



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. Given an array of n elements, rearrange in ascending order by key.

element →

sort 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

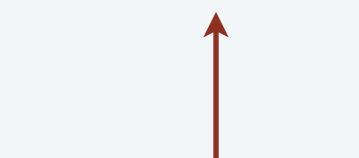
Sorting problem

Goal. Given an array of n elements, rearrange in ascending order by key.

sort key →

element →

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



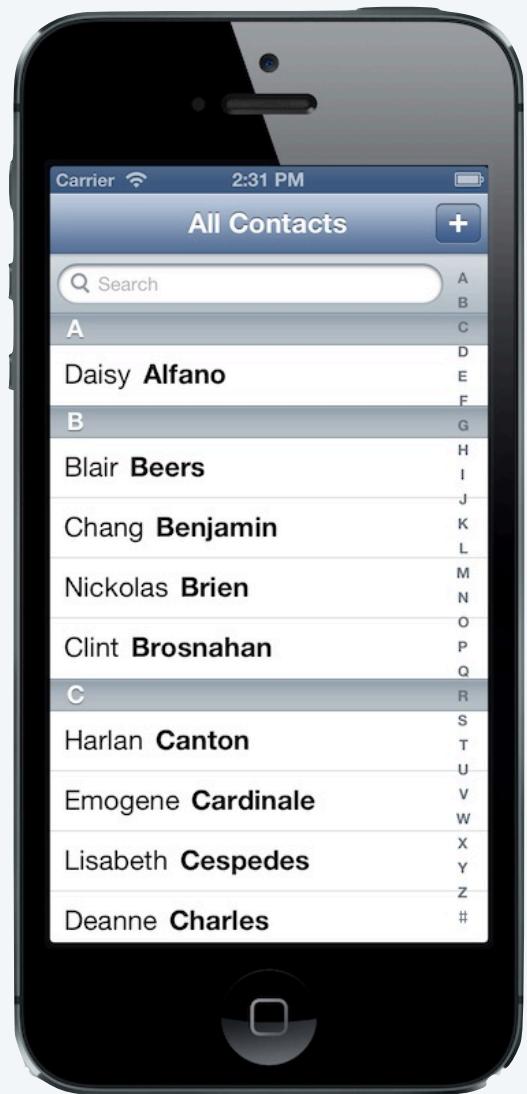
sorted by key



sorting hat

Sorting problem

Familiar examples.



alphabetical order

International Departures				
Flight No	Destination	Time	Gate	Remarks
CX7183	Berlin	7:50	A-11	Gate closed
QF3474	London	7:50	A-12	Gate closed
BA372	Paris	7:55	B-10	Boarding
AY6554	New York	8:00	C-33	Boarding
KL3150	San Francisco	8:00	F-15	Boarding
BA8903	Manchester	8:05	B-12	See ticket desk
BA710	Los Angeles	8:10	C-12	Check-in open
QF3371	Hong Kong	8:15	F-10	Check-in open
MA4866	Barcelona	8:15	F-12	Check-in at kiosks
CX7221	Copenhagen	8:20	G-32	Check-in at kiosks

chronological order

Video name	Views (billions) ▲
Gangnam Style	5.46
Uptown Funk	5.48
Phonics Song	6.28
Shape of You	6.41
See You Again	6.56
Bath Song	7.00
Johny John Yes Papa	7.01
Wheels on the Bus	7.10
Despacito	8.65
Baby Shark Dance	15.59

numerical order (ascending)

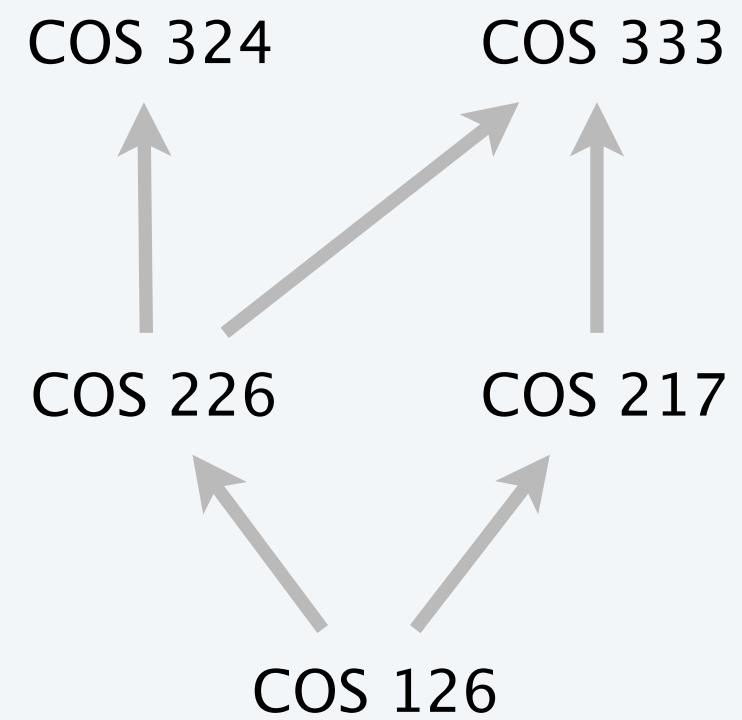
Sorting problem. Is well-defined if there is a binary relation \leq that satisfies:

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

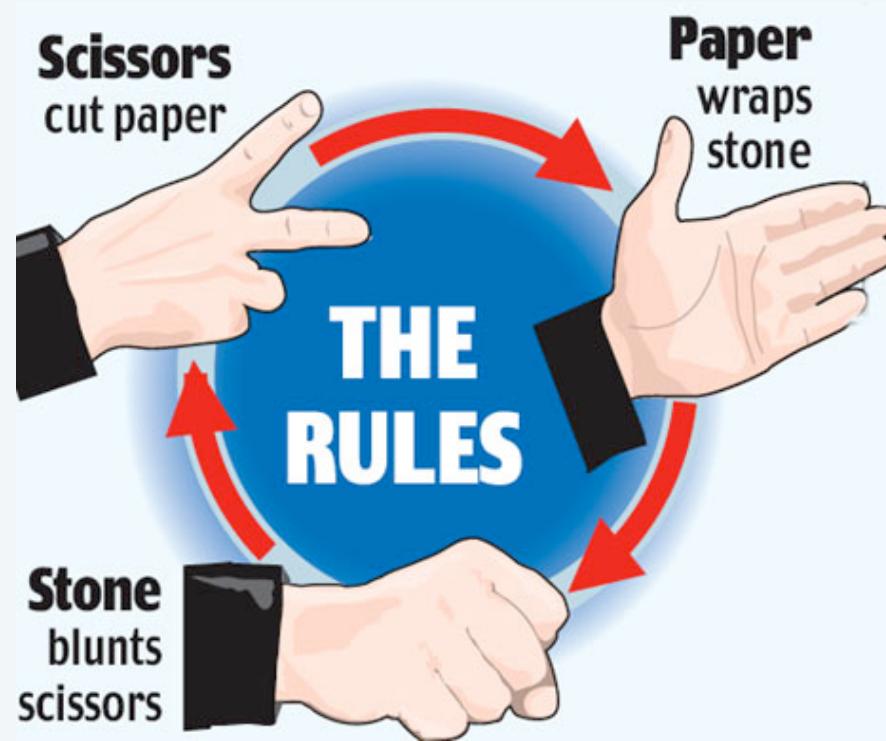
← *mathematically, a “weak order”
(a “total order” but might have both $v \leq w$ and $w \leq v$
for distinct elements v and w)*

Sorting problem

Familiar non-examples.



course prerequisites
(violates comparability)



Ro-sham-bo order
(violates transitivity)

```
~/cos226/sort> jshell
Math.sqrt(-1.0) <= Math.sqrt(-1.0);
false
```

the `<=` operator for double
(violates comparability)

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

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

Sample sort clients

Goal. General-purpose sorting function.

Ex 1. Sort strings in alphabetical order. ← *lexicographic order
(Unicode)*

```
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]);  
    }  
}
```

```
~/cos226/sort> more words3.txt  
BED BUG DAD YET ZOO ... ALL BAD YES  
  
~/cos226/sort> java-algs4 StringSorter < words3.txt  
ALL BAD BED BUG DAD ... YES YET ZOO  
[suppressing newlines]
```

Unicode	value
:	:
A	65
B	66
C	67
D	68
:	:
Θ	1,012
:	:
💖	128,150
:	:
Unicode character ordering	

Sample sort clients

Goal. General-purpose sorting function.

Ex 2. Sort real numbers in numerical order (ascending).

```
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.uniformDouble();  
        Insertion.sort(a);  
        for (int i = 0; i < n; i++)  
            StdOut.println(a[i]);  
    }  
}
```

```
~/cos226/sort> java-algs4 Experiment 10  
0.08614716385210452  
0.09054270895414829  
0.10708746304898642  
0.21166190071646818  
0.363292849257276  
0.460954145685913  
0.5340026311350087  
0.7216129793703496  
0.9003500354411443  
0.9293994908845686
```

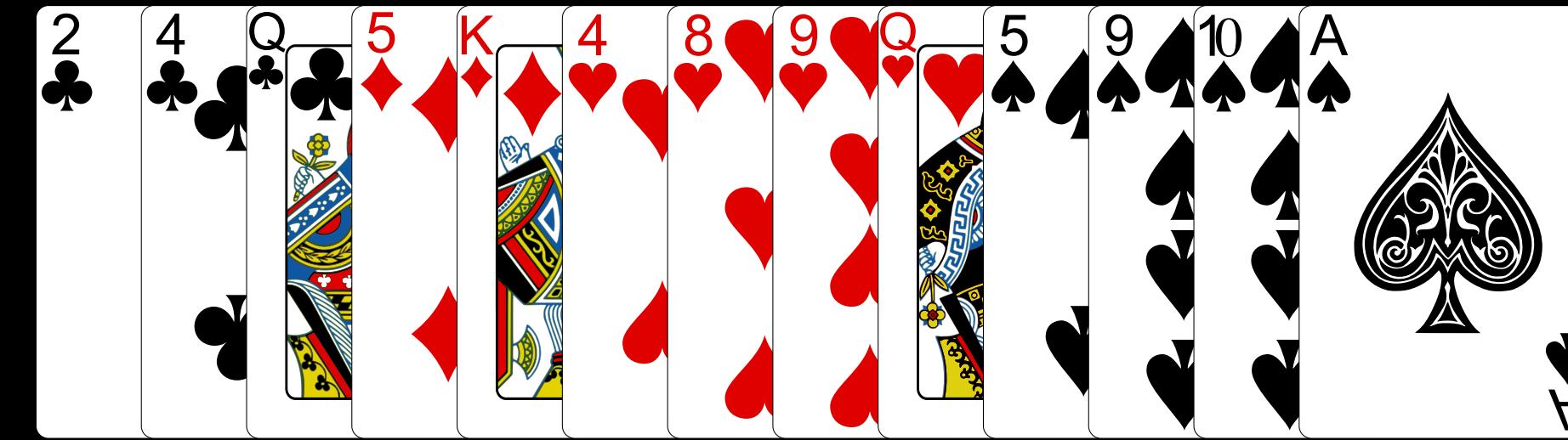
Sample sort clients

Goal. General-purpose sorting function.

Ex 3. Sort playing cards in **suit-major** order.

```
public class Deck {  
    ...  
  
    public static void main(String[] args) {  
        int n = Integer.parseInt(args[0]);  
        PlayingCard[] cards = deal(n);  
        Insertion.sort(cards);  
        draw(cards);  
    }  
}
```

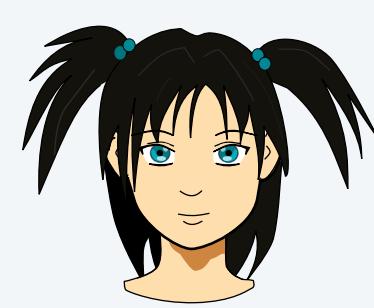
```
~/cos226/sort> java-algs4 Deck 13
```



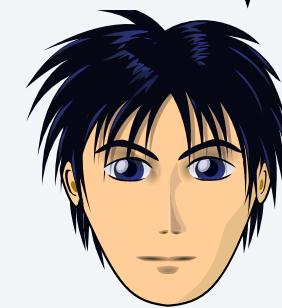
How can a single function sort any type of data?

Goal. General-purpose sorting function.

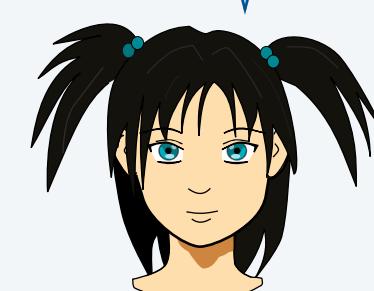
*Please sort these Japanese names for me:
あゆみ, アユミ, Ayumi, 歩美,*



*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 weak order.*



Callbacks

Goal. General-purpose sorting function.

Solution. **Callback** = reference to executable code passed to a function and later executed.

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

*← effectively, client passes compareTo()
method to sort() function;
the callback occurs when
sort() invokes compareTo()*



Implementing callbacks.

- Java: **interfaces**.
- Python, ML, Javascript: first-class functions.
- 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.

interface (java.lang.Comparable)

```
public interface Comparable<Item> {  
    public int compareTo(Item that); ← contract: method with this signature  
(and prescribed behavior)  
}
```

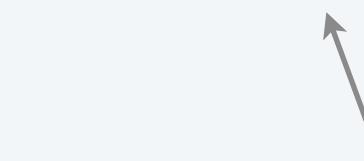
Class that implements interface. Must implement all interface methods.

```
public class String implements Comparable<String> { ← class promises to  
honor the contract  
    ...  
  
    public int compareTo(String that) { ← class abides by  
the contract  
        ...  
    }  
}
```

Java interfaces: properties

Subtype polymorphism.

- Interfaces are reference types.
- A class that implements an interface is a **subtype** of that interface.



e.g., can use an object of the subtype instead of interface type in assignment statements, method arguments, return values, ...

Dynamic dispatch. Java determines which interface method to call using the type of the referenced object at runtime.

Q. Why useful?

A. Enables callbacks.

- Design a single method that can sort strings, integers, or dates.
- Iterate over a collection without knowing the underlying representation.
- Intercept and process mouse clicks in a Java app.
- ...

```
// can assign String object  
// to variable of type Comparable  
Comparable x = "Hello";  
Comparable y = "World";
```

```
// calls String compareTo()  
int result1 = x.compareTo(y);
```

```
// can also assign Date object  
// to variable of type Comparable  
x = new Date(2025, 02, 11);  
y = new Date(1969, 07, 16);
```

```
// calls Date compareTo()  
int result2 = x.compareTo(y);
```

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);  
}
```

sort implementation (Insertion.java)

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

String[] is a subtype
of Comparable[]

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 the clause `implements Comparable<String>` in the class declaration for `String`. What would be the consequence?

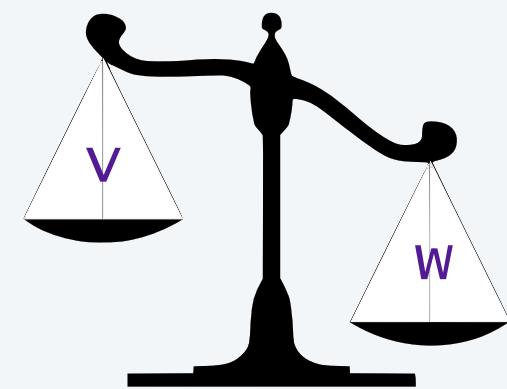
- A. Compile-time error in `String.java`.
- B. Compile-time error in `StringSorter.java`.
- C. Compile-time error in `Insertion.java`.
- D. Run-time exception in `Insertion.java`.

Comparable API

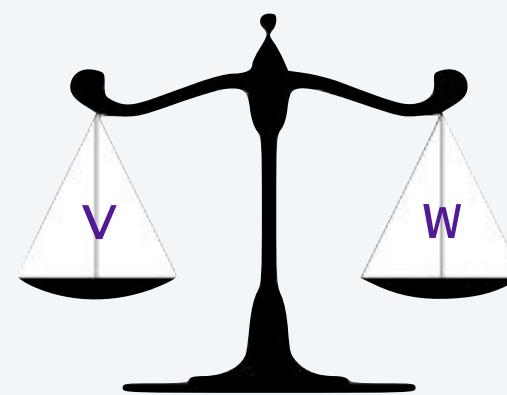
Requirement. Implement `compareTo()` so that `v.compareTo(w)`

- 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`).

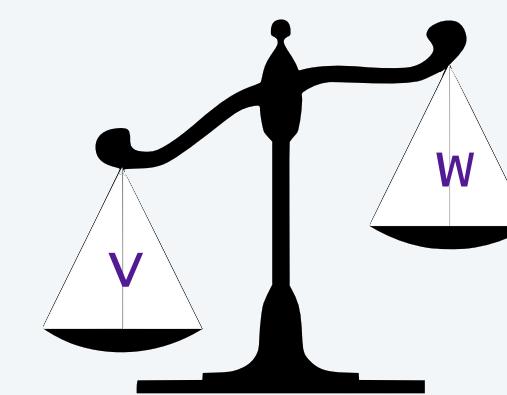
*API requirement:
the binary relation
`v.compareTo(w) <= 0`
is a weak order*



*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*



2.1 ELEMENTARY SORTS

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

ROBERT SEDGEWICK | KEVIN WAYNE

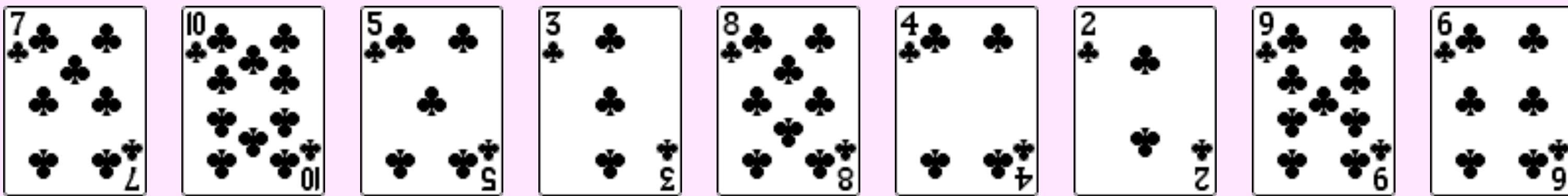
<https://algs4.cs.princeton.edu>

Selection sort demo



Algorithm. For each index i from 0 to $n - 1$:

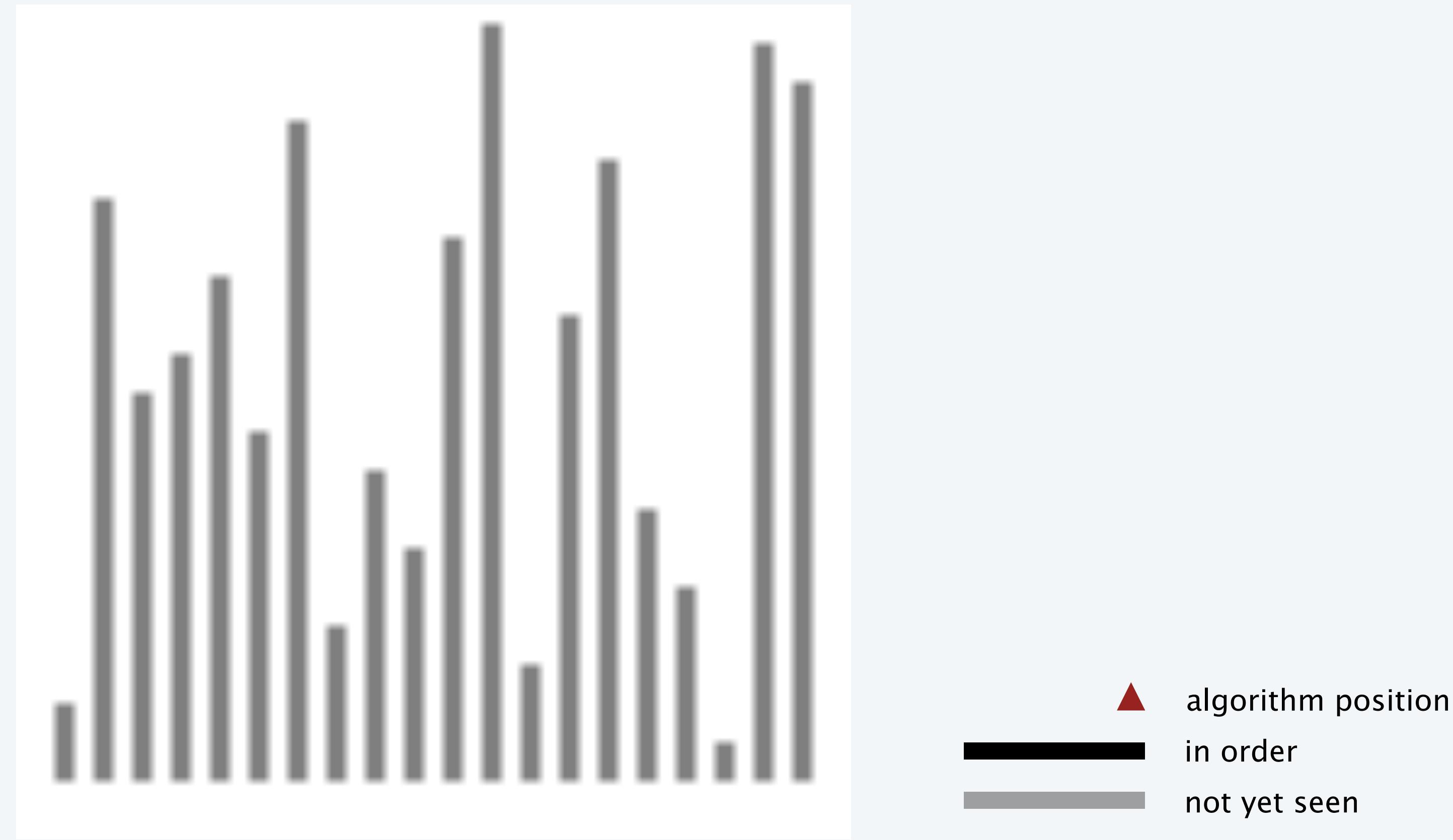
- Find index min of smallest remaining element.
- Swap elements at indices i and min .



initial array

Selection sort: visualization

Visualization. Sort vertical bars by length.



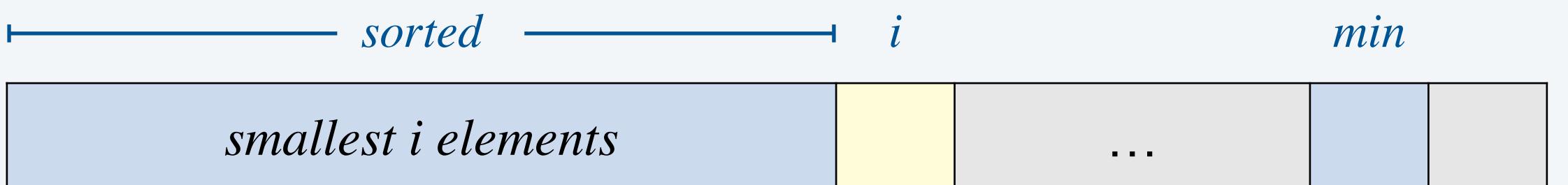
Selection sort invariants

Algorithm. For each index i from 0 to $n - 1$:

- Find index \min of smallest remaining element.
- Swap elements at indices i and \min .

Invariants.

before iteration i



after iteration i



Two useful sorting primitives (and a cost model)

Helper functions. Refer to data only through **compares** and **exchanges**. ← e.g., no calls to equals()

use as our cost model for sorting

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

```
private static boolean less(Comparable v, Comparable w) { ← less("aardvark", "zebra") returns true
    return v.compareTo(w) < 0;
}
```

↑
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

```
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 */  
    }  
}
```



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 $\sim \frac{1}{2} n^2$ compares and n exchanges to sort any array of n items.

Pf. Exactly $(n - i - 1)$ compares and 1 exchange in iteration i .

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

		a[]										
i	min	0	1	2	3	4	5	6	7	8	9	10
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. \leftarrow even if input array is sorted

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

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



2.1 ELEMENTARY SORTS

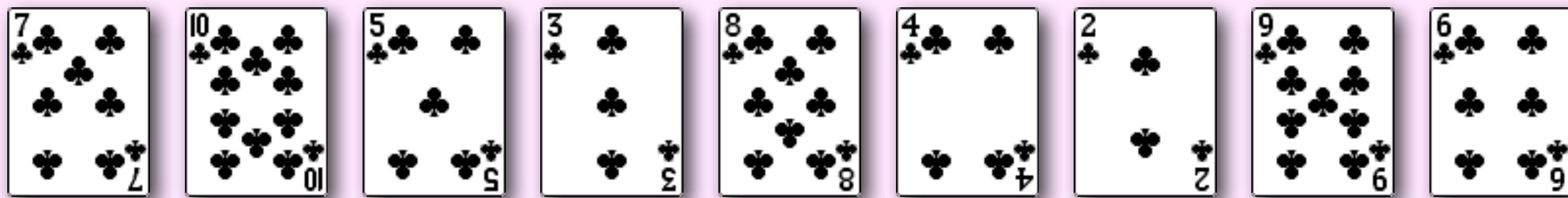
- ▶ *rules of the game*
- ▶ *selection sort*
- ▶ *insertion sort*
- ▶ *binary search*

Insertion sort demo



Algorithm. For each index $i = 0$ to $n - 1$:

- Let x be the element at index i .
- Repeatedly exchange x with each larger element to its immediate left.



initial array

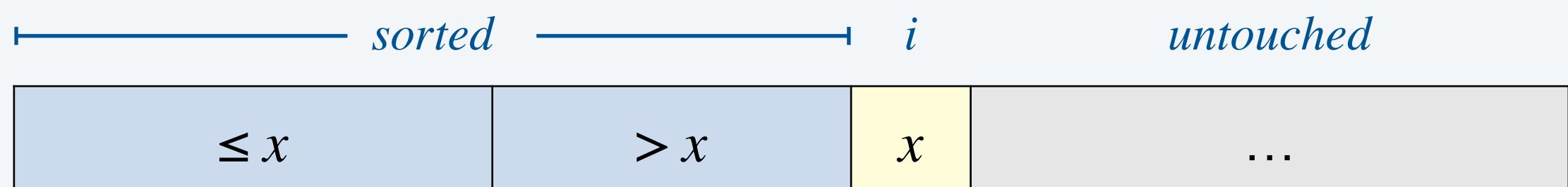
Insertion sort invariants

Algorithm. For each index $i = 0$ to $n - 1$:

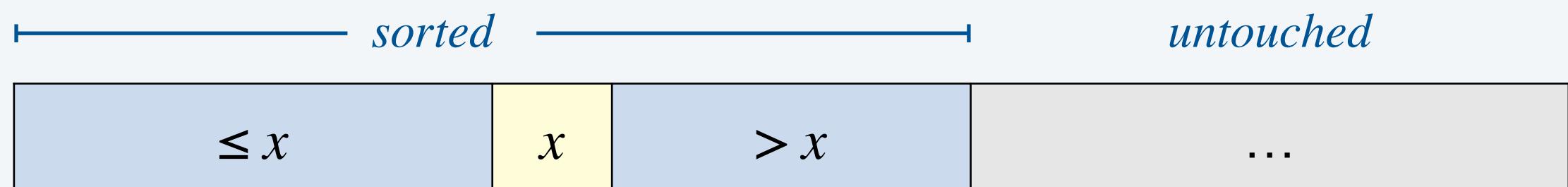
- Let x be the element at index i .
- Repeatedly exchange x with each larger element to its immediate left.

Invariants.

before iteration i



after iteration i



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>



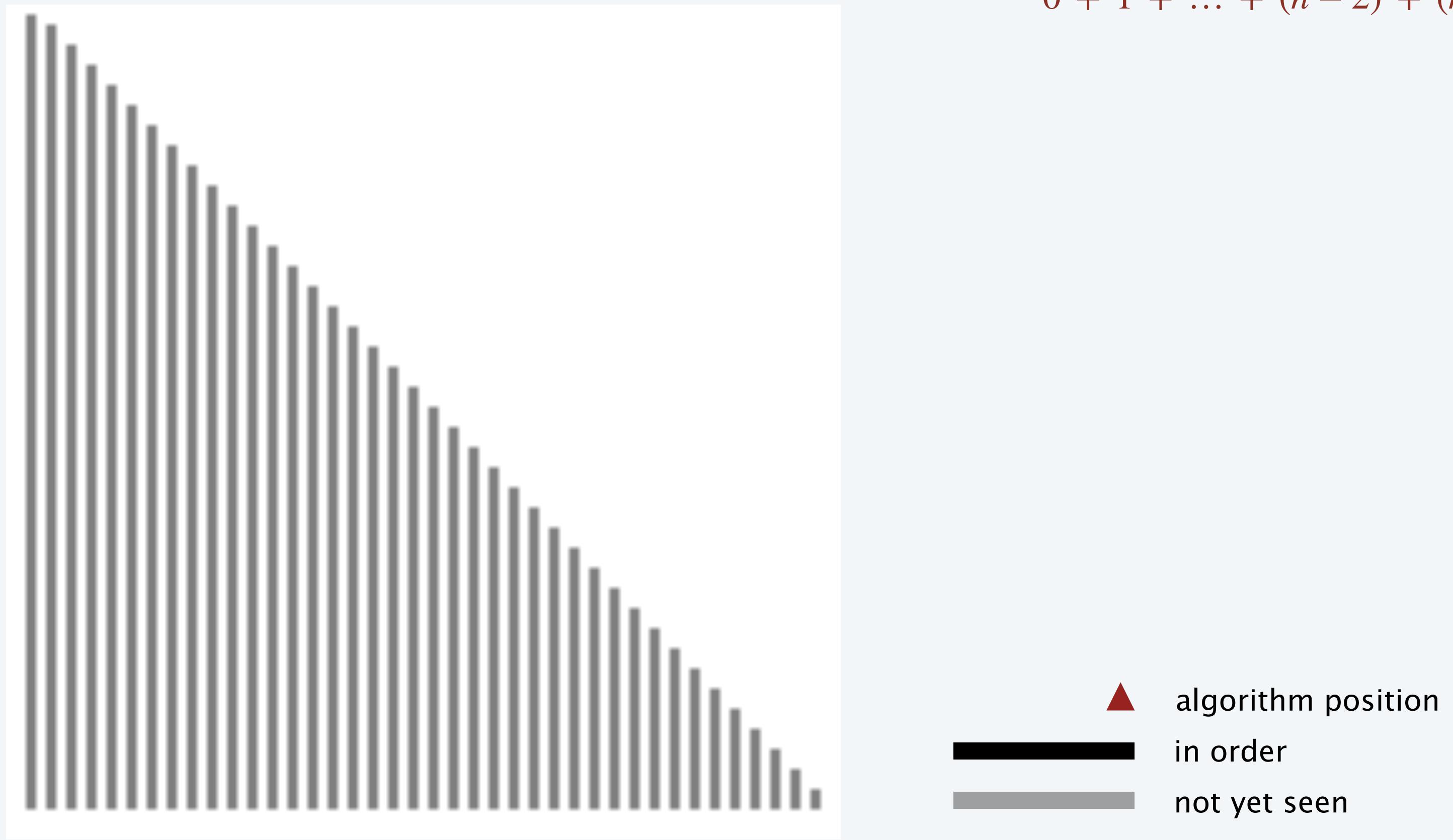
How many compares to insertion sort an array of n distinct keys in reverse order?

- A.** $\sim n$
- B.** $\sim 1/4 n^2$
- C.** $\sim 1/2 n^2$
- D.** $\sim n^2$

Insertion sort: running time 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.

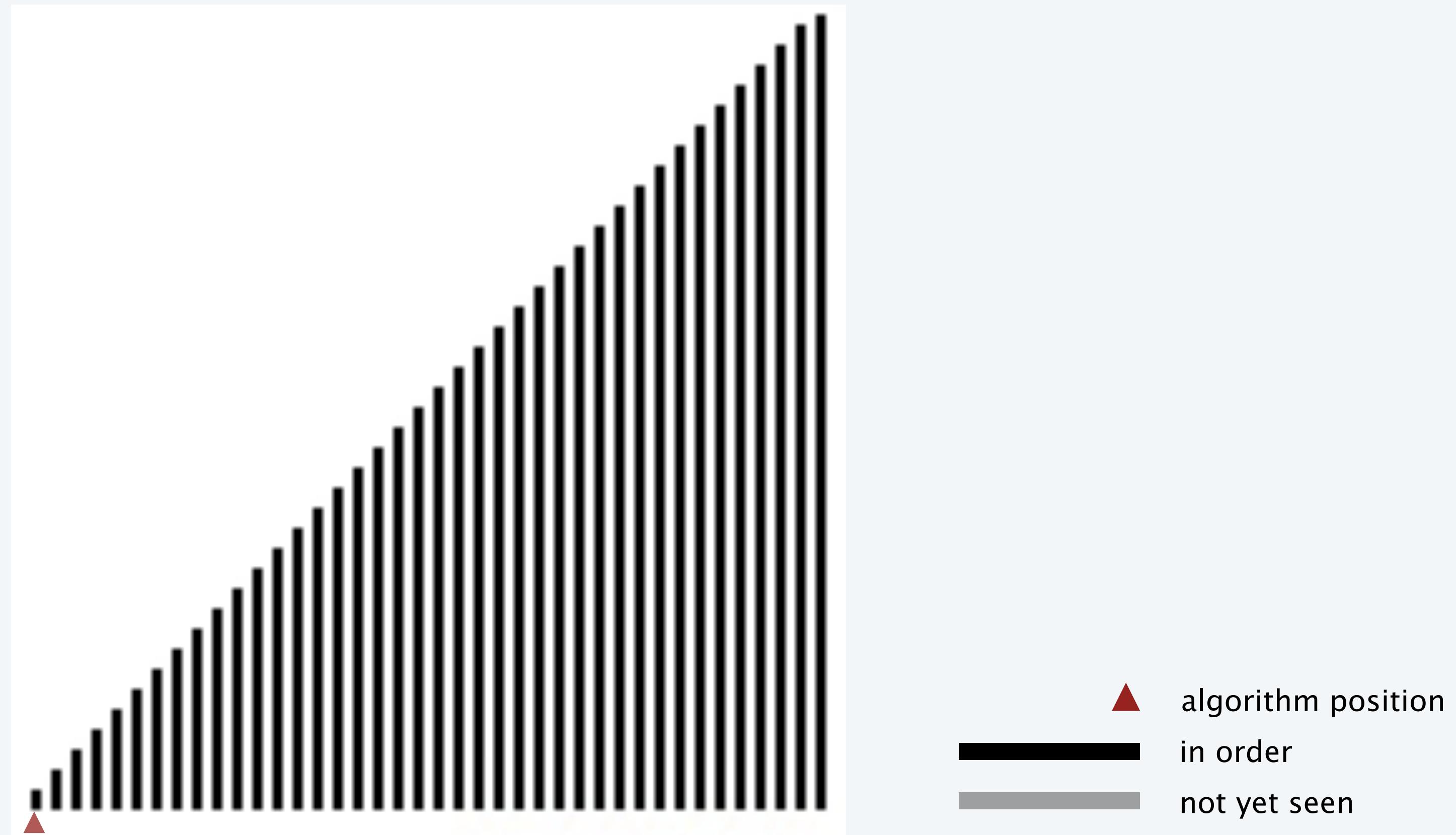
Pf. Exactly i compares and exchanges in iteration i .



Insertion sort: running time analysis

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

Pf. Exactly 1 compares and 0 exchanges in each iteration (except first).

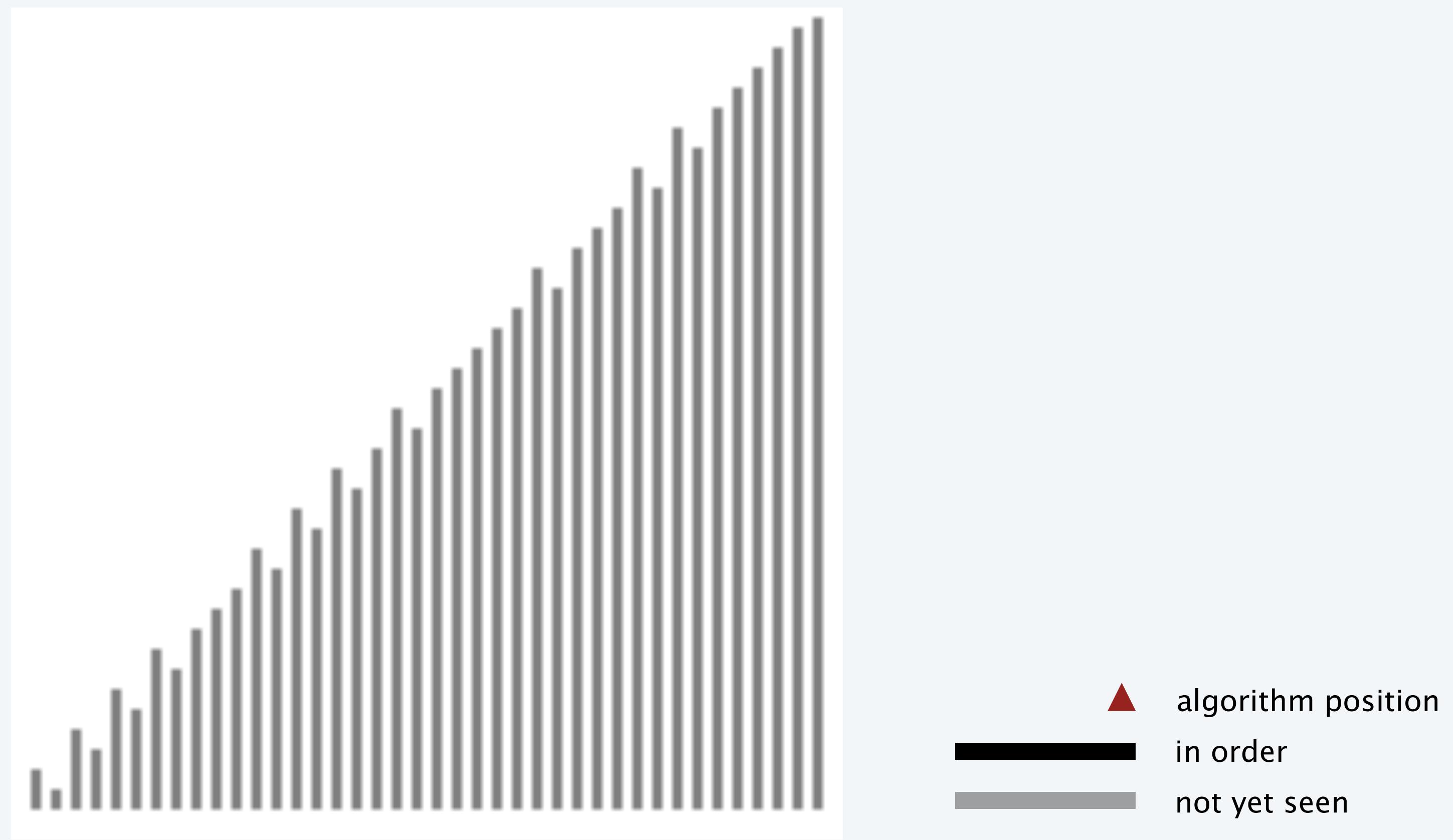


Insertion sort: running time analysis

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

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

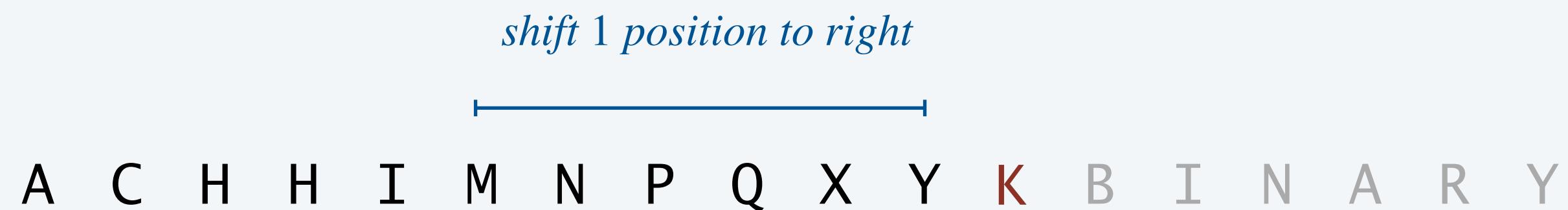
A. Yes, in terms of “inversions” (see textbook).



Insertion sort: practical improvements

Half exchanges. Shift items over (instead of exchanging).

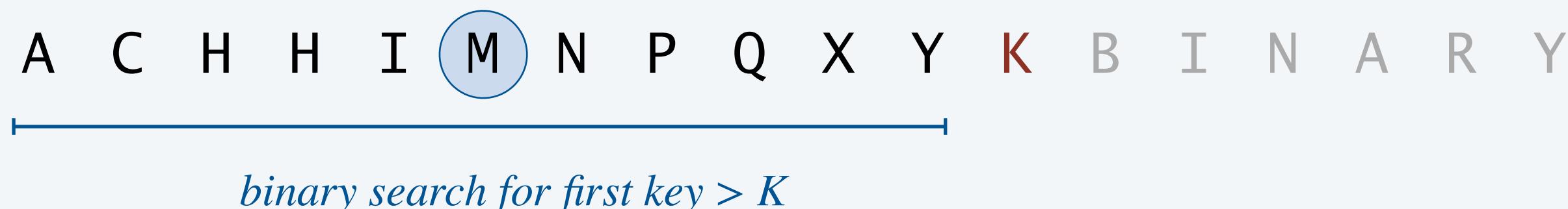
- Same compares; fewer array accesses.
- No longer uses only `less()` and `exch()` to access data.



Binary insertion sort. Use **binary search** to locate insertion point.

- Now, worst-case number of compares is $\sim n \log_2 n$.
- But worst-case number of array access is still $\Theta(n^2)$.

*compares can be very expensive
(relative to data movement),
especially in interpreted languages
(such as Python)*





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?

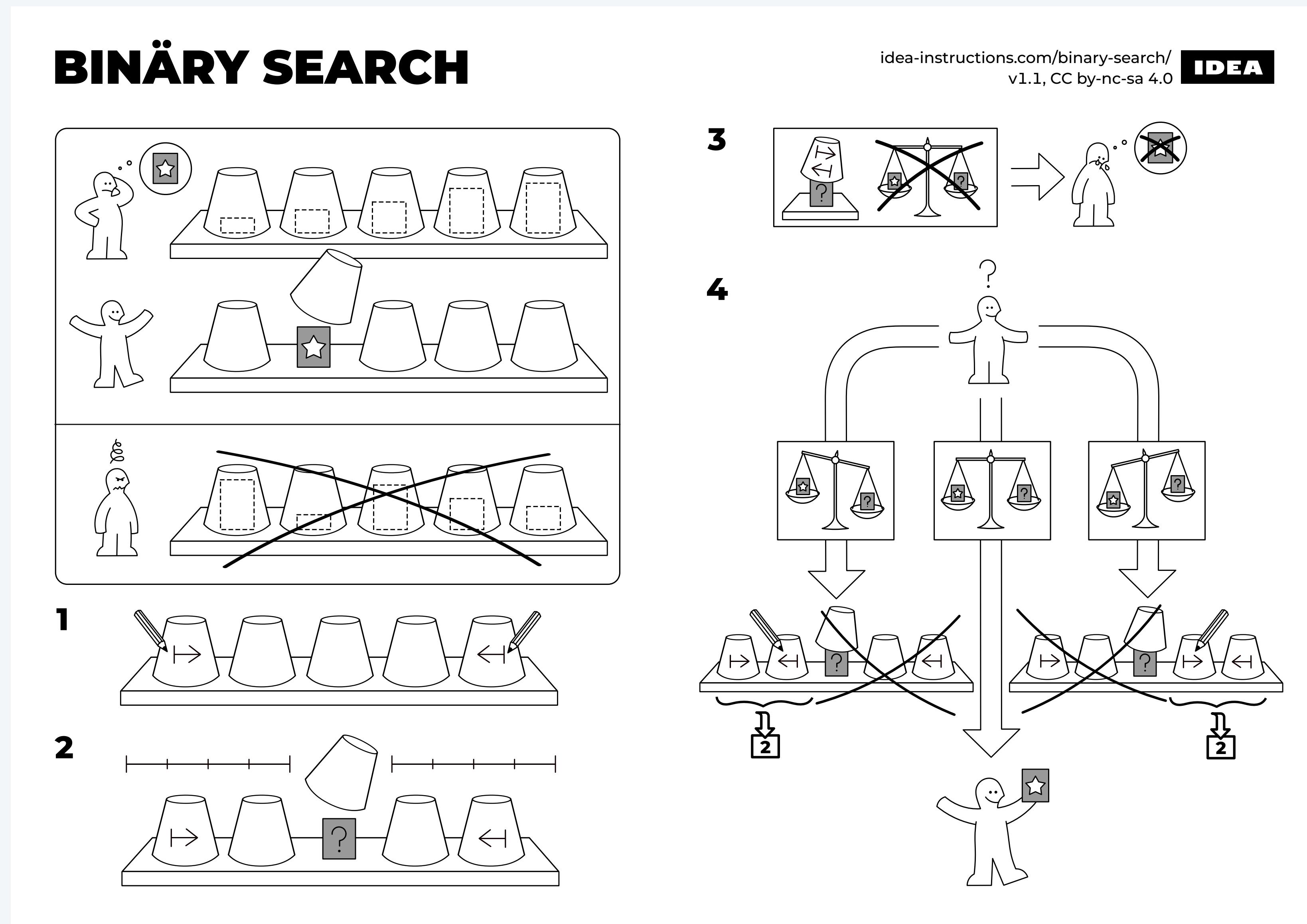
Binary search. Compare search key with middle entry.

- 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: nonverbal algorithm assembly instructions



Binary search: implementation

Trivial to implement?

- First binary search published in 1946.
- First bug-free one in 1962.
- Jon 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/extr-extra-read-all-about-it-nearly.html>

Binary search: implementation

Invariant. If key appears in array a[], then $a[\text{lo}] \leq \text{key} \leq a[\text{hi}]$.

```
public static int binarySearch(String[] a, String key) {  
    int lo = 0, hi = a.length - 1;  
    while (lo <= hi) {  
        int mid = lo + (hi - lo) / 2; why not mid = (lo + hi) / 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;  
}
```

<https://algs4.cs.princeton.edu/11model/BinarySearch.java.html>

Binary search: analysis

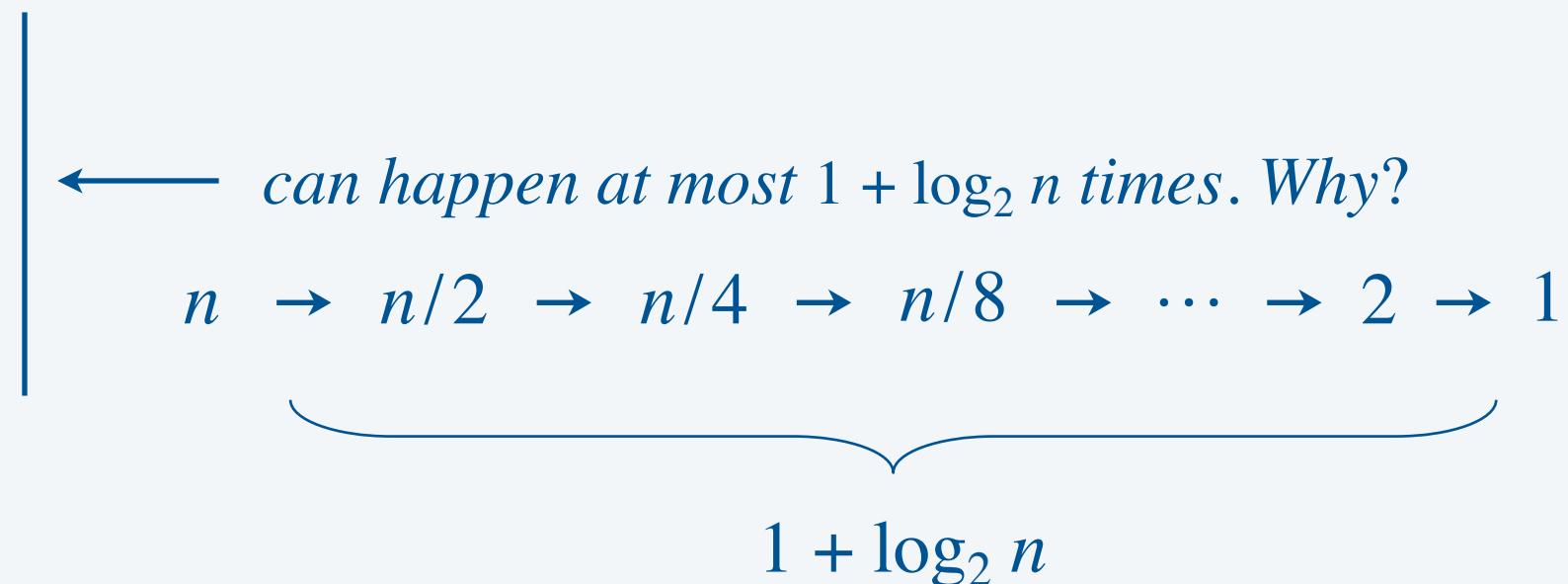
Proposition. Binary search makes at most $1 + \log_2 n$ compares to search in any sorted array of length n .

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 2x,
due to elimination of `a[mid]` from subarray
(or early termination of `while` loop)*



Binary search vs. sequential search



1. less

```
(R. Hendricks 112) int index = 0;
(R. Hendricks 113) while (!element.equals(sortedList.get(index))
(R. Hendricks 114)         && sortedList.size() > ++index);
(R. Hendricks 115) return index < sortedList.size() ? index : -1;
```

SILICON VALLEY



3-SUM. Given an array of n distinct integers, count number of triples that sum to 0.

Version 0. $\Theta(n^3)$ time in worst case. 

Version 1. $\Theta(n^2 \log n)$ time in worst case.

Version 2. $\Theta(n^2)$ time in worst case.

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 $x = -(a[i] + a[j])$.

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

- 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 array $a[]$

30 -40 -20 -10 40 0 10 5

sorted array $a[]$

-40 -20 -10 0 5 10 30 40

binary search

i	j	a[i]	a[j]	x
0	1	-40	-20	60
0	2	-40	-10	50
0	3	-40	0	40
0	4	-40	5	35
0	5	-40	10	30
⋮	⋮	⋮	⋮	⋮
1	2	-20	-10	30
⋮	⋮	⋮	⋮	⋮
2	3	-10	0	10
⋮	⋮	⋮	⋮	⋮
5	6	10	30	>40
5	7	10	40	-50
6	7	30	40	-70

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



3-SUM. Given an array of n distinct integers, count number of triples that sum to 0.

Version 0. $\Theta(n^3)$ time in worst case. ✓

Version 1. $\Theta(n^2 \log n)$ time in worst case. ✓

Version 2. $\Theta(n^2)$ time in worst case. [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

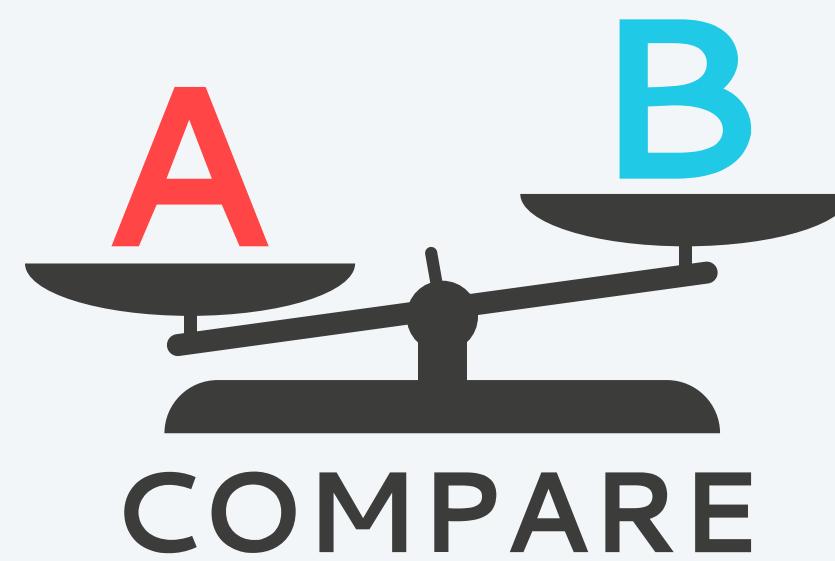
Comparable interface. Java framework for comparing items.

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

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

Binary insertion sort. $\Theta(n \log n)$ compares; $\Theta(n^2)$ exchanges in the worst case.

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



Credits

media	source	license
<i>Sorting Hat</i>	Hannah Hill	CC BY-NC 4.0
<i>Airport Departures</i>	Adobe Stock	education license
<i>iPhone Contacts</i>	StackOverflow	
<i>Playing Cards</i>	Google Code	public domain
<i>Rock, Paper, Scissors</i>	Daily Mail	
<i>Anime Boy</i>	freesvg.org	public domain
<i>Anime Girl</i>	freesvg.org	public domain
<i>Call Back Icon</i>		
<i>Balance</i>	Adobe Stock	education license
<i>Selection Sort Visualization</i>	Toptal	
<i>Insertion Sort Visualization</i>	Toptal	

Credits

media	source	license
<i>Jon Bentley</i>	Amazon	
<i>Binary Search IDEA</i>	idea-instructions.com	CC BY-NC-SA 4.0
<i>Binary vs. Sequential Search</i>	Silicon Valley S6E4	
<i>Compare A and B</i>	Adobe Stock	education license
<i>Selection Sort Graphic</i>	Adobe Stock	education license
<i>Insertion Sort Graphic</i>	Adobe Stock	education license
<i>Insertion Sort Dance</i>	AlgoRhythmics	

Insertion sort with Romanian folk dance



<https://www.youtube.com/watch?v=ROalU379I3U>