https://algs4.cs.princeton.edu

2.1 Elementary Sorts

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

https://algs4.cs.princeton.edu

2.1 Elementary Sorts

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

Sorting problem

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

item →

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

sorting hat

Sorting problem

Goal. Rearrange an 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 → (Malfoy)

item → (Potter row)

sorted by key ↑

sorting hat

Sorting problem

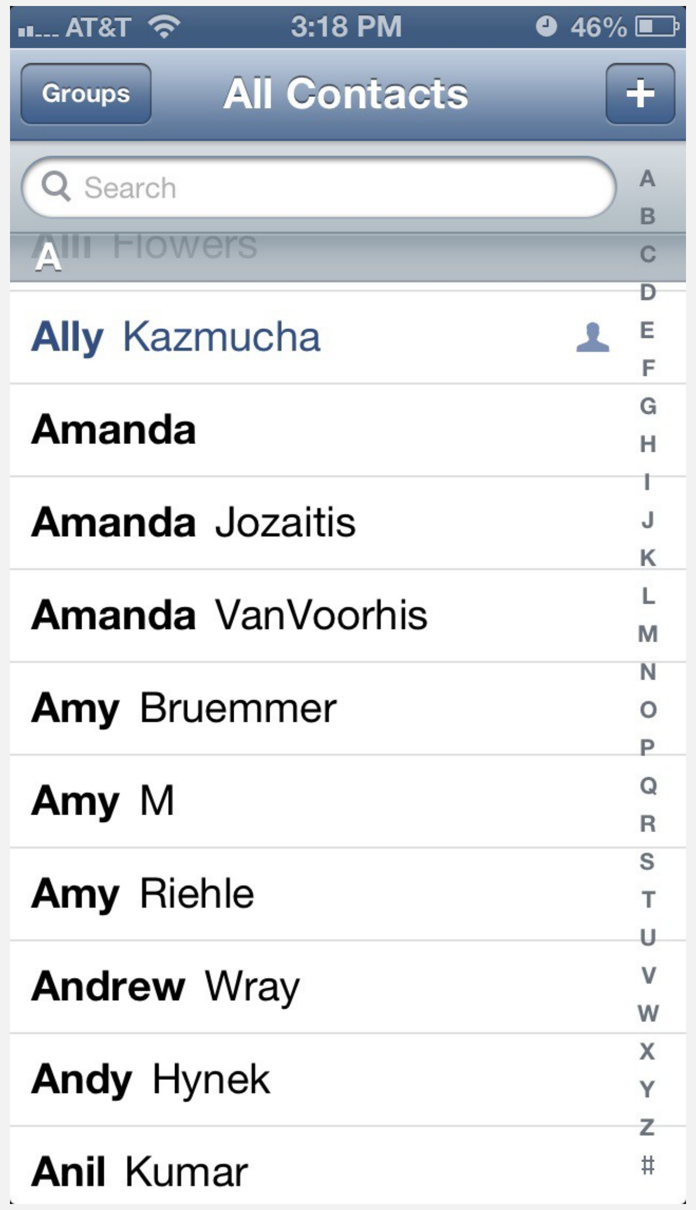
Sorting is a well-defined problem if there is a binary relation $\leq$ that satisfies:

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

← mathematically, a "weak order"

Examples.

chronological order



lexicographic order

No.	Video name	Views (billions)
1.	"Baby Shark Dance"[3]	10.15
2.	"Despacito"[6]	7.73
3.	"Johny Johny Yes Papa"[12]	6.15
4.	"Shape of You"[13]	5.61
5.	"See You Again"[15]	5.41

numerical order (descending)

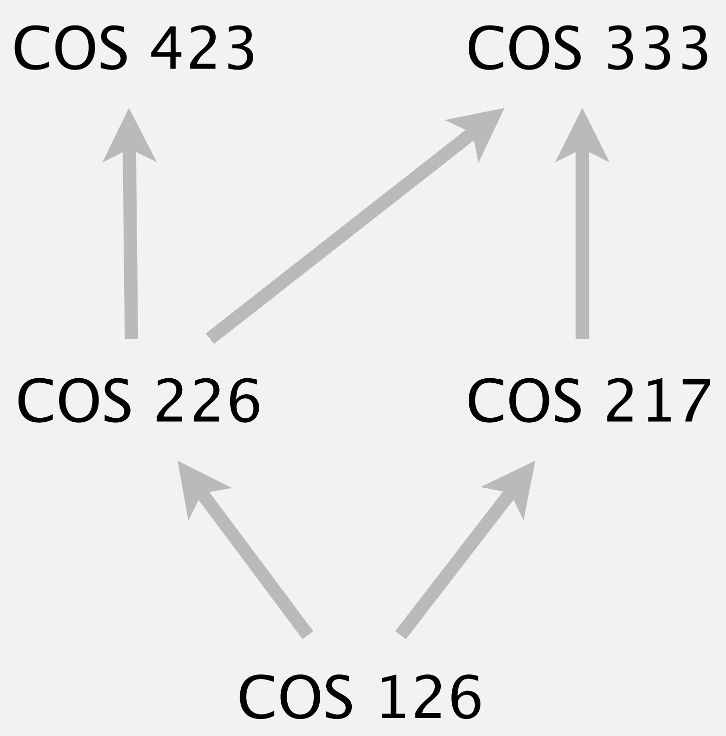
Sorting problem

Sorting is a well-defined problem if there is a binary relation ≤ that satisfies:

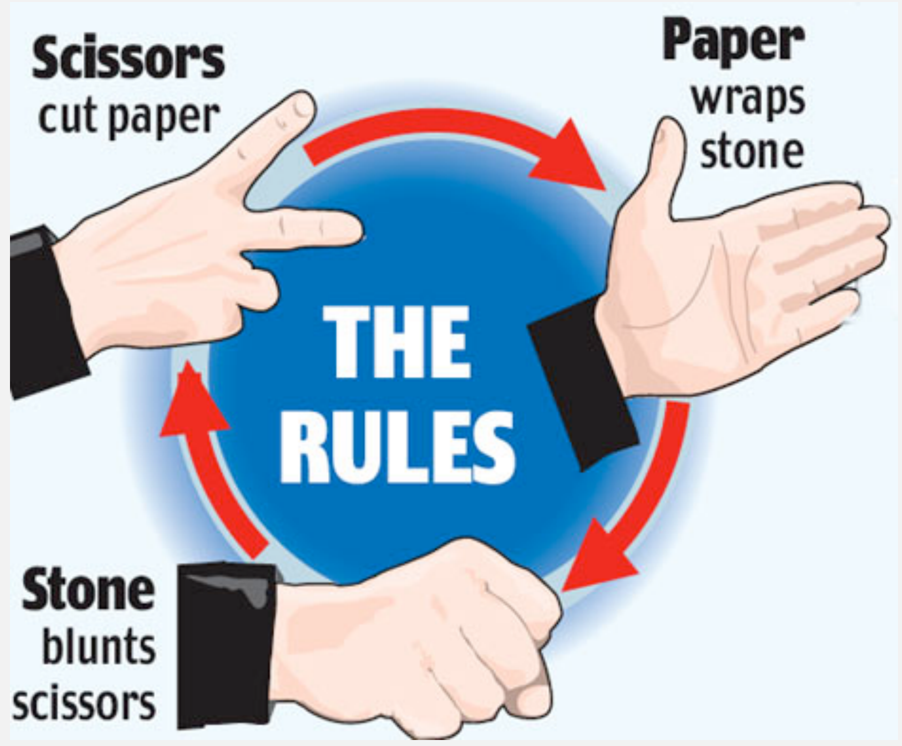
- Totality: either $v \le w$ or $w \le v$ or both.
- Transitivity: if both $v \le w$ and $w \le x$, then $v \le x$.

← mathematically, a "weak order"

Non-examples.



course prerequisites (violates totality)



Ro-sham-bo order (violates transitivity)

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

the <= operator for double (irreflexive, which violates totality)

Sample sort clients

Goal. General-purpose sorting function.

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]);
   }
}
```

```
~/Desktop/sort> more words3.txt
bed bug dad yet zoo ... all bad yes

~/Desktop/sort> java StringSorter < words3.txt
all bad bed bug dad ... yes yet zoo
[suppressing newlines]
```

Sample sort clients

Goal. General-purpose sorting function.

Ex 2. Sort real numbers in numerical 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]);
   }
}
```

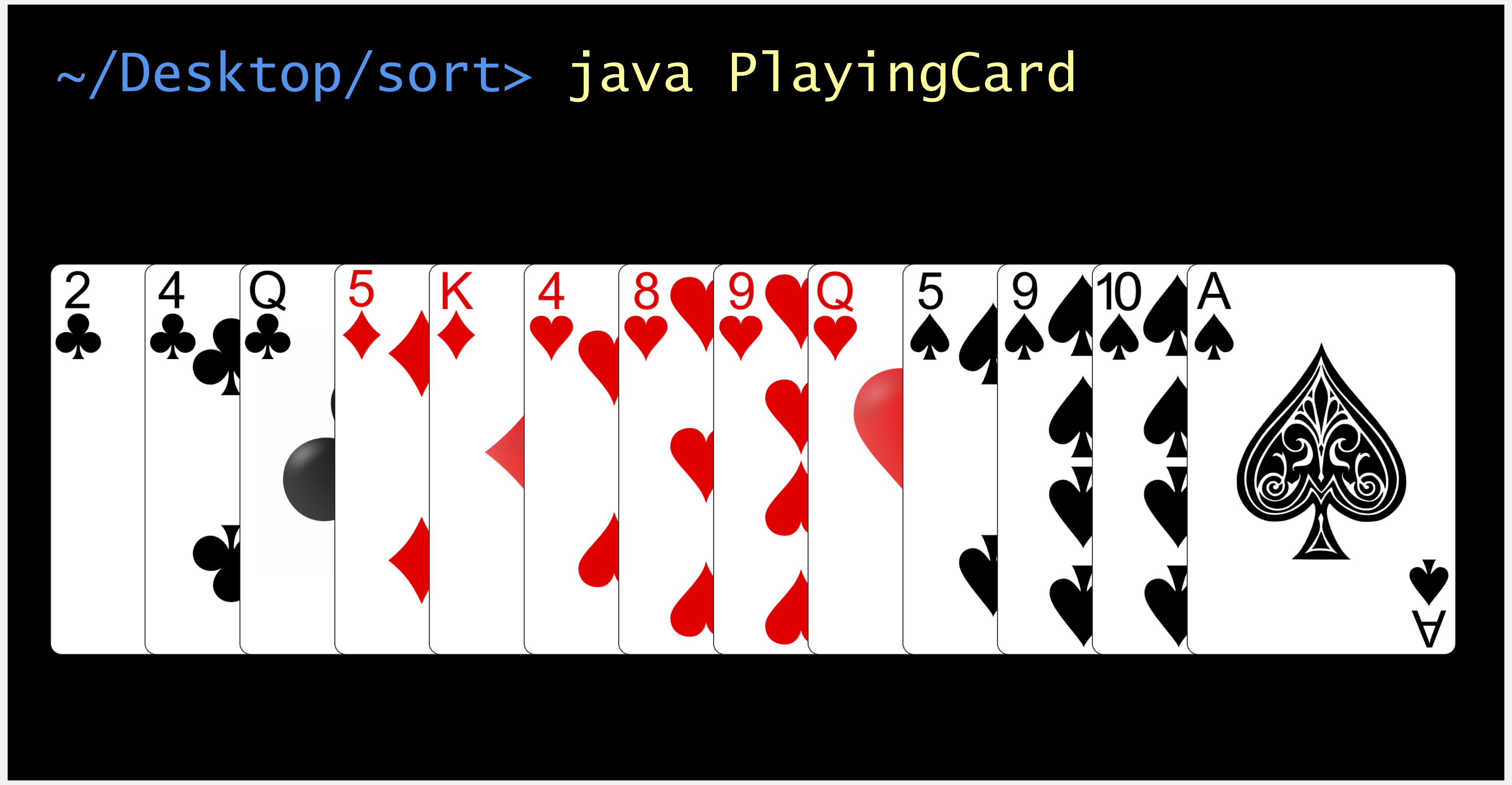
```
~/Desktop/sort> java Experiment 10
0.08614716385210452
0.09054270895414829
0.10708746304898642
0.21166190071646818
0.363292849257276
0.460954145685913
0.5340026311350087
0.7216129793703496
0.9003500354411443
0.9293994908845686
```

Sample sort clients

Goal. General-purpose sorting function.

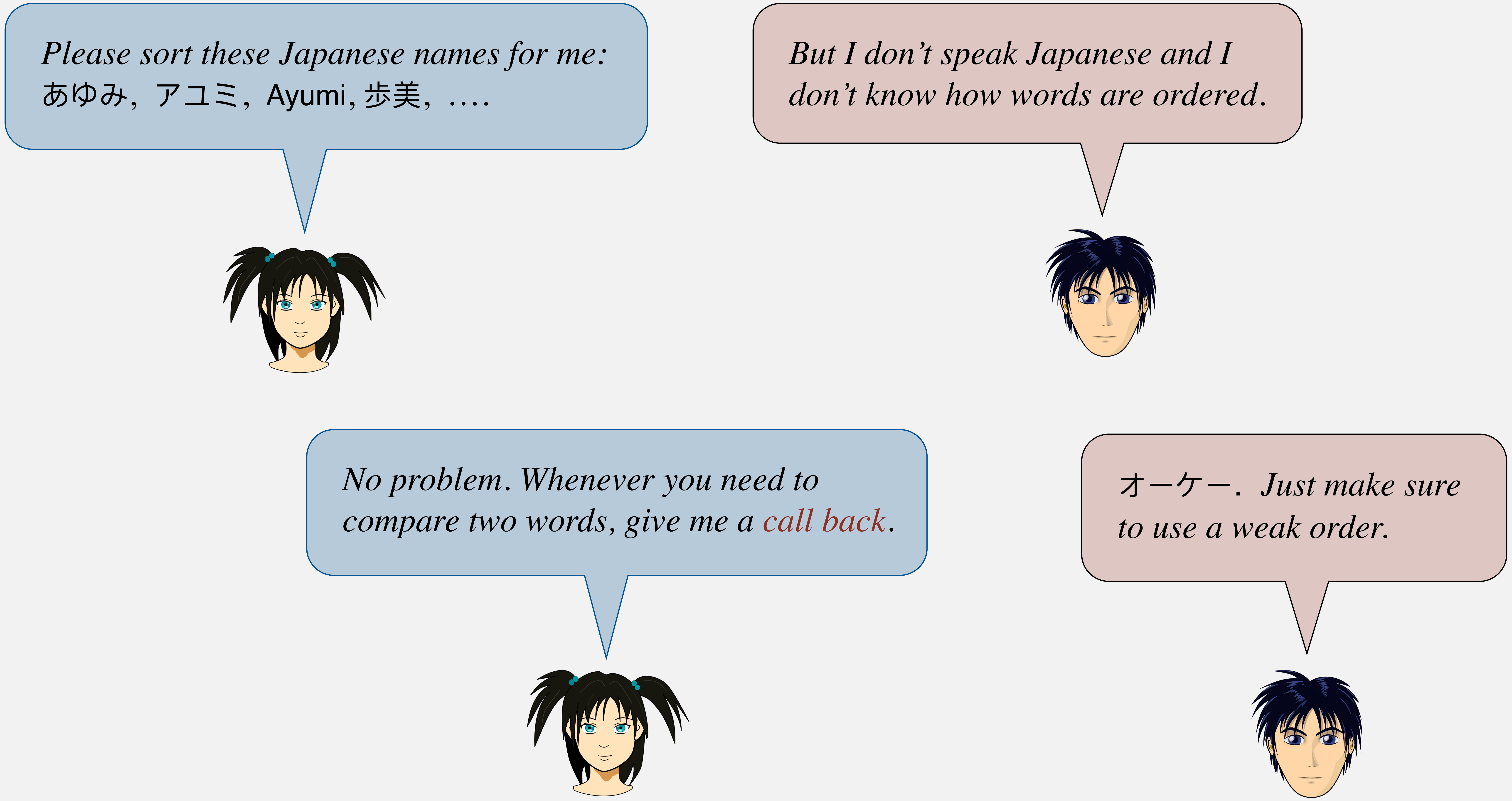
Ex 3. Sort playing cards by suit and rank.

```
public class PlayingCard
{
   public static void main(String[] args)
   {
      PlayingCard[] cards = deal(13);
      Insertion.sort(cards);
      draw(cards);
   }
}
```



How can a single function sort any type of data?

Goal. General-purpose sorting function.



Callbacks

Goal. General-purpose sorting function.

Solution. Callback = reference to executable code passed to other code and later executed.

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

← in effect, client passes `compareTo()` method to `sort()` function; the callback occurs when `sort()` invokes `compareTo()`

Implementing callbacks.

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

Java interfaces

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

java.lang.Comparable

```
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

Callbacks in Java: roadmap

client (StringSorter.java)

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

interface (Comparable.java)

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

String[] is a subtype of Comparable[]

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: sorting 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?**

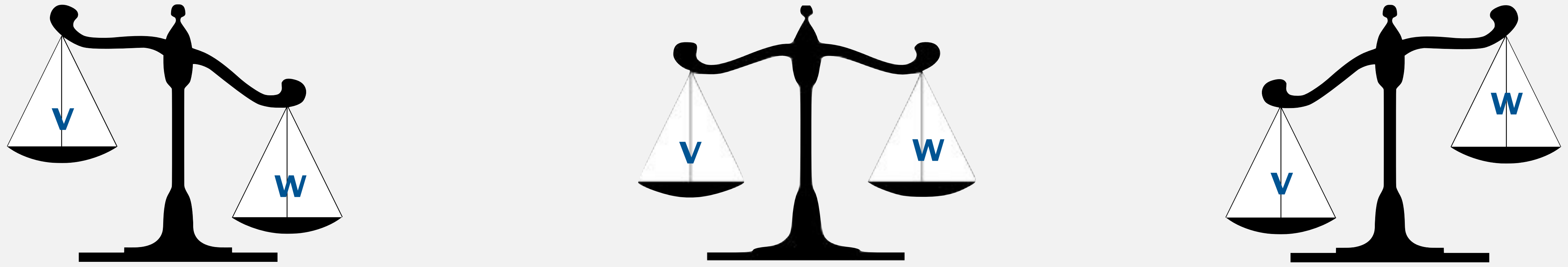
- **A.** String.java won't compile.
- **B.** StringSorter.java won't compile.
- **C.** Insertion.java won't compile.
- **D.** Insertion.java will throw a run-time exception.

Comparable API

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

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

API requirement:
the binary relation
`v.compareTo(w) <= 0`
is a weak order



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`, `java.util.Date`, ...

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

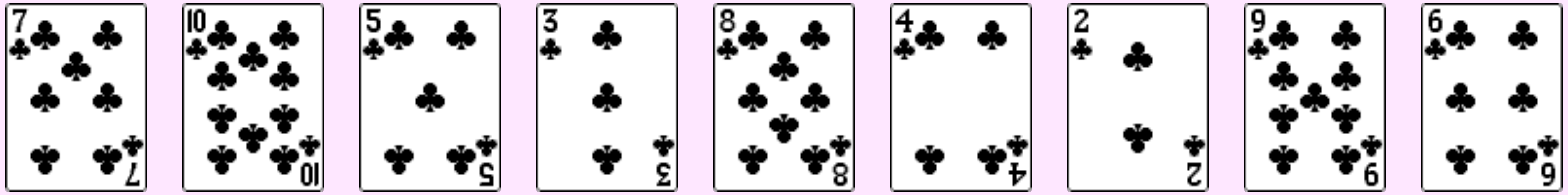
https://algs4.cs.princeton.edu

2.1 Elementary Sorts

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

Selection sort demo

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



initial array

Selection sort: visualization

Visualization. Sort vertical bars by length.

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

▲ algorithm position

in order

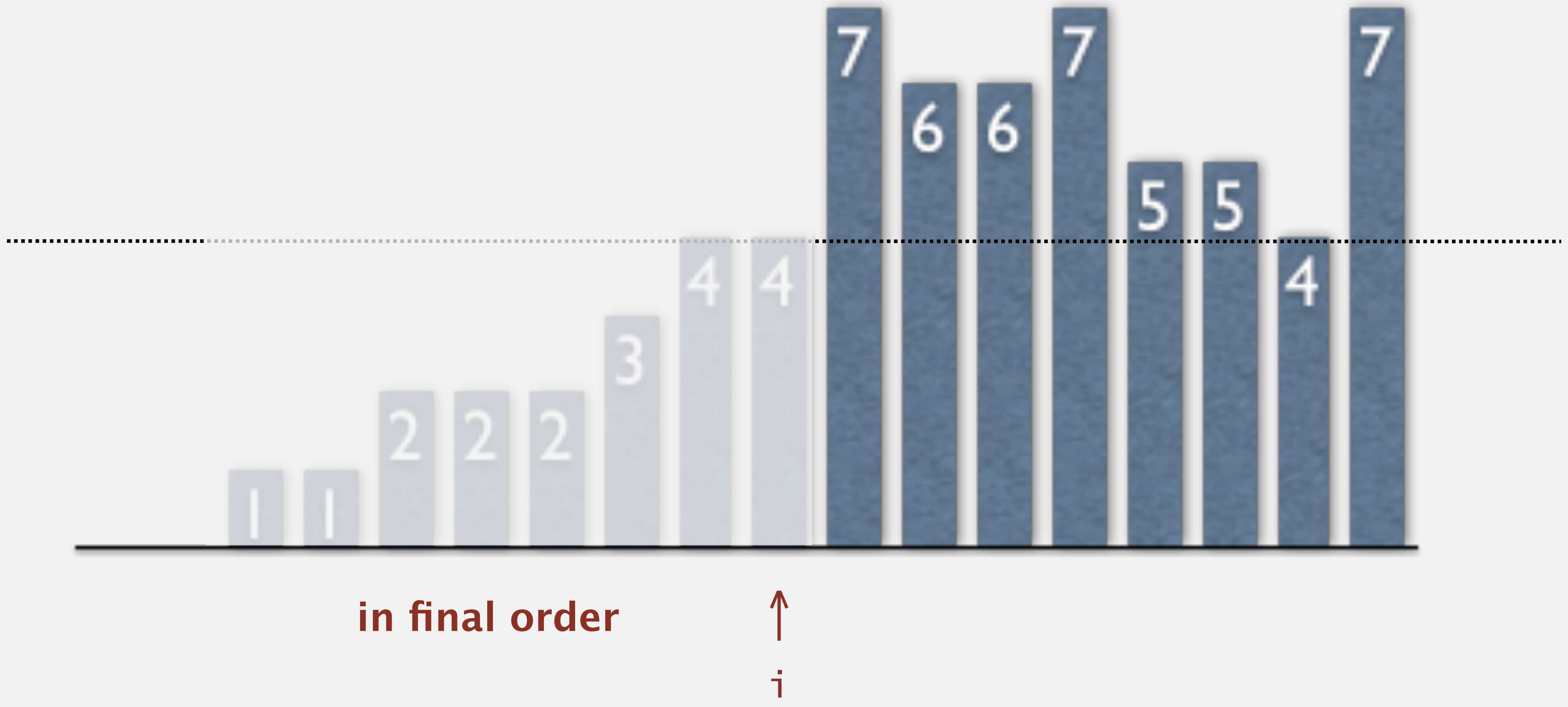
not yet seen

Selection sort

Algorithm. ↑ scans from left to right.

Invariants.

- Entries the left of ↑ (including ↑) are 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:

- Advance pointer i one position to right.

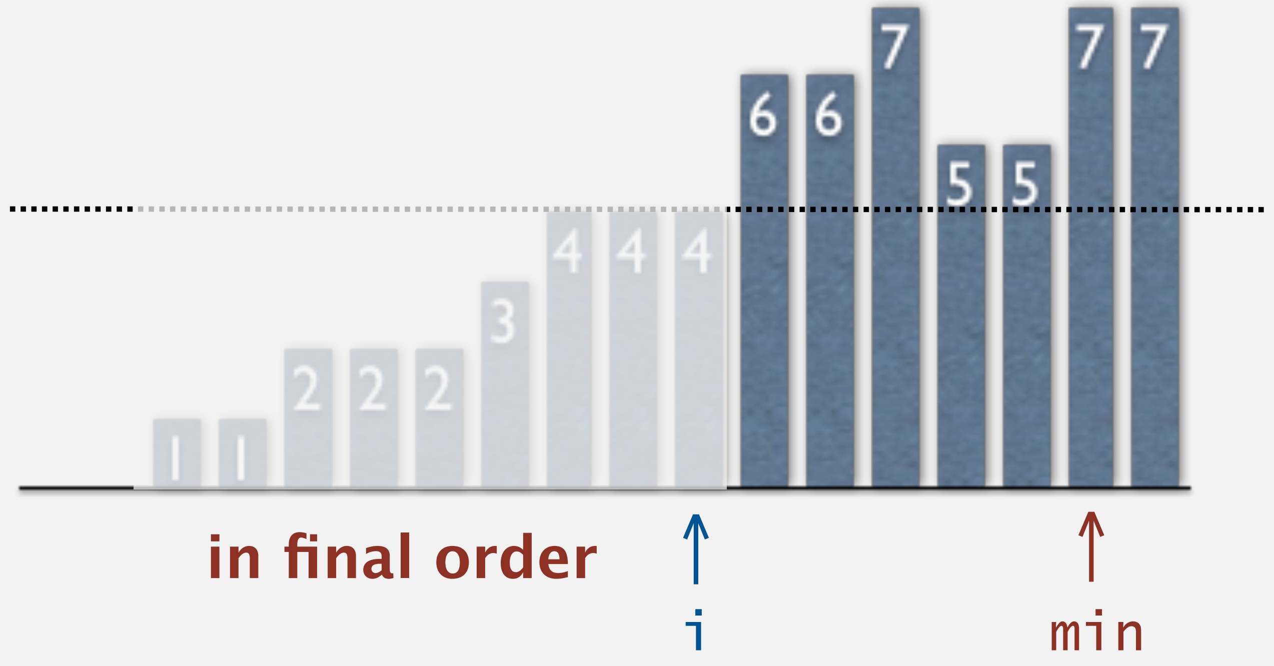
```
i++;
```

- Identify index min 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 a[i] and a[min].

```
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 item w ?

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

less("aardvark", "zebra") returns true

polymorphic method call

use interface type as argument
⇒ method works for all subtypes

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

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

Java arrays are "covariant"
(e.g., String[] is a subtype of Object[])

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

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

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 $(n-1) + (n-2) + \ldots + 1 + 0 \sim \frac{1}{2} n^2$ compares and n exchanges to sort any array of n items.

i	min	a[] 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

Running time insensitive to input. $\Theta(n^2)$ compares. ← even if input array is sorted

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

In place. $\Theta(1)$ extra space.

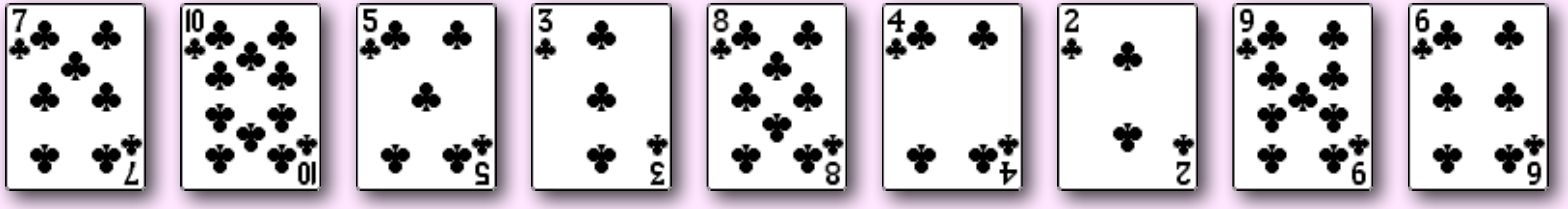
https://algs4.cs.princeton.edu

2.1 Elementary Sorts

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

Insertion sort demo

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



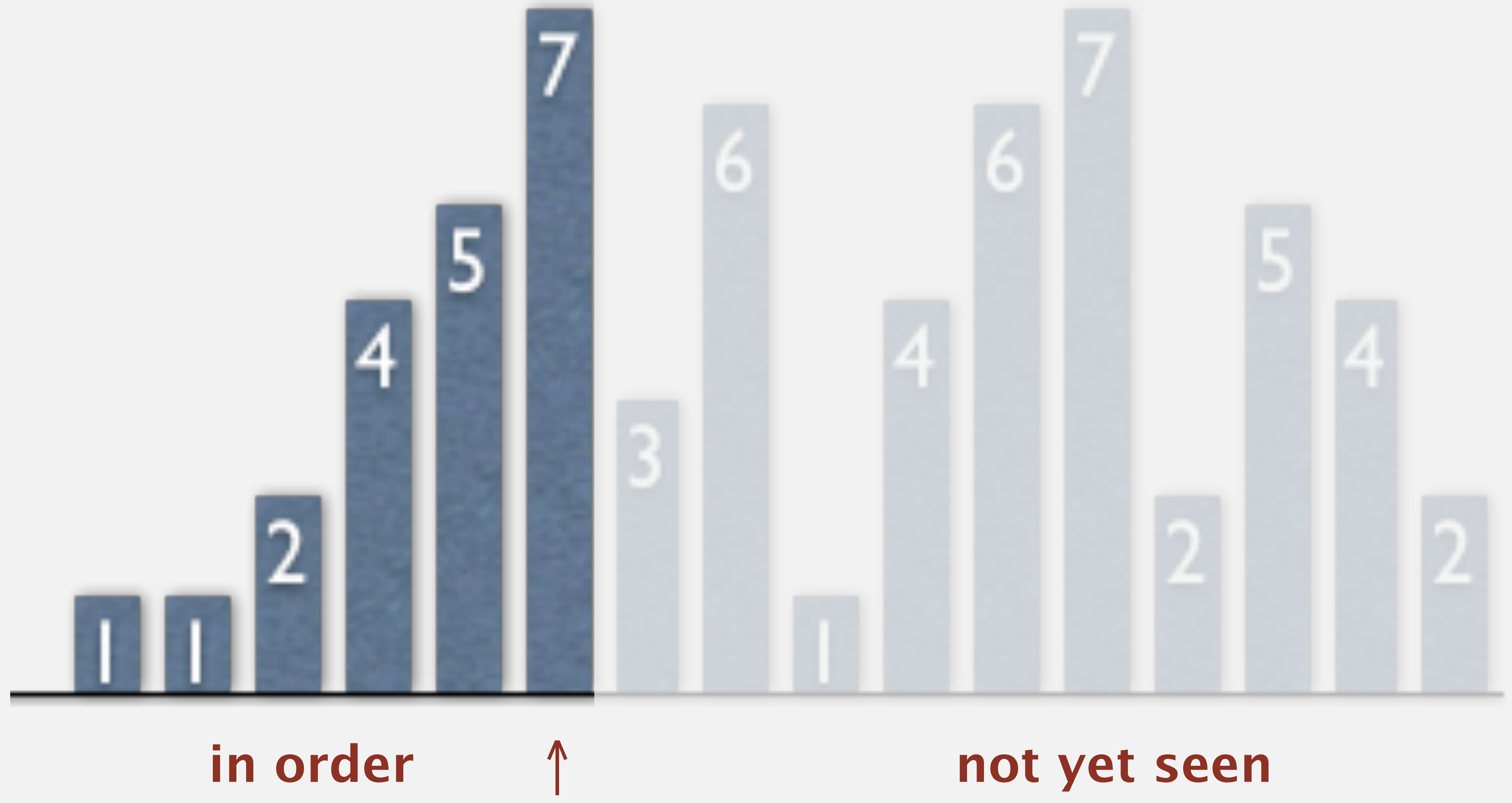
initial array

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.

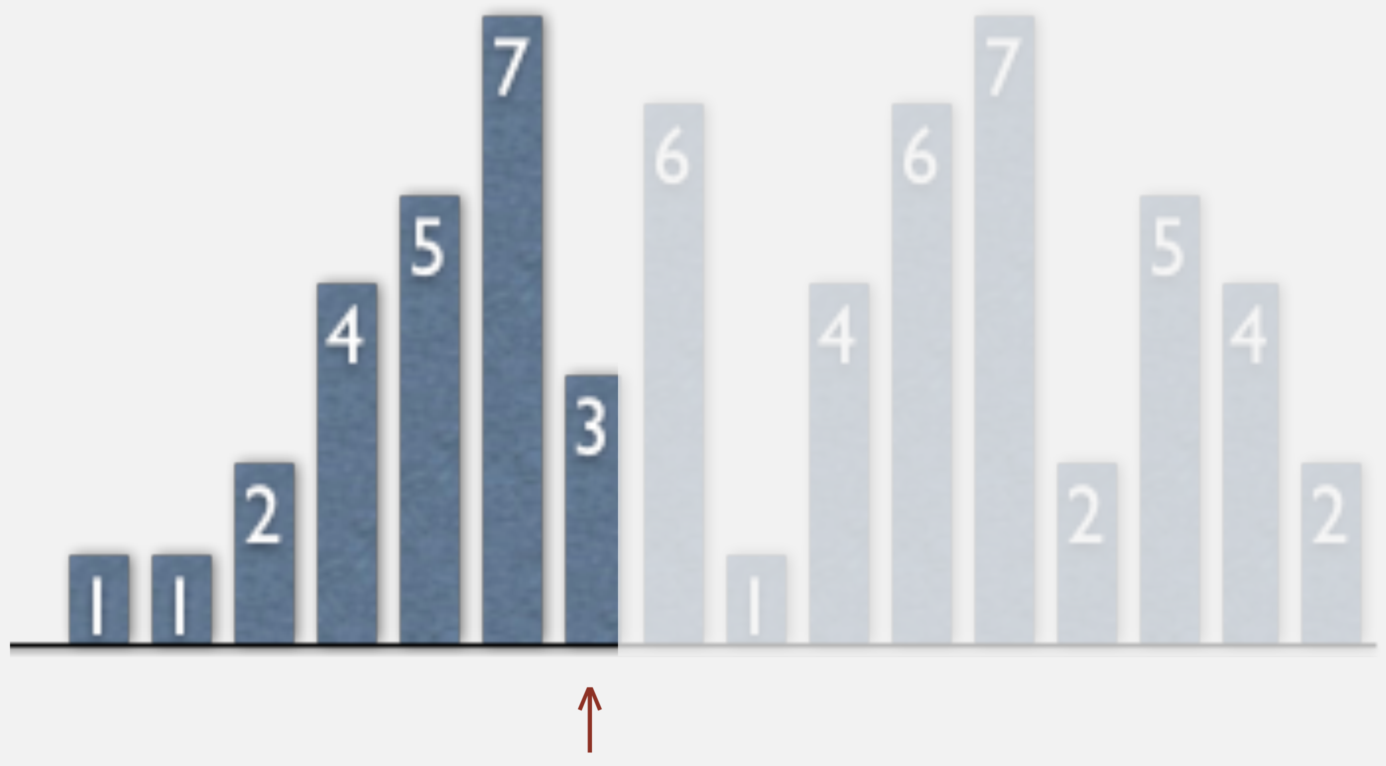


Insertion sort: inner loop

To maintain algorithm invariants:

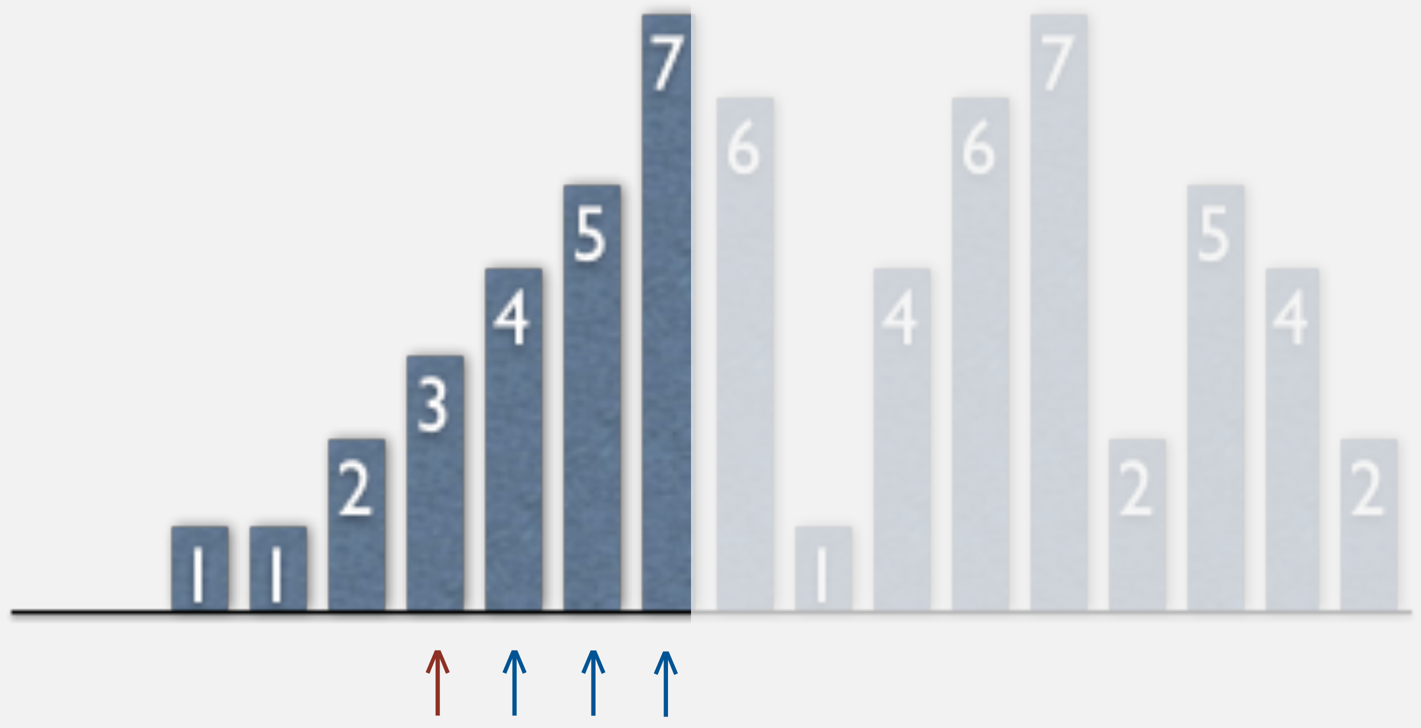
- Advance pointer i one position to right.

```
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 */  }

}
```

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

Elementary sorts: quiz 4

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

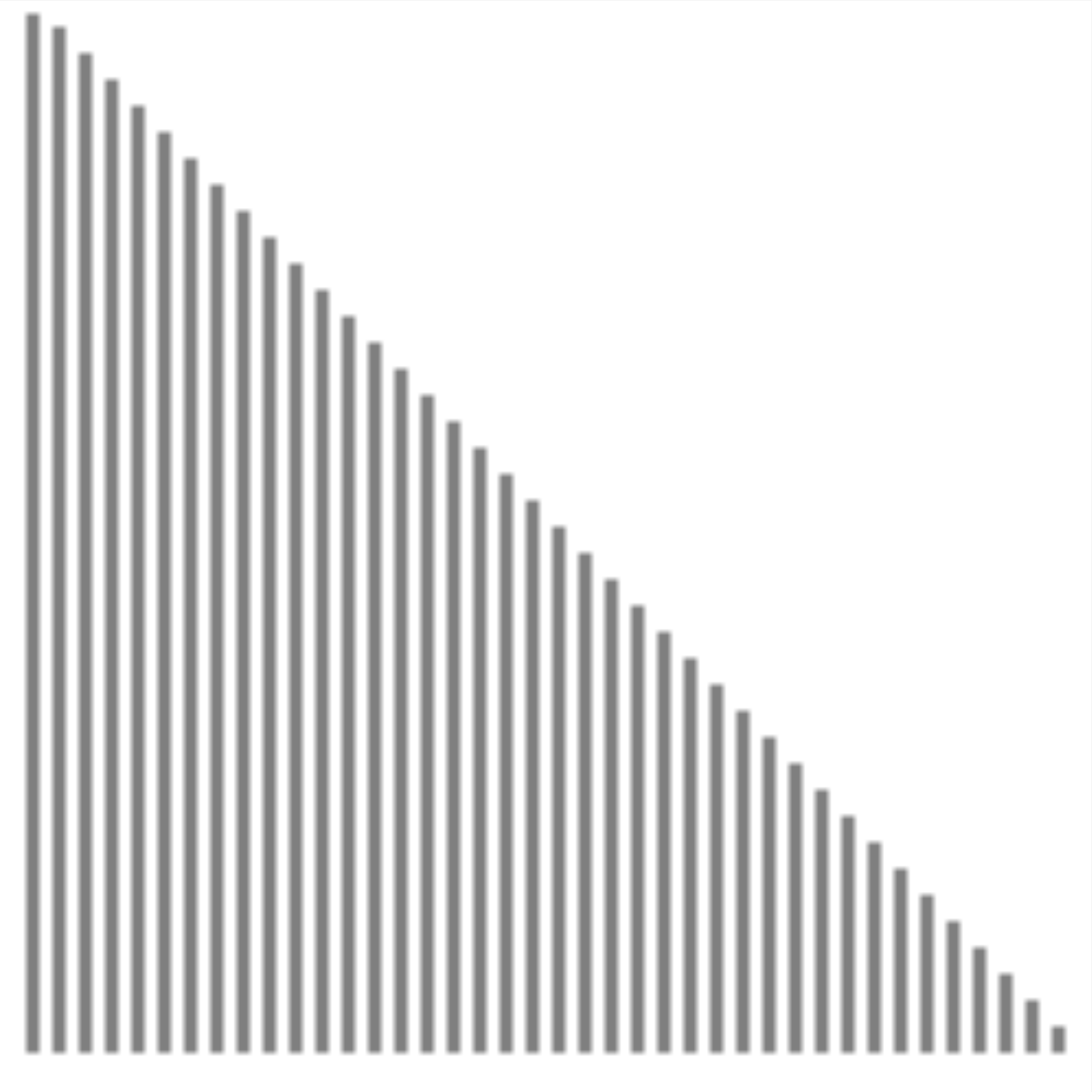
D. $\sim n^2$

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.

$0 + 1 + 2 + \ldots + (n-1) \sim \frac{1}{2} n^2$



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

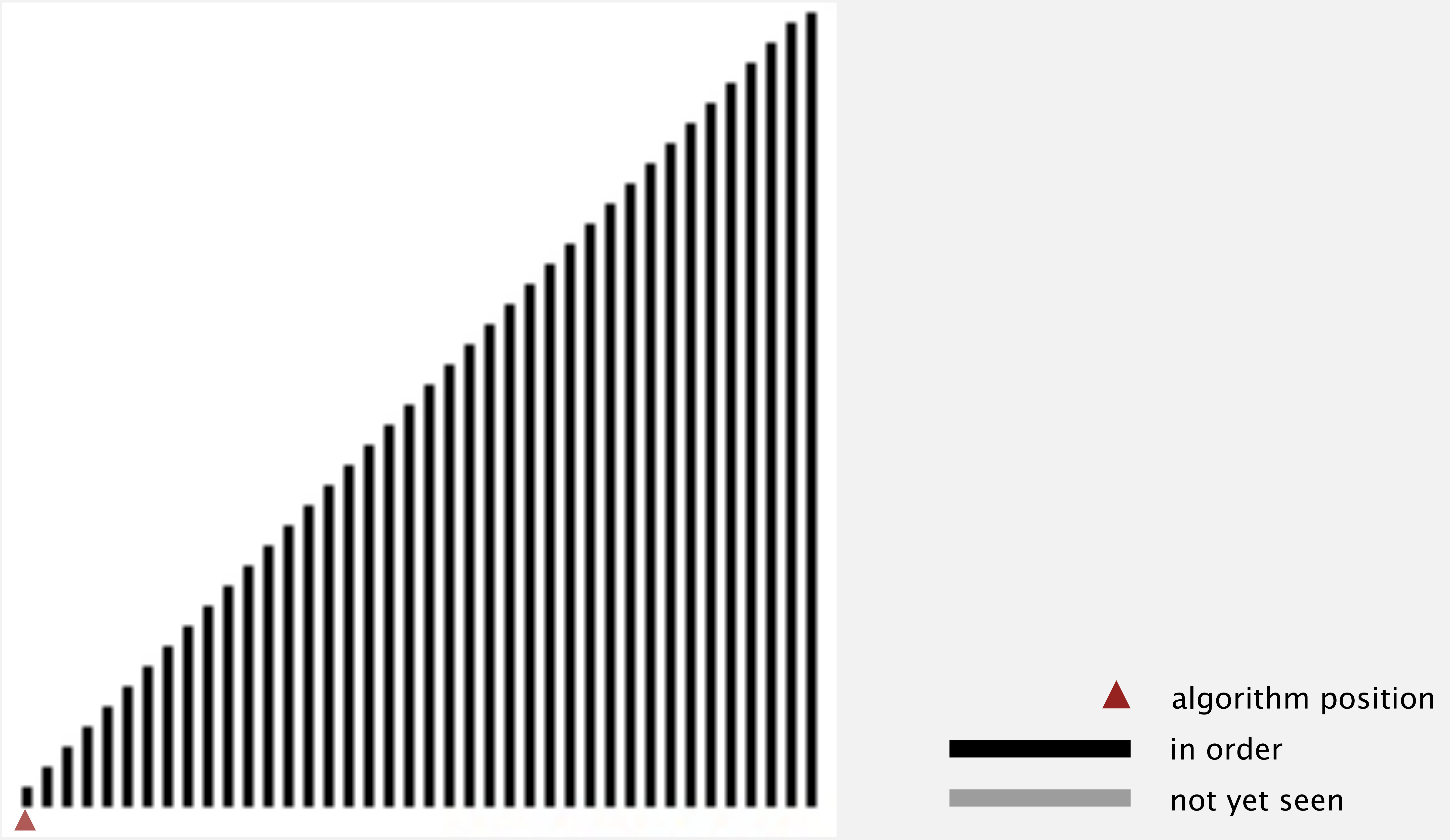
▲ algorithm position

in order

not yet seen

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.



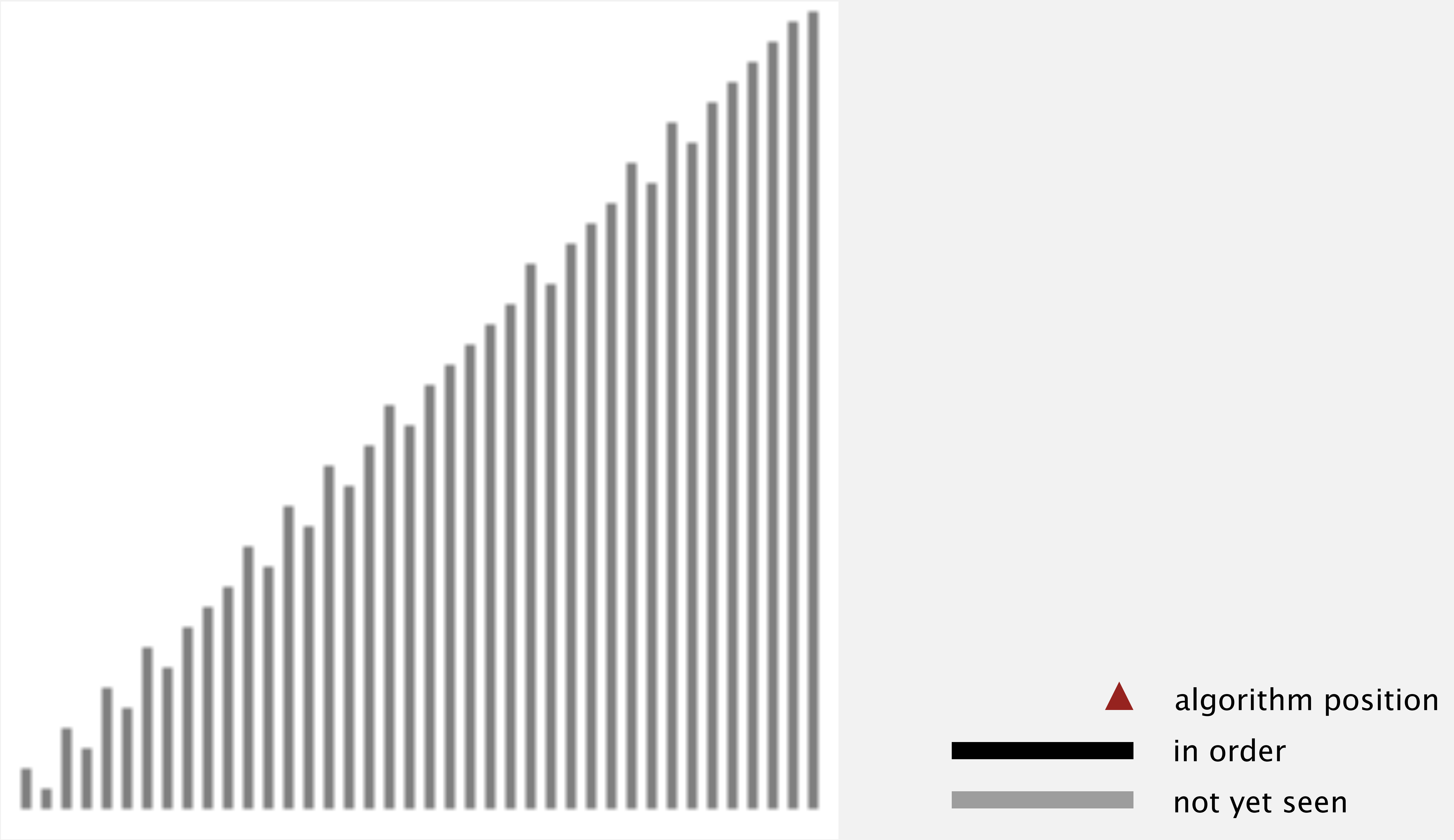
http://www.sorting-algorithms.com/insertion-sort

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; 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 still makes $\Theta(n^2)$ array accesses in worst case.

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***

Binary search

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

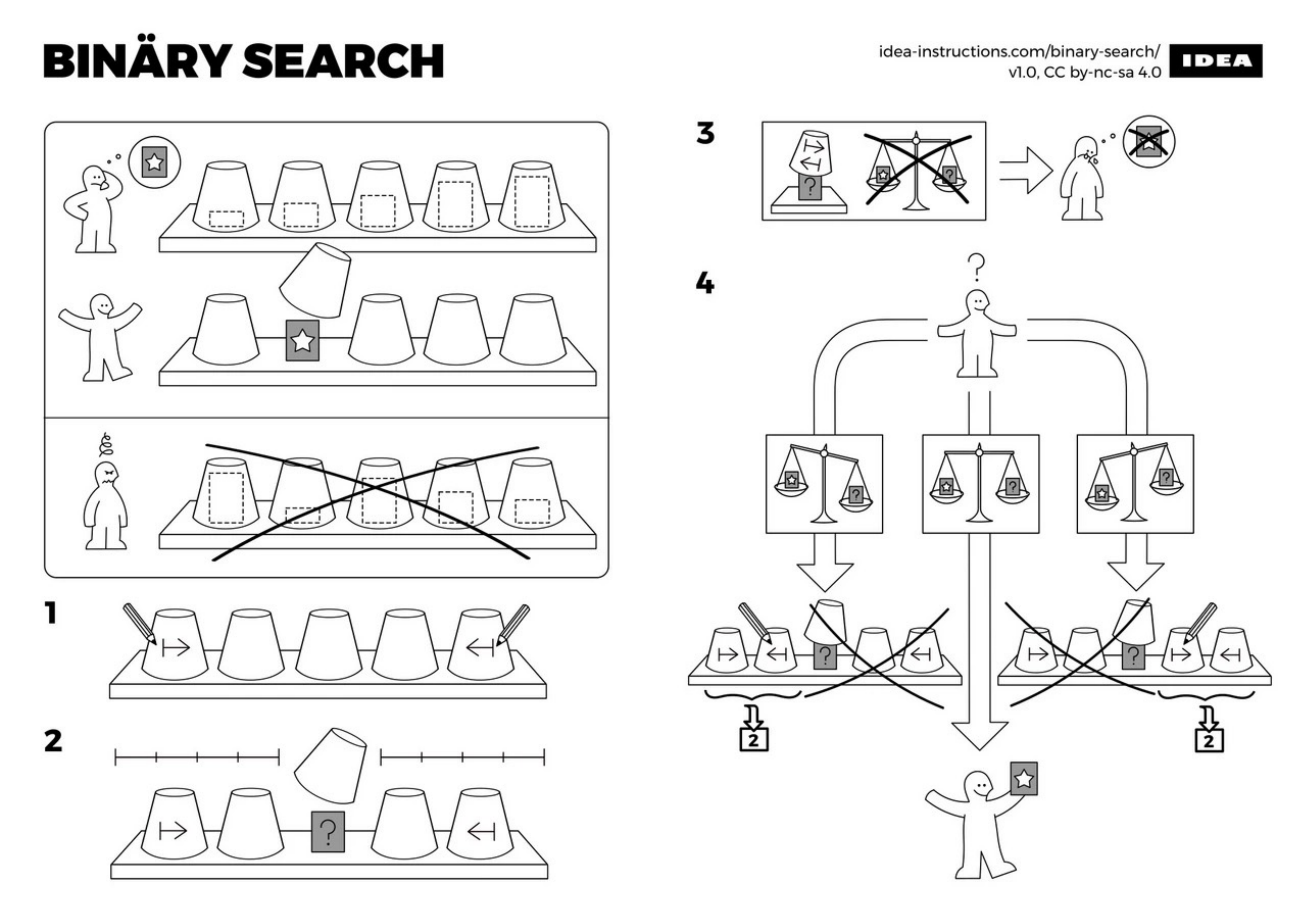
Binary search. Compare search key with middle entry.

- 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: assembly instructions



Binary search: implementation

Trivial to implement?

- First binary search published in 1946.
- First bug-free one in 1962.
- Bentley experiment: 90% of programmers implement it incorrectly.
- Bug in Java's `Arrays.binarySearch()` discovered in 2006.

and in C, C++, ...

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]`.

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

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

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

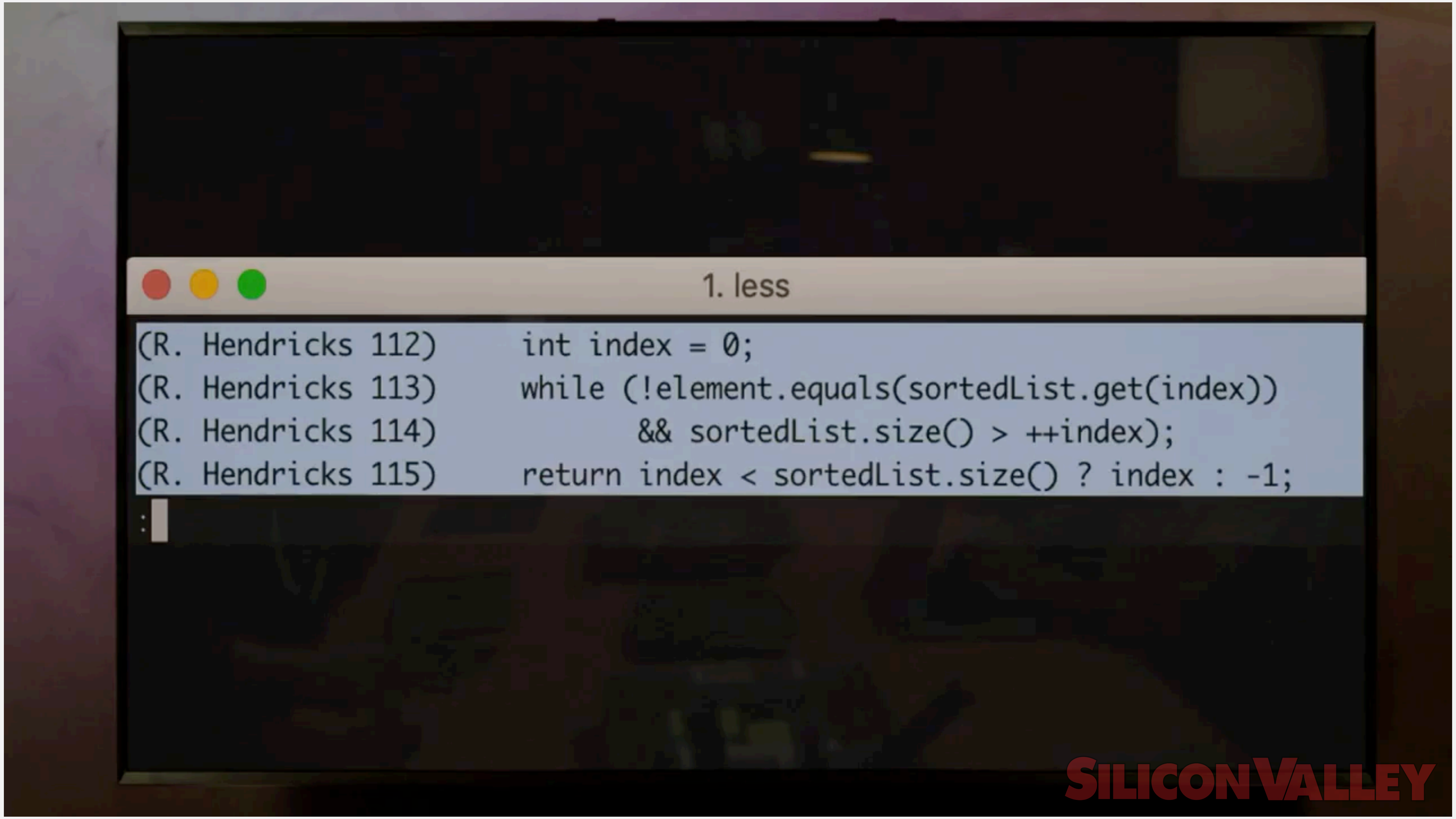
slightly better than 2×, due to elimination of `a[mid]` from subarray (or early termination of `while` loop)

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

$$n \rightarrow n/2 \rightarrow n/4 \rightarrow n/8 \rightarrow \cdots \rightarrow 2 \rightarrow 1$$

$$\underbrace{\qquad\qquad}_{1 + \log_2 n}$$

Binary search vs. sequential search



3-SUM

3–SUM. Given an array of n distinct integers, count number of triples that sum to 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.

3-SUM: A $\Theta(N^2 \log N)$ ALGORITHM

Algorithm.

- Step 1: Sort the n distinct numbers.
- Step 2: For each pair `a[i]` and `a[j]`:
  binary search for `-(a[i] + a[j])`.

Analysis. Running time is $\Theta(n^2 \log n)$.

- Step 1: $\Theta(n^2)$ with selection sort.
- Step 2: $\Theta(n^2 \log n)$ with binary search.

$\Theta(n^2)$ binary searches in an array of length n

input

`30 -40 -20 -10 40 0 10 5`

sort

`-40 -20 -10 0 5 10 30 40`

binary search

```
(-40, -20)   60
(-40, -10)   50
(-40,   0)   40
(-40,   5)   35
(-40,  10)   30
     ⋮        ⋮
(-20, -10)   30
     ⋮        ⋮
(-10,   0)   10
     ⋮        ⋮
( 10,  30)  -40
( 10,  40)  -50
( 30,  40)  -70
```

count only if i < j < k to avoid both triple counting and 10 + 10 + −20

3-Sum

3–Sum. Given an array of n distinct integers, find three such that $x + y + z = 0$.

Version 0. $\Theta(n^3)$ time. ✔

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

Version 2. $\Theta(n^2)$ time. [not much harder]

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 no $\Theta(n)$ time algorithm is possible.

Summary

Comparable interface. Java framework for comparing items.

Selection sort. $\Theta(n^2)$ compares; $\Theta(n)$ exchanges.

Insertion sort. $\Theta(n^2)$ compares and exchanges in the worst case.

Binary search. Search a sorted array using $\Theta(\log n)$ compares.