

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

## 4.2 ALGORITHMS

---

- ▶ *sequential search*
- ▶ *binary search*
- ▶ *insertion sort*
- ▶ *mergesort*







Intuition. An **algorithm** is like a recipe. ← *but unambiguous and mechanically executable*

### Pita bread ~~algorithm~~ recipe

**1** 20ml olive oil + 200ml water

**2** 7g yeast + 500g flour

**3** 1/2 teaspoon salt

**4** Cover the dough for 20 minutes.

**5** Cut the dough into 10 pieces. Roll the dough into circles.

**6** Bake in oven on high for 7 minutes.

[rhurbans.com](http://rhurbans.com)

# Algorithms

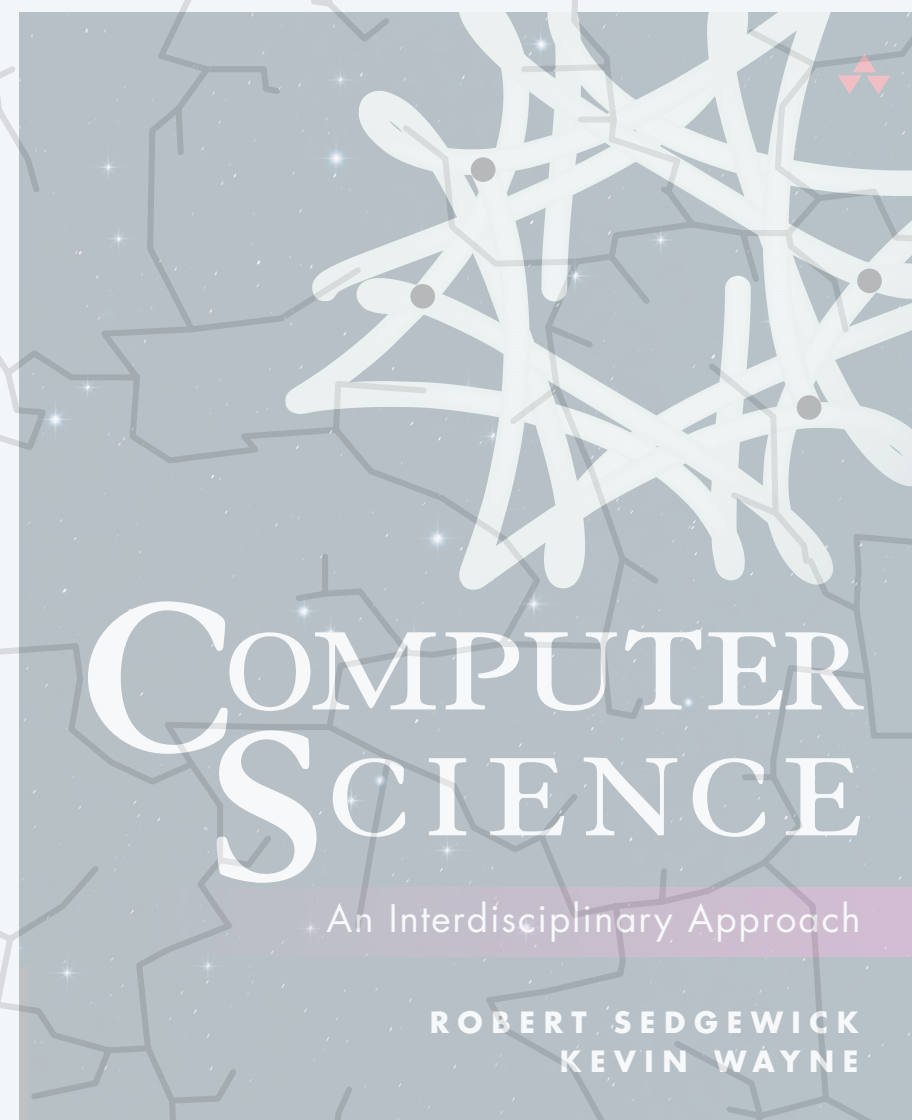


**Algorithm.** Step-by-step procedure for solving a problem.

- Takes input; produces output.
- Unambiguous and mechanically executable (e.g., in Java).

← formalized by  
Turing machines  
(stay tuned)

category	famous algorithms
historic	<i>Euclid's gcd algorithm, gradient descent, Newton's method</i>
sorting and searching	<i>binary search, insertion sort, mergesort</i> ← today's lecture
graphs	<i>DFS, BFS, Dijkstra, Kruskal, Ford–Fulkerson</i>
linear algebra	<i>Gaussian elimination, simplex method, QR method, PageRank</i>
scientific computing	<i>Smith–Waterman, Metropolis–Hastings, k-means, FFT</i>
machine learning	<i>A*-search, multiplicative weights, neural network, transformer</i>



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

## 4.2 ALGORITHMS

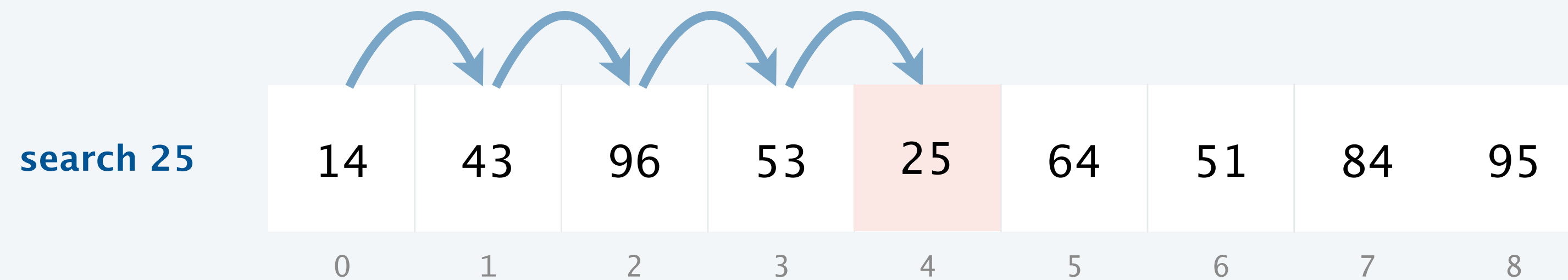
---

- ▶ *sequential search*
- ▶ *binary search*
- ▶ *insertion sort*
- ▶ *mergesort*

# Sequential search

**Problem.** Given an **array** of  $n$  elements and a **search key**, find index of search key in array.

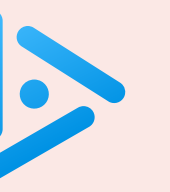
**Sequential search.** Check each element in array until match is found.



```
public static int sequentialSearch(String[] a, String key) {  
    int n = a.length;  
    for (int i = 0; i < n; i++) {  
        if (a[i].equals(key)) return i;  
    }  
    return -1;   
}
```

*return -1 to indicate key is not in array*

*we'll write code with String keys*



In the worst case, how many equality tests (or array accesses) does **sequential search** make to search for a key in an array of length  $n$ ?

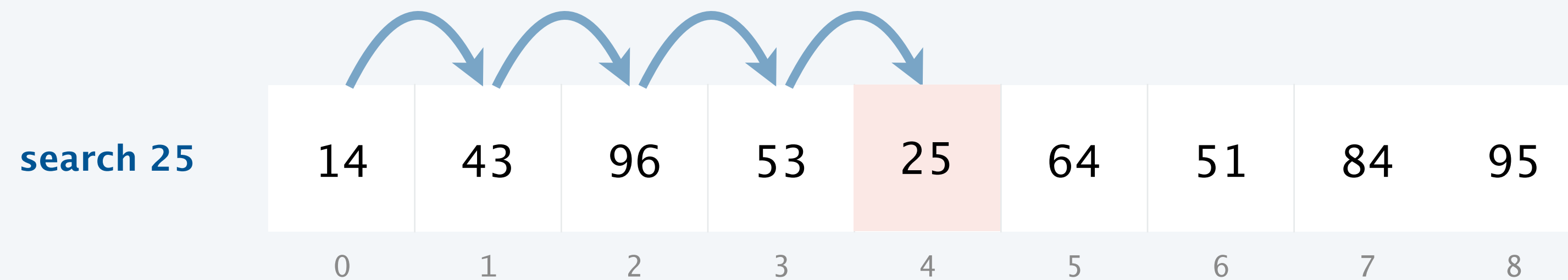
- A.  $\Theta(1)$
- B.  $\Theta(\log n)$
- C.  $\Theta(n)$
- D.  $\Theta(n^2)$

# Sequential search

---

**Problem.** Given an array of  $n$  elements and a search key, find index of search key in array.

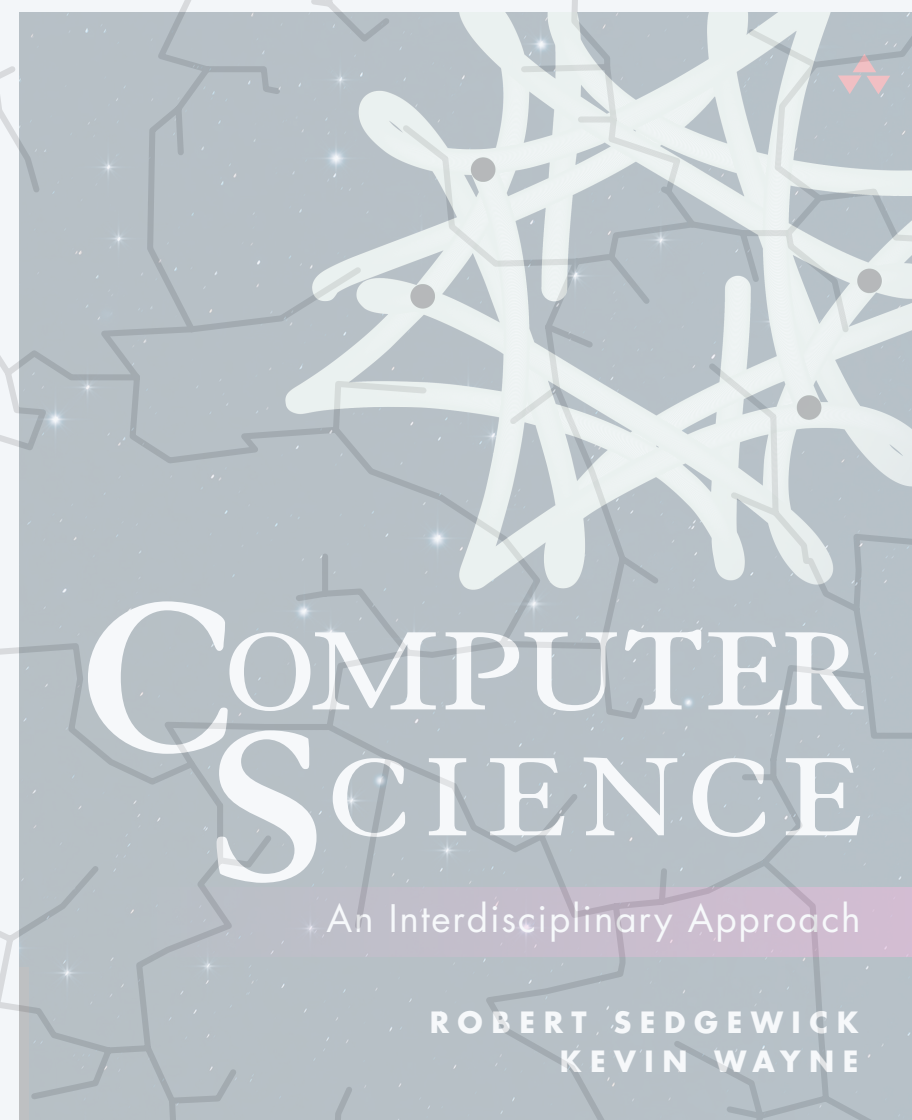
**Sequential search (linear search).** Check each element in array until match is found.



**Cost model.** Equality tests (or array accesses).

**Performance.** Sequential search solves the problem using  $\leq n$  equality tests.





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

## 4.2 ALGORITHMS

---

- ▶ *sequential search*
- ▶ *binary search*
- ▶ *insertion sort*
- ▶ *mergesort*

# Binary search



**Problem.** Given a **sorted array** of  $n$  elements and a **search key**, find index of search key in array.

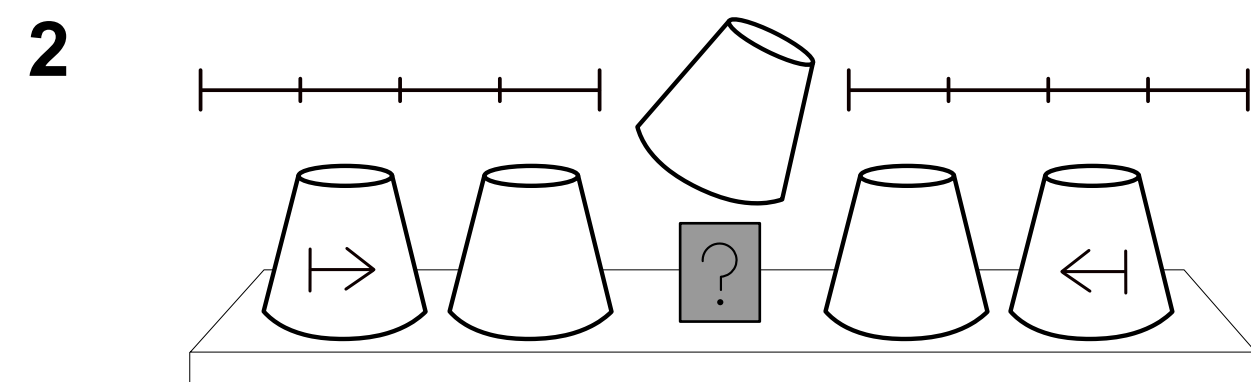
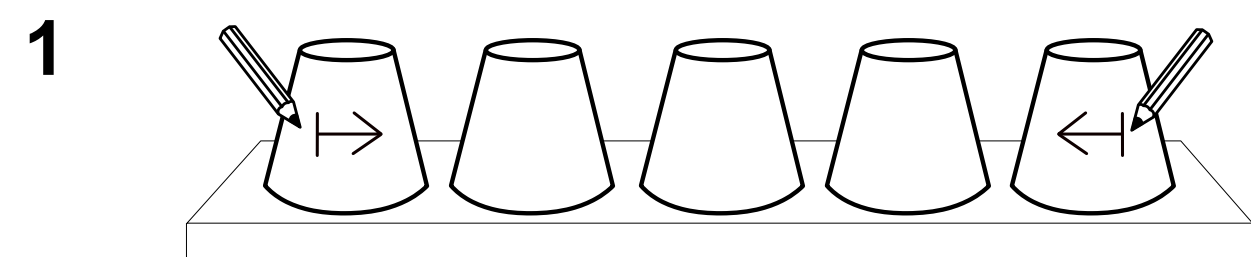
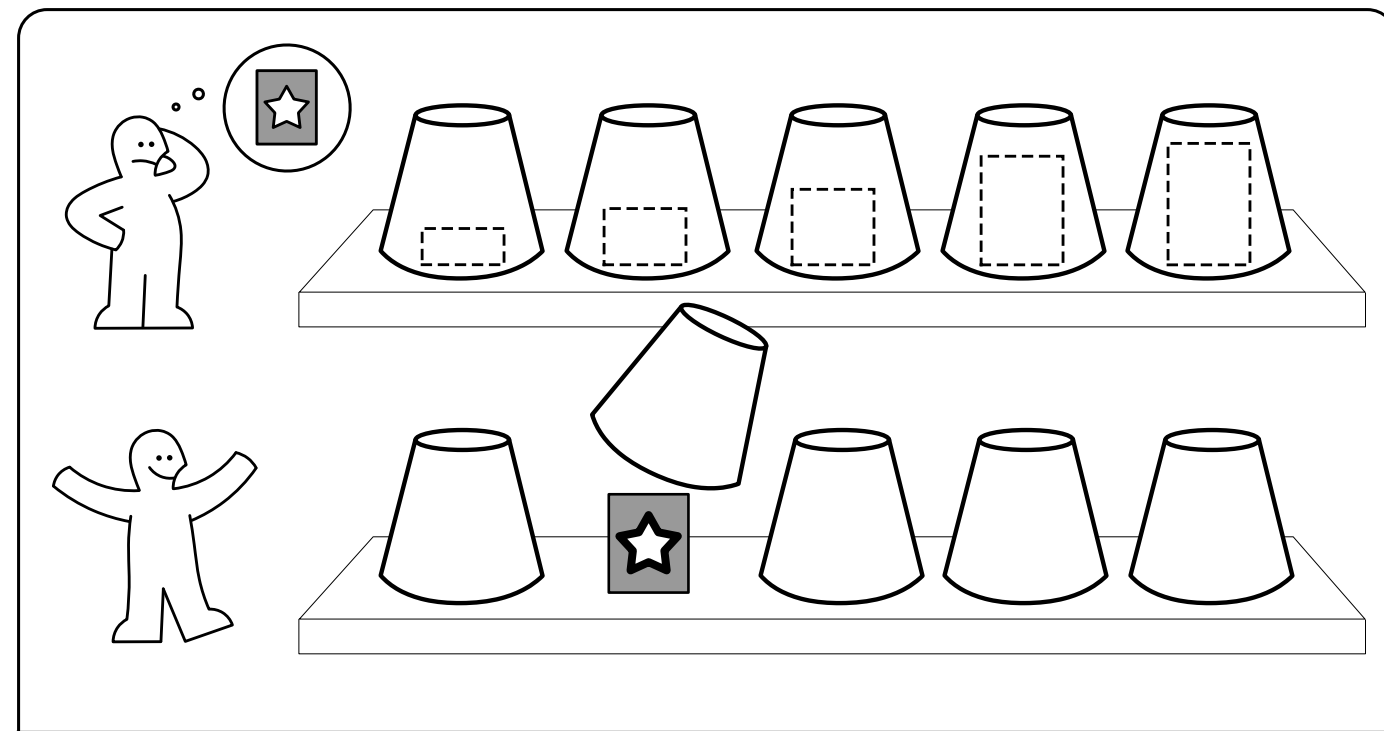
**Binary search.** Compare search key with middle element.

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

**sorted array**

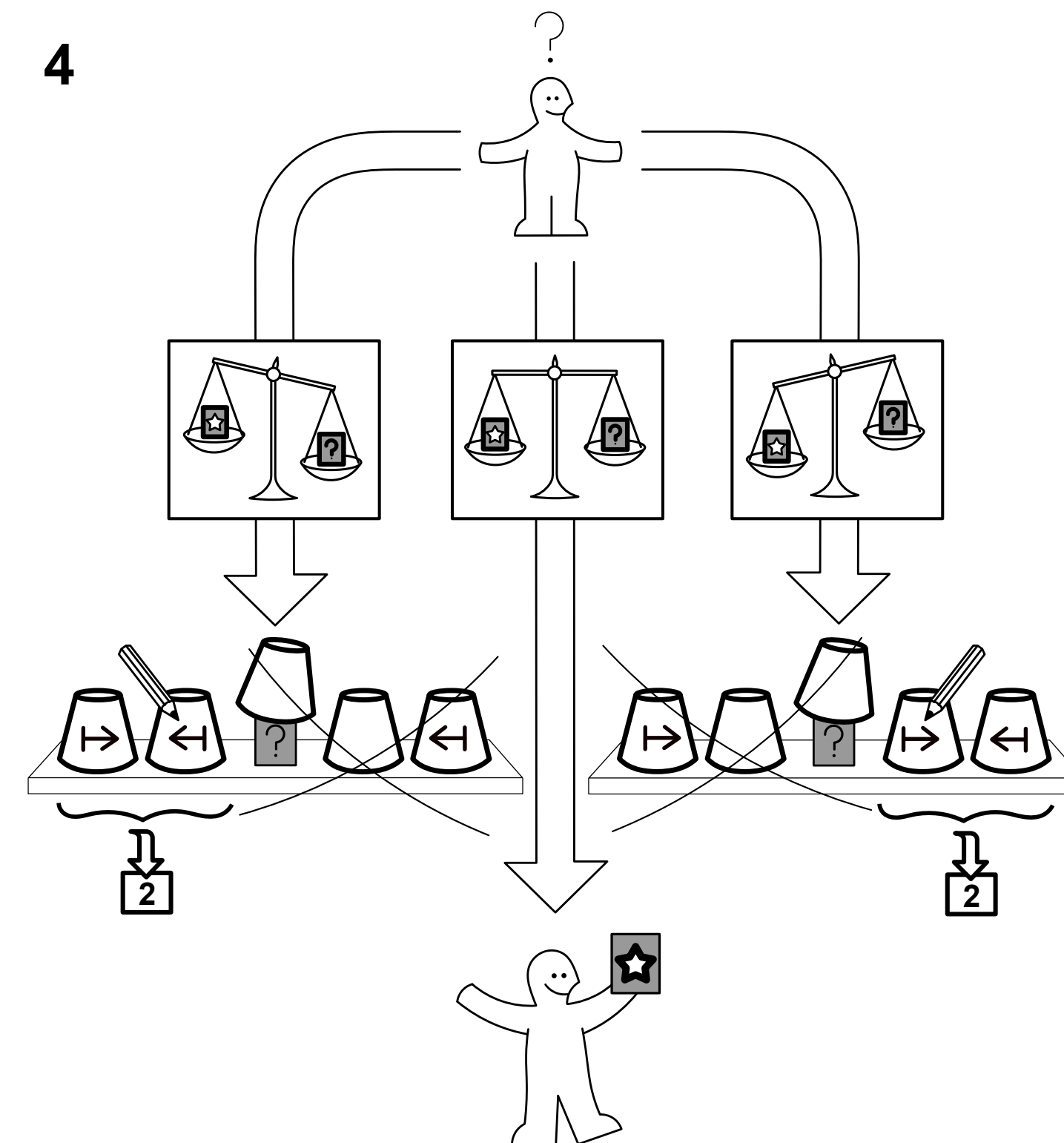
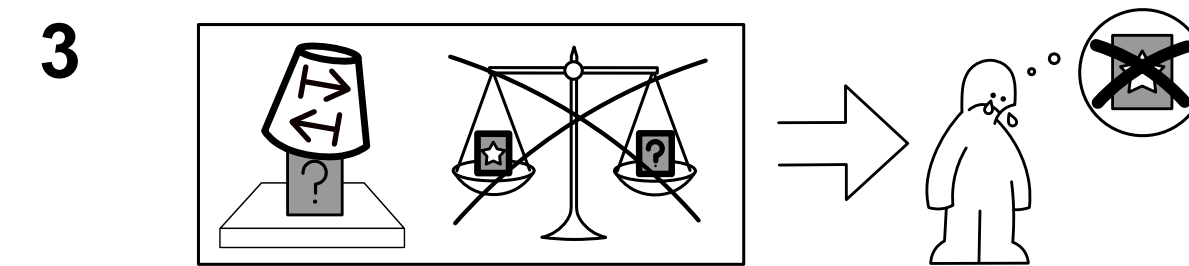
10	13	14	25	33	43	51	53	64	72	84	93	95	96	97
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
↑														↑
<i>lo</i>														<i>hi</i>

## BINÄRY SEARCH



[idea-instructions.com/binary-search/](http://idea-instructions.com/binary-search/)  
v1.1, CC by-nc-sa 4.0

**IDEA**



# Binary search: Java implementation

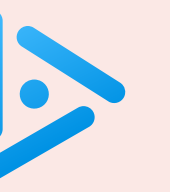
---

**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 - 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;
}
```

*essentially equivalent to*  
`mid = (lo + hi) / 2`  
*(but avoids arithmetic overflow)*

*return zero (if equal),*  
*negative integer (if less),*  
*positive integer (if greater)*



In the worst case, how many compares (or array accesses) does **binary search** make to search for a key in a sorted array of length  $n$ ?

- A.  $\Theta(1)$
- B.  $\Theta(\log n)$
- C.  $\Theta(n)$
- D.  $\Theta(n^2)$

# Binary search: analysis

---

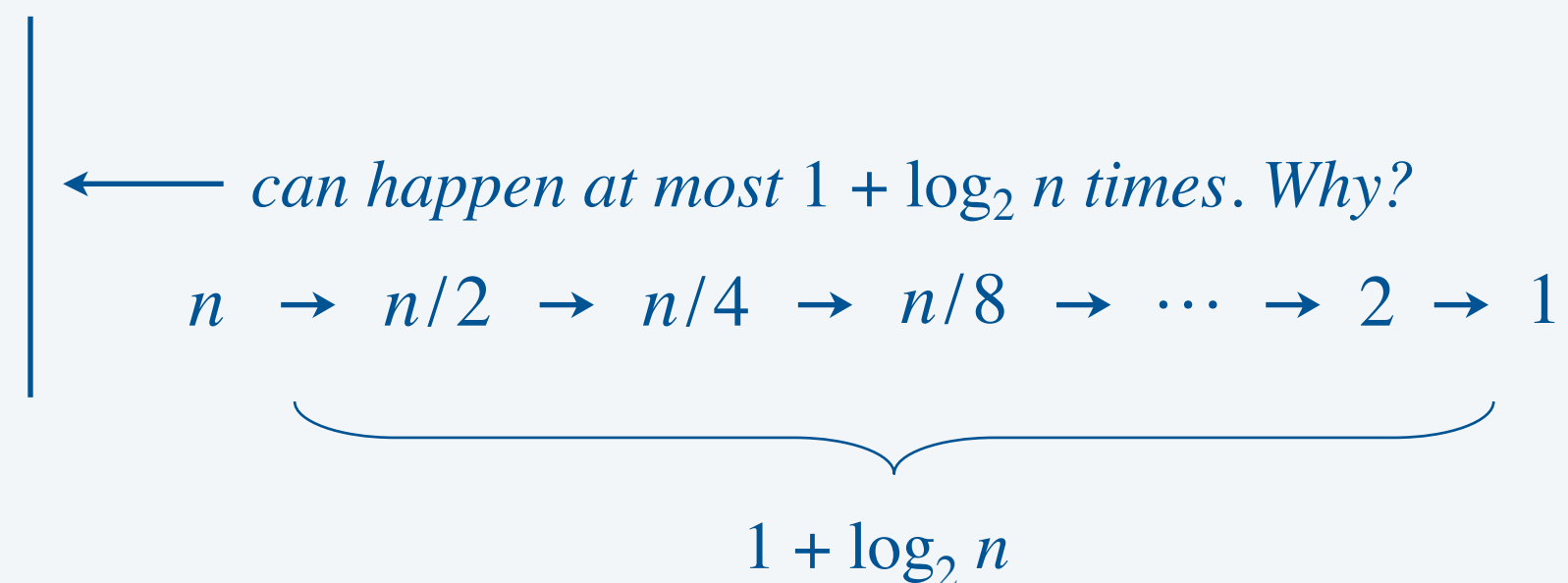
**Problem.** Given a **sorted array** of  $n$  elements and a **search key**, find index of search key in array.

**Proposition.** Binary search solves problem using  $\leq 1 + \log_2 n$  compares.

**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)*



# Sequential search vs. binary search: empirical analysis

Running time and energy estimates (approximate):

- CPU core executes  $10^8$  compares/second.
- CPU core consumes 18 watts power.

Resources required. [on array of length  $n = 10^9$ ]

queries/hour	CPU cores	power
thousand	2.78	50 watts
million	2.78 thousand	50 kilowatts
billion	2.78 million	50 megawatts

sequential search  $\Theta(n)$

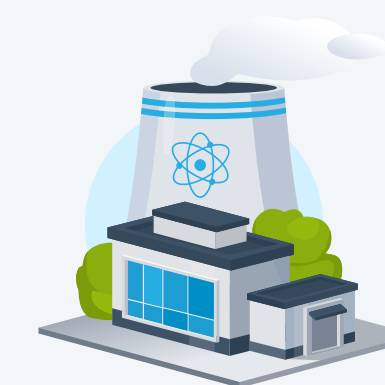
queries/hour	CPU cores	power
million	–	–
billion	0.083	1.5 watts
trillion	83	1.5 kilowatts

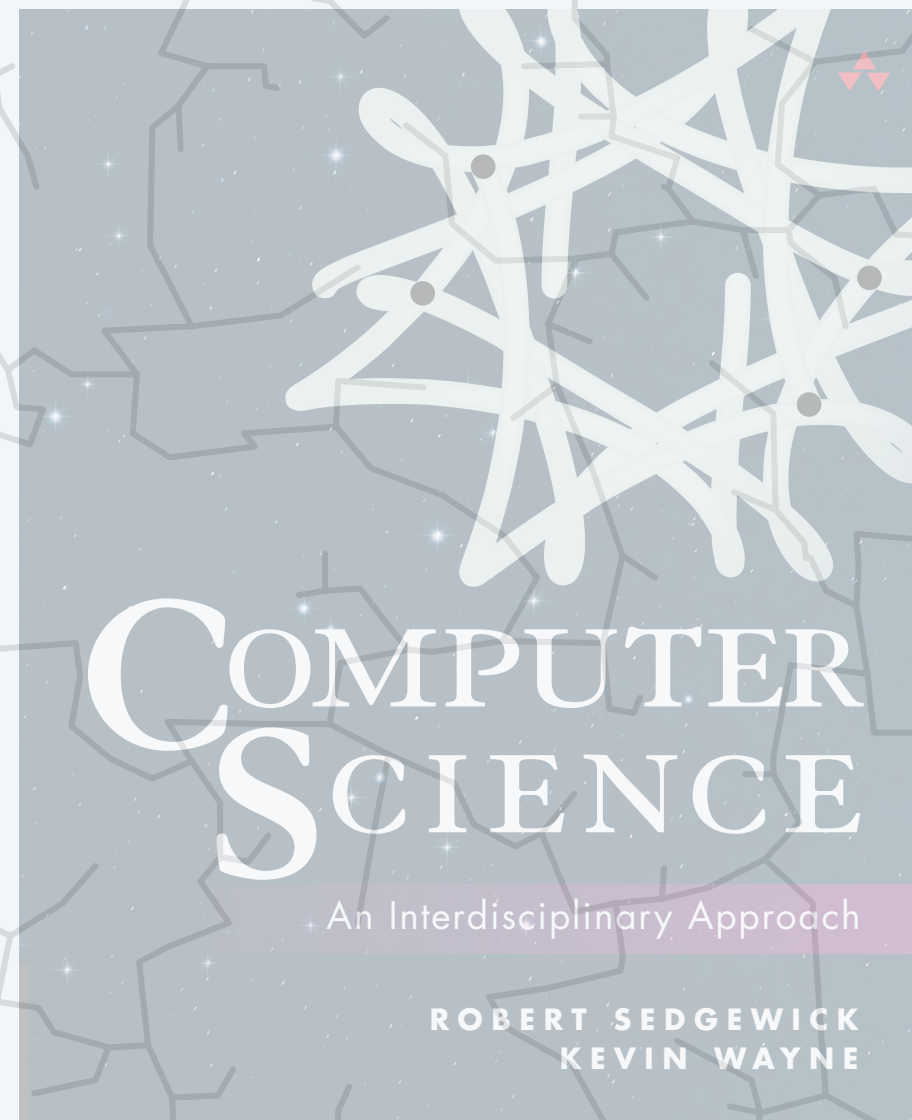
binary search  $\Theta(\log n)$

← *about 33M times more efficient*  
( $10^9$  vs.  $\log_2 10^9$ )

Bottom line. Great algorithms can replace server farms. ←

500K ChatGPT queries per hour  
( $\approx 1$  gigawatt)





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

## 4.2 ALGORITHMS

---

- ▶ *sequential search*
- ▶ *binary search*
- ▶ *insertion sort*
- ▶ *mergesort*



# Sorting problem

---

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

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

*element* →

*key* →



**sorting hat**

# Sorting problem

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

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

*key* →

*element* →

↑  
*sorted by key*



**sorting hat**

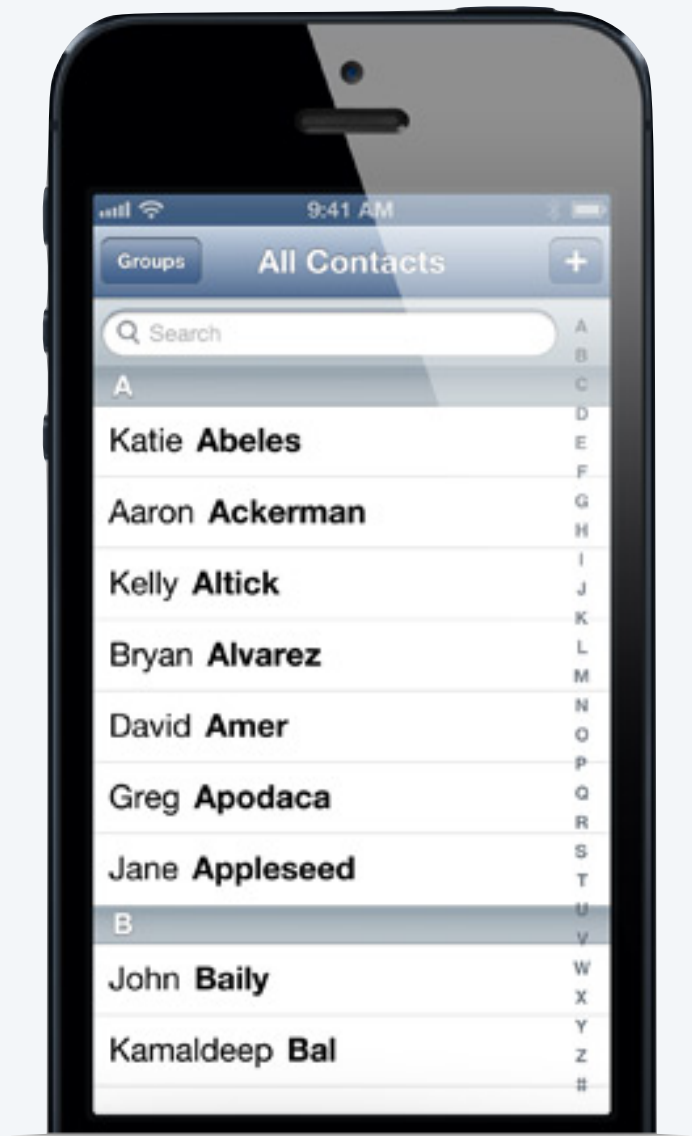
# Sorting applications

International Departures				
Flight No	Destination	Time	Gate	Remarks
CX7183	Berlin	7:50	A-11	Gate closing
QF3474	London	7:50	A-12	Gate closing
BA372	Paris	7:55	B-10	Boarding
AY6554	New York	8:00	C-33	Boarding
KL3160	San Francisco	8:00	F-15	Boarding
BA8903	Manchester	8:05	B-12	Gate lounge open
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

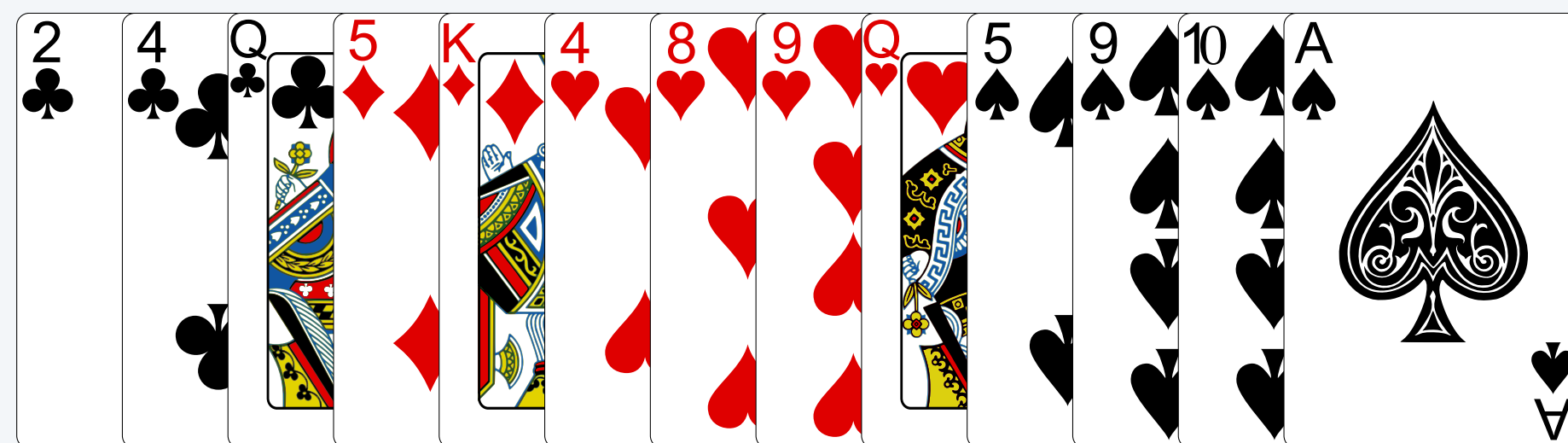
chronological order

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

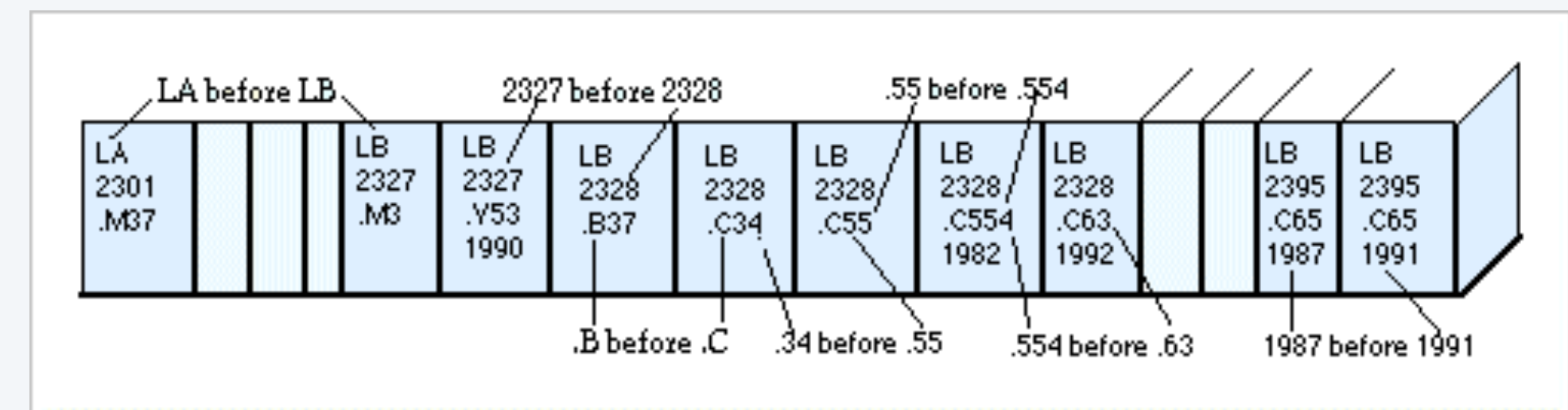
numerical order (descending)



alphabetical order



suit and rank order



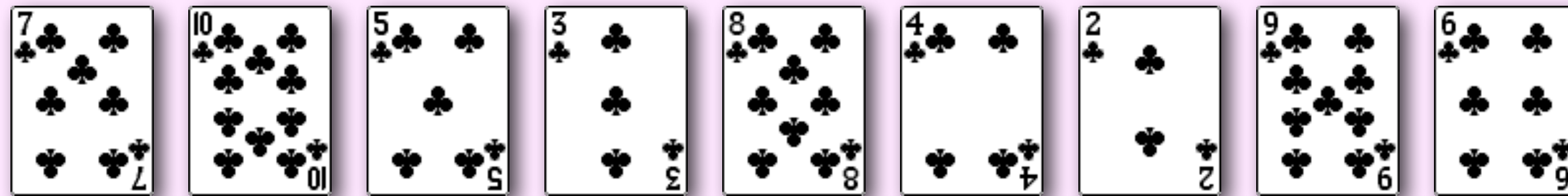
Library of Congress order

# 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 demo (Romanian folk dance)



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 left.



# Insertion sort

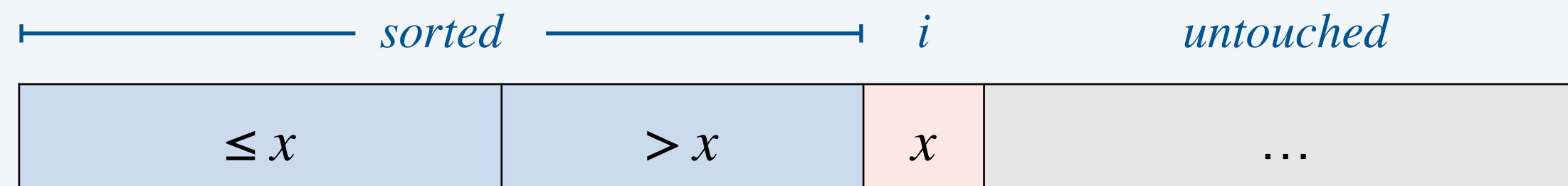
---

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

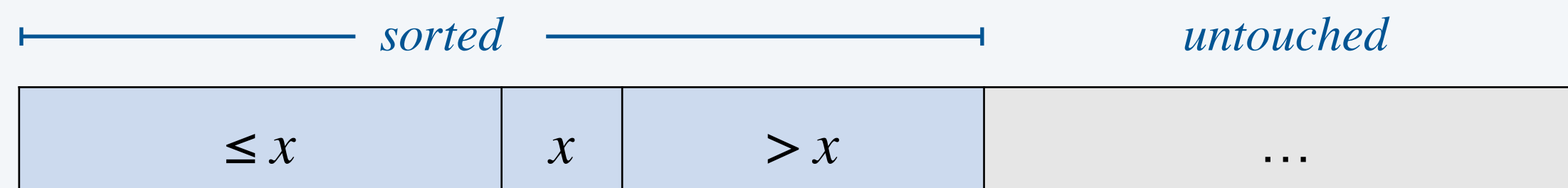
- Let  $x$  be the element at index  $i$ .
- Repeatedly exchange  $x$  with each larger element to its left.

**Invariants.**

before iteration  $i$



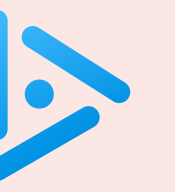
after iteration  $i$



# Insertion sort: Java implementation

---

```
public class Insertion {  
  
    public static void sort(String[] 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(String v, String w) {  
        return v.compareTo(w) < 0;  
    }  
  
    private static void exch(String[] a, int i, int j) {  
        String temp = a[i];  
        a[i] = a[j];  
        a[j] = temp;  
    }  
  
}
```



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

- A.  $\Theta(1)$
- B.  $\Theta(\log n)$
- C.  $\Theta(n)$
- D.  $\Theta(n^2)$



## Insertion sort: analysis

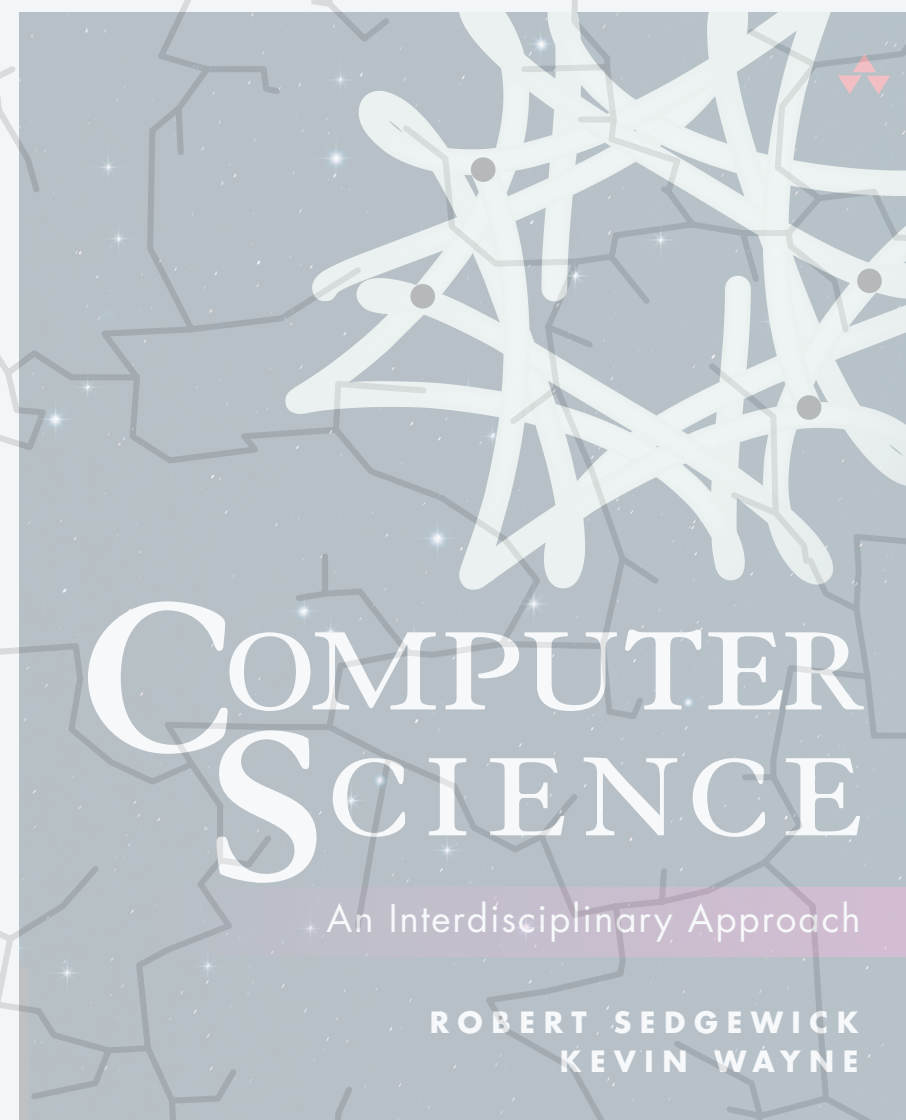
---

**Sorting cost model.** Number of compares (or array accesses).

**Proposition.** Insertion sort never makes more than  $\sim \frac{1}{2} n^2$  compares to sort an array of length  $n$ .

**Pf.**

- The worst case is a reverse-sorted array of  $n$  distinct keys.
- In iteration  $i$ , insertion sort makes  $i$  compares.
- Total number of compares =  $0 + 1 + 2 + \dots + (n-1) = \frac{1}{2} n(n-1)$ .



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

## 4.2 ALGORITHMS

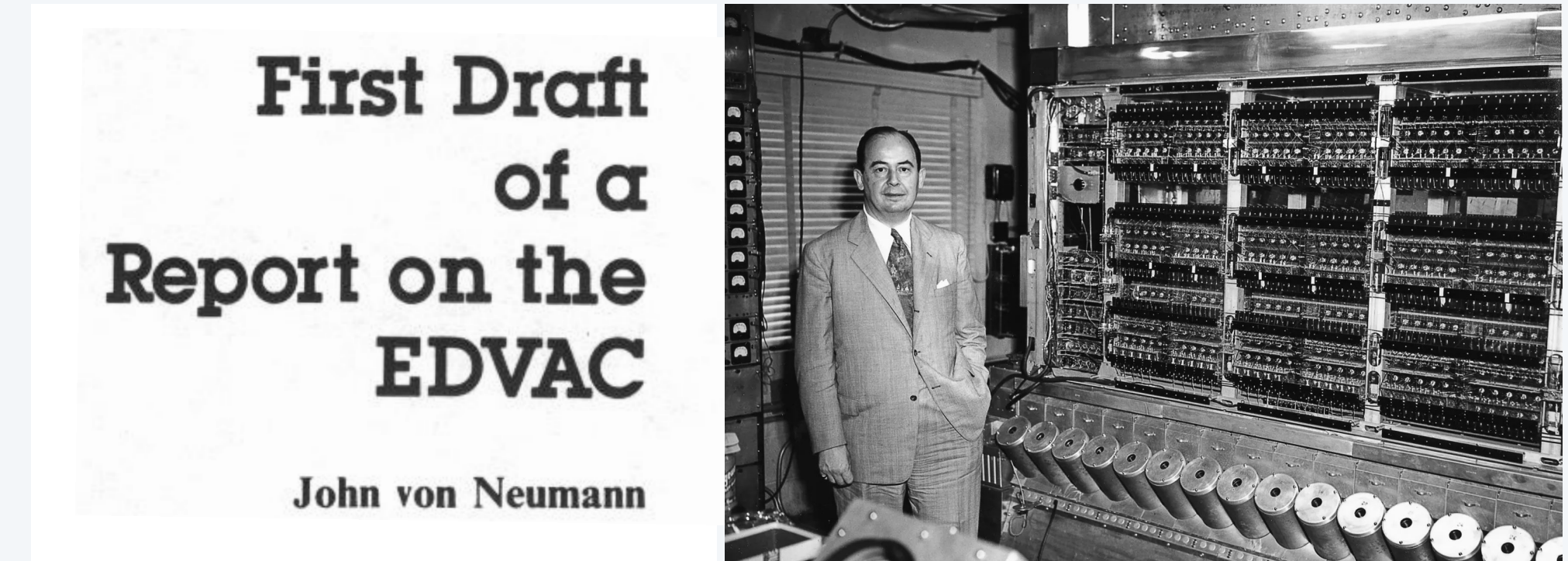
---

- ▶ *sequential search*
- ▶ *binary search*
- ▶ *insertion sort*
- ▶ ***mergesort***

# Mergesort overview

## Basic plan.

- Divide array into two halves.
- Recursively sort left half.
- Recursively sort right half.
- **Merge** two sorted halves.

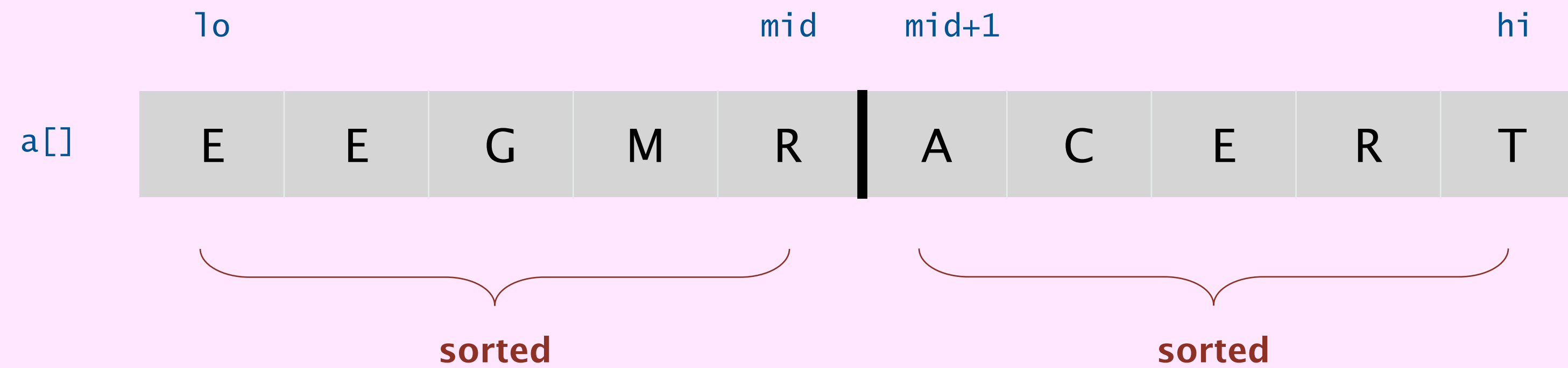


<b>input</b>	M	E	R	G	E	S	O	R		T	E	X	A	M	P	L	E
<b>sort left half</b>	E	E	G	M	O	R	R	S		T	E	X	A	M	P	L	E
<b>sort right half</b>	E	E	G	M	O	R	R	S		A	E	E	L	M	P	T	X
<b>merge results</b>	A	E	E	E	E	G	L	M	M	O	P	R	R	S	T	X	

# Abstract in-place merge demo



**Goal.** Given two sorted subarrays  $a[lo]$  to  $a[mid]$  and  $a[mid+1]$  to  $a[hi]$ , replace with sorted subarray  $a[lo]$  to  $a[hi]$ .



# Merging: Java implementation

```
private static void merge(String[] a, String[] aux, int lo, int mid, int hi) {
```

```
    for (int k = lo; k <= hi; k++) copy  
        aux[k] = a[k];
```

```
    int i = lo, j = mid+1; merge  
    for (int k = lo; k <= hi; k++) {  
        if (i > mid) a[k] = aux[j++];  
        else if (j > hi) a[k] = aux[i++];  
        else if (less(aux[j], aux[i])) a[k] = aux[j++];  
        else a[k] = aux[i++];  
    }
```

```
}
```



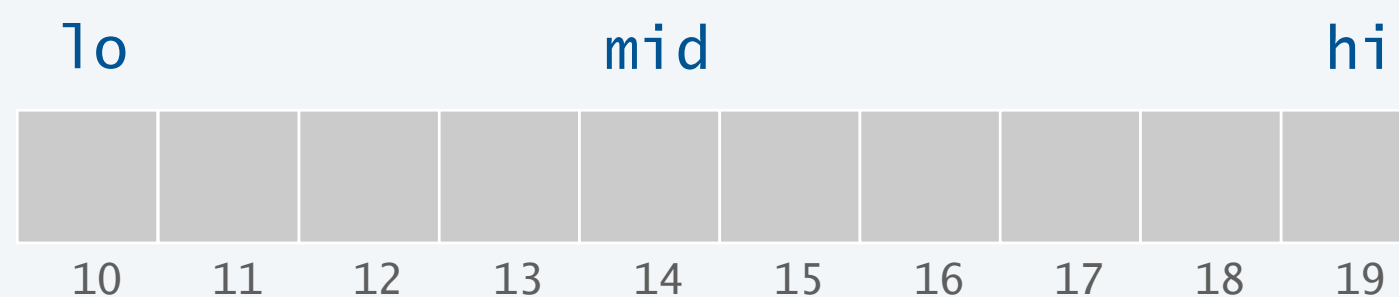
# Mergesort: Java implementation

---

```
public class Merge {  
    private static void merge(...) {  
        /* as before */  
    }  
}
```

```
private static void sort(String[] a, String[] aux, int lo, int hi) {  
    if (hi <= lo) return;  
    int mid = lo + (hi - lo) / 2;  
    sort(a, aux, lo, mid);  
    sort(a, aux, mid+1, hi);  
    merge(a, aux, lo, mid, hi);  
}
```

```
public static void sort(String[] a) {  
    String[] aux = new String[a.length];  
    sort(a, aux, 0, a.length - 1);  
}
```



# Mergesort: trace

	a[]															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	M	E	R	G	E	S	O	R	T	E	X	A	M	P	L	E
merge(a, aux, <sup>lo</sup> 0, 0, <sup>hi</sup> 1)	E	M	R	G	E	S	O	R	T	E	X	A	M	P	L	E
merge(a, aux, 2, 2, 3)	E	M	G	R	E	S	O	R	T	E	X	A	M	P	L	E
merge(a, aux, 0, 1, 3)	E	G	M	R	E	S	O	R	T	E	X	A	M	P	L	E
merge(a, aux, 4, 4, 5)	E	G	M	R	E	S	O	R	T	E	X	A	M	P	L	E
merge(a, aux, 6, 6, 7)	E	G	M	R	E	S	O	R	T	E	X	A	M	P	L	E
merge(a, aux, 4, 5, 7)	E	G	M	R	E	O	R	S	T	E	X	A	M	P	L	E
merge(a, aux, 0, 3, 7)	E	E	G	M	O	R	R	S	T	E	X	A	M	P	L	E
merge(a, aux, 8, 8, 9)	E	E	G	M	O	R	R	S	E	T	X	A	M	P	L	E
merge(a, aux, 10, 10, 11)	E	E	G	M	O	R	R	S	E	T	A	X	M	P	L	E
merge(a, aux, 8, 9, 11)	E	E	G	M	O	R	R	S	A	E	T	X	M	P	L	E
merge(a, aux, 12, 12, 13)	E	E	G	M	O	R	R	S	A	E	T	X	M	P	L	E
merge(a, aux, 14, 14, 15)	E	E	G	M	O	R	R	S	A	E	T	X	M	P	E	L
merge(a, aux, 12, 13, 15)	E	E	G	M	O	R	R	S	A	E	T	X	E	L	M	P
merge(a, aux, 8, 11, 15)	E	E	G	M	O	R	R	S	A	E	E	L	M	P	T	X
merge(a, aux, 0, 7, 15)	A	E	E	E	E	G	L	M	M	O	P	R	R	S	T	X

← result after recursive call

# Insertion sort vs. mergesort: empirical analysis

---

## Running time estimates (approximate):

- Laptop executes  $10^8$  compares/second.
- Supercomputer executes  $10^{12}$  compares/second.

n	laptop	super
thousand	<i>instant</i>	<i>instant</i>
million	<i>2.8 hours</i>	<i>1 second</i>
billion	<i>317 years</i>	<i>1 week</i>

**insertion sort**

n	laptop	super
thousand	<i>instant</i>	<i>instant</i>
million	<i>1 second</i>	<i>instant</i>
billion	<i>18 minutes</i>	<i>instant</i>

**mergesort**

**Bottom line.** Great algorithms are better than supercomputers.





In the worst case, how many compares to merges two sorted subarrays of length  $n/2$  to produce a sorted array of length  $n$  ?

merging two sorted arrays, each of length  $n/2$

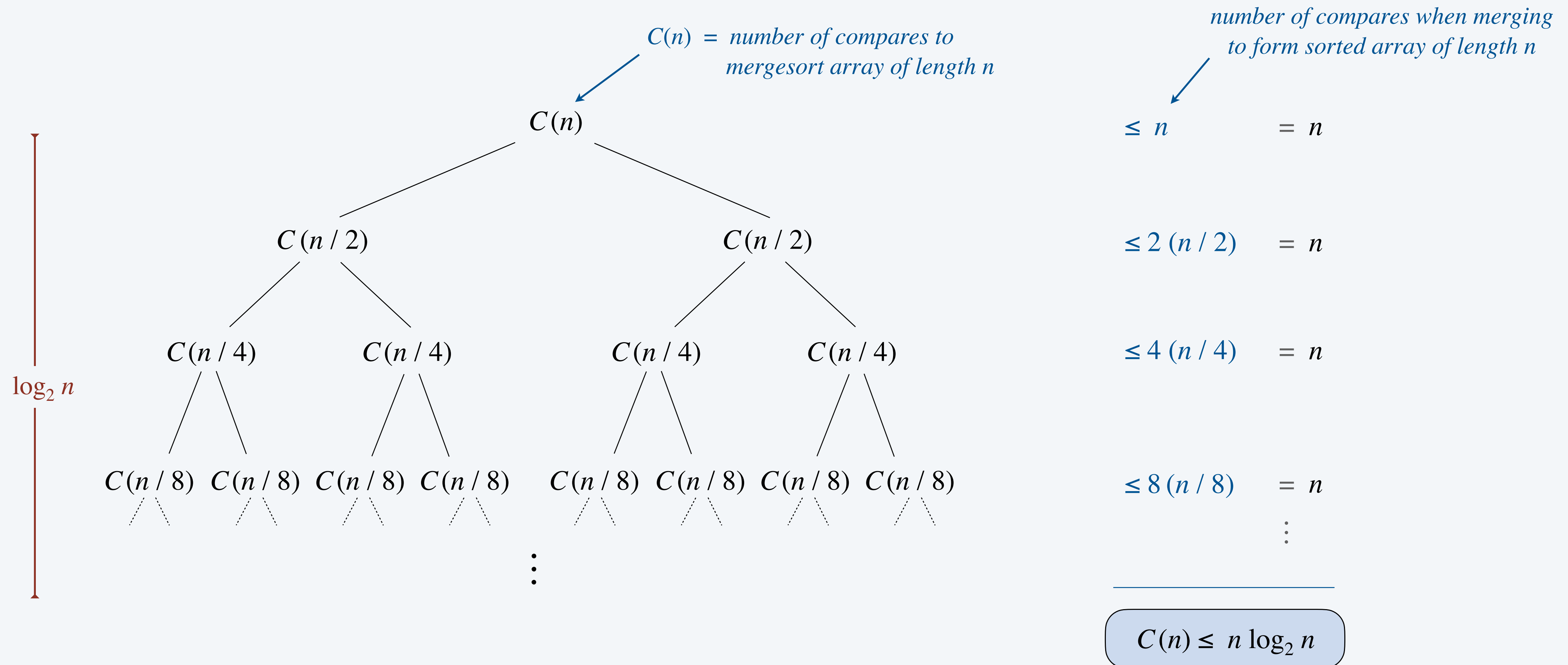


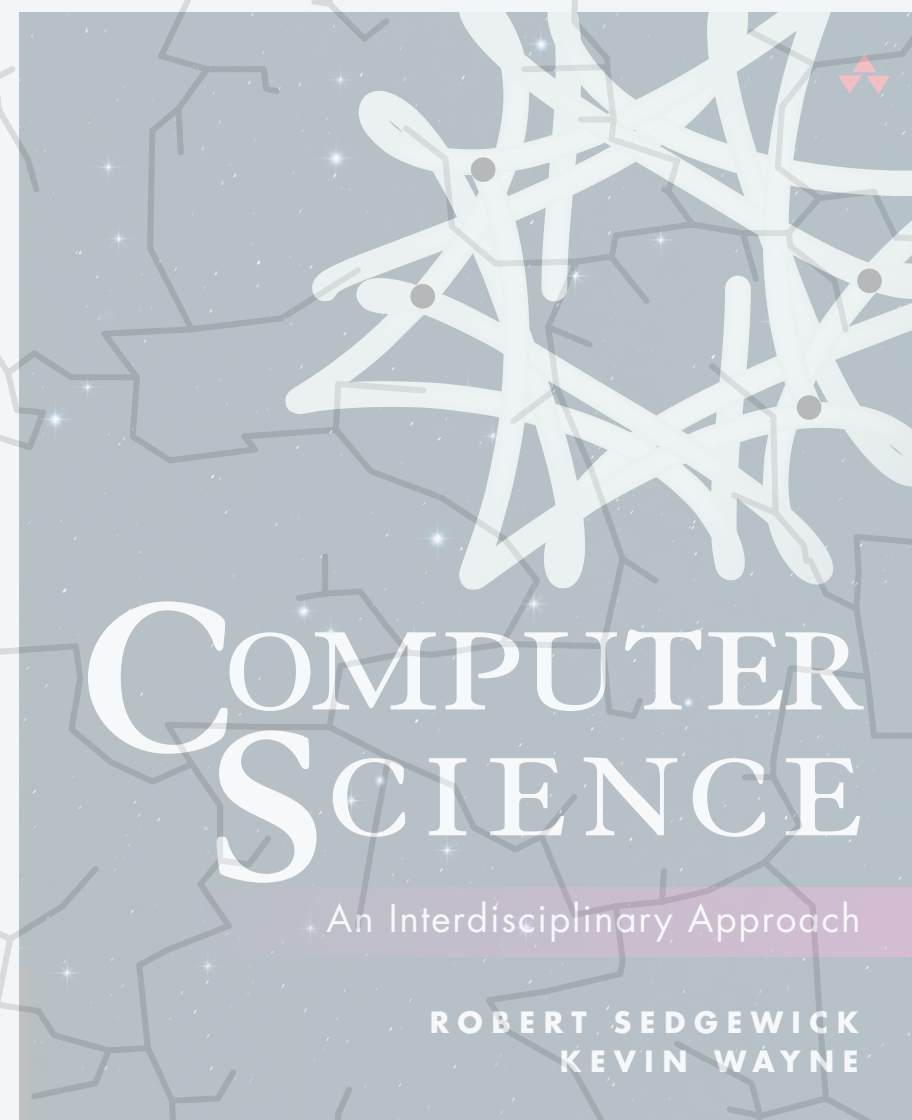
- A.  $n/2$
- B.  $n - 1$
- C.  $\Theta(n \log n)$
- D.  $\Theta(n^2)$

# Mergesort analysis

**Proposition.** Mergesort makes  $\leq n \log_2 n$  compares to sort any array of length  $n$ .

Pf by picture. [assuming  $n$  is a power of 2]





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

## 4.2 ALGORITHMS

---

- ▶ *sequential search*
- ▶ *binary search*
- ▶ *insertion sort*
- ▶ *mergesort*
- ▶ *system libraries*

# Arrays Javadoc

---

`java.util.Arrays` contains static methods for manipulating arrays.  *sorting, searching, comparing, ...*

**Module** `java.base`

**Package** `java.util`

## **Class Arrays**

`java.lang.Object`

`java.util.Arrays`

---

```
public class Arrays
```

```
extends Object
```

This class contains various methods for manipulating arrays (such as sorting and searching). This class also contains a static factory that allows arrays to be viewed as lists.


The methods in this class all throw a `NullPointerException`, if the specified array reference is null, except where noted.

The documentation for the methods contained in this class includes brief descriptions of the *implementations*. Such descriptions should be regarded as *implementation notes*, rather than parts of the *specification*. Implementors should feel free to substitute other algorithms, so long as the specification itself is adhered to. (For example, the algorithm used by `sort(Object[])` does not have to be a MergeSort, but it does have to be *stable*.)

# System sort and binary search

---

`Arrays.sort()` and `Arrays.binarySearch()` each refer to several overloaded methods.

- For primitive types.
- For reference types.  *must be Comparable  
(such as String, stay tuned)*
- For subarrays.

```
import java.util.Arrays;

public class TestSort {
    public static void main(String[] args) {
        Arrays.sort(args);
        StdOut.println(Arrays.toString(args));
    }
}
```

*to access methods defined  
in java.util.Arrays*



*useful function to print elements in an array  
(separated by commas, enclosed in square braces)*



```
~/cos126/algorithms> java-introcs TestSort S O R T M E  
[E, M, O, R, S, T]
```

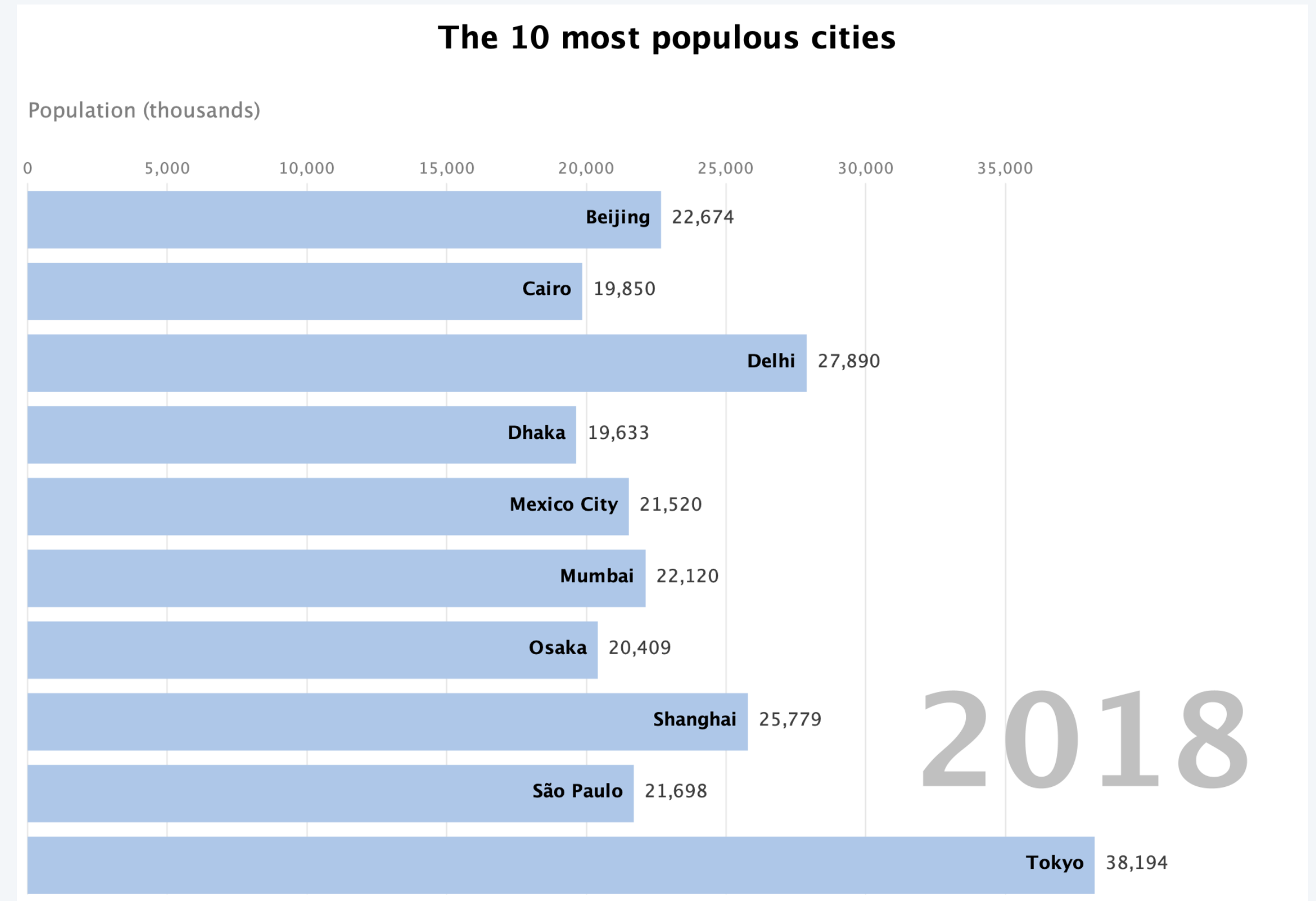
# Bar charts

`BarChart` is a data type (with simple API) to draw bar charts.

```
// create the bar chart
String title = "The 10 most populous cities";
String xlabel = "Population (thousands)";
String source = "Source: United Nations";
BarChart chart = new BarChart(title, xlabel, source);

// add the bars and caption to the bar chart
chart.add("Beijing", 22674, "East Asia");
chart.add("Cairo", 19850, "Middle East");
chart.add("Delhi", 27890, "South Asia");
chart.add("Dhaka", 19633, "South Asia");
chart.add("Mexico City", 21520, "Latin America");
chart.add("Mumbai", 22120, "South Asia");
chart.add("Osaka", 20409, "East Asia");
chart.add("Shanghai", 25779, "East Asia");
chart.add("São Paulo", 21698, "Latin America");
chart.add("Tokyo", 38194, "East Asia");
chart.setCaption("2018");

// draw the bar chart
chart.draw();
```



# Sorted bar chart

**Sort** the bars in descending order by value (e.g., population).

**Color** the bars by category (e.g., region).

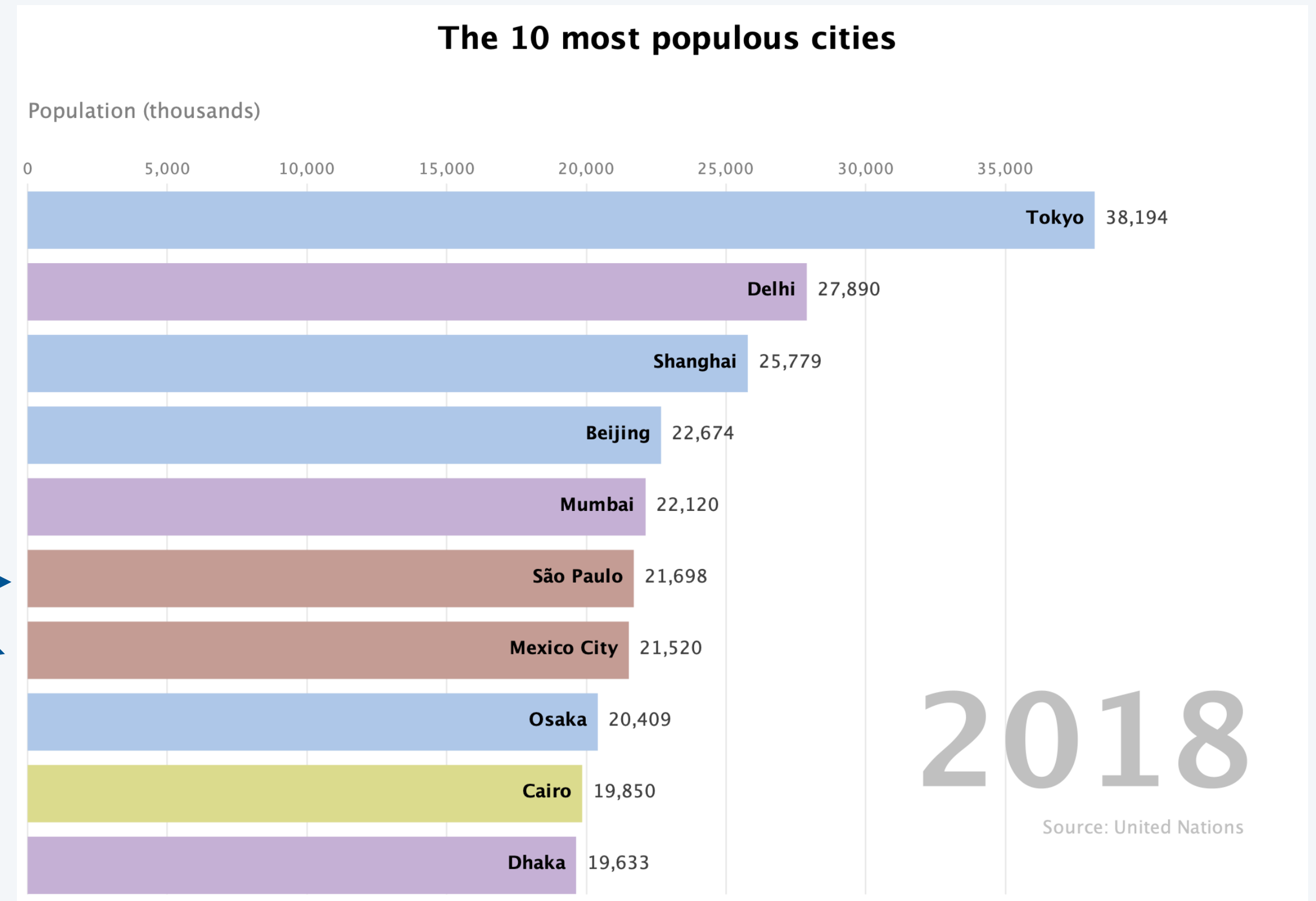
*name and region*



**Challenge.** Maintain auxiliary info when sorting.

**Solution.** Define helper **Bar** data type.

*Latin America*



*bars appear in descending  
order by population*

# Bar data type

---

To create a user-defined data type for use with `Arrays.sort()`:

- Declare the data type to be `Comparable`.
- Include a `compareTo()` method that compares two objects.

```
public class Bar implements Comparable<Bar> {  
    private final String name;  
    private final int value;  
    private final String category;  
  
    public Bar(String name, int value, String category) {  
        this.name = name;  
        this.value = value;  
        this.category = category;  
    }  
  
    public int compareTo(Bar that) {  
        if (this.value < that.value) return -1;  
        if (this.value > that.value) return +1;  
        return 0;  
    }  
    ...  
}
```

```
public int compareTo(Bar that) {  
    if (this.value < that.value) return -1;  
    if (this.value > that.value) return +1;  
    return 0;  
}
```

*return a negative integer (smaller),  
positive integer (larger), or zero (equal)*



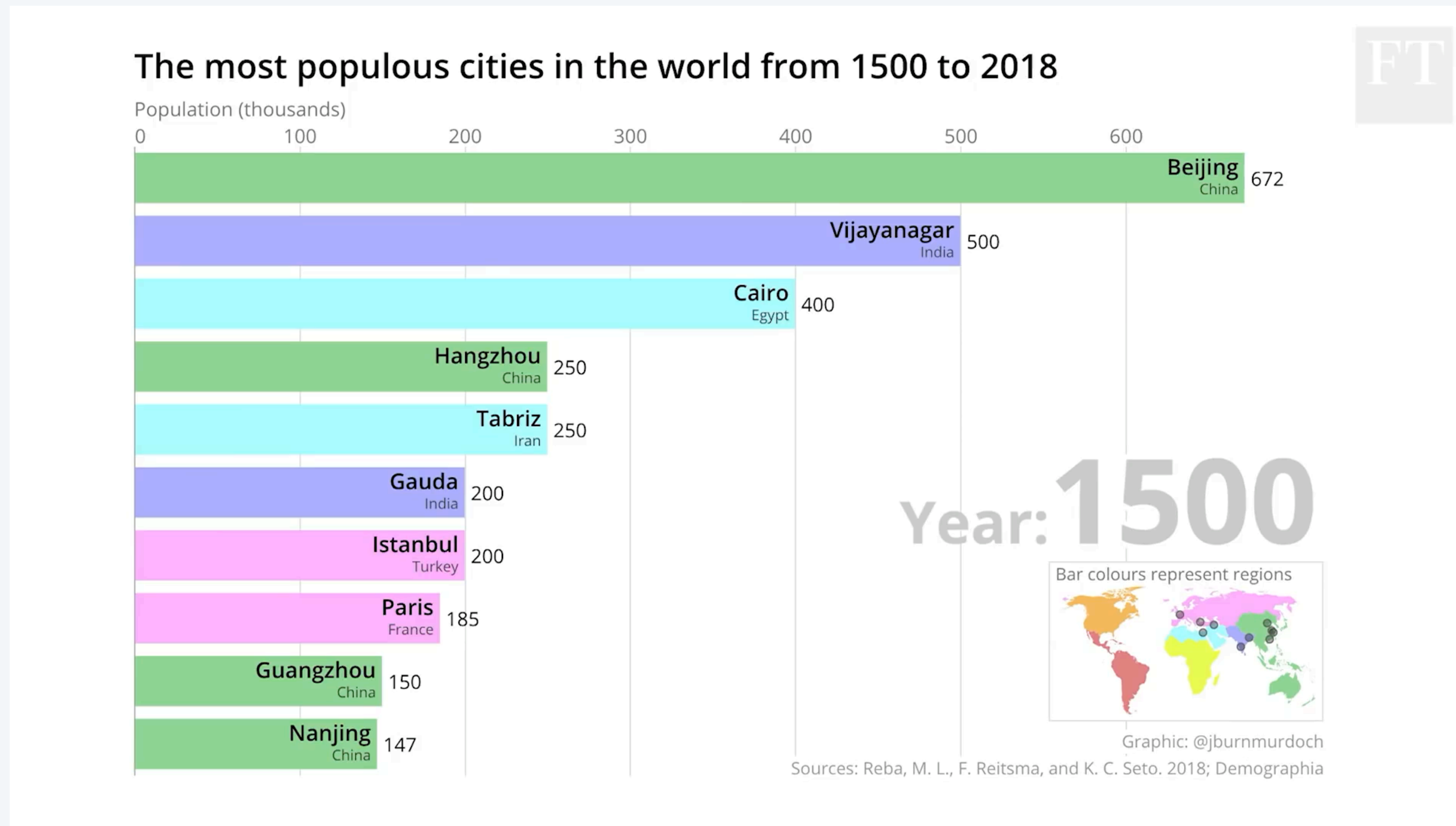
```
Bar[] bars = new Bar[10];  
...  
Arrays.sort(bars);
```

**sort bars in ascending order by value**



# Bar chart race

Animated bar chart. Draw sorted bar chart (largest  $k = 10$ ) for each year.



<https://www.youtube.com/watch?v=pMs5xapBewM>

# Credits

---

<b>media</b>	<b>source</b>	<b>license</b>
<i>Algorithm Icon</i>	<u>Adobe Stock</u>	<u>education license</u>
<i>Gospel of Algorithms</i>	<u>Jamis Buck</u>	
<i>Pita Bread Algorithm</i>	<u>Rishal Hurbans</u>	
<i>Algorithms T-shirt</i>	<u>Hapo Team</u>	
<i>Binary Search Instructions</i>	<u>IDEA</u>	<u>CC BY-NC-SA 4.0</u>
<i>Nuclear Power Plant</i>	<u>Adobe Stock</u>	<u>education license</u>
<i>Sorting Hat</i>	<u>Hannah Hill</u>	<u>CC BY-NC 4.0</u>
<i>Playing Cards</i>	<u>Google Code</u>	<u>public domain</u>
<i>Jon von Neumann</i>	<u>IAS / Alan Richards</u>	
<i>java.util.Arrays API</i>	<u>Oracle</u>	
<i>Most Populous Cities</i>	<u>John Burn-Murdoch</u>	