

2.2 Mergesort



- ▶ mergesort
- ▶ bottom-up mergesort
- ▶ sorting complexity
- ▶ comparators
- ▶ stability

Two classic sorting algorithms

Critical components in the world's computational infrastructure.

- Full scientific understanding of their properties has enabled us to develop them into practical system sorts.
- Quicksort honored as one of top 10 algorithms of 20th century in science and engineering.

Mergesort.

← today

- Java sort for objects.
- Perl, C++ stable sort, Python stable sort, Firefox JavaScript, ...

Quicksort.

← next lecture

- Java sort for primitive types.
- C qsort, Unix, Visual C++, Python, Matlab, Chrome JavaScript, ...

- ▶ mergesort
- ▶ bottom-up mergesort
- ▶ sorting complexity
- ▶ comparators
- ▶ stability

Mergesort

Basic plan.

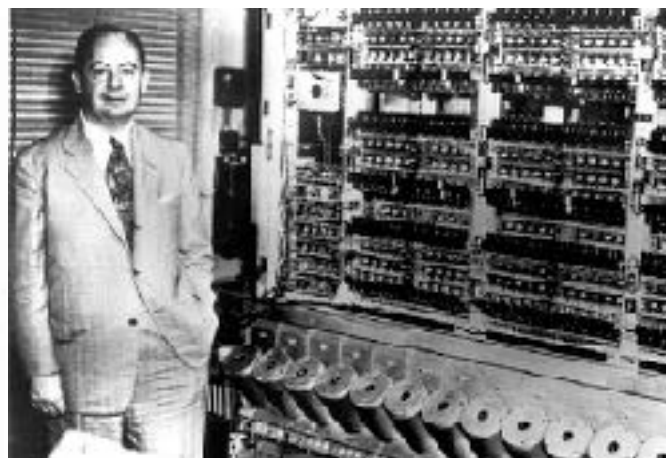
- Divide array into two halves.
- **Recursively** sort each half.
- Merge two halves.

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

Mergesort overview

First Draft
of a
Report on the
EDVAC

John von Neumann



Merging

- Q. How to combine two sorted subarrays into a sorted whole.
- A. Use an auxiliary array.

a[]											aux[]												
	k	0	1	2	3	4	5	6	7	8	9	i	j	0	1	2	3	4	5	6	7	8	9
input		E	E	G	M	R	A	C	E	R	T			-	-	-	-	-	-	-	-	-	-
copy		E	E	G	M	R	A	C	E	R	T			E	E	G	M	R	A	C	E	R	T
												0	5										
	0	A										0	6	E	E	G	M	R	A	C	E	R	T
	1	A	C									0	7	E	E	G	M	R		C	E	R	T
	2	A	C	E								1	7	E	E	G	M	R			E	R	T
	3	A	C	E	E							2	7		E	G	M	R			E	R	T
	4	A	C	E	E	E						2	8			G	M	R		E	R	T	
	5	A	C	E	E	E	G					3	8			G	M	R				R	T
	6	A	C	E	E	E	G	M				4	8				M	R				R	T
	7	A	C	E	E	E	G	M	R			5	8					R				R	T
	8	A	C	E	E	E	G	M	R	R		5	9								R	T	
	9	A	C	E	E	E	G	M	R	R	T	6	10										T
merged result		A	C	E	E	E	G	M	R	R	T												

Abstract in-place merge trace

Merging: Java implementation

```
private static void merge(Comparable[] a, Comparable[] aux, int lo, int mid, int hi)
{
    assert isSorted(a, lo, mid);    // precondition: a[lo..mid]    sorted
    assert isSorted(a, mid+1, hi);  // precondition: a[mid+1..hi] sorted

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

    int i = lo, j = mid+1;
    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++];
    }

    assert isSorted(a, lo, hi);    // postcondition: a[lo..hi] sorted
}
```

	lo			i	mid			j		hi	
aux[]	A	G	L	O	R		H	I	M	S	T
							k				
a[]	A	G	H	I	L	M					

Assertions

Assertion. Statement to test assumptions about your program.

- Helps detect logic bugs.
- Documents code.

Java assert statement. Throws an exception unless boolean condition is true.

```
assert isSorted(a, lo, hi);
```

Can enable or disable at runtime. \Rightarrow No cost in production code.

```
java -ea MyProgram    // enable assertions
java -da MyProgram    // disable assertions (default)
```

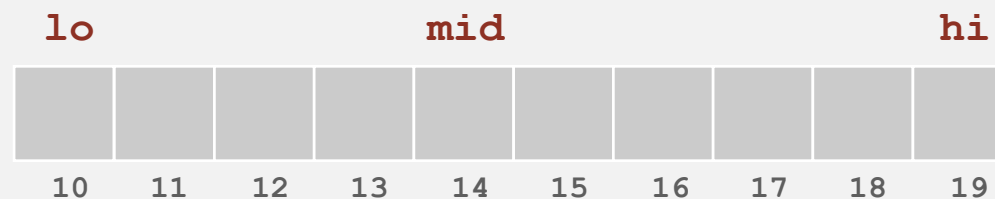
Best practices. Use to check internal invariants. Assume assertions will be disabled in production code (e.g., don't use for external argument-checking).

Mergesort: Java implementation

```
public class Merge
{
    private static void merge(Comparable[] a, Comparable[] aux, int lo, int mid, int hi)
    { /* as before */ }

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



Mergesort trace

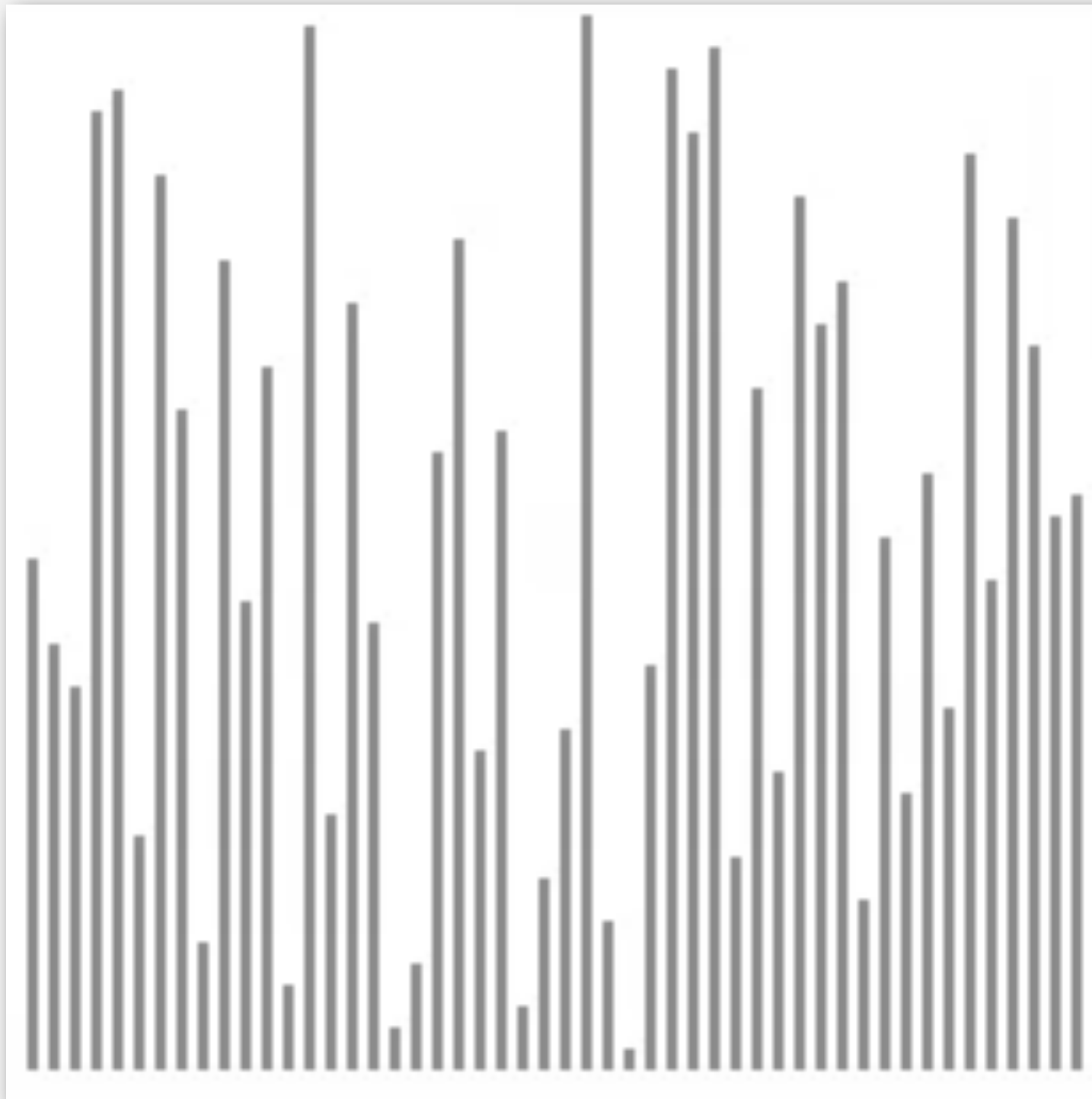
	lo	hi	a[]															
			0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
			M	E	R	G	E	S	O	R	T	E	X	A	M	P	L	E
merge(a,	0	0, 1)	E	M	R	G	E	S	O	R	T	E	X	A	M	P	L	E
merge(a,	2	2, 3)	E	M	G	R	E	S	O	R	T	E	X	A	M	P	L	E
merge(a,	0	1, 3)	E	G	M	R	E	S	O	R	T	E	X	A	M	P	L	E
merge(a,	4	4, 5)	E	G	M	R	E	S	O	R	T	E	X	A	M	P	L	E
merge(a,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

Trace of merge results for top-down mergesort

result after recursive call

Mergesort animation

50 random elements

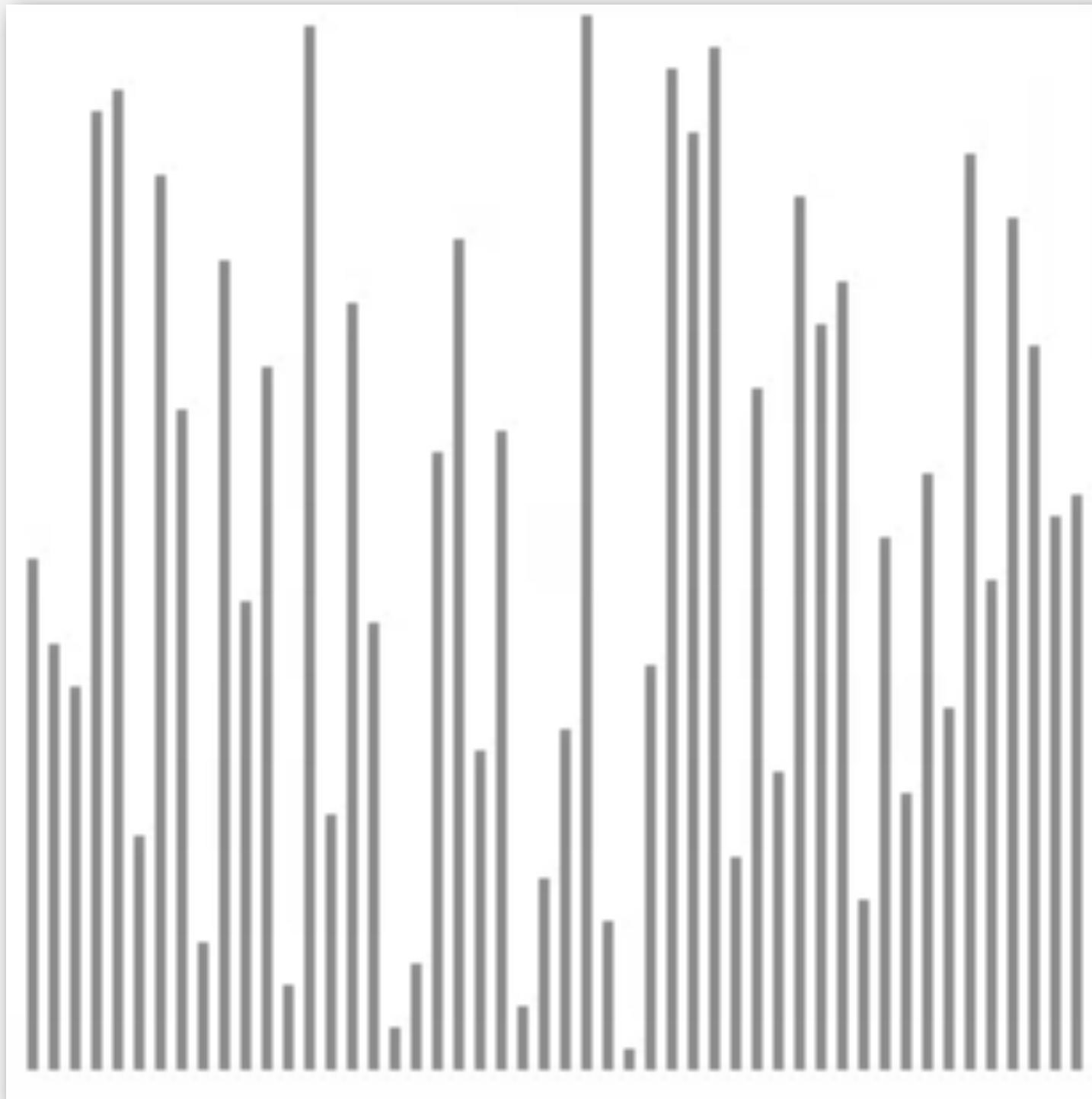


- ▲ algorithm position
- █ in order
- █ current subarray
- █ not in order

<https://www.cs.purdue.edu/homes/cs251/slides/media/merge-sort.mov>

Mergesort animation

50 random elements

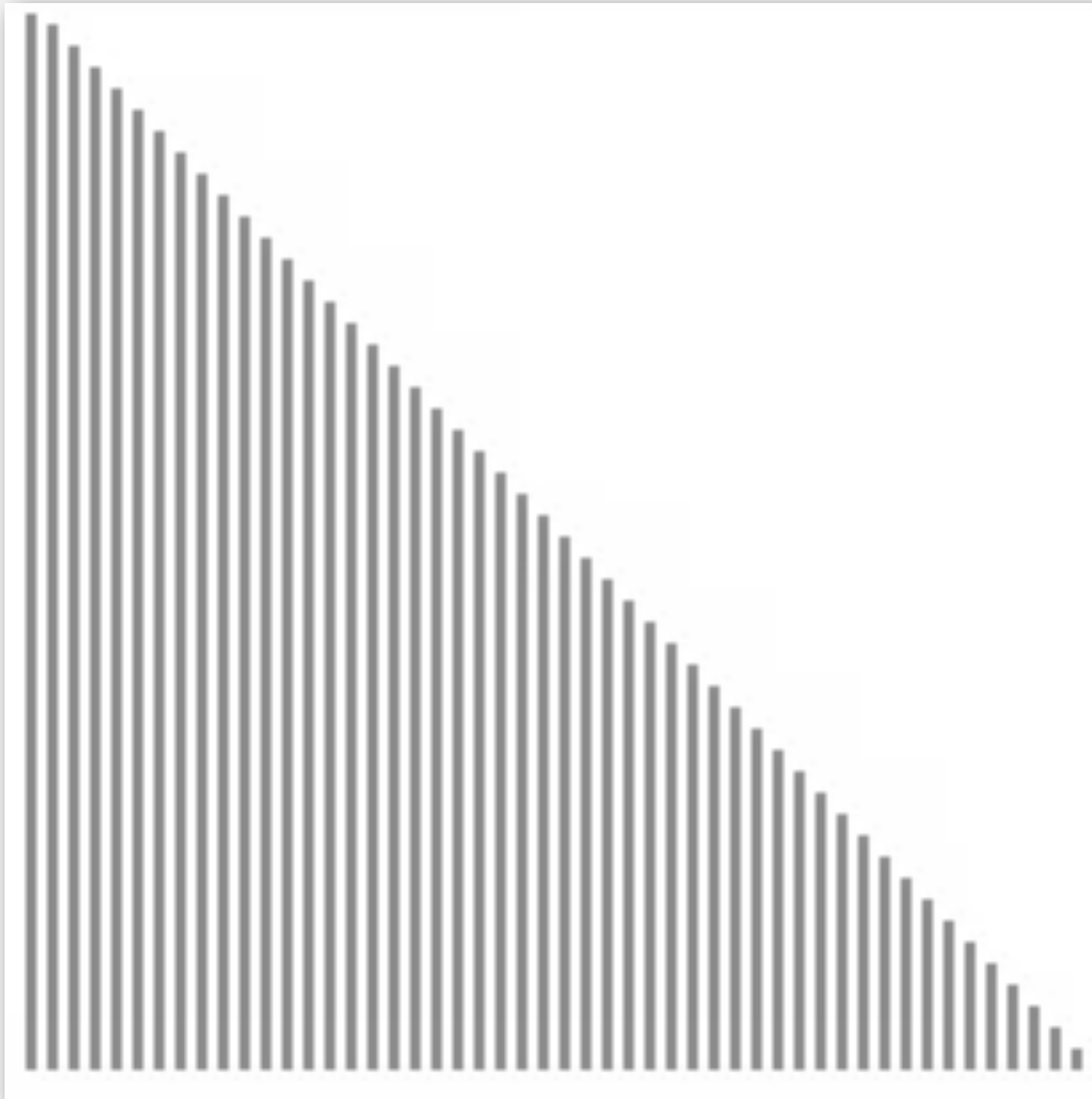


- ▲ algorithm position
- in order
- current subarray
- not in order

<https://www.cs.purdue.edu/homes/cs251/slides/media/merge-sort.mov>

Mergesort animation

50 reverse-sorted elements

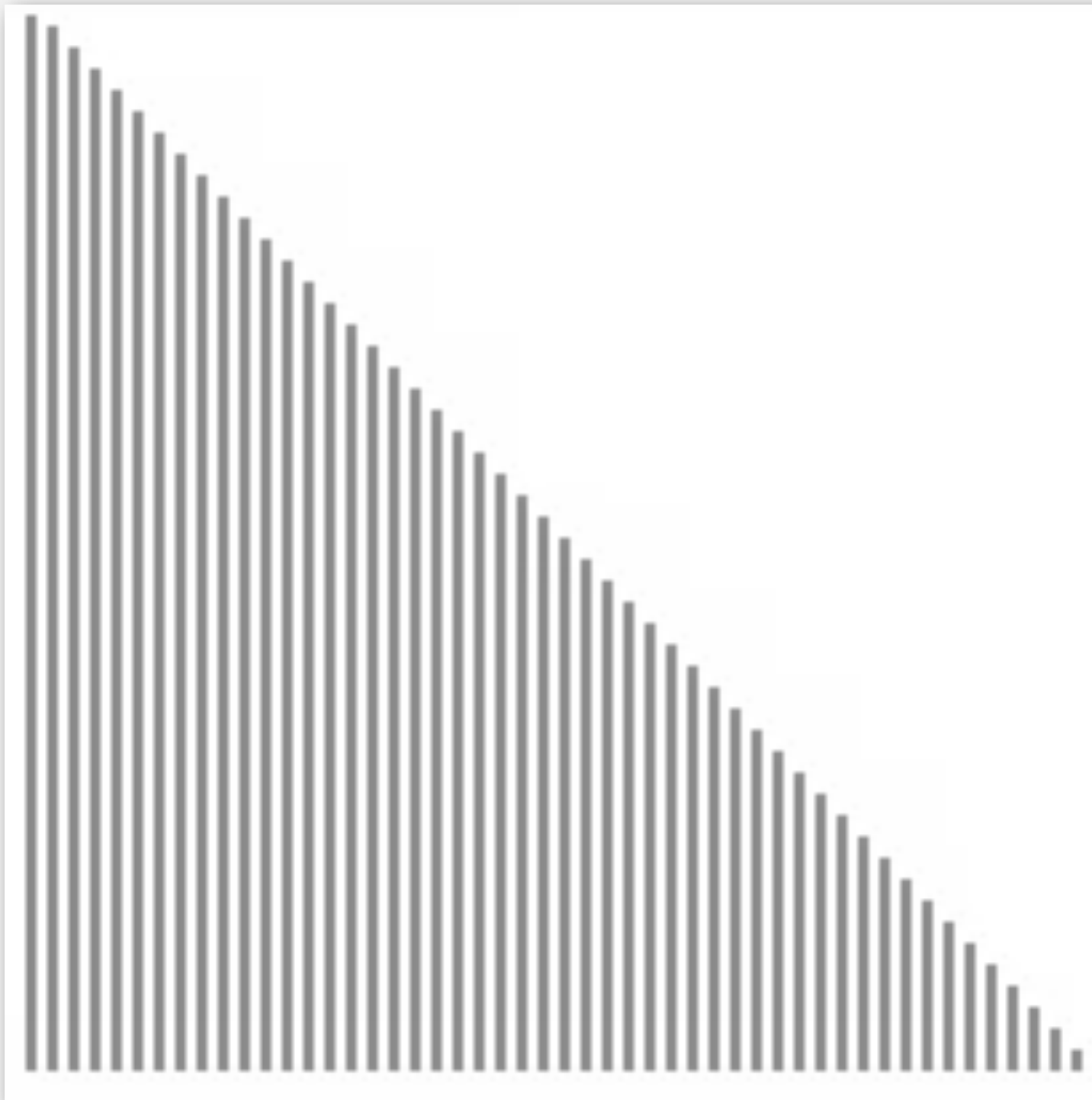


- ▲ algorithm position
- █ in order
- █ current subarray
- █ not in order

<https://www.cs.purdue.edu/homes/cs251/slides/media/merge-sort1.mov>

Mergesort animation

50 reverse-sorted elements



- ▲ algorithm position
- █ in order
- █ current subarray
- █ not in order

<https://www.cs.purdue.edu/homes/cs251/slides/media/merge-sort1.mov>

Mergesort: empirical analysis

Running time estimates:

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

	insertion sort (N^2)			mergesort ($N \log N$)		
computer	thousand	million	billion	thousand	million	billion
home	instant	2.8 hours	317 years	instant	1 second	18 min
super	instant	1 second	1 week	instant	instant	instant

Bottom line. Good algorithms are better than supercomputers.

Mergesort: number of compares and array accesses

Proposition. Mergesort uses at most $N \lg N$ compares and $6 N \lg N$ array accesses to sort any array of size N .

Pf sketch. The number of compares $C(N)$ and array accesses $A(N)$ to mergesort an array of size N satisfies the recurrences:

$$C(N) \leq C(\lfloor N/2 \rfloor) + C(\lceil N/2 \rceil) + N \quad \text{for } N > 1, \text{ with } C(1) = 0.$$

↑
left half
↓

↑
right half
↓

↑
merge
↓

$$A(N) \leq A(\lfloor N/2 \rfloor) + A(\lceil N/2 \rceil) + 6N \quad \text{for } N > 1, \text{ with } A(1) = 0.$$

We solve the simpler divide-and-conquer recurrence when N is a power of 2.



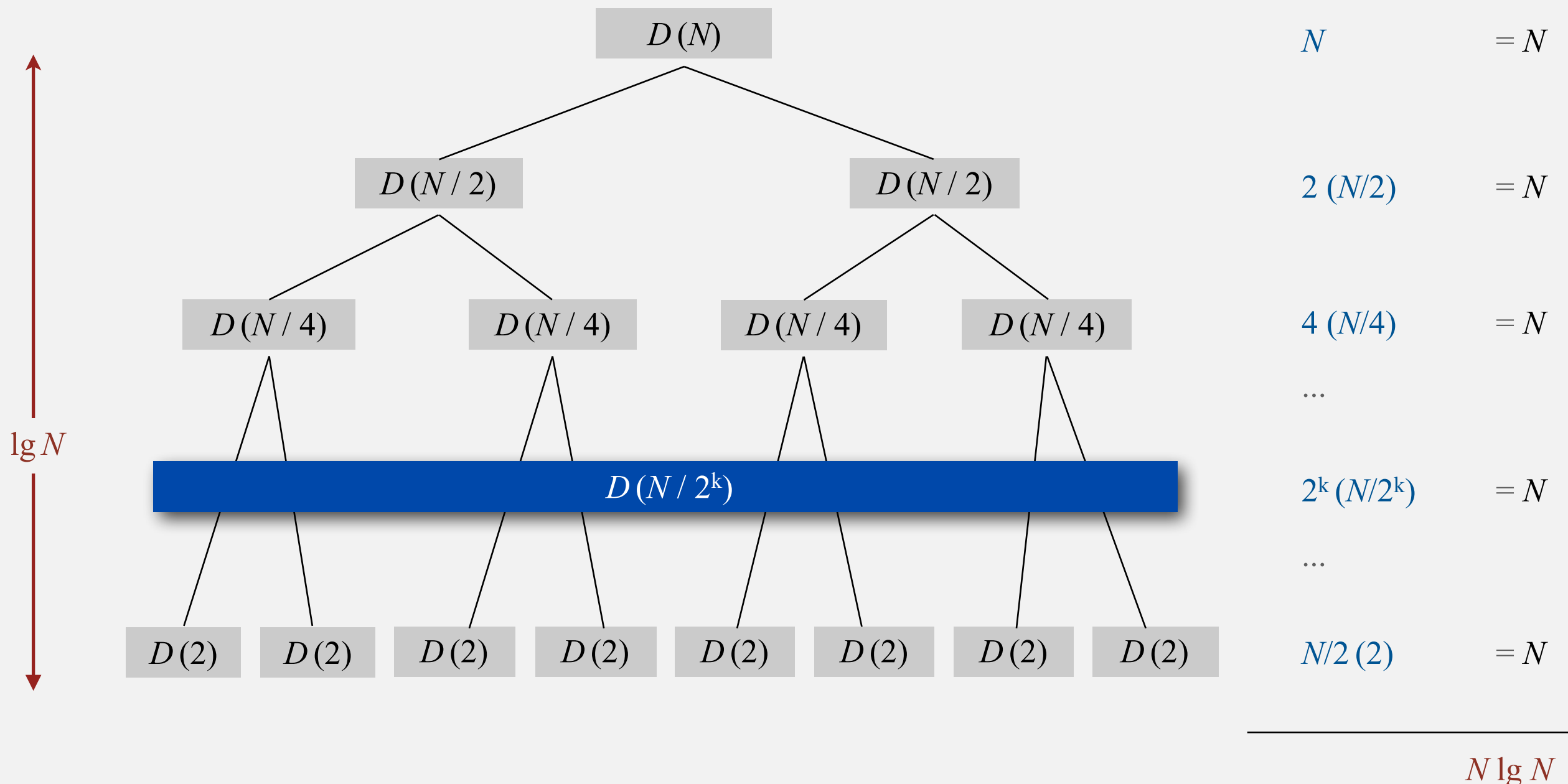
result holds for all N

$$D(N) = 2 D(N/2) + N, \text{ for } N > 1, \text{ with } D(1) = 0.$$

Divide-and-conquer recurrence: proof by picture

Proposition. If $D(N)$ satisfies $D(N) = 2 D(N/2) + N$ for $N > 1$, with $D(1) = 0$, then $D(N) = N \lg N$.

Pf 1. [assuming N is a power of 2]



Divide-and-conquer recurrence: proof by expansion

Proposition. If $D(N)$ satisfies $D(N) = 2 D(N/2) + N$ for $N > 1$, with $D(1) = 0$, then $D(N) = N \lg N$.

Pf 2. [assuming N is a power of 2]

$$D(N) = 2 D(N/2) + N$$

given

$$D(N) / N = 2 D(N/2) / N + 1$$

divide both sides by N

$$= D(N/2) / (N/2) + 1$$

algebra

$$= D(N/4) / (N/4) + 1 + 1$$

apply to first term

$$= D(N/8) / (N/8) + 1 + 1 + 1$$

apply to first term again

...

$$= D(N/N) / (N/N) + 1 + 1 + \dots + 1$$

$$= \lg N$$

stop applying, $D(1) = 0$

Divide-and-conquer recurrence: proof by induction

Proposition. If $D(N)$ satisfies $D(N) = 2 D(N/2) + N$ for $N > 1$, with $D(1) = 0$, then $D(N) = N \lg N$.

Pf 3. [assuming N is a power of 2]

- Base case: $N = 1$.
- Inductive hypothesis: $D(N) = N \lg N$.
- Goal: show that $D(2N) = (2N) \lg (2N)$.

$$D(2N) = 2 D(N) + 2N$$

given

$$= 2 N \lg N + 2N$$

inductive hypothesis

$$= 2 N (\lg (2N) - 1) + 2N$$

algebra

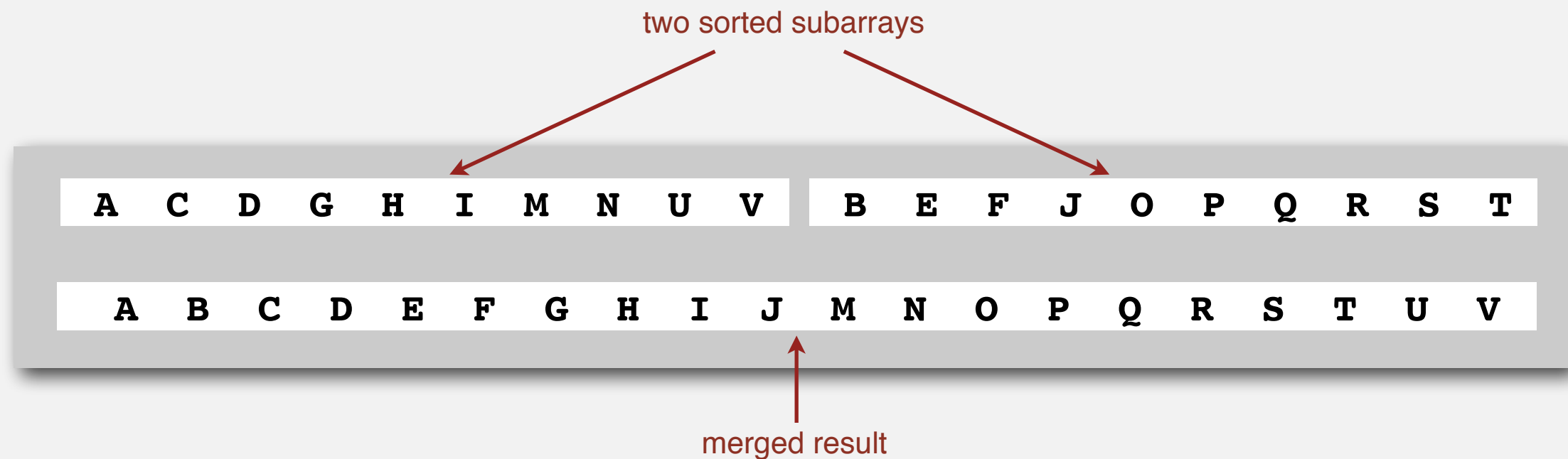
$$= 2 N \lg (2N)$$

QED

Mergesort analysis: memory

Proposition. Mergesort uses extra space proportional to N .

Pf. The array `aux[]` needs to be of size N for the last merge.



Def. A sorting algorithm is **in-place** if it uses $O(\log N)$ extra memory.

Ex. Insertion sort, selection sort, shellsort.

Challenge for the bored. In-place merge. [Kronrod, 1969]

Mergesort: practical improvements

Use insertion sort for small subarrays.

- Mergesort has too much overhead for tiny subarrays.
- Cutoff to insertion sort for ≈ 7 elements.

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

Mergesort: practical improvements

Stop if already sorted.

- Is biggest element in first half \leq smallest element in second half?
- Helps for partially-ordered arrays.

A B C D E F G H I **J**

M N O P Q R S T U V

A B C D E F G H I J M N O P Q R S T U V

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

Mergesort: practical improvements

Eliminate the copy to the auxiliary array. Save time (but not space) by switching the role of the input and auxiliary array in each recursive call.

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

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

← merge from a[] to aux[]

↑ switch roles of aux[] and a[]

Mergesort: practical improvements

Use insertion sort for small subarrays.

- Mergesort has too much overhead for tiny subarrays.
- Cutoff to insertion sort for ≈ 7 elements.

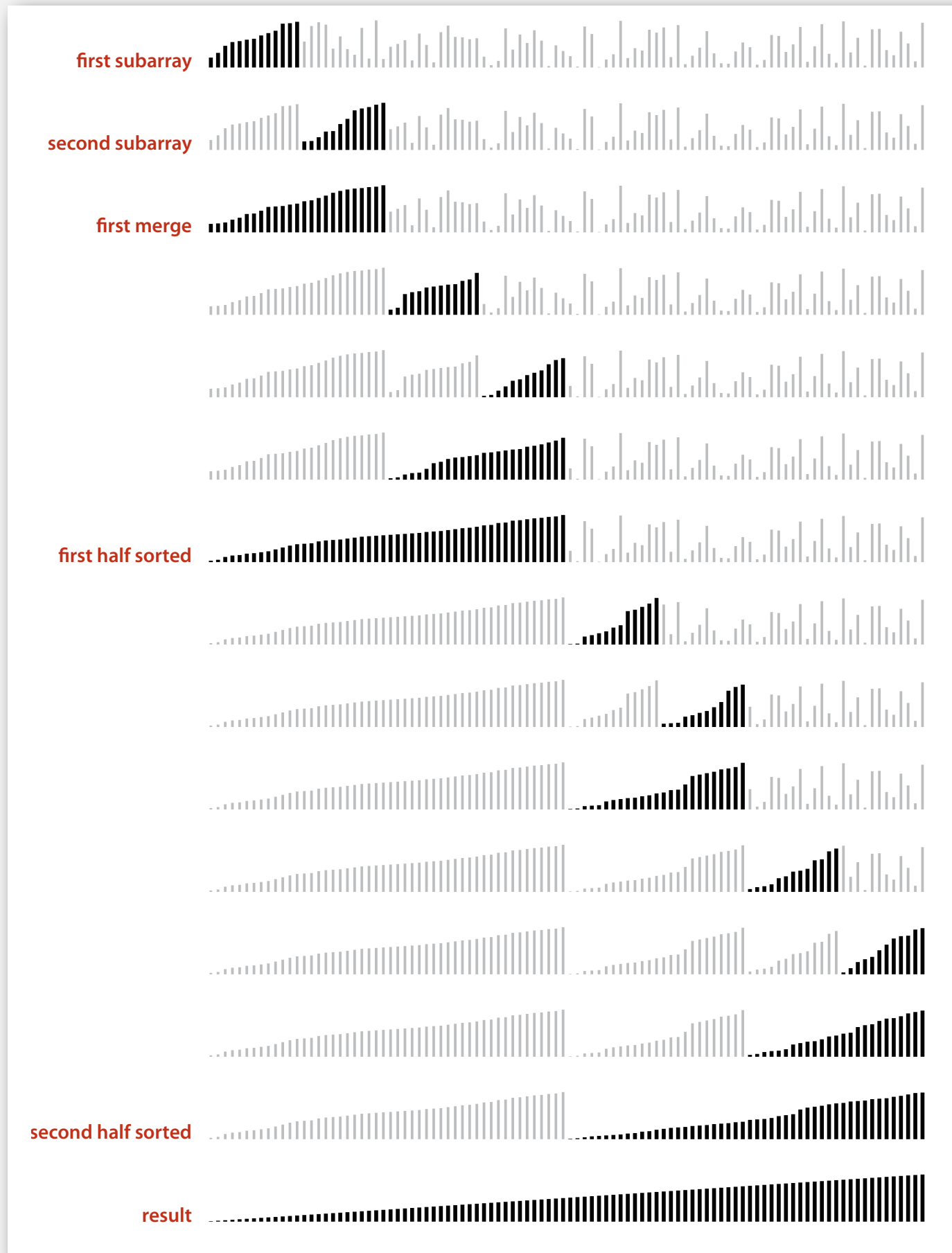
Stop if already sorted.

- Is biggest element in first half \leq smallest element in second half?
- Helps for partially-ordered arrays.

Eliminate the copy to the auxiliary array. Save time (but not space) by switching the role of the input and auxiliary array in each recursive call.

Ex. See `MergeX.java` or `Arrays.sort()`.

Mergesort visualization



- ▶ mergesort
- ▶ **bottom-up mergesort**
- ▶ sorting complexity
- ▶ comparators
- ▶ stability

Bottom-up mergesort

Basic plan.

- Pass through array, merging subarrays of size 1.
- Repeat for subarrays of size 2, 4, 8, 16,

				a[i]															
				0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
sz = 1				M	E	R	G	E	S	O	R	T	E	X	A	M	P	L	E
merge(a,	0,	0,	1)	E	M	R	G	E	S	O	R	T	E	X	A	M	P	L	E
merge(a,	2,	2,	3)	E	M	G	R	E	S	O	R	T	E	X	A	M	P	L	E
merge(a,	4,	4,	5)	E	M	G	R	E	S	O	R	T	E	X	A	M	P	L	E
merge(a,	6,	6,	7)	E	M	G	R	E	S	O	R	T	E	X	A	M	P	L	E
merge(a,	8,	8,	9)	E	M	G	R	E	S	O	R	E	T	X	A	M	P	L	E
merge(a,	10,	10,	11)	E	M	G	R	E	S	O	R	E	T	A	X	M	P	L	E
merge(a,	12,	12,	13)	E	M	G	R	E	S	O	R	E	T	A	X	M	P	L	E
merge(a,	14,	14,	15)	E	M	G	R	E	S	O	R	E	T	A	X	M	P	E	L
sz = 2				E	G	M	R	E	S	O	R	E	T	A	X	M	P	E	L
merge(a,	0,	1,	3)	E	G	M	R	E	O	R	S	E	T	A	X	M	P	E	L
merge(a,	4,	5,	7)	E	G	M	R	E	O	R	S	A	E	T	X	M	P	E	L
merge(a,	8,	9,	11)	E	G	M	R	E	O	R	S	A	E	T	X	E	L	M	P
merge(a,	12,	13,	15)	E	G	M	R	E	O	R	S	A	E	T	X	E	L	M	P
sz = 4				E	E	G	M	O	R	R	S	A	E	T	X	E	L	M	P
merge(a,	0,	3,	7)	E	E	G	M	O	R	R	S	A	E	E	L	M	P	T	X
merge(a,	8,	11,	15)	E	E	G	M	O	R	R	S	A	E	E	L	M	P	T	X
sz = 8				A	E	E	E	E	G	L	M	M	O	P	R	R	S	T	X
merge(a,	0,	7,	15)	A	E	E	E	E	G	L	M	M	O	P	R	R	S	T	X

Bottom line. No recursion needed!

Bottom-up mergesort: Java implementation

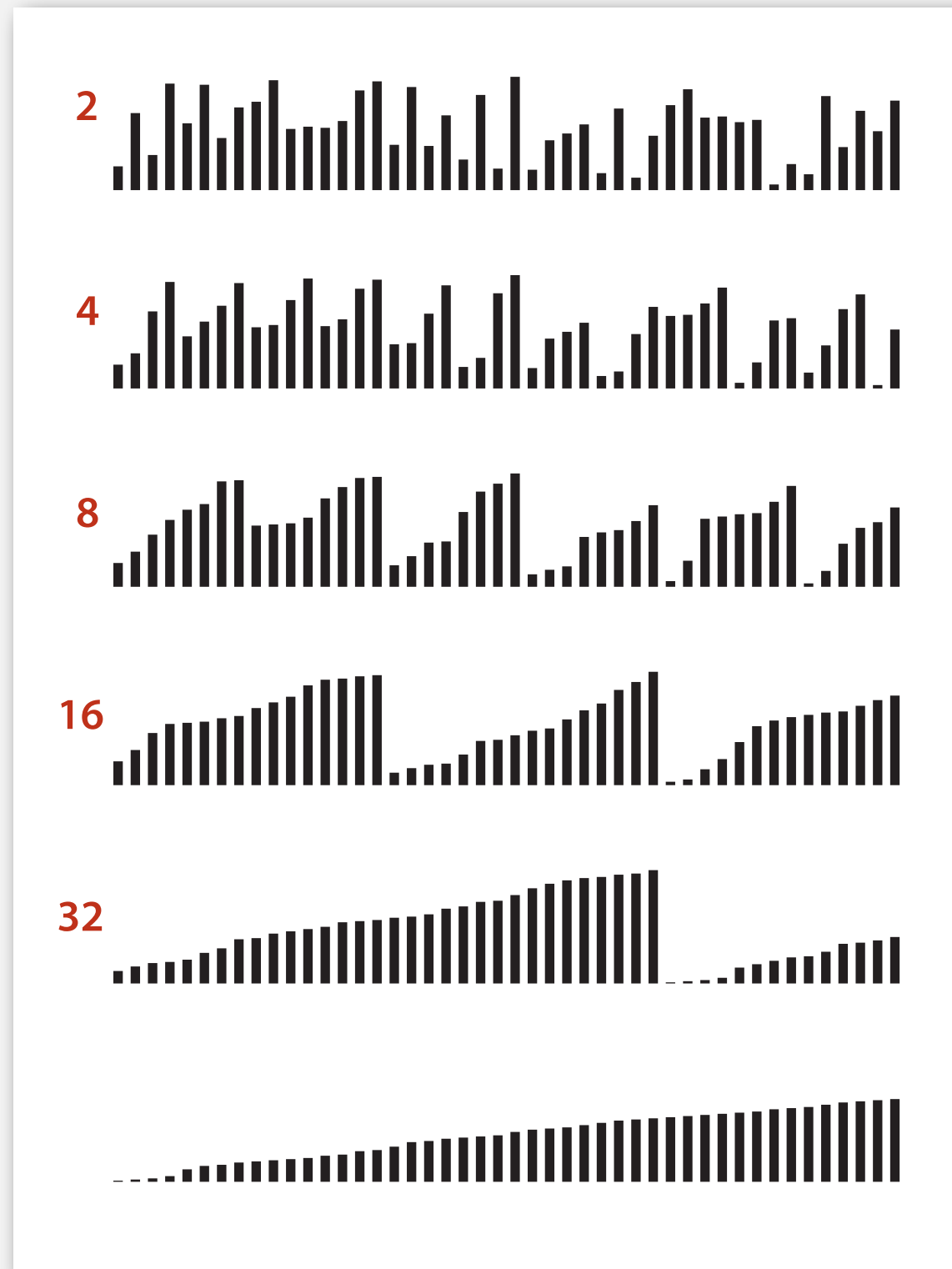
```
public class MergeBU
{
    private static Comparable[] aux;

    private static void merge(Comparable[] a, int lo, int mid, int hi)
    { /* as before */ }

    public static void sort(Comparable[] a)
    {
        int N = a.length;
        aux = new Comparable[N];
        for (int sz = 1; sz < N; sz = sz+sz)
            for (int lo = 0; lo < N-sz; lo += sz+sz)
                merge(a, lo, lo+sz-1, Math.min(lo+sz+sz-1, N-1));
    }
}
```

Bottom line. Concise industrial-strength code, if you have the space.

Bottom-up mergesort: visual trace



- ▶ mergesort
- ▶ bottom-up mergesort
- ▶ **sorting complexity**
- ▶ comparators
- ▶ stability

Complexity of sorting

Computational complexity. Framework to study efficiency of algorithms for solving a particular problem X .

Model of computation. Specify allowable operations.

Cost model. Focus on fundamental operations.

Upper bound. Cost guarantee provided by **some** algorithm for X .

Lower bound. Proven limit on cost guarantee of **all** algorithms for X .


Optimal algorithm. Algorithm with best cost guarantee for X .



lower bound ~ upper bound

Example: sorting.

- Model of computation: decision tree.
- Cost model: # compares.
- Upper bound: $\sim N \lg N$ from mergesort.
- Lower bound: $\sim N \lg N$???
- Optimal algorithm: mergesort ???

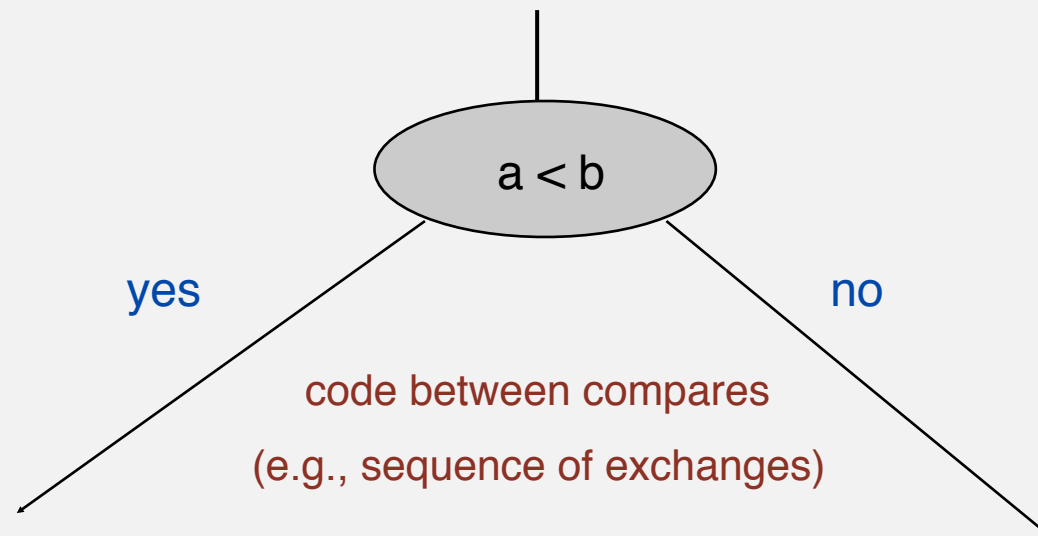


can access information
only through compares
(e.g., our Java sorting framework)

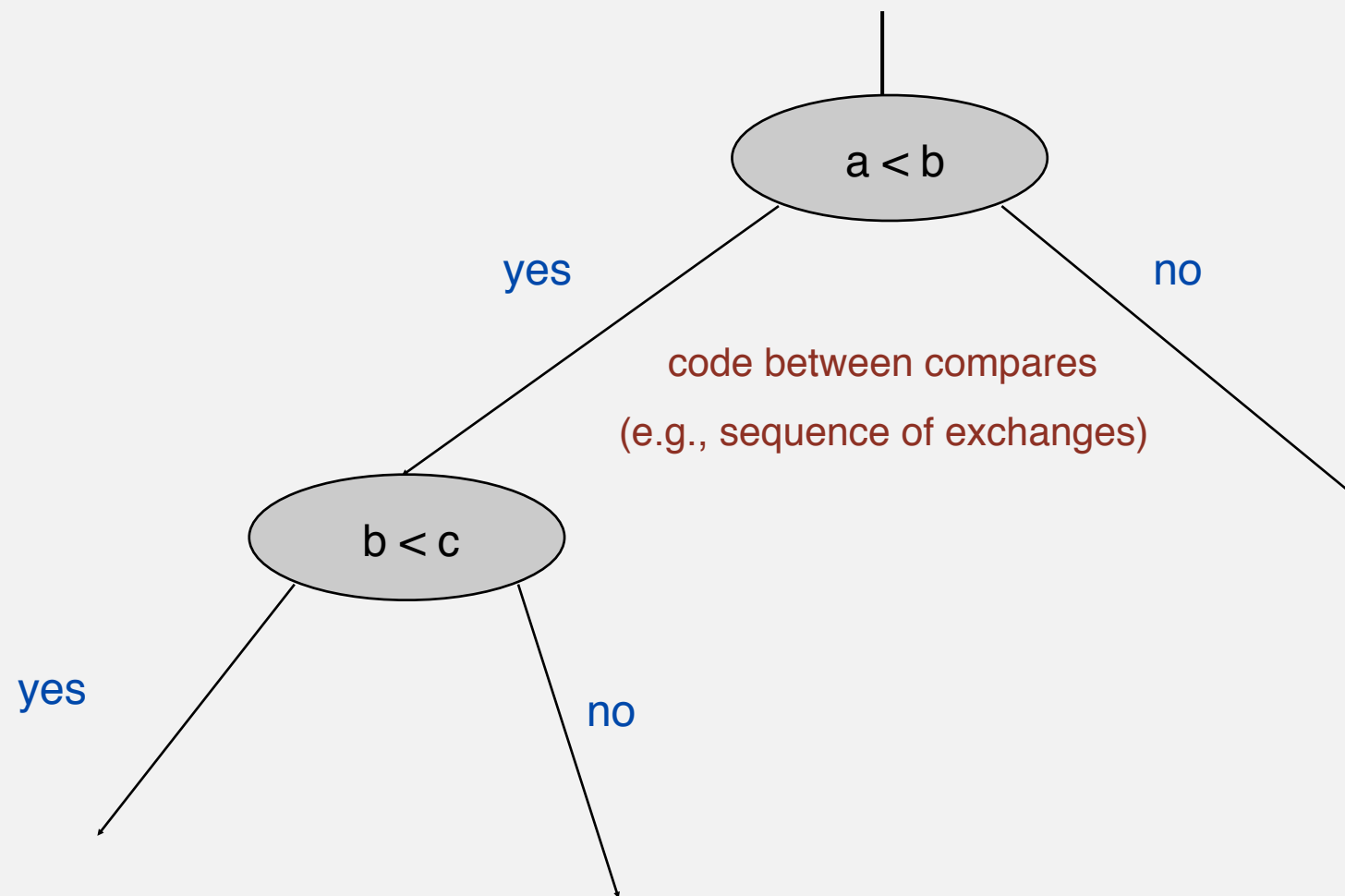
Decision tree (for 3 distinct elements a, b, and c)

|

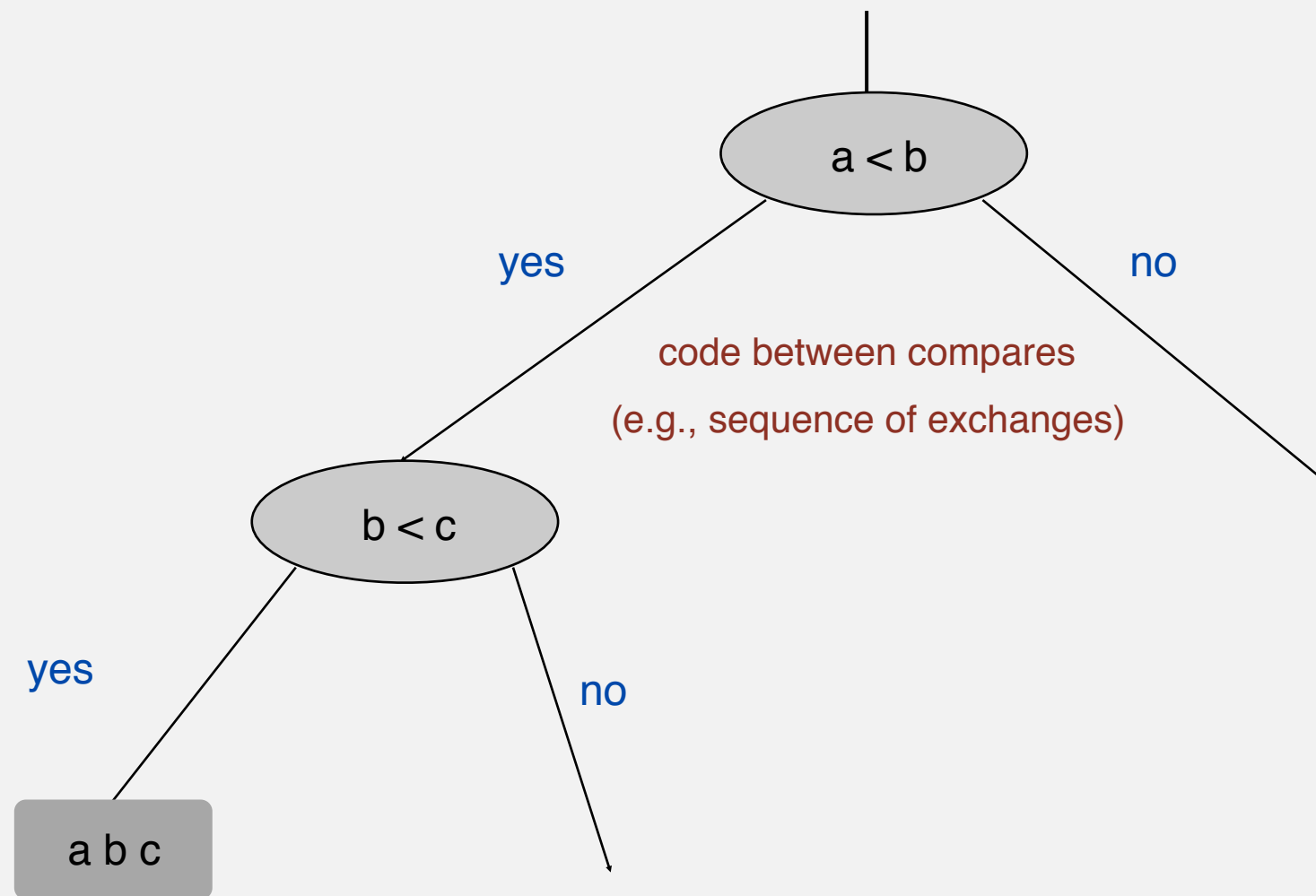
Decision tree (for 3 distinct elements a, b, and c)



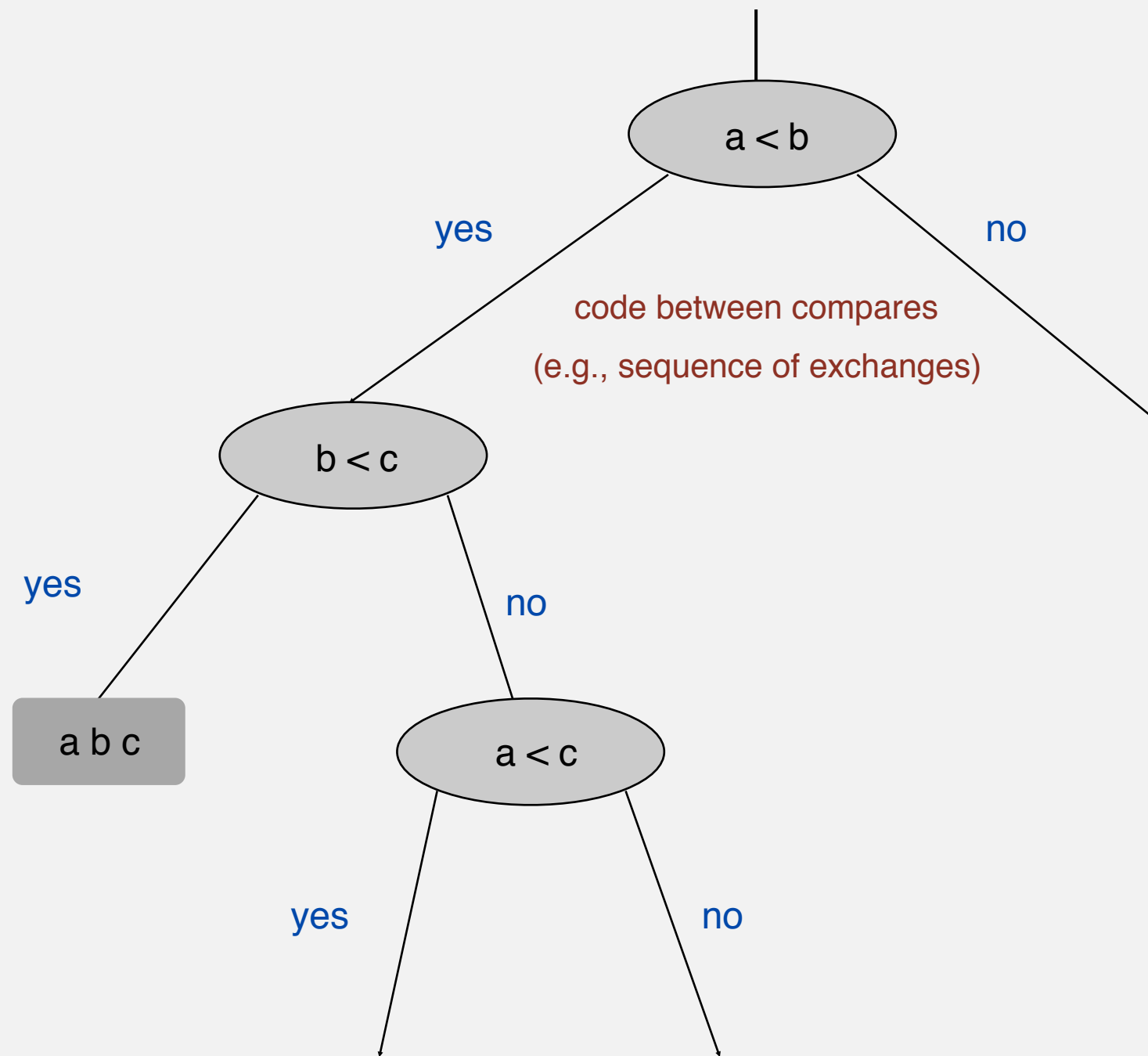
Decision tree (for 3 distinct elements a, b, and c)



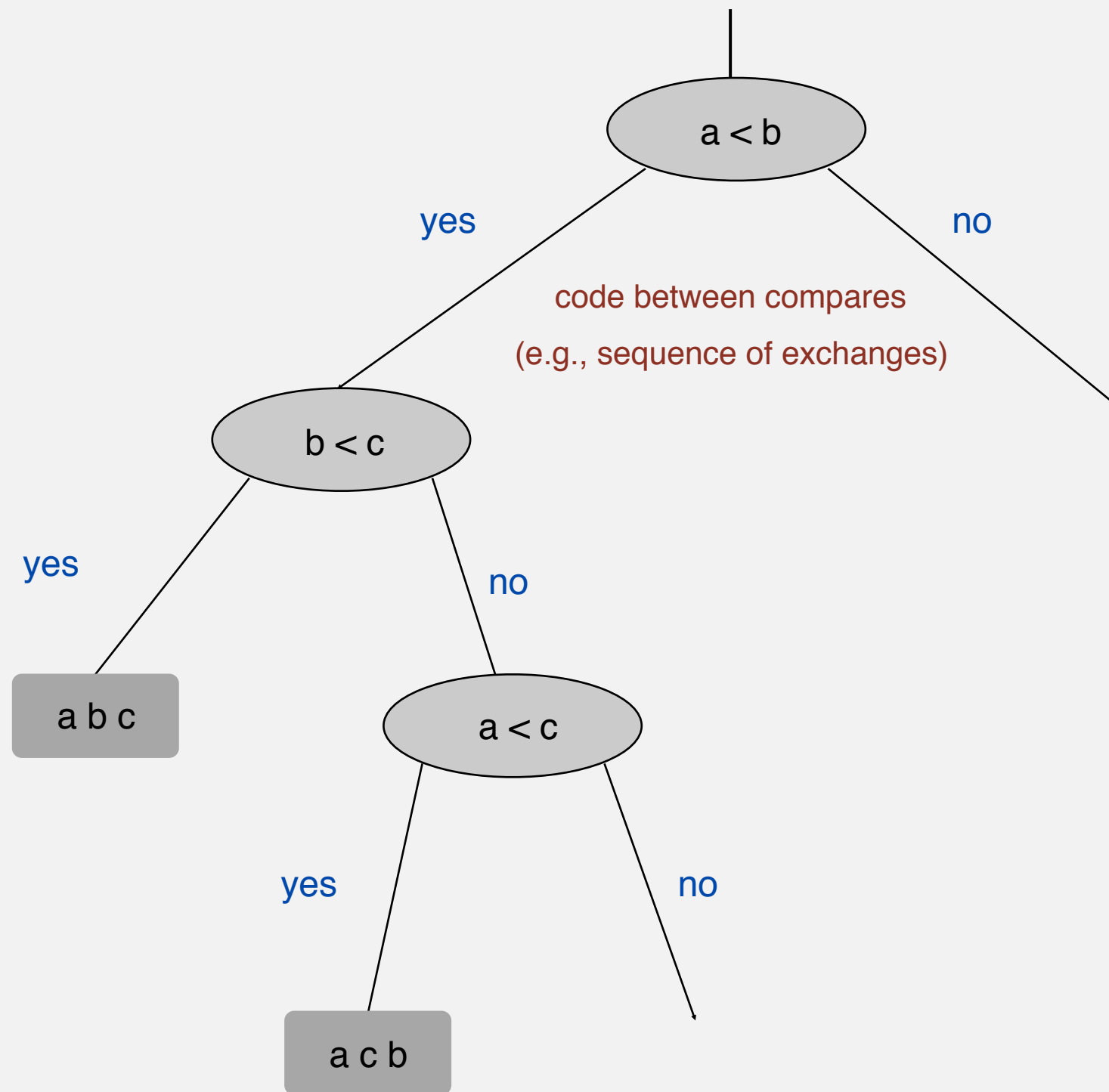
Decision tree (for 3 distinct elements a, b, and c)



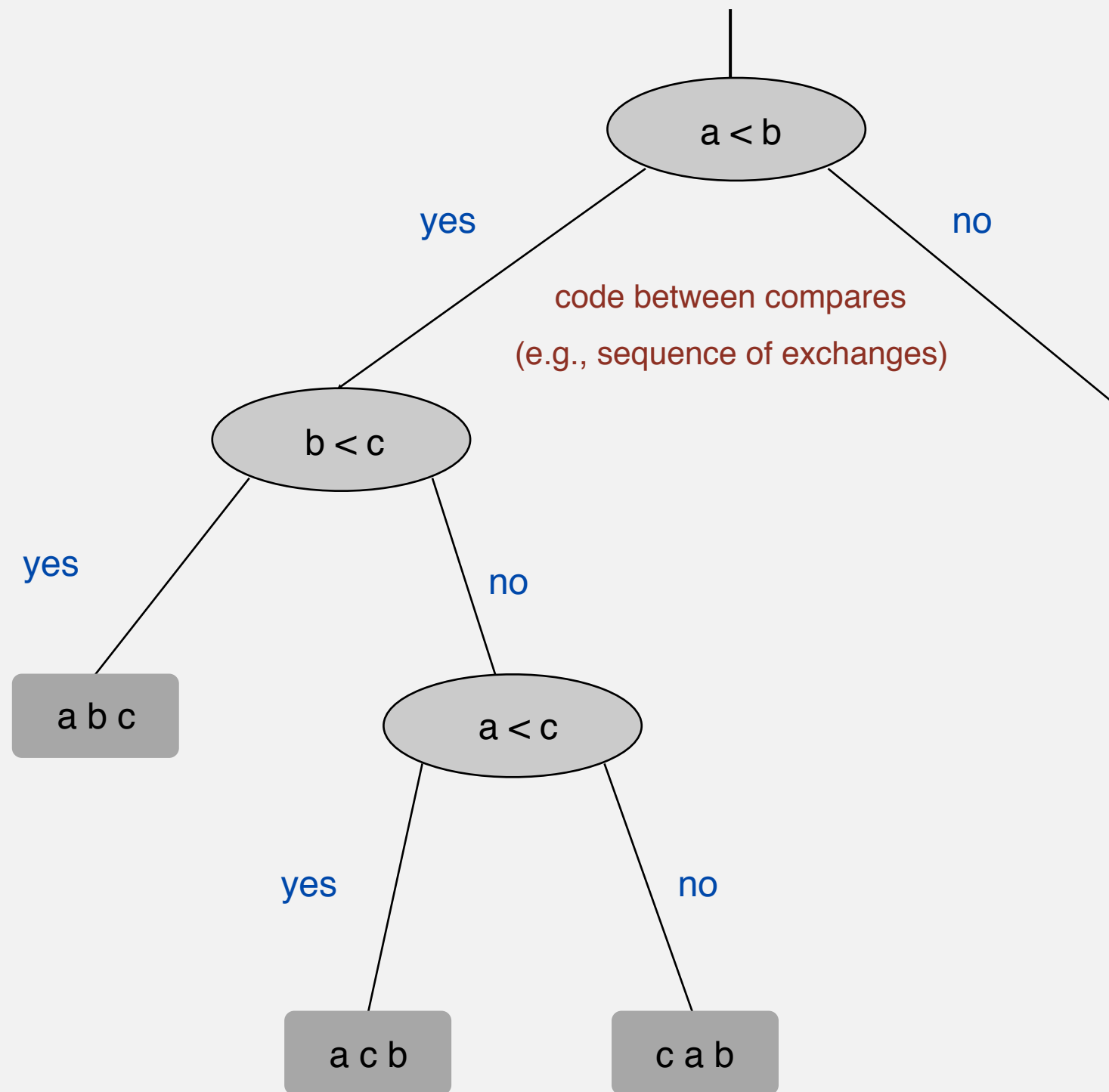
Decision tree (for 3 distinct elements a, b, and c)



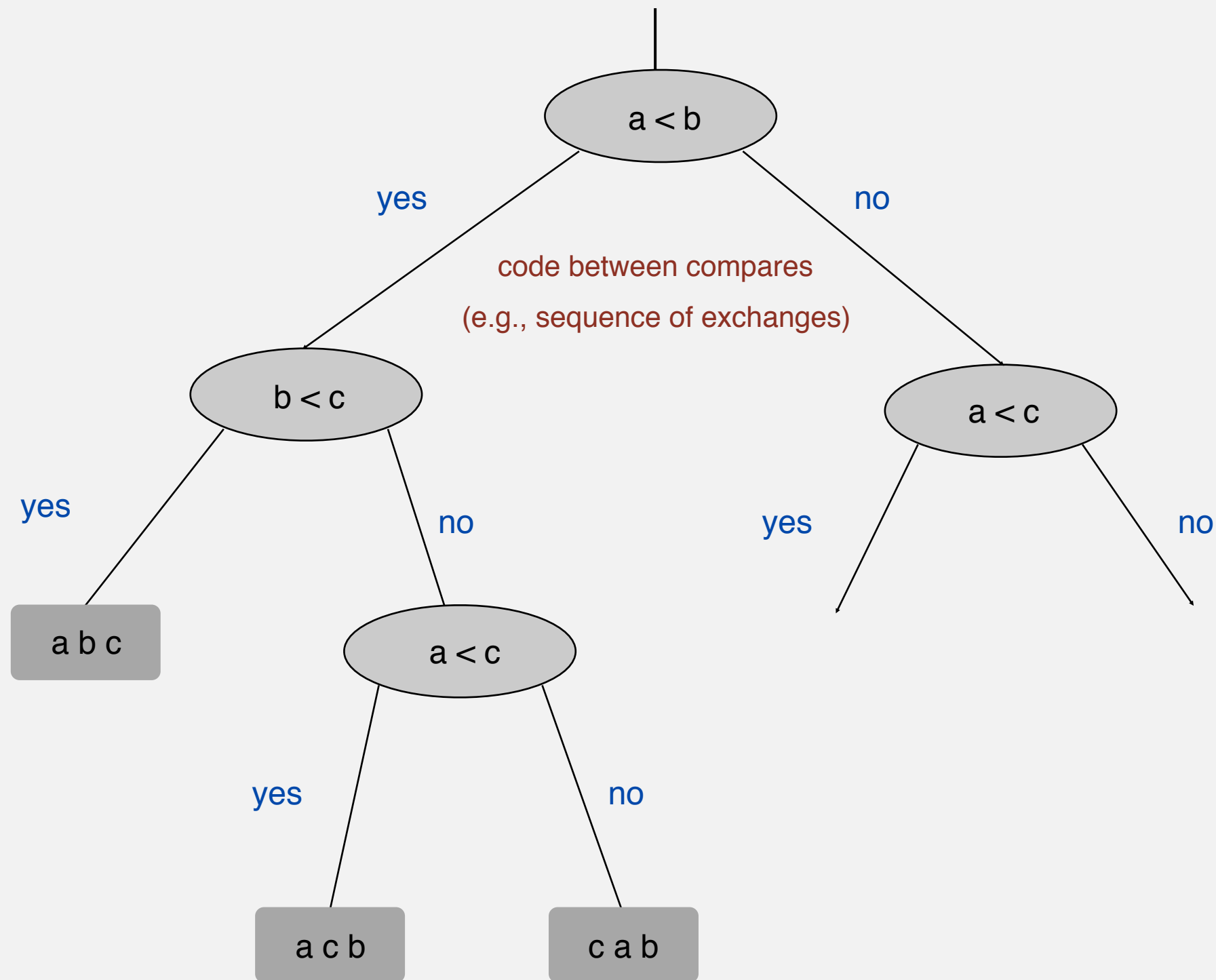
Decision tree (for 3 distinct elements a, b, and c)



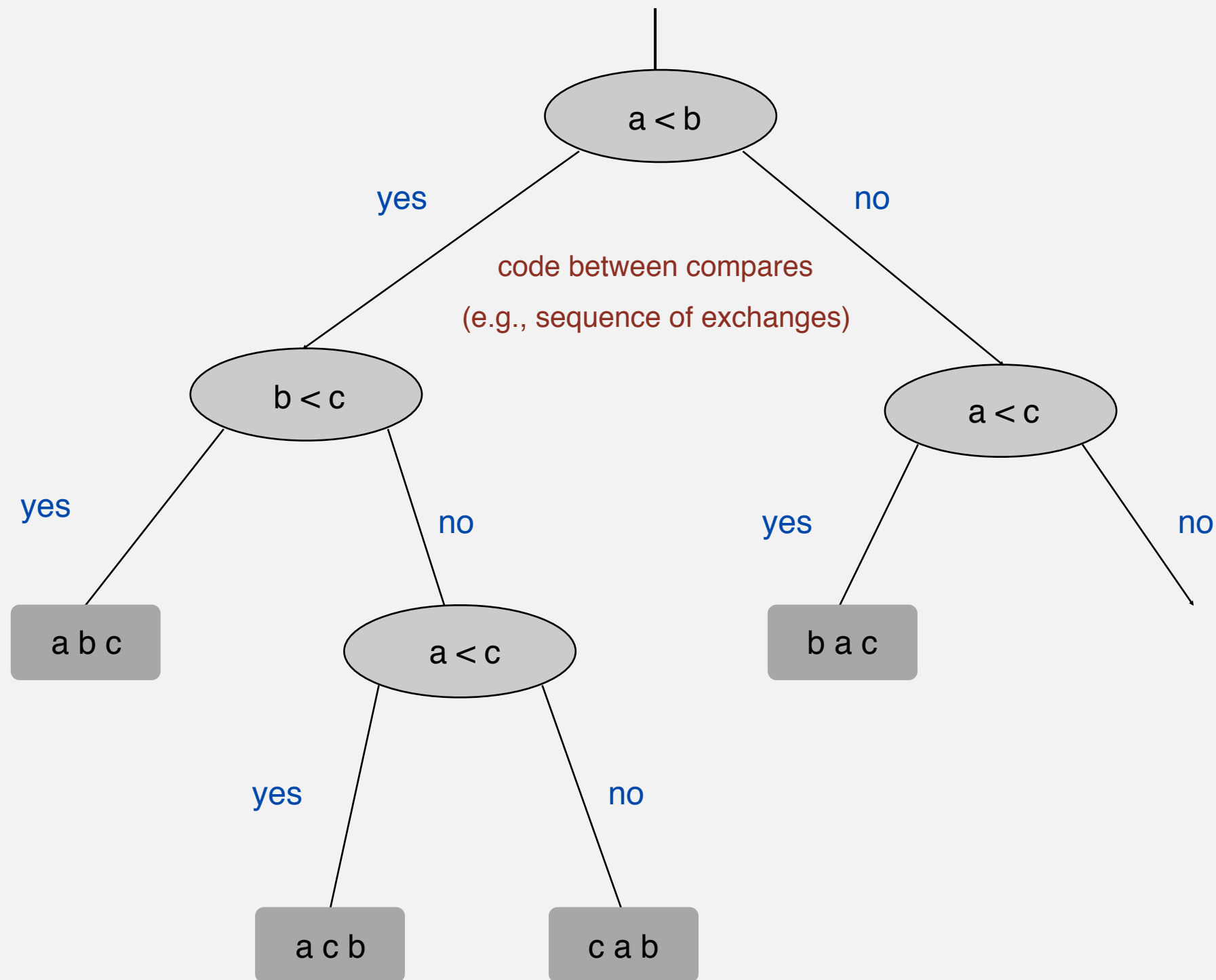
Decision tree (for 3 distinct elements a, b, and c)



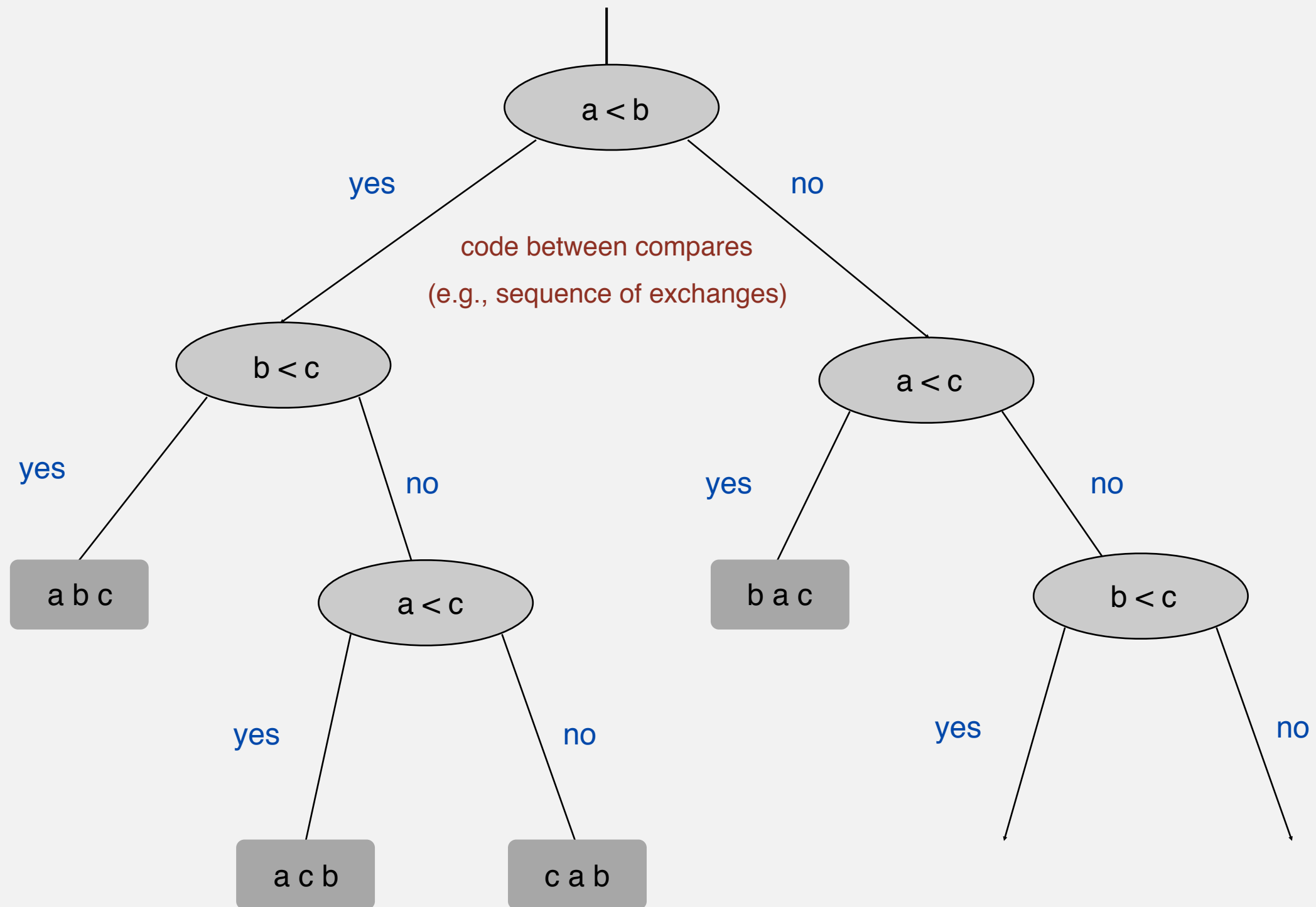
Decision tree (for 3 distinct elements a, b, and c)



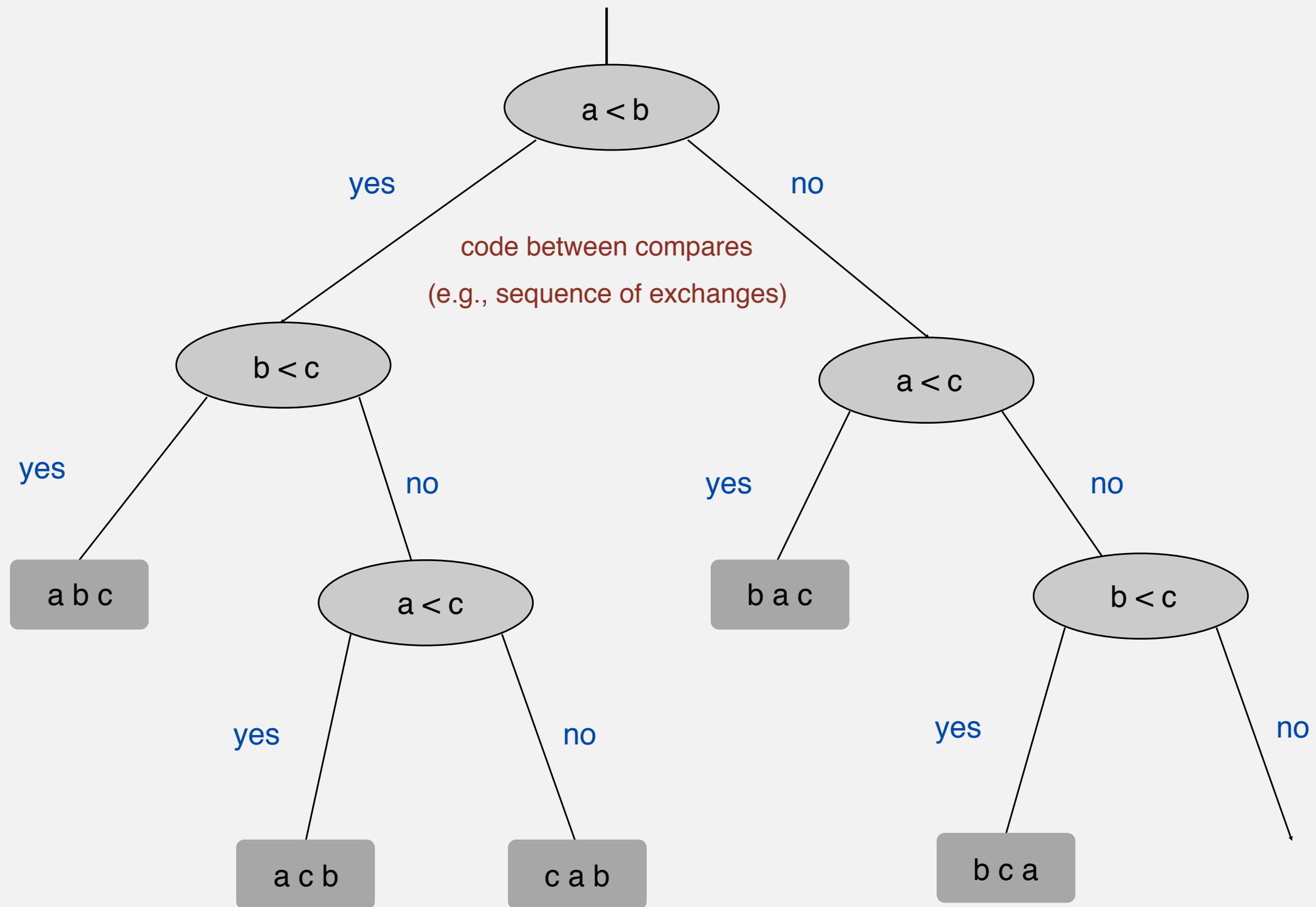
Decision tree (for 3 distinct elements a, b, and c)



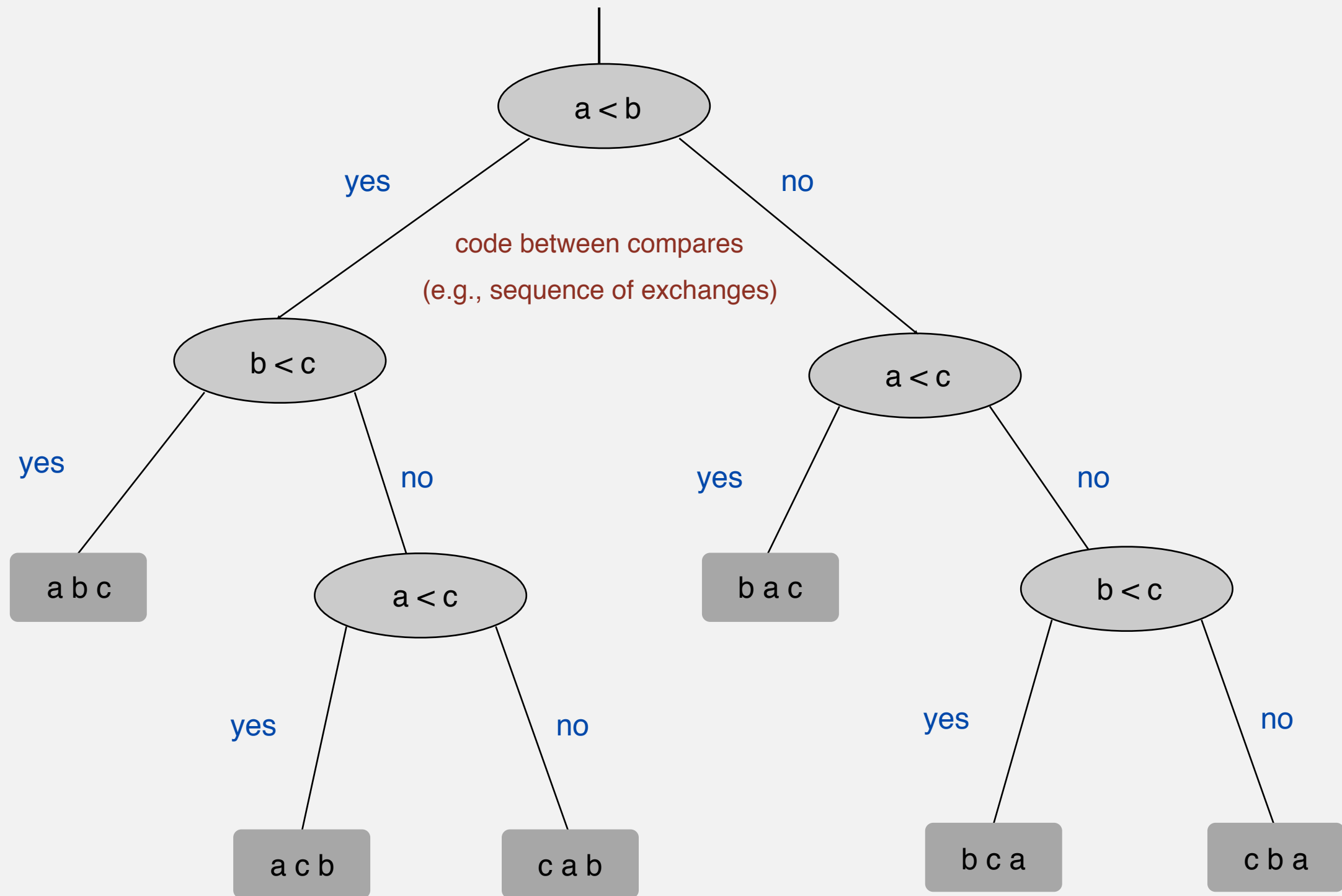
Decision tree (for 3 distinct elements a, b, and c)



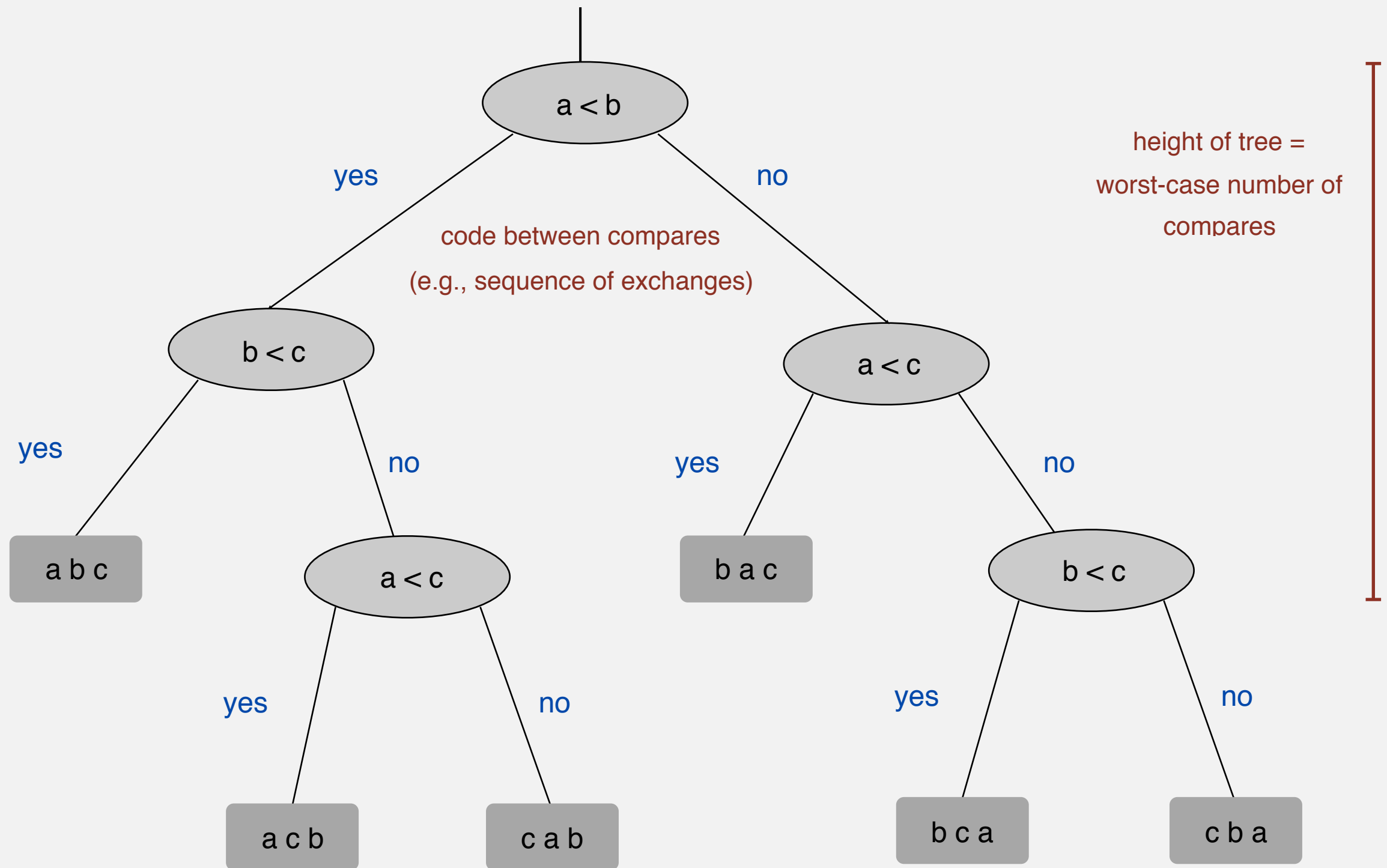
Decision tree (for 3 distinct elements a, b, and c)



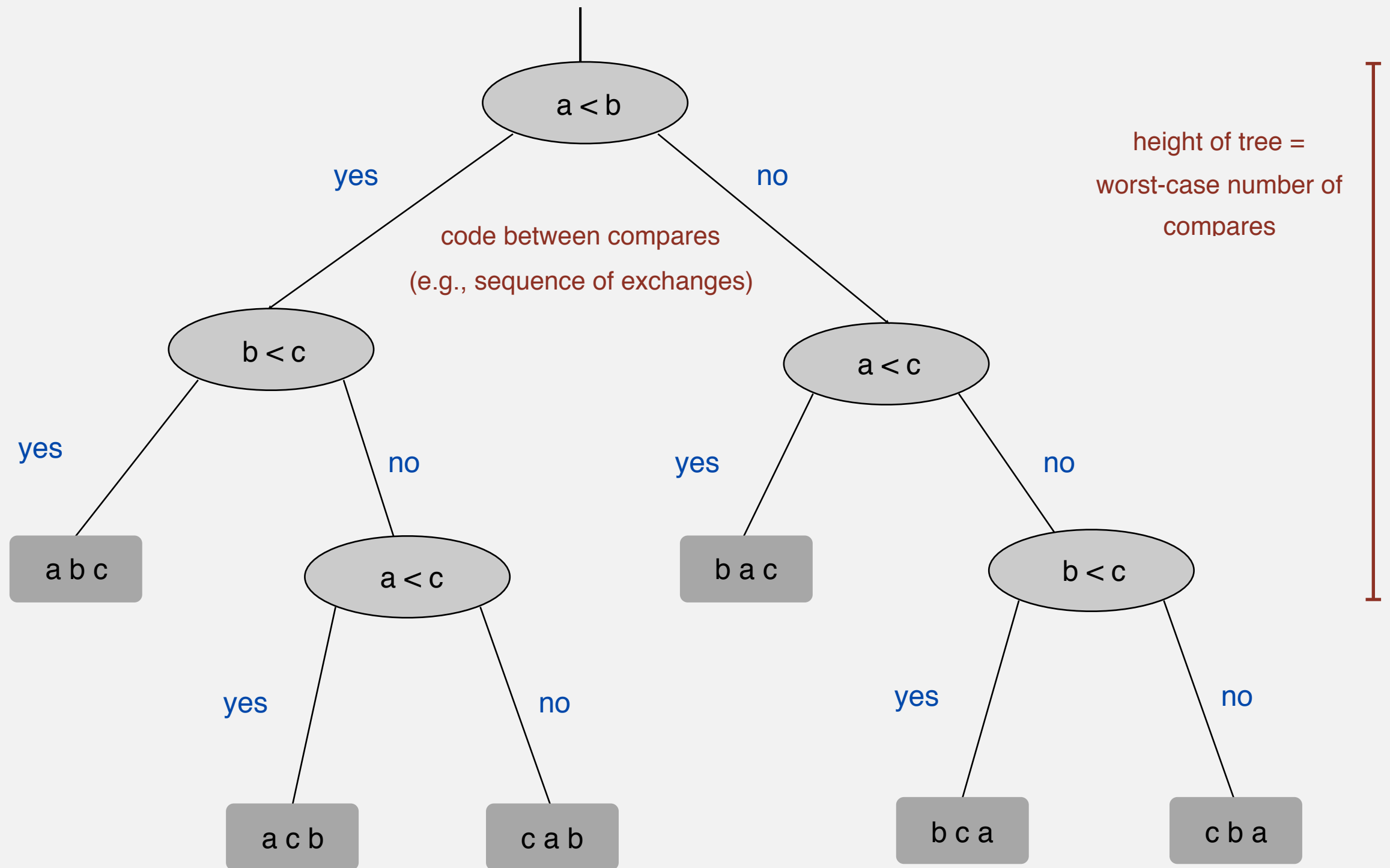
Decision tree (for 3 distinct elements a, b, and c)



Decision tree (for 3 distinct elements a, b, and c)



Decision tree (for 3 distinct elements a, b, and c)



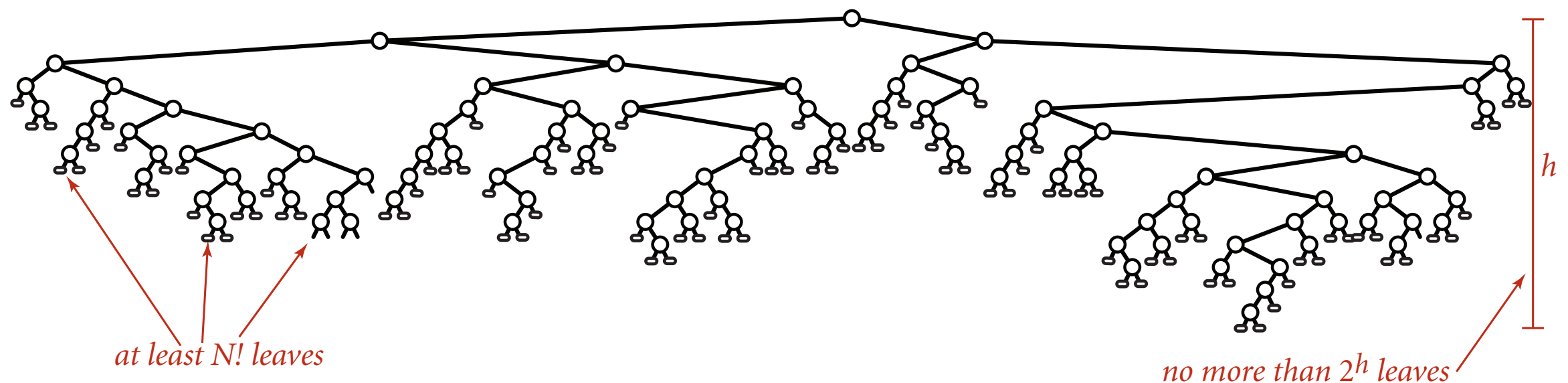
(at least) one leaf for each possible ordering

Compare-based lower bound for sorting

Proposition. Any compare-based sorting algorithm must use at least $\lg(N!) \sim N \lg N$ compares in the worst-case.

Pf.

- Assume array consists of N distinct values a_1 through a_N .
- Worst case dictated by **height** h of decision tree.
- Binary tree of height h has at most 2^h leaves.
- $N!$ different orderings \Rightarrow at least $N!$ leaves.



Compare-based lower bound for sorting

Proposition. Any compare-based sorting algorithm must use at least $\lg(N!) \sim N \lg N$ compares in the worst-case.

Pf.

- Assume array consists of N distinct values a_1 through a_N .
- Worst case dictated by **height** h of decision tree.
- Binary tree of height h has at most 2^h leaves.
- $N!$ different orderings \Rightarrow at least $N!$ leaves.

$$2^h \geq \# \text{ leaves} \geq N!$$

$$\Rightarrow h \geq \lg(N!) \sim N \lg N$$

↑
Stirling's formula

Complexity of sorting

Model of computation. Specify allowable operations.

Cost model. Focus on fundamental operations.

Upper bound. Cost guarantee provided by some algorithm for X .

Lower bound. Proven limit on cost guarantee of all algorithms for X .

Optimal algorithm. Algorithm with best cost guarantee for X .

Example: sorting.

- Model of computation: decision tree.
- Cost model: # compares.
- Upper bound: $\sim N \lg N$ from mergesort.
- Lower bound: $\sim N \lg N$.
- Optimal algorithm = mergesort.

First goal of algorithm design: optimal algorithms.

Complexity results in context

Other operations? Mergesort is optimal with respect to number of compares (e.g., but not to number of array accesses).

Space?

- Mergesort is **not optimal** with respect to space usage.
- Insertion sort, selection sort, and shellsort are space-optimal.

Challenge. Find an algorithm that is both time- and space-optimal. [stay tuned]

Lessons. Use theory as a guide.

Ex. Don't try to design sorting algorithm that guarantees $\frac{1}{2} N \lg N$ compares.

Complexity results in context (continued)

Lower bound may not hold if the algorithm has information about:

- The initial order of the input.
- The distribution of key values.
- The representation of the keys.

Partially-ordered arrays. Depending on the initial order of the input, we may not need $N \lg N$ compares.

↖ insertion sort requires only $N-1$ compares if input array is sorted

Duplicate keys. Depending on the input distribution of duplicates, we may not need $N \lg N$ compares.

↖ stay tuned for 3-way quicksort

Digital properties of keys. We can use digit/character compares instead of key compares for numbers and strings.

↖ stay tuned for radix sorts

- ▶ mergesort
- ▶ bottom-up mergesort
- ▶ sorting complexity
- ▶ **comparators**
- ▶ stability

Sort by artist name



	Name	Artist	Time	Album
12	<input checked="" type="checkbox"/> Let It Be	The Beatles	4:03	Let It Be
13	<input checked="" type="checkbox"/> Take My Breath Away	BERLIN	4:13	Top Gun – Soundtrack
14	<input checked="" type="checkbox"/> Circle Of Friends	Better Than Ezra	3:27	Empire Records
15	<input checked="" type="checkbox"/> Dancing With Myself	Billy Idol	4:43	Don't Stop
16	<input checked="" type="checkbox"/> Rebel Yell	Billy Idol	4:49	Rebel Yell
17	<input checked="" type="checkbox"/> Piano Man	Billy Joel	5:36	Greatest Hits Vol. 1
18	<input checked="" type="checkbox"/> Pressure	Billy Joel	3:16	Greatest Hits, Vol. II (1978 – 1985) (Disc 2)
19	<input checked="" type="checkbox"/> The Longest Time	Billy Joel	3:36	Greatest Hits, Vol. II (1978 – 1985) (Disc 2)
20	<input checked="" type="checkbox"/> Atomic	Blondie	3:50	Atomic: The Very Best Of Blondie
21	<input checked="" type="checkbox"/> Sunday Girl	Blondie	3:15	Atomic: The Very Best Of Blondie
22	<input checked="" type="checkbox"/> Call Me	Blondie	3:33	Atomic: The Very Best Of Blondie
23	<input checked="" type="checkbox"/> Dreaming	Blondie	3:06	Atomic: The Very Best Of Blondie
24	<input checked="" type="checkbox"/> Hurricane	Bob Dylan	8:32	Desire
25	<input checked="" type="checkbox"/> The Times They Are A-Changin'	Bob Dylan	3:17	Greatest Hits
26	<input checked="" type="checkbox"/> Livin' On A Prayer	Bon Jovi	4:11	Cross Road
27	<input checked="" type="checkbox"/> Beds Of Roses	Bon Jovi	6:35	Cross Road
28	<input checked="" type="checkbox"/> Runaway	Bon Jovi	3:53	Cross Road
29	<input checked="" type="checkbox"/> Rasputin (Extended Mix)	Boney M	5:50	Greatest Hits
30	<input checked="" type="checkbox"/> Have You Ever Seen The Rain	Bonnie Tyler	4:10	Faster Than The Speed Of Night
31	<input checked="" type="checkbox"/> Total Eclipse Of The Heart	Bonnie Tyler	7:02	Faster Than The Speed Of Night
32	<input checked="" type="checkbox"/> Straight From The Heart	Bonnie Tyler	3:41	Faster Than The Speed Of Night
33	<input checked="" type="checkbox"/> Holding Out For A Hero	Bonny Tyler	5:49	Meat Loaf And Friends
34	<input checked="" type="checkbox"/> Dancing In The Dark	Bruce Springsteen	4:05	Born In The U.S.A.
35	<input checked="" type="checkbox"/> Thunder Road	Bruce Springsteen	4:51	Born To Run
36	<input checked="" type="checkbox"/> Born To Run	Bruce Springsteen	4:30	Born To Run
37	<input checked="" type="checkbox"/> Jungleland	Bruce Springsteen	9:34	Born To Run
38	<input checked="" type="checkbox"/> Turn Turn Turn (To Simple)	The Byrds	3:57	Forrest Gump The Soundtrack (Disc 2)

Sort by song name



	Name	Artist	Time	Album
1	<input checked="" type="checkbox"/> Alive	Pearl Jam	5:41	Ten
2	<input checked="" type="checkbox"/> All Over The World	Pixies	5:27	Bossanova
3	<input checked="" type="checkbox"/> All Through The Night	Cyndi Lauper	4:30	She's So Unusual
4	<input checked="" type="checkbox"/> Allison Road	Gin Blossoms	3:19	New Miserable Experience
5	<input checked="" type="checkbox"/> Ama, Ama, Ama Y Ensancha El ...	Extremoduro	2:34	Daltoya (1992)
6	<input checked="" type="checkbox"/> And We Danced	Hooters	3:50	Nervous Night
7	<input checked="" type="checkbox"/> As I Lay Me Down	Sophie B. Hawkins	4:09	Whaler
8	<input checked="" type="checkbox"/> Atomic	Blondie	3:50	Atomic: The Very Best Of Blondie
9	<input checked="" type="checkbox"/> Automatic Lover	Jay-Jay Johanson	4:19	Antenna
10	<input checked="" type="checkbox"/> Baba O'Riley	The Who	5:01	Who's Better, Who's Best
11	<input checked="" type="checkbox"/> Beautiful Life	Ace Of Base	3:40	The Bridge
12	<input checked="" type="checkbox"/> Beds Of Roses	Bon Jovi	6:35	Cross Road
13	<input checked="" type="checkbox"/> Black	Pearl Jam	5:44	Ten
14	<input checked="" type="checkbox"/> Bleed American	Jimmy Eat World	3:04	Bleed American
15	<input checked="" type="checkbox"/> Borderline	Madonna	4:00	The Immaculate Collection
16	<input checked="" type="checkbox"/> Born To Run	Bruce Springsteen	4:30	Born To Run
17	<input checked="" type="checkbox"/> Both Sides Of The Story	Phil Collins	6:43	Both Sides
18	<input checked="" type="checkbox"/> Bouncing Around The Room	Phish	4:09	A Live One (Disc 1)
19	<input checked="" type="checkbox"/> Boys Don't Cry	The Cure	2:35	Staring At The Sea: The Singles 1979-1985
20	<input checked="" type="checkbox"/> Brat	Green Day	1:43	Insomniac
21	<input checked="" type="checkbox"/> Breakdown	Deerheart	3:40	Deerheart
22	<input checked="" type="checkbox"/> Bring Me To Life (Kevin Roen Mix)	Evanescence Vs. Pa...	9:46	
23	<input checked="" type="checkbox"/> Californication	Red Hot Chili Pepp...	1:40	
24	<input checked="" type="checkbox"/> Call Me	Blondie	3:33	Atomic: The Very Best Of Blondie
25	<input checked="" type="checkbox"/> Can't Get You Out Of My Head	Kylie Minogue	3:50	Fever
26	<input checked="" type="checkbox"/> Celebration	Kool & The Gang	3:45	Time Life Music Sounds Of The Seventies - C
27	<input checked="" type="checkbox"/> Chateau Chateau	Suburban Station	5:11	Bombay Dreams

Natural order

Comparable interface: sort uses 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

Generalized compare

Comparable interface: sort uses type's **natural order**.

Problem 1. May want to use a non-natural order.

Problem 2. Desired data type may not come with a "natural" order.

Ex. Sort strings by:

- Natural order.
- Case insensitive.
- Spanish.
- British phone book.

Now is the time

is Now the time

café cafetero cuarto churro nube ñoño

McKinley Mackintosh

pre-1994 order for
digraphs ch and ll and rr



```
String[] a;  
...  
Arrays.sort(a);  
Arrays.sort(a, String.CASE_INSENSITIVE_ORDER);  
Arrays.sort(a, Collator.getInstance(Locale.SPANISH));
```

import java.text.Collator;

Comparators

Solution. Use Java's Comparator interface.

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

Remark. `compare()` must implement a total order like `compareTo()`.

Advantages. Decouples the definition of the data type from the definition of what it means to compare two objects of that type.

- Can add any number of new orders to a data type.
- Can add an order to a library data type with no natural order.

Comparator example

Reverse order. Sort an array of strings in reverse order.

comparator implementation

```
public class ReverseOrder implements Comparator<String>
{
    public int compare(String a, String b)
    {
        return b.compareTo(a) ;
    }
}
```

client

```
...
Arrays.sort(a, new ReverseOrder()) ;
...
```


Sort implementation with comparators

To support comparators in our sort implementations:

- Use object instead of Comparable.
- Pass Comparator to sort() and less().
- Use it in less().

Ex. Insertion sort.

```
public static void sort(Object[] a, Comparator comparator)
{
    int N = a.length;
    for (int i = 0; i < N; i++)
        for (int j = i; j > 0 && less(comparator, a[j], a[j-1]); j--)
            exch(a, j, j-1);
}

private static boolean less(Comparator c, Object v, Object w)
{ return c.compare(v, w) < 0; }

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

Generalized compare

Comparators enable multiple sorts of a single array (by different keys).

Ex. Sort students by name **or** by section.

```
Arrays.sort(students, Student.BY_NAME);  
Arrays.sort(students, Student.BY_SECT);
```

sort by name



Andrews	3	A	664-480-0023	097 Little
Battle	4	C	874-088-1212	121 Whitman
Chen	2	A	991-878-4944	308 Blair
Fox	1	A	884-232-5341	11 Dickinson
Furia	3	A	766-093-9873	101 Brown
Gazsi	4	B	665-303-0266	22 Brown
Kanaga	3	B	898-122-9643	22 Brown
Rohde	3	A	232-343-5555	343 Forbes

sort by section



Fox	1	A	884-232-5341	11 Dickinson
Chen	2	A	991-878-4944	308 Blair
Andrews	3	A	664-480-0023	097 Little
Furia	3	A	766-093-9873	101 Brown
Kanaga	3	B	898-122-9643	22 Brown
Rohde	3	A	232-343-5555	343 Forbes
Battle	4	C	874-088-1212	121 Whitman
Gazsi	4	B	665-303-0266	22 Brown

Generalized compare

Ex. Enable sorting students by name or by section.

```
public class Student
{
    public static final Comparator<Student> BY_NAME = new ByName();
    public static final Comparator<Student> BY_SECT = new BySect();

    private final String name;
    private final int section;
    ...
    private static class ByName implements Comparator<Student>
    {
        public int compare(Student a, Student b)
        { return a.name.compareTo(b.name); }
    }

    private static class BySect implements Comparator<Student>
    {
        public int compare(Student a, Student b)
        { return a.section - b.section; }
    }
}
```

use this trick only if no danger of overflow

- ▶ mergesort
- ▶ bottom-up mergesort
- ▶ sorting complexity
- ▶ comparators
- ▶ **stability**

Generalized compare problem

A typical application. First, sort by name; then sort by section.

`Arrays.sort(students, Student.BY_NAME);`



Andrews	3	A	664-480-0023	097 Little
Battle	4	C	874-088-1212	121 Whitman
Chen	2	A	991-878-4944	308 Blair
Fox	1	A	884-232-5341	11 Dickinson
Furia	3	A	766-093-9873	101 Brown
Gazsi	4	B	665-303-0266	22 Brown
Kanaga	3	B	898-122-9643	22 Brown
Rohde	3	A	232-343-5555	343 Forbes

`Arrays.sort(students, Student.BY_SECT);`



Fox	1	A	884-232-5341	11 Dickinson
Chen	2	A	991-878-4944	308 Blair
Kanaga	3	B	898-122-9643	22 Brown
Andrews	3	A	664-480-0023	097 Little
Furia	3	A	766-093-9873	101 Brown
Rohde	3	A	232-343-5555	343 Forbes
Battle	4	C	874-088-1212	121 Whitman
Gazsi	4	B	665-303-0266	22 Brown

@#%&@!! Students in section 3 no longer in order by name.

A **stable** sort preserves the relative order of records with equal keys.

Sorting challenge 5

Q. Which sorts are stable?

Insertion sort? Selection sort? Shellsort? Mergesort?

sorted by time	sorted by location (not stable)	sorted by location (stable)
Chicago 09:00:00	Chicago 09:25:52	Chicago 09:00:00
Phoenix 09:00:03	Chicago 09:03:13	Chicago 09:00:59
Houston 09:00:13	Chicago 09:21:05	Chicago 09:03:13
Chicago 09:00:59	Chicago 09:19:46	Chicago 09:19:32
Houston 09:01:10	Chicago 09:19:32	Chicago 09:19:46
Chicago 09:03:13	Chicago 09:00:00	Chicago 09:21:05
Seattle 09:10:11	Chicago 09:35:21	Chicago 09:25:52
Seattle 09:10:25	Chicago 09:00:59	Chicago 09:35:21
Phoenix 09:14:25	Houston 09:01:10	Houston 09:00:13
Chicago 09:19:32	Houston 09:00:13	Houston 09:01:10
Chicago 09:19:46	Phoenix 09:37:44	Phoenix 09:00:03
Chicago 09:21:05	Phoenix 09:00:03	Phoenix 09:14:25
Seattle 09:22:43	Phoenix 09:14:25	Phoenix 09:37:44
Seattle 09:22:54	Seattle 09:10:25	Seattle 09:10:11
Chicago 09:25:52	Seattle 09:36:14	Seattle 09:10:25
Chicago 09:35:21	Seattle 09:22:43	Seattle 09:22:43
Seattle 09:36:14	Seattle 09:10:11	Seattle 09:22:54
Phoenix 09:37:44	Seattle 09:22:54	Seattle 09:36:14

Stability when sorting on a second key

Sorting challenge 5A

Q. Is insertion sort 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

A. Yes, equal elements never move past each other.

Sorting challenge 5B

Q. Is selection sort 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

A. No, long-distance exchange might move left element to the right of some equal element.

Sorting challenge 5C

Q. Is shellsort stable?

```
public class Shell
{
    public static void sort(Comparable[] a)
    {
        int N = a.length;
        int h = 1;
        while (h < N/3) h = 3*h + 1;
        while (h >= 1)
        {
            for (int i = h; i < N; i++)
            {
                for (int j = i; j > h && less(a[j], a[j-h]); j -= h)
                    exch(a, j, j-h);
            }
            h = h/3;
        }
    }
}
```

h	0	1	2	3	4
	B ₁	B ₂	B ₃	B ₄	A ₁
4	A ₁	B ₂	B ₃	B ₄	B ₁
1	A ₁	B ₂	B ₃	B ₄	B ₁
	A ₁	B ₂	B ₃	B ₄	B ₁

A. No. Long-distance exchanges.

Sorting challenge 5D

Q. Is mergesort stable?

```
public class Merge
{
    private static Comparable[] aux;
    private static void merge(Comparable[] a, int lo, int mid, int hi)
    { /* as before */ }

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

    public static void sort(Comparable[] a)
    { /* as before */ }
}
```

Sorting challenge 5D

Q. Is mergesort stable?

	a[i]															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
sz = 1	M	E	R	G	E	S	O	R	T	E	X	A	M	P	L	E
merge(a, 0, 0, 1)	E	M	R	G	E	S	O	R	T	E	X	A	M	P	L	E
merge(a, 2, 2, 3)	E	M	G	R	E	S	O	R	T	E	X	A	M	P	L	E
merge(a, 4, 4, 5)	E	M	G	R	E	S	O	R	T	E	X	A	M	P	L	E
merge(a, 6, 6, 7)	E	M	G	R	E	S	O	R	T	E	X	A	M	P	L	E
merge(a, 8, 8, 9)	E	M	G	R	E	S	O	R	E	T	X	A	M	P	L	E
merge(a, 10, 10, 11)	E	M	G	R	E	S	O	R	E	T	A	X	M	P	L	E
merge(a, 12, 12, 13)	E	M	G	R	E	S	O	R	E	T	A	X	M	P	L	E
merge(a, 14, 14, 15)	E	M	G	R	E	S	O	R	E	T	A	X	M	P	E	L
sz = 2																
merge(a, 0, 1, 3)	E	G	M	R	E	S	O	R	E	T	A	X	M	P	E	L
merge(a, 4, 5, 7)	E	G	M	R	E	O	R	S	E	T	A	X	M	P	E	L
merge(a, 8, 9, 11)	E	G	M	R	E	O	R	S	A	E	T	X	M	P	E	L
merge(a, 12, 13, 15)	E	G	M	R	E	O	R	S	A	E	T	X	E	L	M	P
sz = 4																
merge(a, 0, 3, 7)	E	E	G	M	O	R	R	S	A	E	T	X	E	L	M	P
merge(a, 8, 11, 15)	E	E	G	M	O	R	R	S	A	E	E	L	M	P	T	X
sz = 8																
merge(a, 0, 7, 15)	A	E	E	E	E	G	L	M	M	O	P	R	R	S	T	X

Trace of merge results for bottom-up mergesort

A. Yes, if merge is stable.

Sorting challenge 5D (continued)

Q. Is merge stable?

```
private static void merge(Comparable[] a, int lo, int mid, int hi)
{
    for (int k = lo; k <= hi; k++)
        aux[k] = a[k];

    int i = lo, j = mid+1;
    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++];
    }
}
```

0	1	2	3	4	5	6	7	8	9	10
A ₁	A ₂	A ₃	B	D	A ₄	A ₅	C	E	F	G

A. Yes, if implemented carefully (take from left subarray if equal).

Sorting challenge 5 (summary)

Q. Which sorts are stable ?

Yes. Insertion sort, mergesort.

No. Selection sort, shellsort.

Note. Need to carefully check code ("less than" vs "less than or equal to").