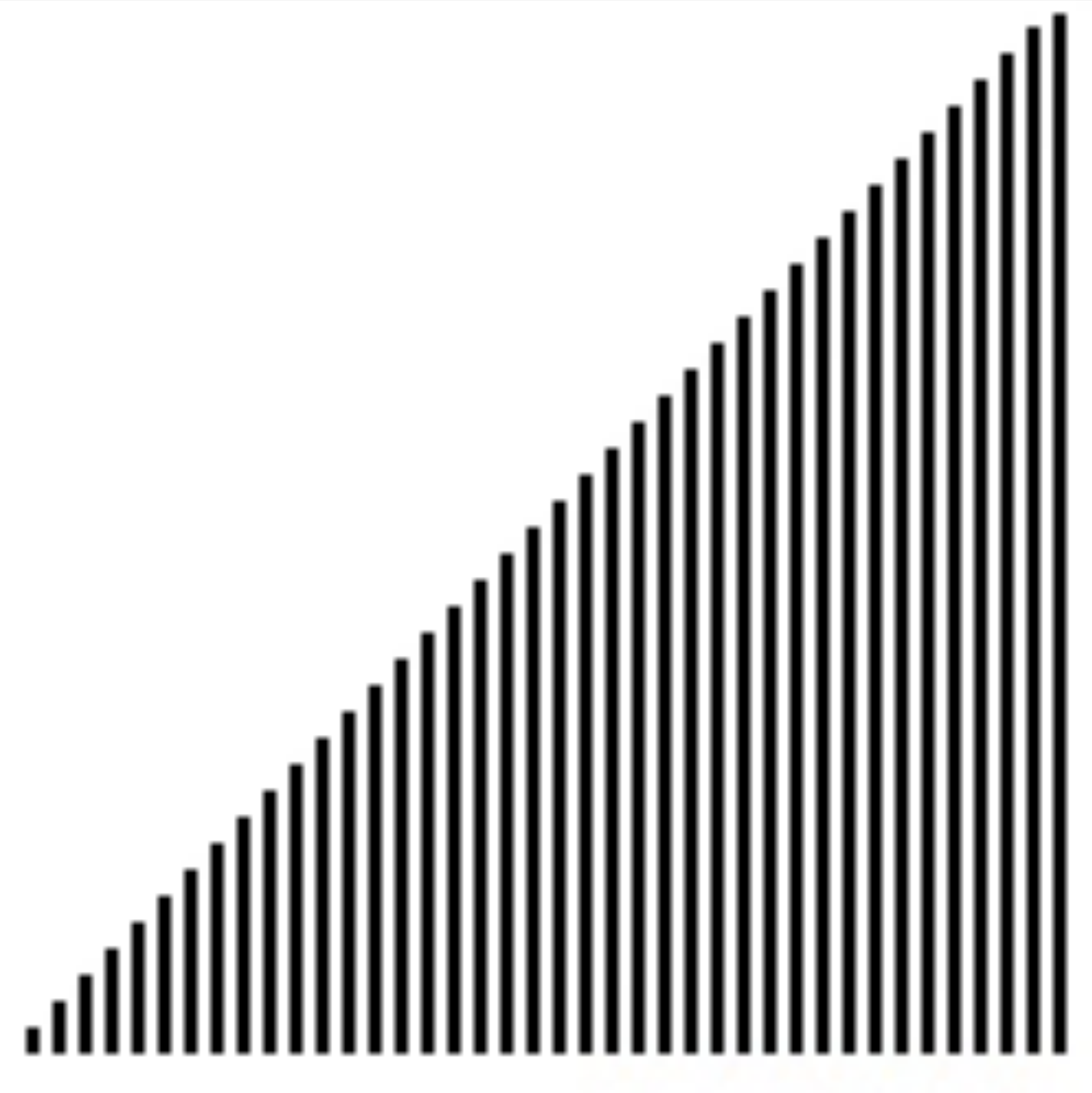




http://www.sorting-algorithms.com/insertion-sort

# 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

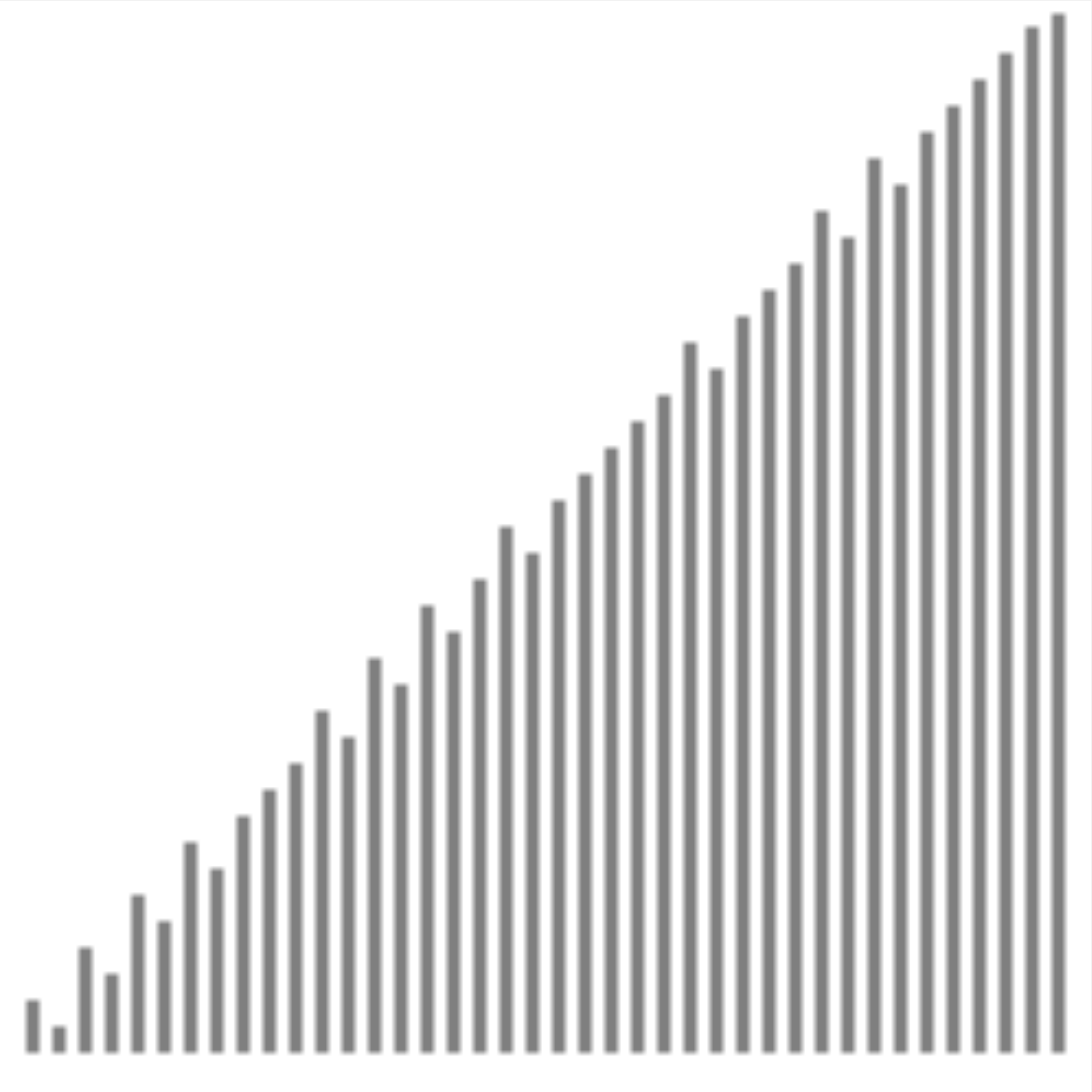
algorithm position
in order
not yet seen

# Insertion sort: analysis

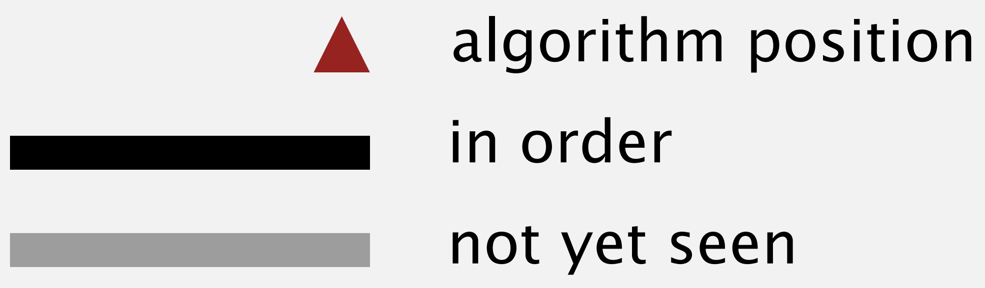
Good case. Insertion sort takes $\Theta(n)$ time on "partially sorted" arrays.

Q. Can we formalize what we mean by partially sorted?

A. Yes, in terms of "inversions" (see textbook).



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



# 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


https://algs4.cs.princeton.edu

# 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**

| | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 6 | 13 | 14 | 25 | 33 | 43 | 51 | 53 | 64 | 72 | 84 | 93 | 95 | 96 | 97 |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
| lo | | | | | | | | | | | | | | hi |

# 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

## Binary search: Java implementation

Invariant. If `key` appears in array `a[]`, then `a[lo]` ≤ `key` ≤ `a[hi]`.

why not `mid = (lo + hi) / 2` ?

```
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

can happen at most $1 + \log_2 n$ times. Why?

$$\underbrace{n \rightarrow n/2 \rightarrow n/4 \rightarrow n/8 \rightarrow \cdots \rightarrow 2 \rightarrow 1}_{1 + \log_2 n}$$

slightly better than 2×,
due to elimination of `a[mid]` from subarray
(or early termination of `while` loop)

# 3-SUM

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.

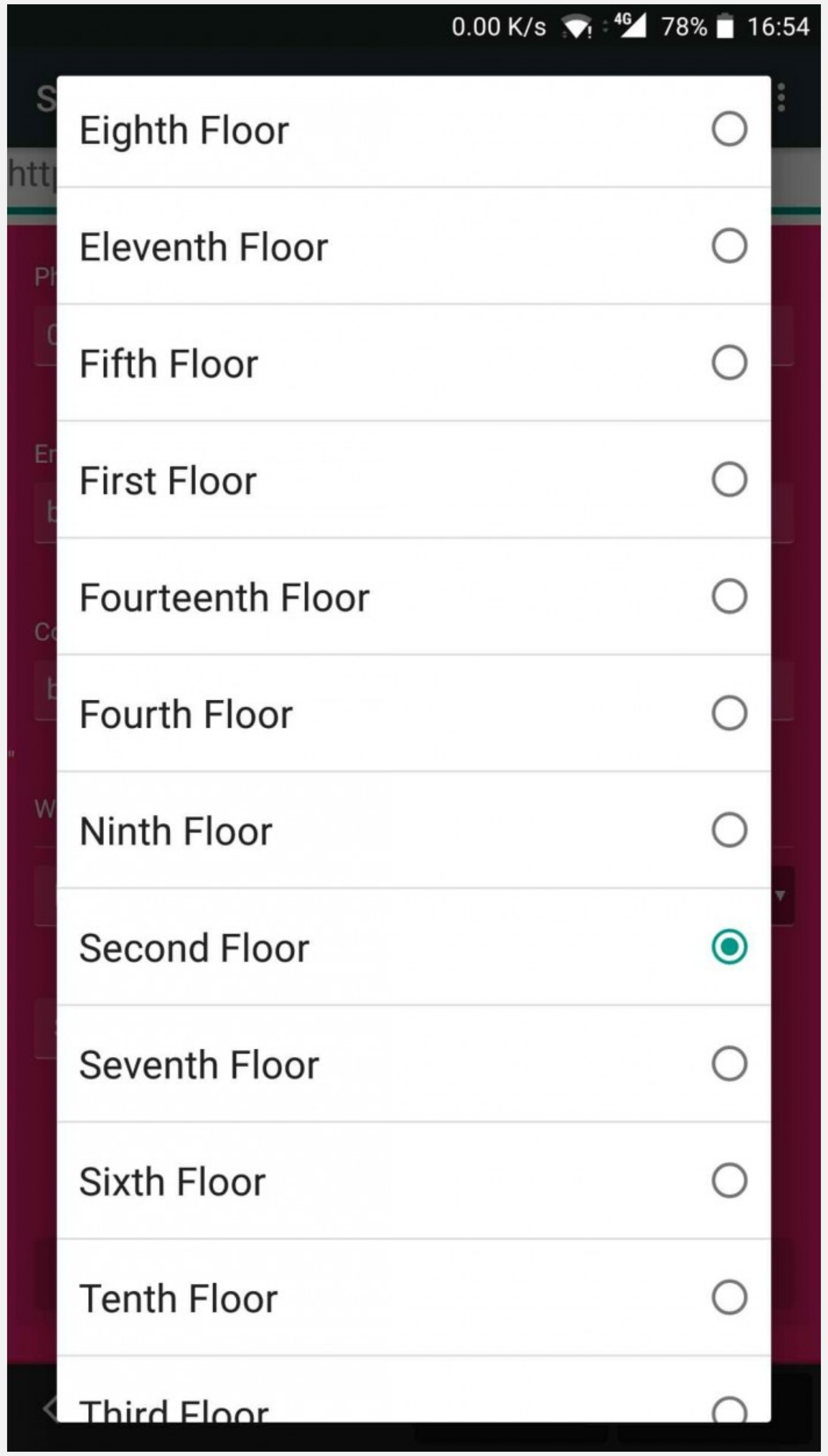

https://algs4.cs.princeton.edu

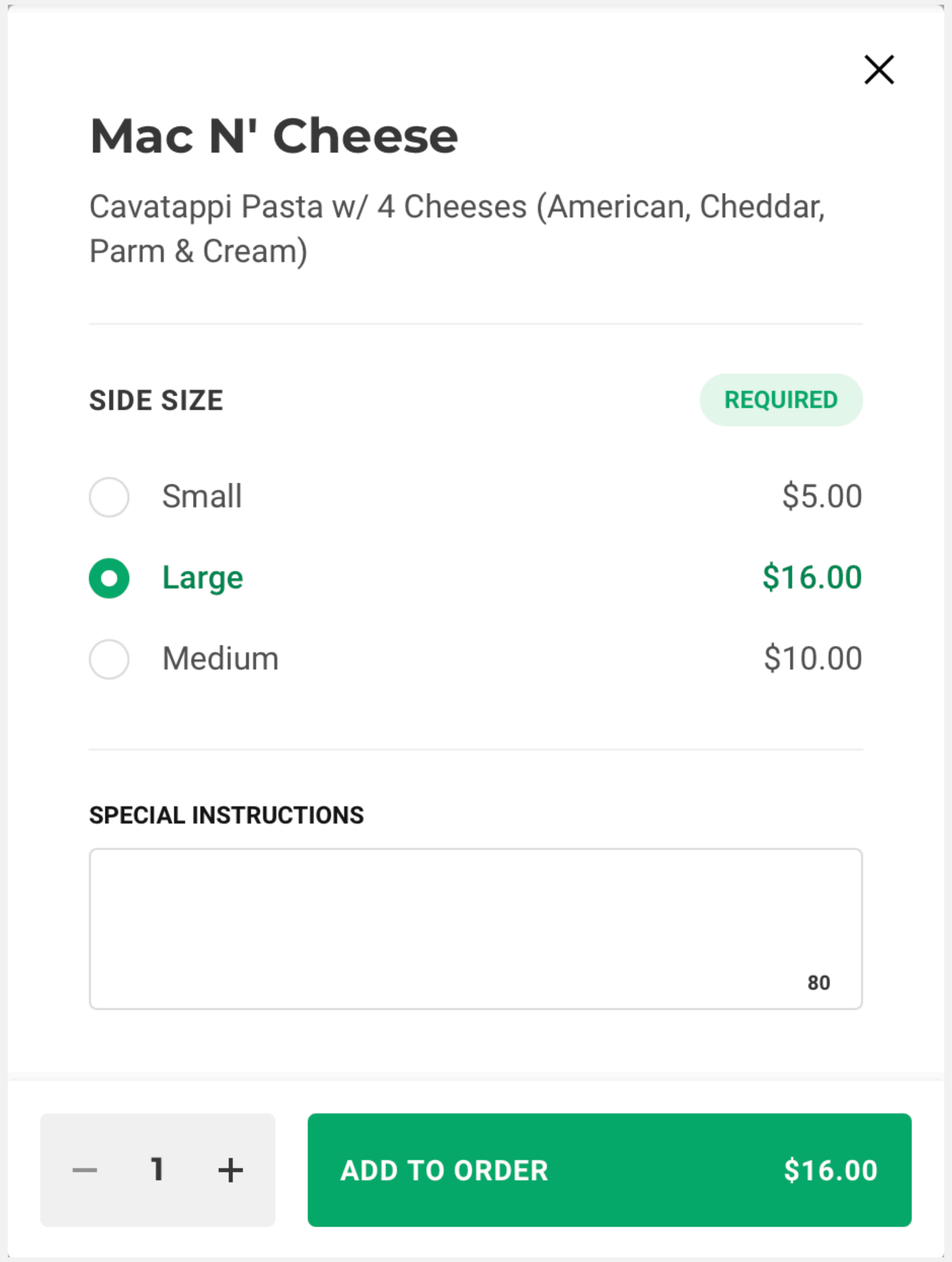
# 2.1 Elementary Sorts

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

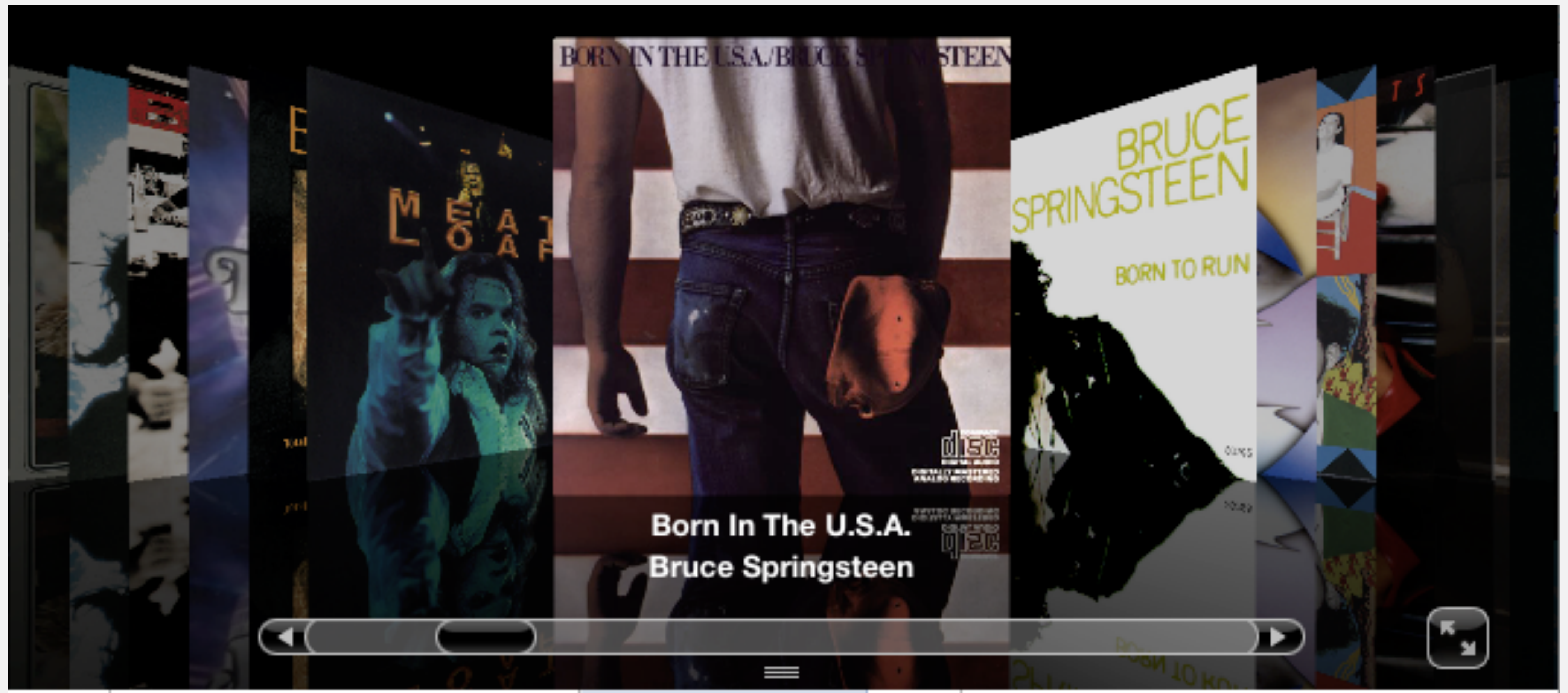
# Different orderings

Q. When might we need to define different sort orderings?



# Sort music library by artist




| | 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 |

# 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 |

# 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

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

# Comparator interface

Comparator interface: sort using an alternate order.

**java.util.Comparator interface**

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

Required property. Induces a total preorder.

| string order | example |
|---|---|
| **natural** | Now is the time |
| **case insensitive** | is Now the time |
| **Spanish (modern)** | café cafetero churro cuarto nube ñoño ocasión |
| **diacritic insensitive** | Aaron Ådne Ævarr Ágnes Älke Ayşegül |

ñ is between n and o

# 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

```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
import java.util.Comparator;

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

   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();  }

}
```

`static` = one per class (not per instance of class)

https://algs4.cs.princeton.edu/12oop/Student.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;
   ...

   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();  }

}
```

useful library method

# Comparator interface: using lambda expressions

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

```
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>

lambdas expressions
not needed in this course

# 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());`

| | | | | |
|---|---|---|---|---|
| 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 |

`Arrays.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 |

# 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.


https://algs4.cs.princeton.edu

# 2.1 Elementary Sorts

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

skipped in lecture
(see precept)

# 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.

**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);
   }
}
```

| 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 an equal item past another one.