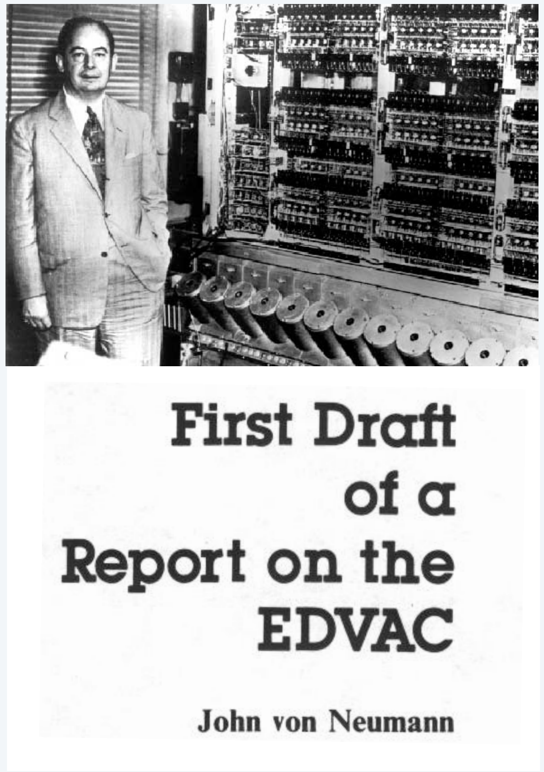
Today:

- merge sort → analyzing runtime of recursive algs
- proving a lower bound on sorting runtime
- counting inversions
- survey

Mergesort

- Recursively sort left half.
- Recursively sort right half.
- Merge two halves to make sorted whole.



input

A	L	G	O	R	I	T	H	M	S

sort left half

A	G	L	O	R	I	T	H	M	S

n/2 log n

sort right half

A	G	L	O	R	H	I	M	S	T

merge results

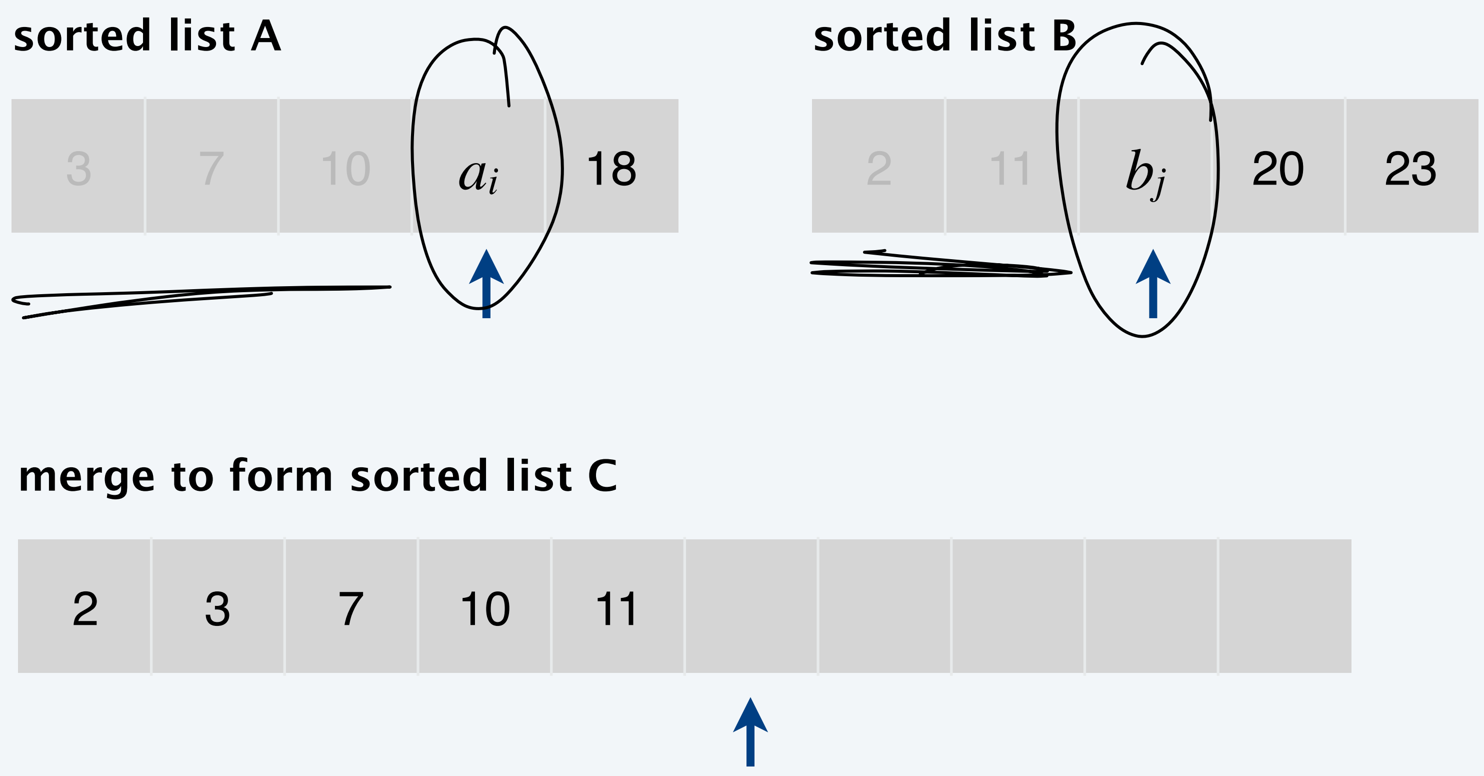
A	G	H	I	L	M	O	R	S	T

n

Merging

Goal. Combine two sorted lists A and B into a sorted whole C.

- Scan A and B from left to right.
- Compare a_i and b_j.
- If $a_i \le b_j$, append a_i to C (no larger than any remaining element in B).
- If $a_i > b_j$, append b_j to C (smaller than every remaining element in A).



Merge demo

Given two sorted lists A and B, merge into sorted list C.

sorted list A

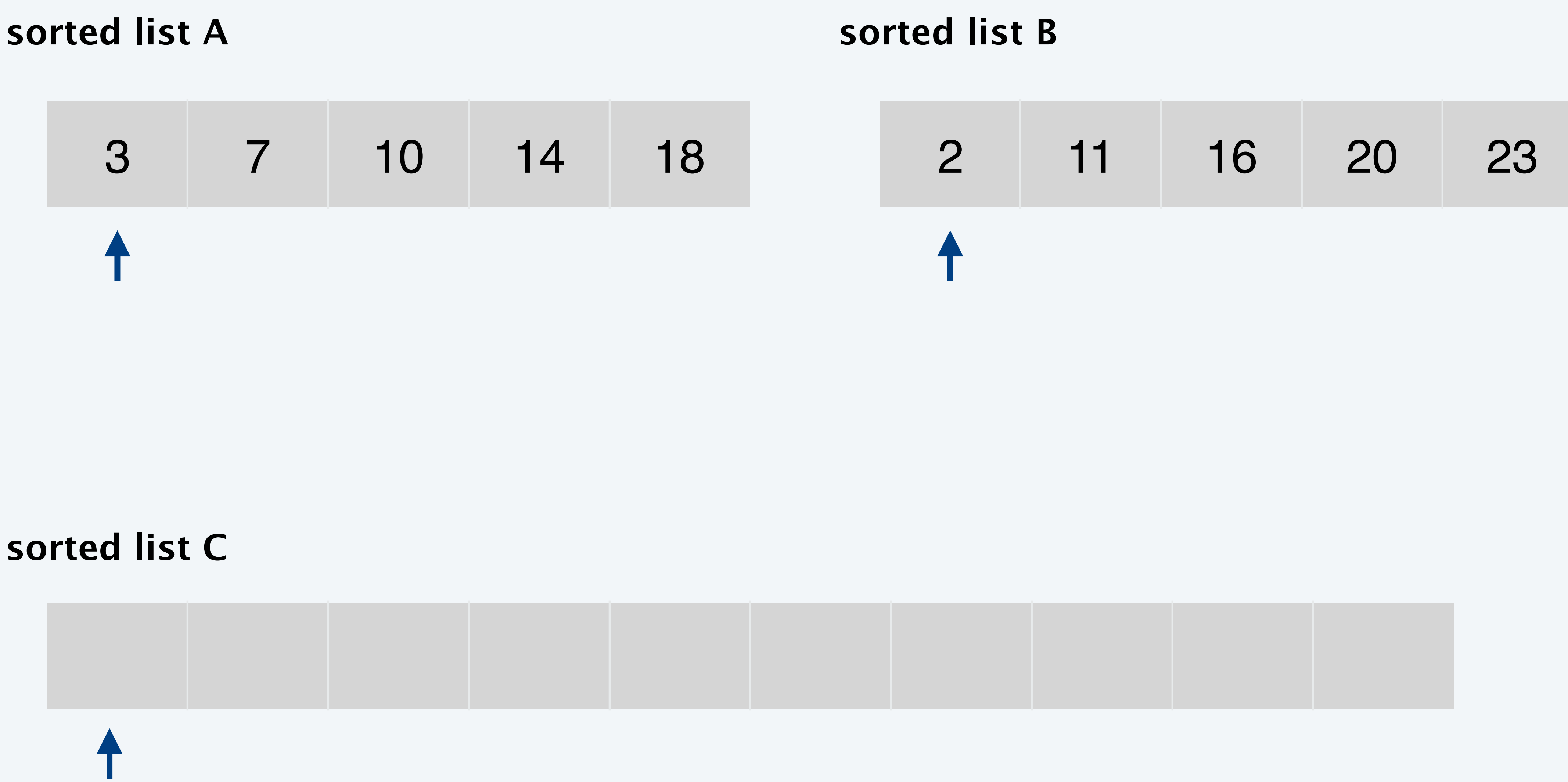
3	7	10	14	18

sorted list B

2	11	16	20	23

Merge demo

Given two sorted lists A and B, merge into sorted list C.



Merge demo

Given two sorted lists A and B, merge into sorted list C.

sorted list A

3	7	10	14	18

sorted list B

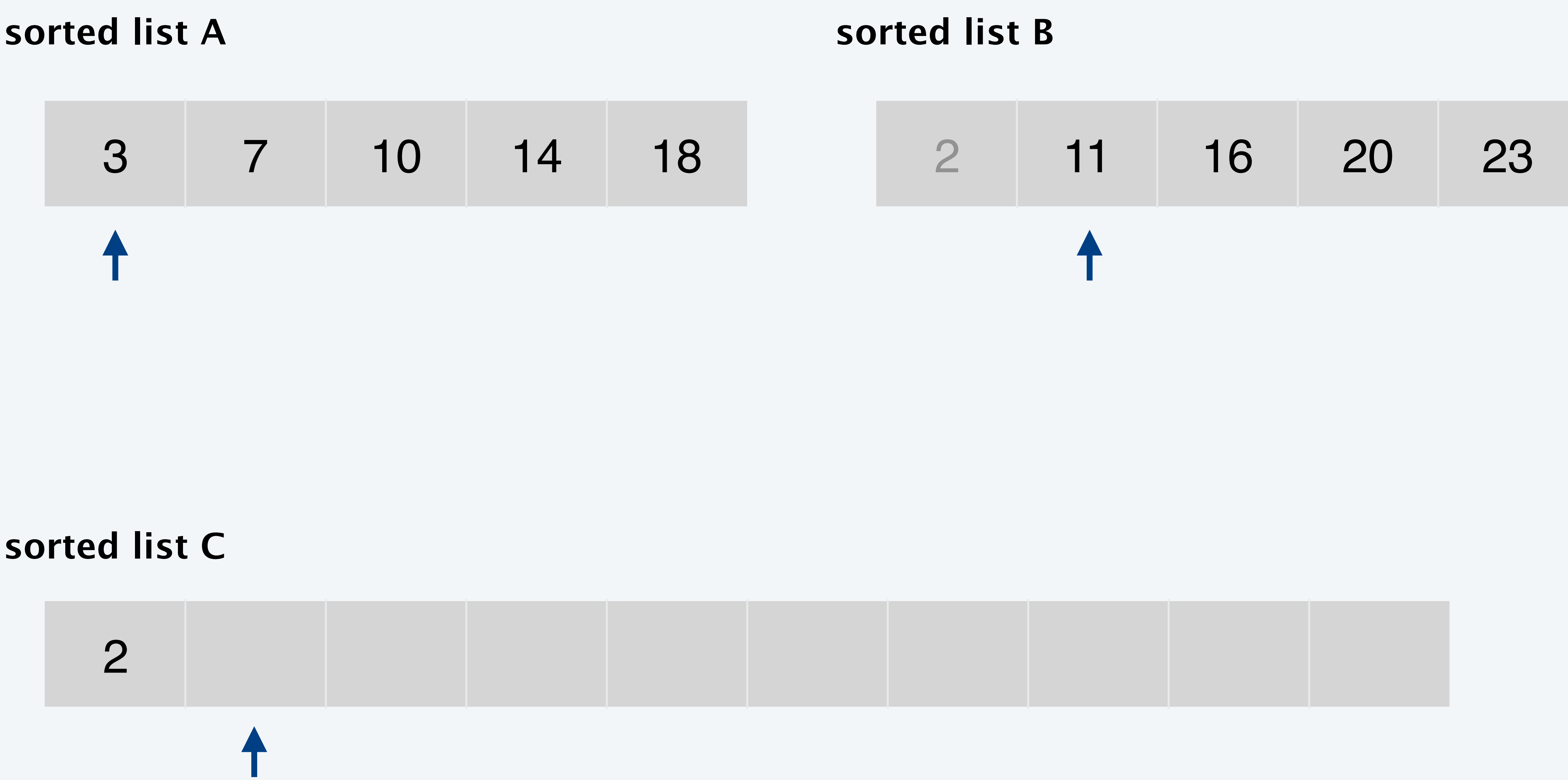
2	11	16	20	23

compare minimum entry in each list: copy 2

sorted list C

Merge demo

Given two sorted lists A and B, merge into sorted list C.

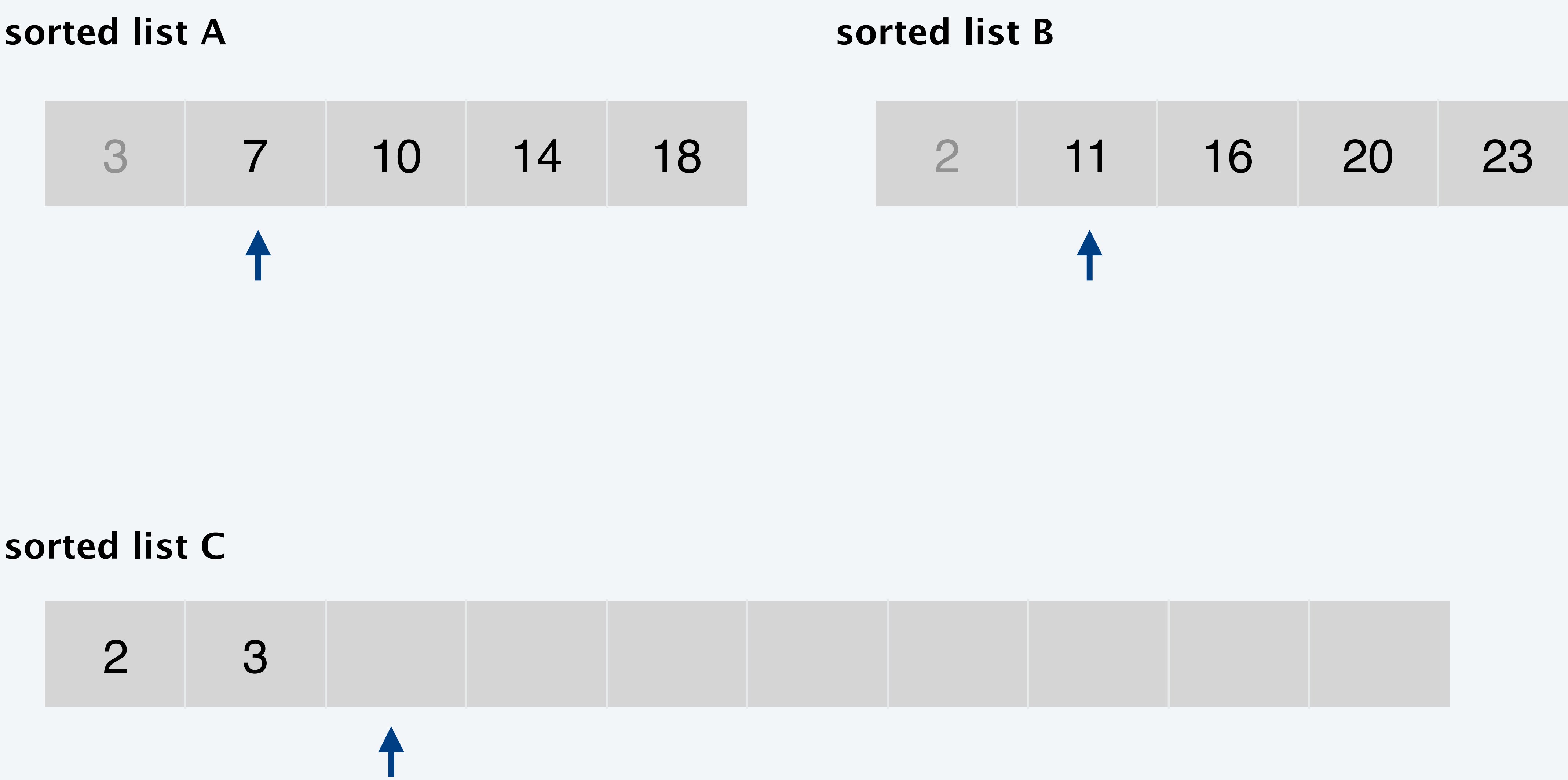


Merge demo

Given two sorted lists A and B, merge into sorted list C.

Merge demo

Given two sorted lists A and B, merge into sorted list C.



Merge demo

Given two sorted lists A and B, merge into sorted list C.

sorted list A

3	7	10	14	18

sorted list B

2	11	16	20	23

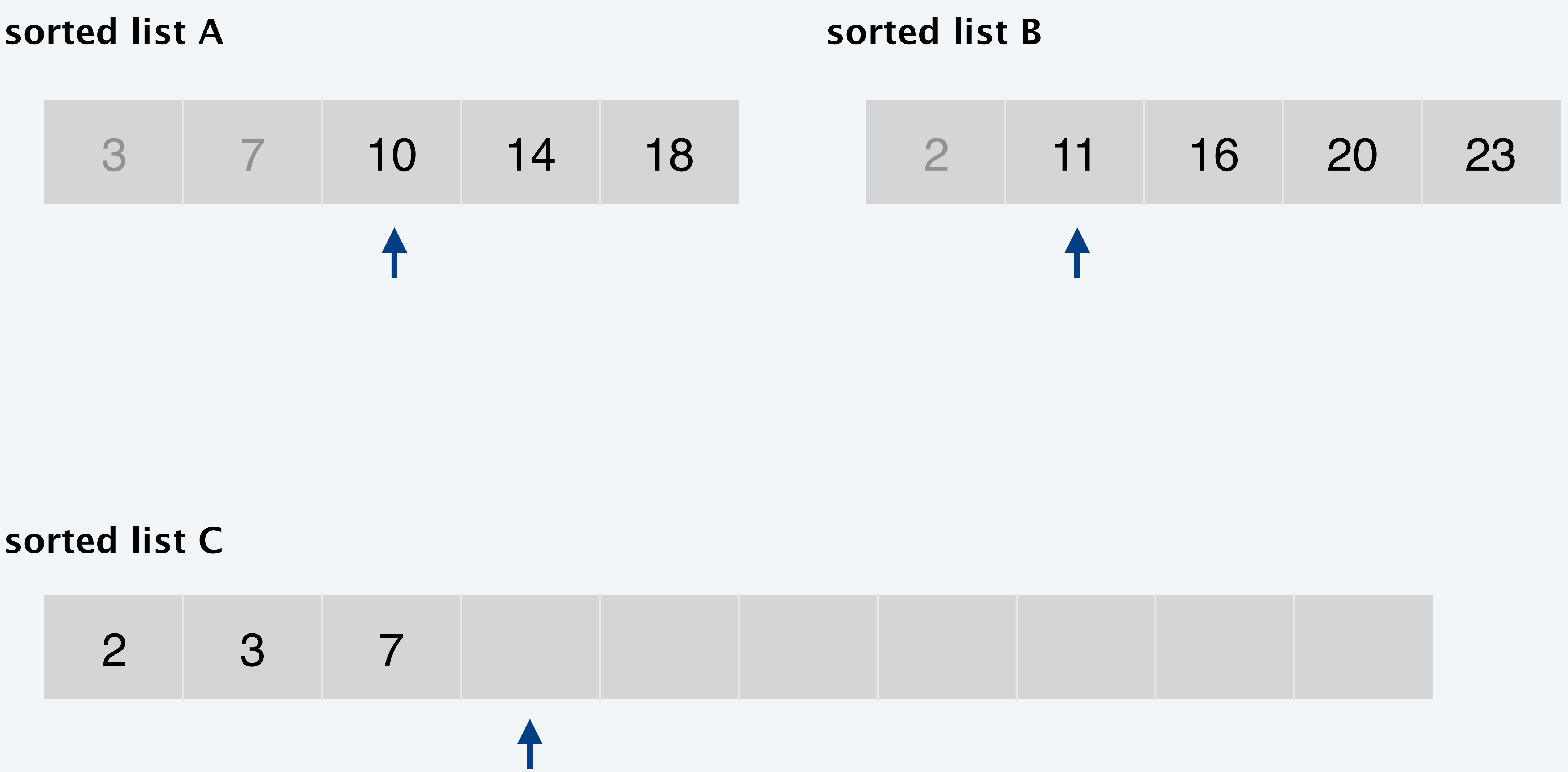
compare minimum entry in each list: copy 7

sorted list C

2	3								

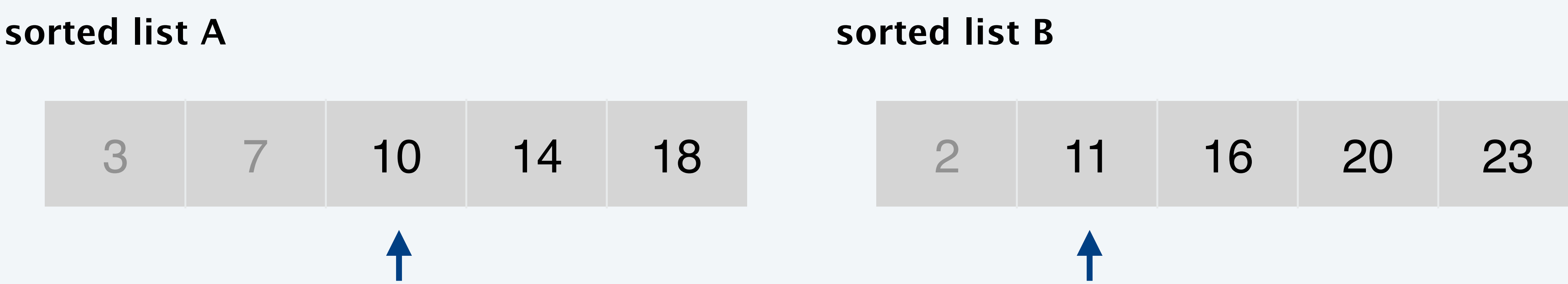
Merge demo

Given two sorted lists A and B, merge into sorted list C.



Merge demo

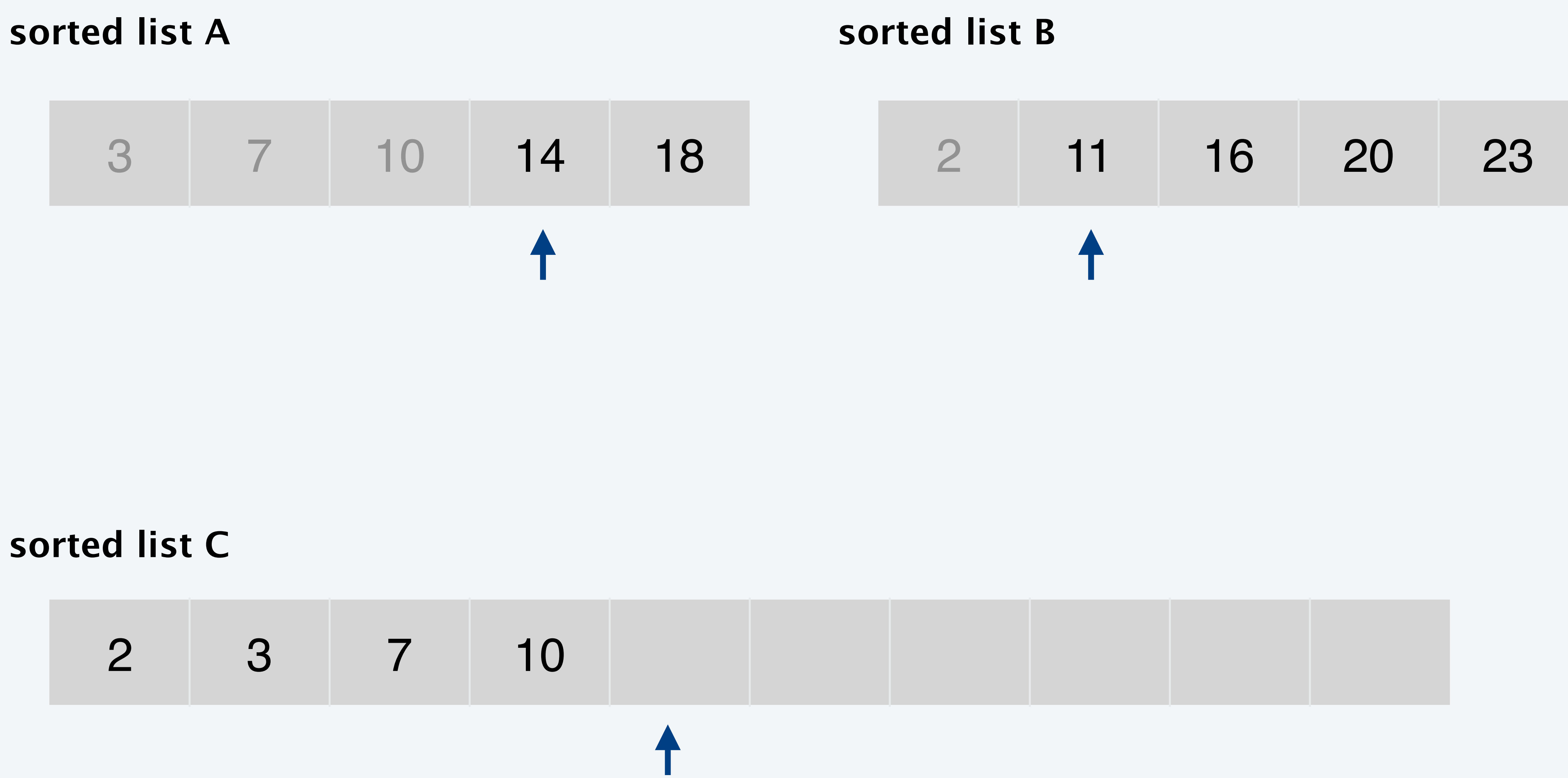
Given two sorted lists A and B, merge into sorted list C.



compare minimum entry in each list: copy 10

Merge demo

Given two sorted lists A and B, merge into sorted list C.



Merge demo

Given two sorted lists A and B, merge into sorted list C.

sorted list A

3	7	10	14	18

sorted list B

2	11	16	20	23

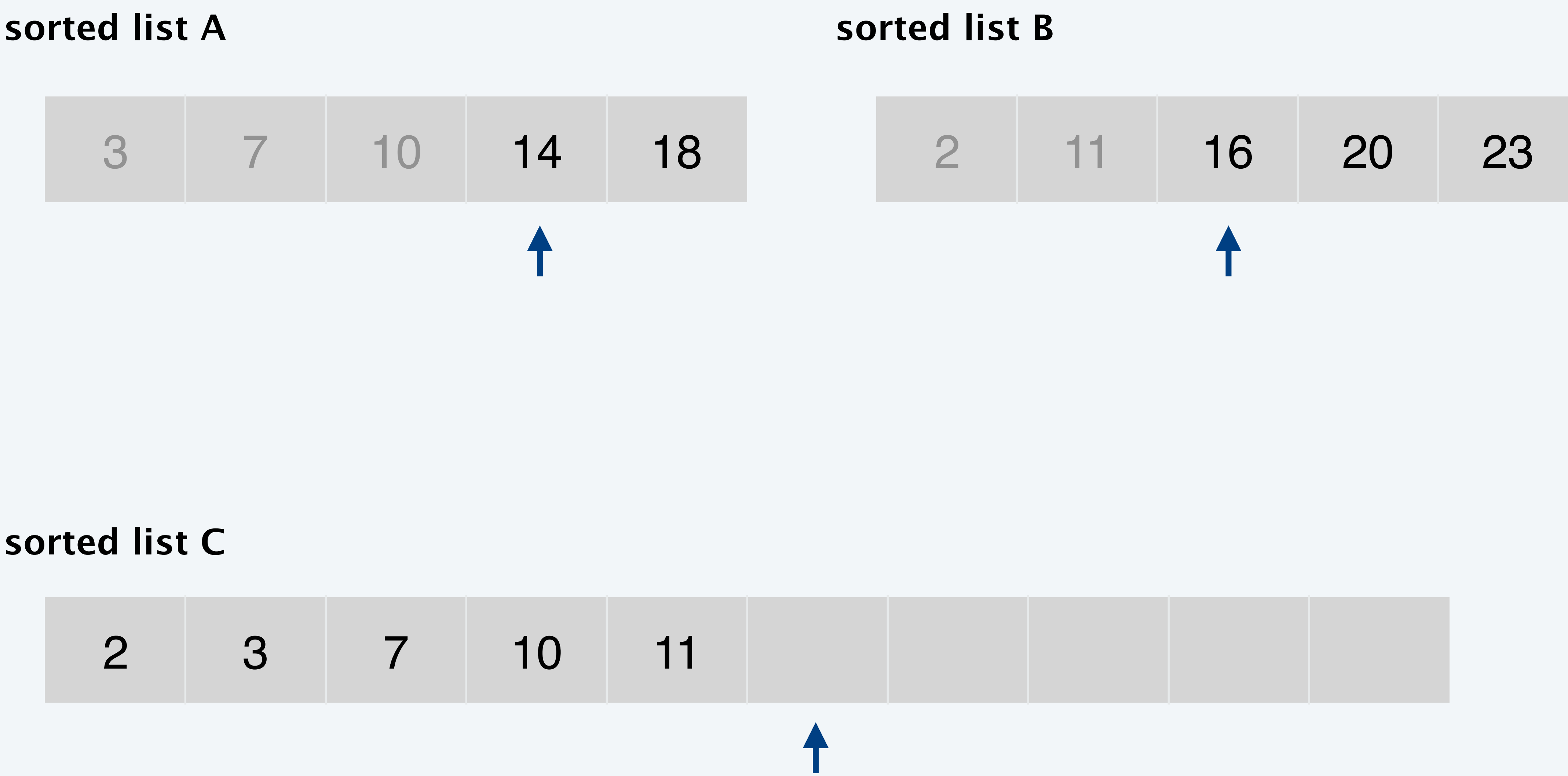
compare minimum entry in each list: copy 11

sorted list C

2	3	7	10						

Merge demo

Given two sorted lists A and B, merge into sorted list C.



Merge demo

Given two sorted lists A and B, merge into sorted list C.

Merge demo

Given two sorted lists A and B, merge into sorted list C.

sorted list A

3	7	10	14	18

sorted list B

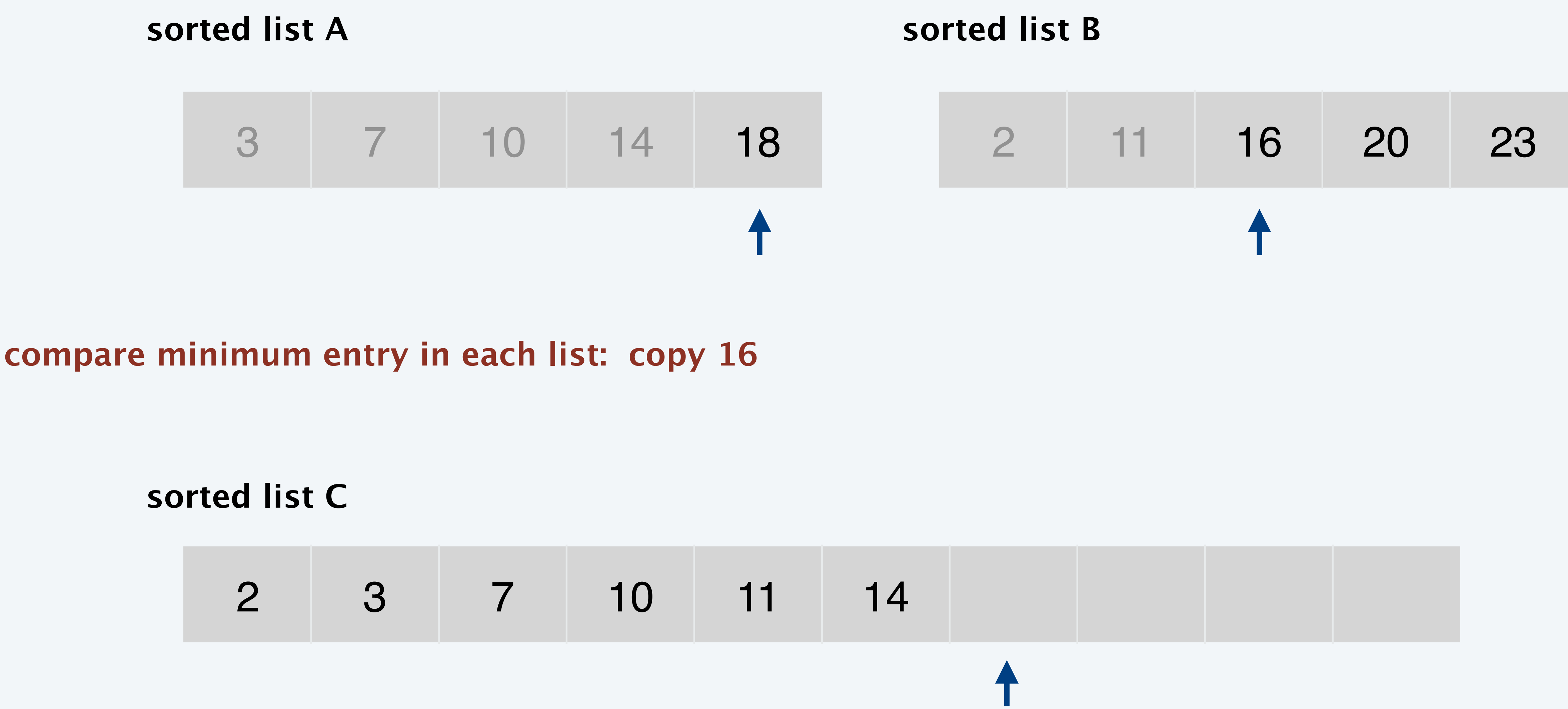
2	11	16	20	23

sorted list C

2	3	7	10	11	14				

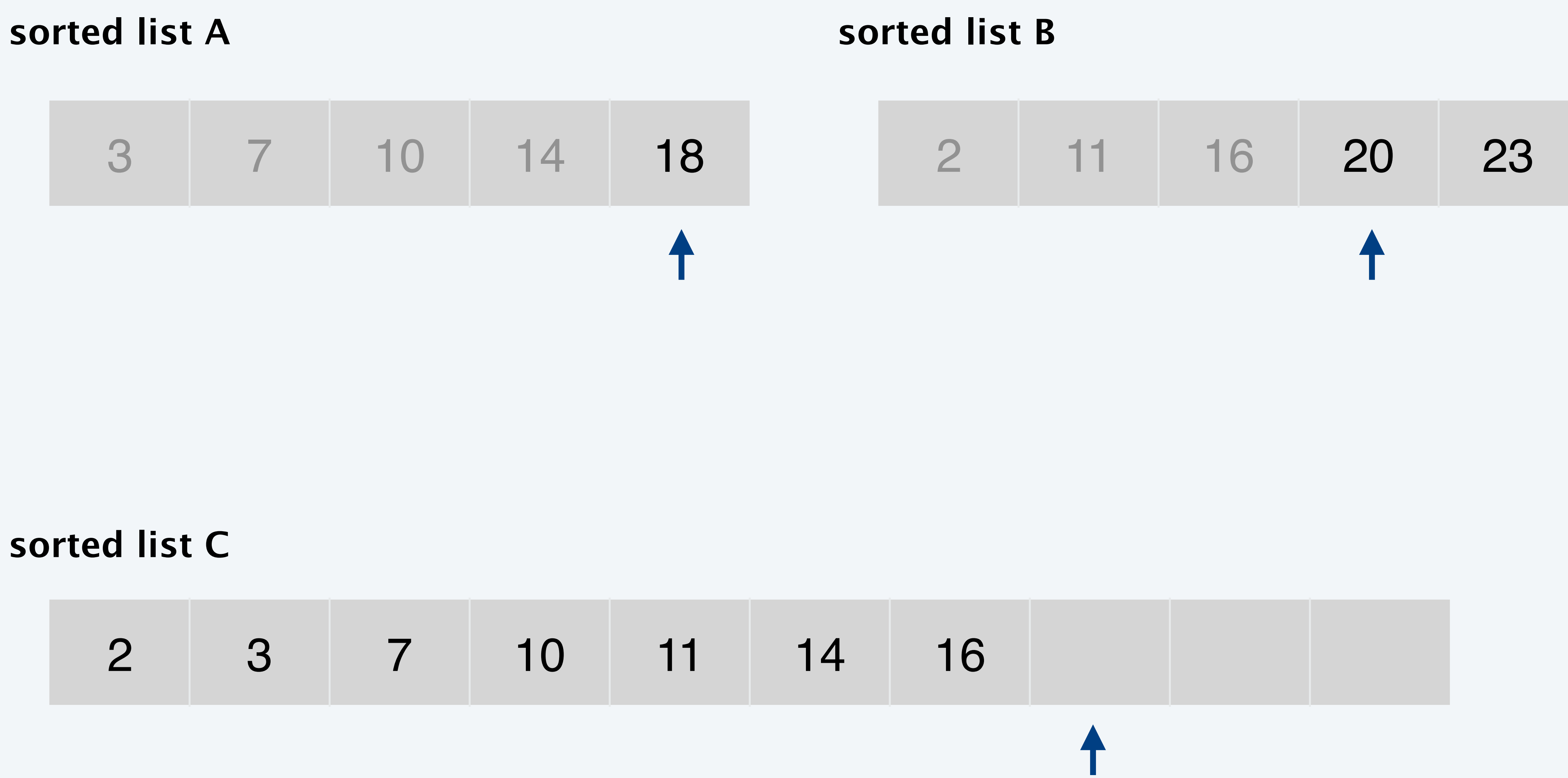
Merge demo

Given two sorted lists A and B, merge into sorted list C.



Merge demo

Given two sorted lists A and B, merge into sorted list C.



Merge demo

Given two sorted lists A and B, merge into sorted list C.

sorted list A

3	7	10	14	18

sorted list B

2	11	16	20	23

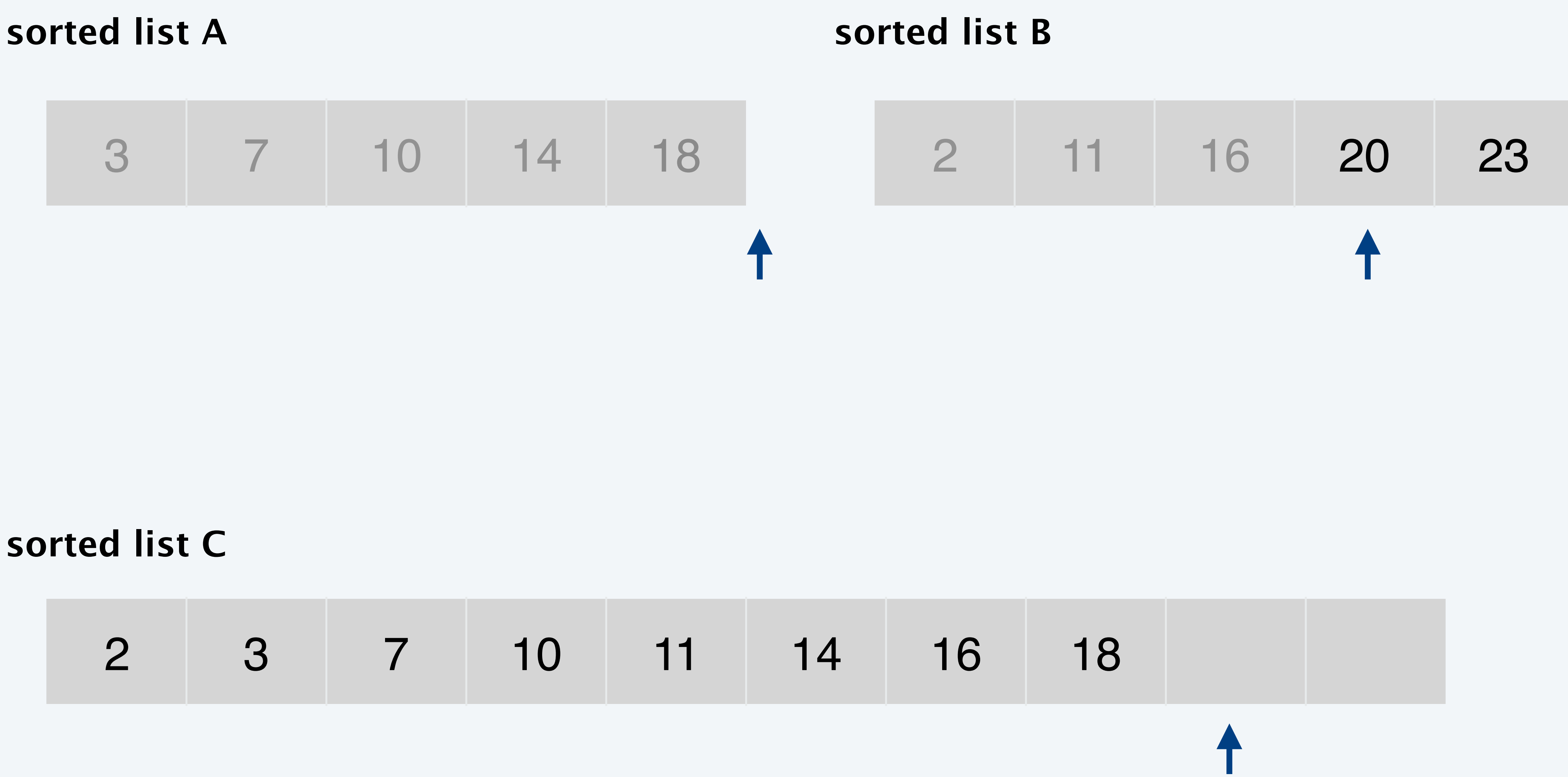
compare minimum entry in each list: copy 18

sorted list C

2	3	7	10	11	14	16			

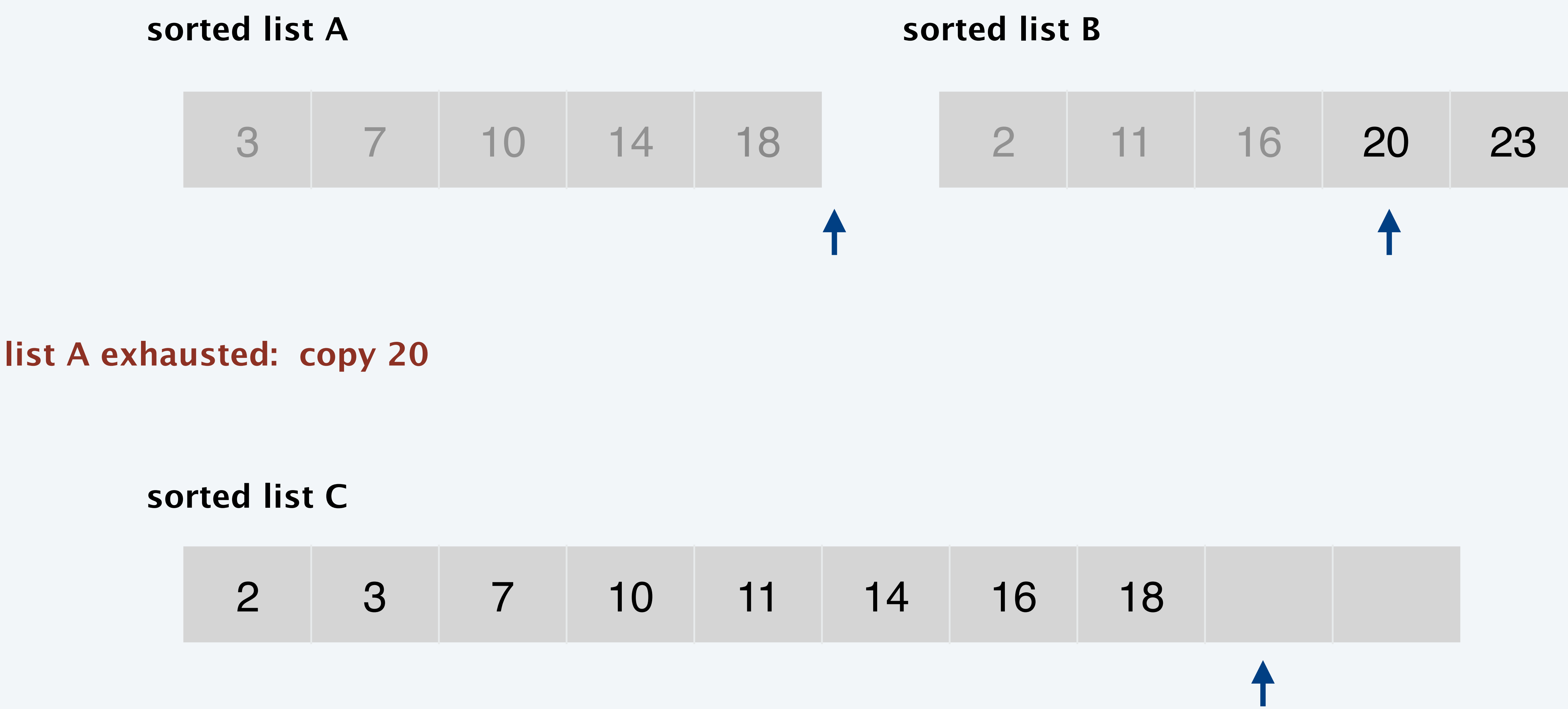
Merge demo

Given two sorted lists A and B, merge into sorted list C.



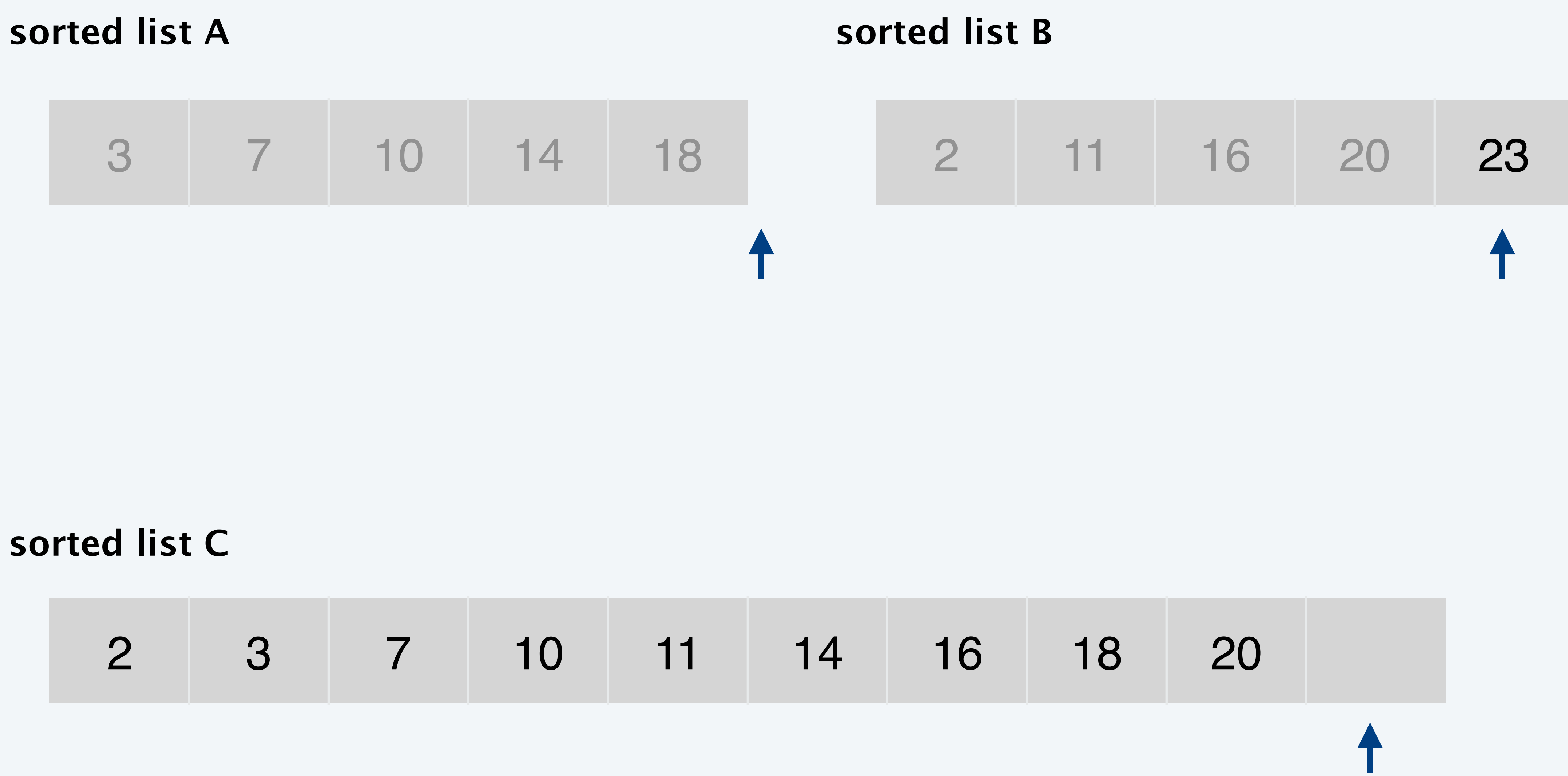
Merge demo

Given two sorted lists A and B, merge into sorted list C.



Merge demo

Given two sorted lists A and B, merge into sorted list C.



Merge demo

Given two sorted lists A and B, merge into sorted list C.

sorted list A

3	7	10	14	18

sorted list B

2	11	16	20	23

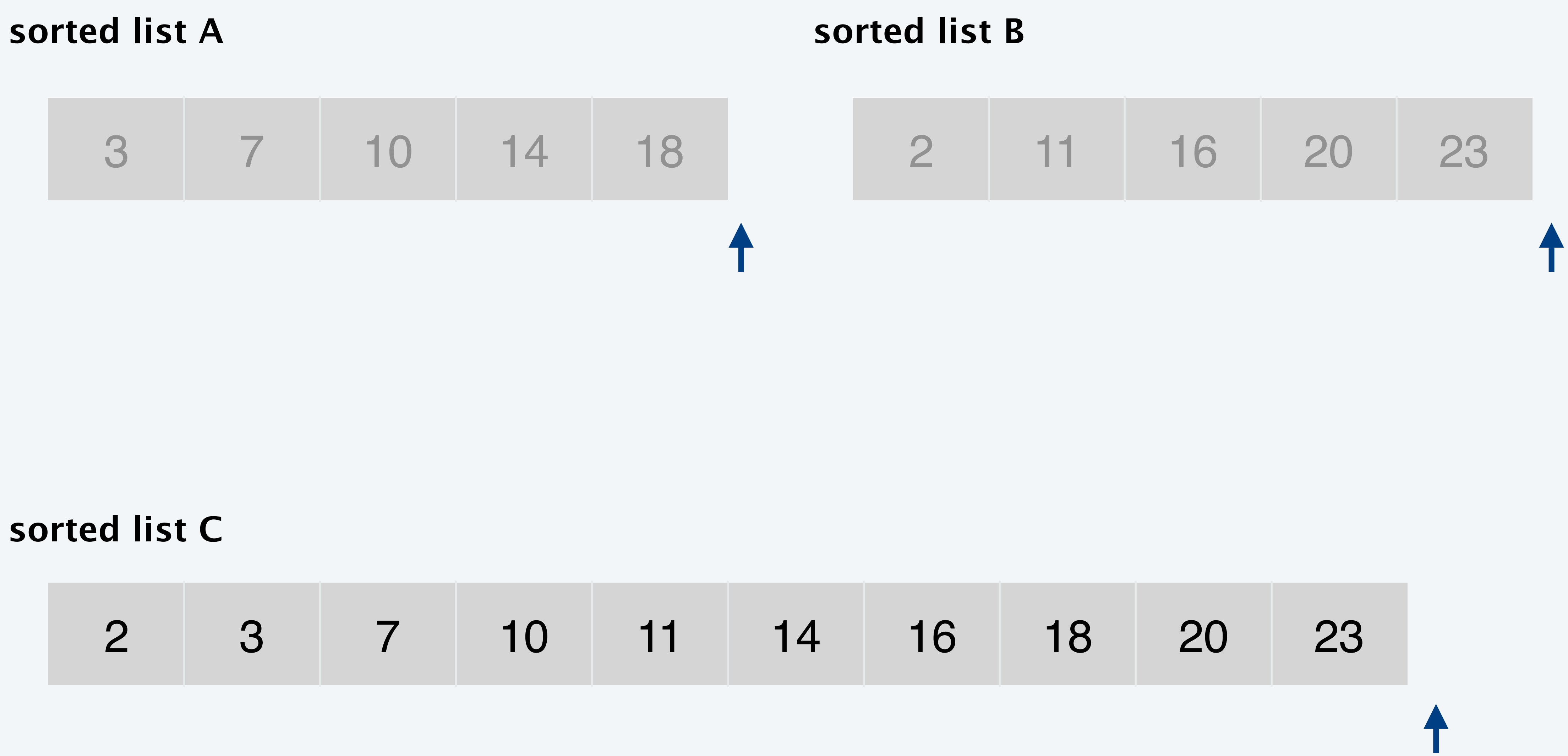
list A exhausted: copy 23

sorted list C

2	3	7	10	11	14	16	18	20	

Merge demo

Given two sorted lists A and B, merge into sorted list C.



Merge demo

Is there a difference between a worst-case and best-case input for number of steps taken by this algorithm? (just the merge)

1. Yes
2. No

$\frac{n}{2}$ is $\Theta(n)$

sorted list A

3	7	10	14	18

sorted list B

2	11	16	20	23

1 2 3 4 5 6 7 8 9 10

sorted list C

2	3	7	10	11	14	16	18	20	23

n

$n = 10$

Merge demo

Is there a difference between a worst-case and best-case input for number of steps taken by this algorithm? Discuss with table

1. Yes
2. No

sorted list A

3	7	10	14	18

sorted list B

2	11	16	20	23

sorted list C

2	3	7	10	11	14	16	18	20	23

Merge demo

Is there a difference between a worst-case and best-case input for number of steps taken by this algorithm? Discuss with table

1. Yes
2. **No**

sorted list A

3	7	10	14	18

sorted list B

2	11	16	20	23

sorted list C

2	3	7	10	11	14	16	18	20	23

Merge demo

Is there a difference between a worst-case and best-case input for number of steps taken by this algorithm? Discuss with table: what is the runtime?

1. Yes
2. **No**

sorted list A

3	7	10	14	18

sorted list B

2	11	16	20	23

sorted list C

2	3	7	10	11	14	16	18	20	23

Mergesort

- Recursively sort left half.
- Recursively sort right half.
- Merge two halves to make sorted whole.

$T(n) = 2T\left(\frac{n}{2}\right) + cn$

```
mergesort(L):
  L1 = first half of L
  L2 = first half of L
  sorted_L1 = mergesort(L1)
  sorted_L2 = mergesort(L2)
  return merged L1 and L2
```

n steps

n steps

$T(n)$ = worst-case runtime of mergesort on an input of length n

$T(n)$ = time to split list (n) + time to make 2 recursive calls ($2T(\frac{n}{2})$) + time to merge (n)

Approach # 1: unroll the recurrence

Approach # 1: unroll the recurrence

recurrence relation

Proposition. If $T(n)$ satisfies the following recurrence, then T(n) is…

$$T(n) = \begin{cases} 0 & \text{if } n = 1 \\ 2\,T(n/2) + n & \text{if } n > 1 \end{cases}$$

Approach # 1: unroll the recurrence

Proposition. If $T(n)$ satisfies the following recurrence, then $T(n) = n \log_2 n$.

assuming n is a power of 2

$$T(n) = \begin{cases} 0 & \text{if } n = 1 \\ 2\,T(n/2) + n & \text{if } n > 1 \end{cases}$$

Approach # 1: unroll the recurrence

Proposition. If $T(n)$ satisfies the following recurrence, then $T(n) = n \log_2 n$.

assuming n is a power of 2

$$T(n) = \begin{cases} 0 & \text{if } n = 1 \\ 2\,T(n/2) + n & \text{if } n > 1 \end{cases}$$

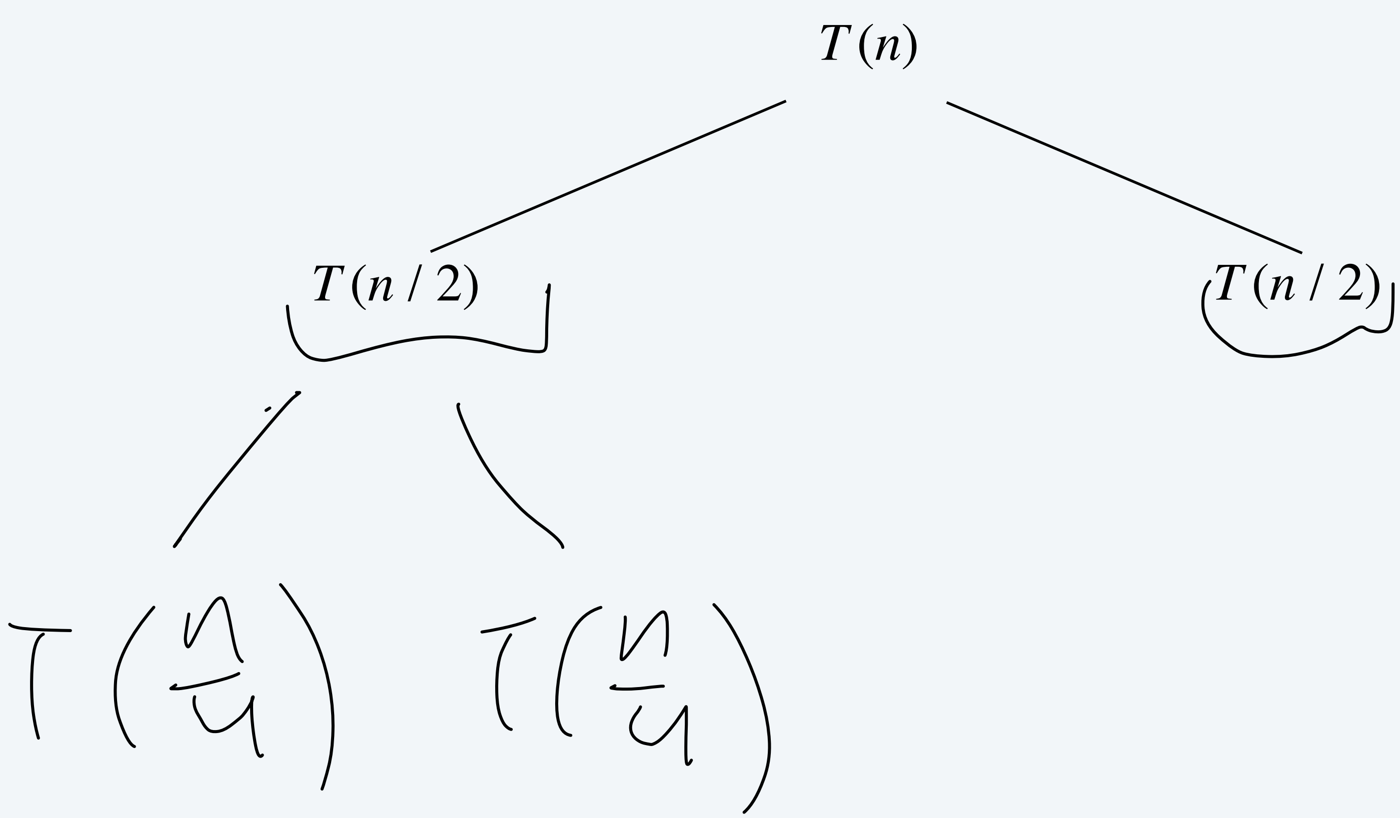
$T(n)$

Approach # 1: unroll the recurrence

Proposition. If $T(n)$ satisfies the following recurrence, then $T(n) = n \log_2 n$.

assuming n is a power of 2

$$T(n) = \begin{cases} 0 & \text{if } n = 1 \\ 2\,T(n/2) + n & \text{if } n > 1 \end{cases}$$

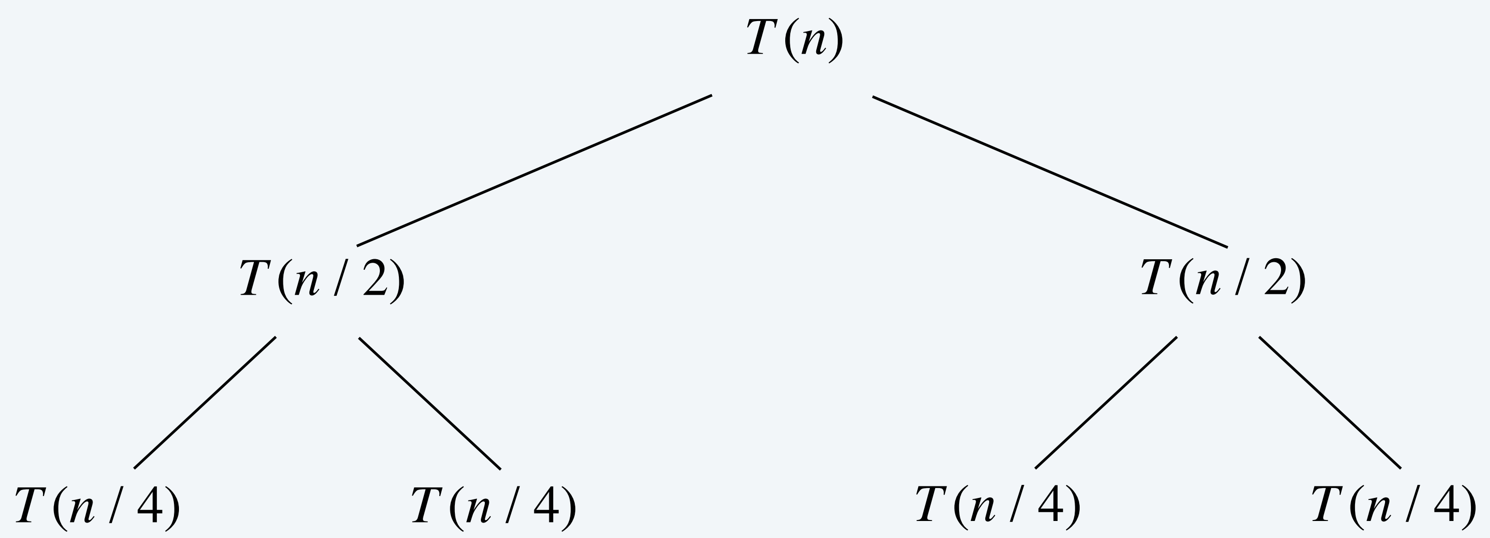


Approach # 1: unroll the recurrence

Proposition. If $T(n)$ satisfies the following recurrence, then $T(n) = n \log_2 n$.

assuming n is a power of 2

$$T(n) = \begin{cases} 0 & \text{if } n = 1 \\ 2\,T(n/2) + n & \text{if } n > 1 \end{cases}$$

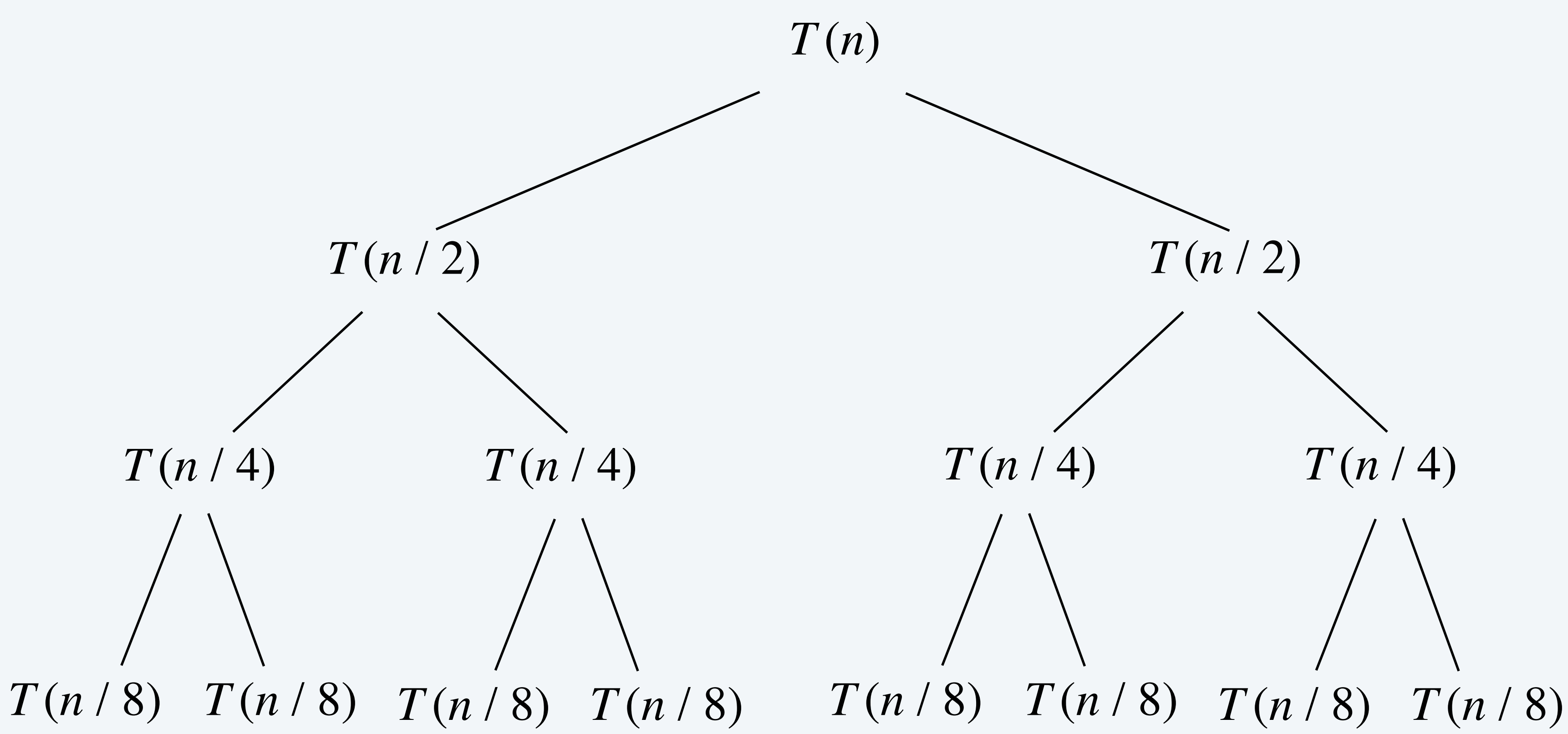


Approach # 1: unroll the recurrence

Proposition. If $T(n)$ satisfies the following recurrence, then $T(n) = n \log_2 n$.

assuming n is a power of 2

$$T(n) = \begin{cases} 0 & \text{if } n = 1 \\ 2\,T(n/2) + n & \text{if } n > 1 \end{cases}$$

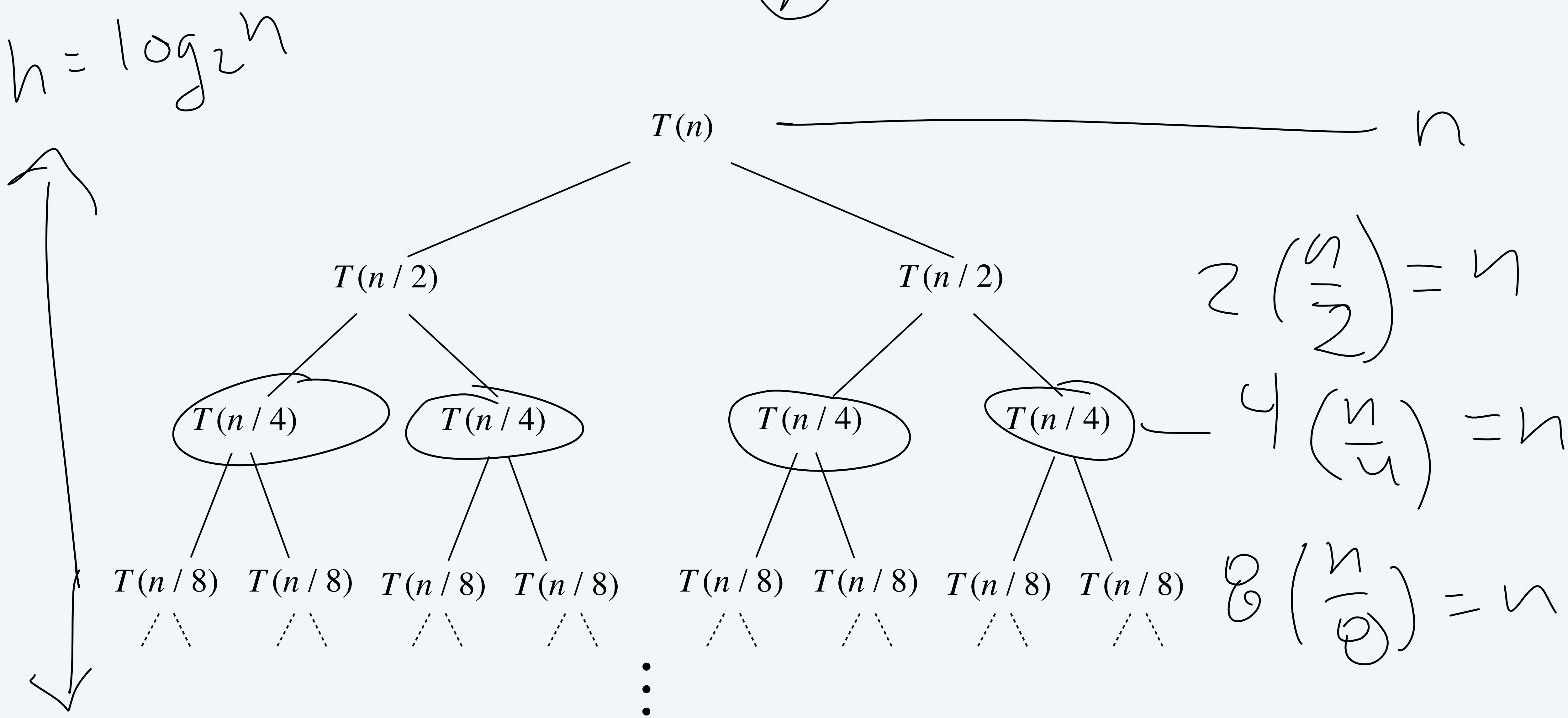


Approach # 1: unroll the recurrence

Proposition. If $T(n)$ satisfies the following recurrence, then $T(n) = n \log_2 n$.

assuming n is a power of 2

$$T(n) = \begin{cases} 0 & \text{if } n = 1 \\ 2\,T(n/2) + n & \text{if } n > 1 \end{cases}$$

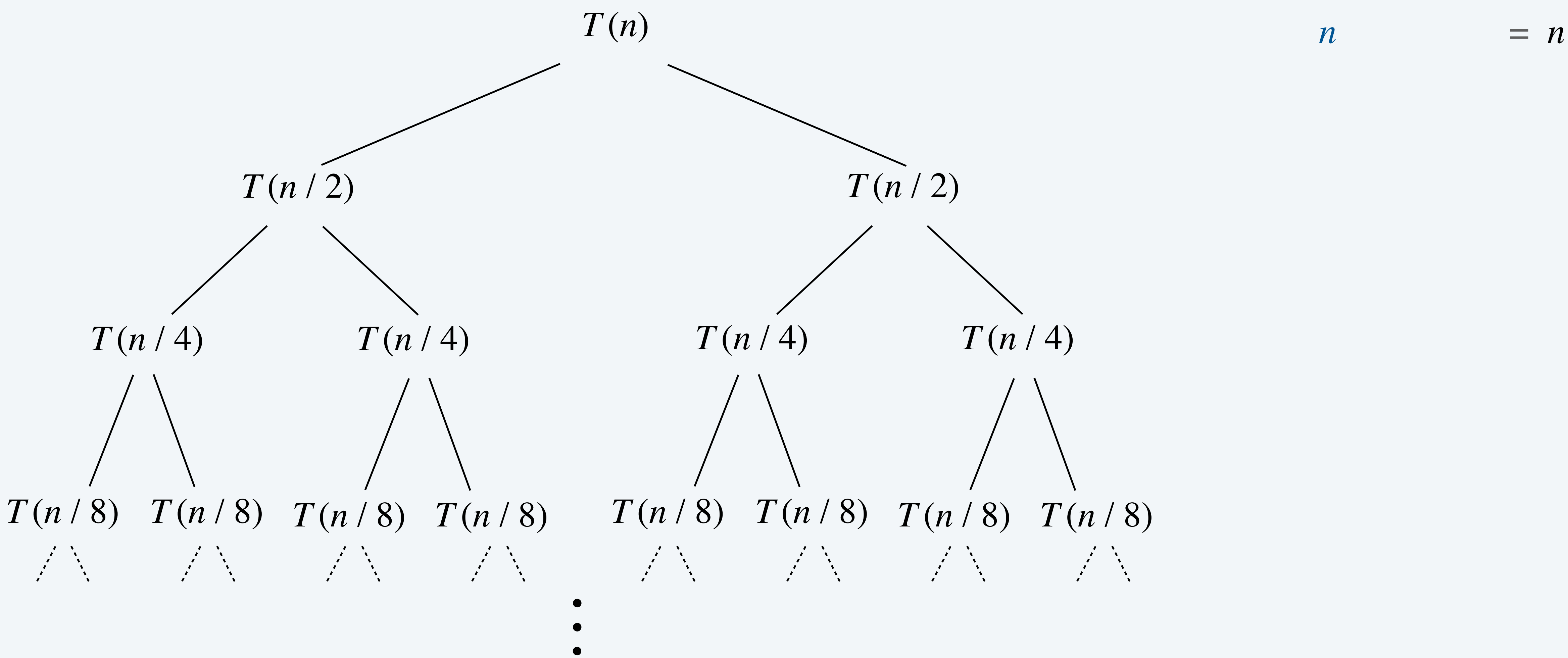


Approach # 1: unroll the recurrence

Proposition. If $T(n)$ satisfies the following recurrence, then $T(n) = n \log_2 n$.

assuming n is a power of 2

$$T(n) = \begin{cases} 0 & \text{if } n = 1 \\ 2\,T(n/2) + n & \text{if } n > 1 \end{cases}$$

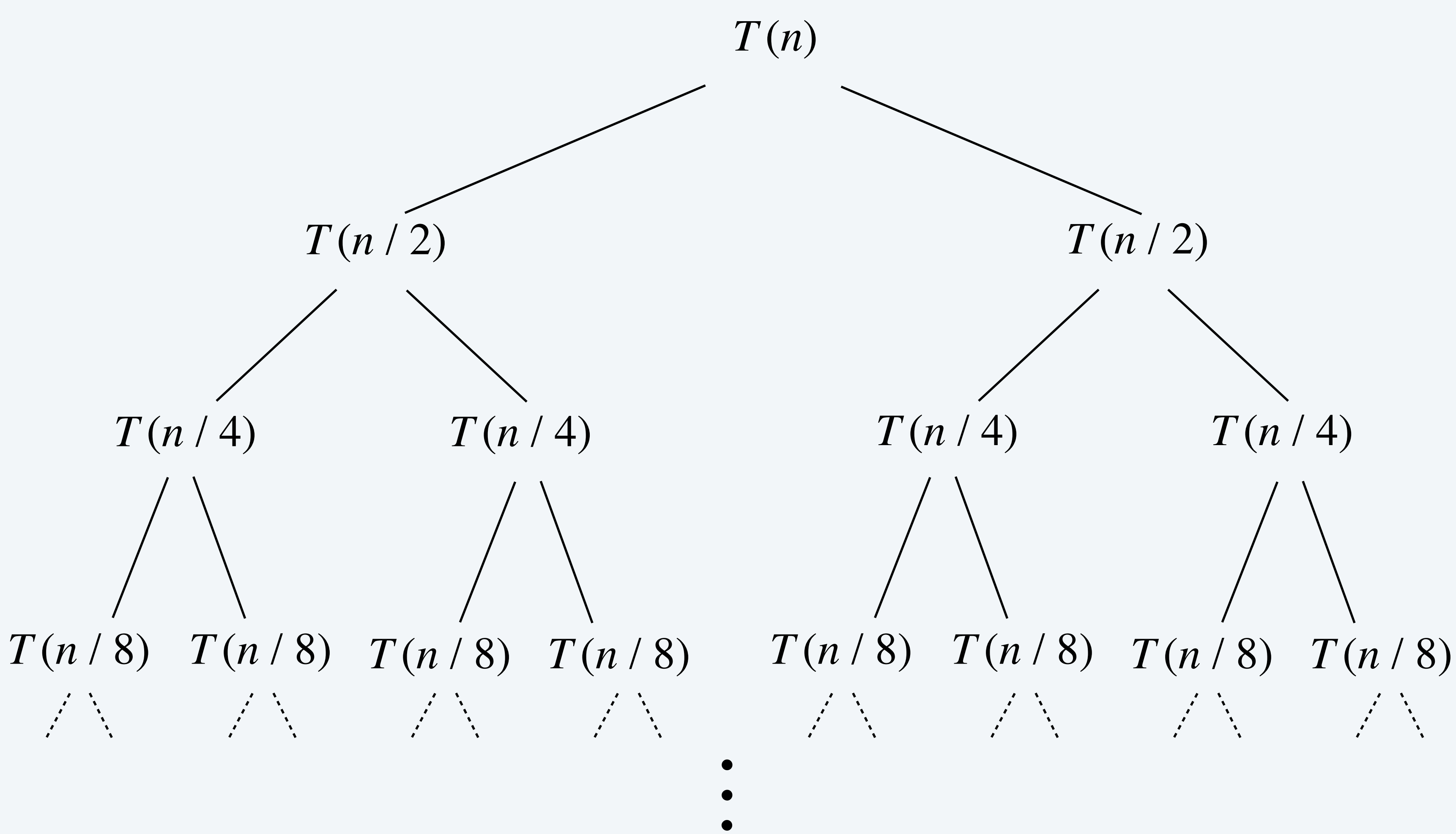


Approach # 1: unroll the recurrence

Proposition. If $T(n)$ satisfies the following recurrence, then $T(n) = n \log_2 n$.

assuming n is a power of 2

$$T(n) = \begin{cases} 0 & \text{if } n = 1 \\ 2\,T(n/2) + n & \text{if } n > 1 \end{cases}$$



$n = n$

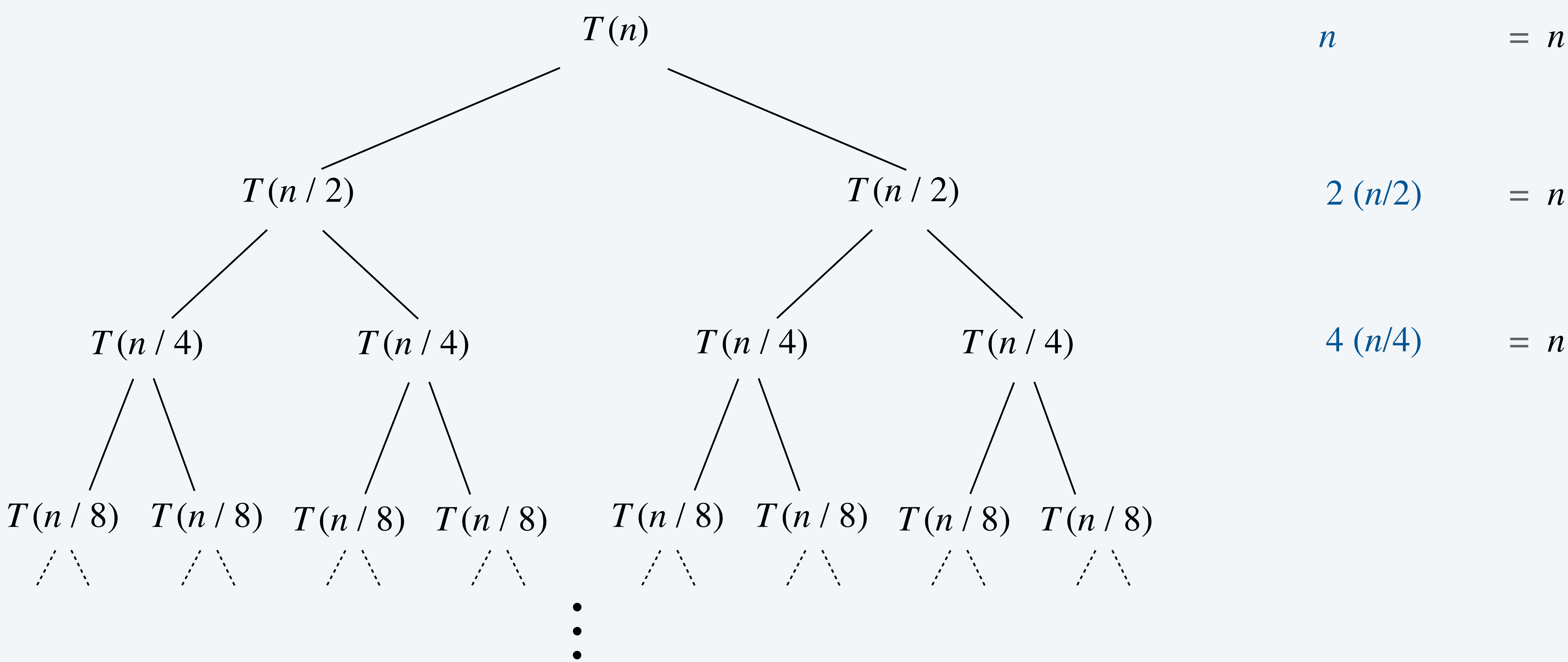
$2\,(n/2) = n$

Approach # 1: unroll the recurrence

Proposition. If $T(n)$ satisfies the following recurrence, then $T(n) = n \log_2 n$.

assuming n is a power of 2

$$T(n) = \begin{cases} 0 & \text{if } n = 1 \\ 2\,T(n/2) + n & \text{if } n > 1 \end{cases}$$

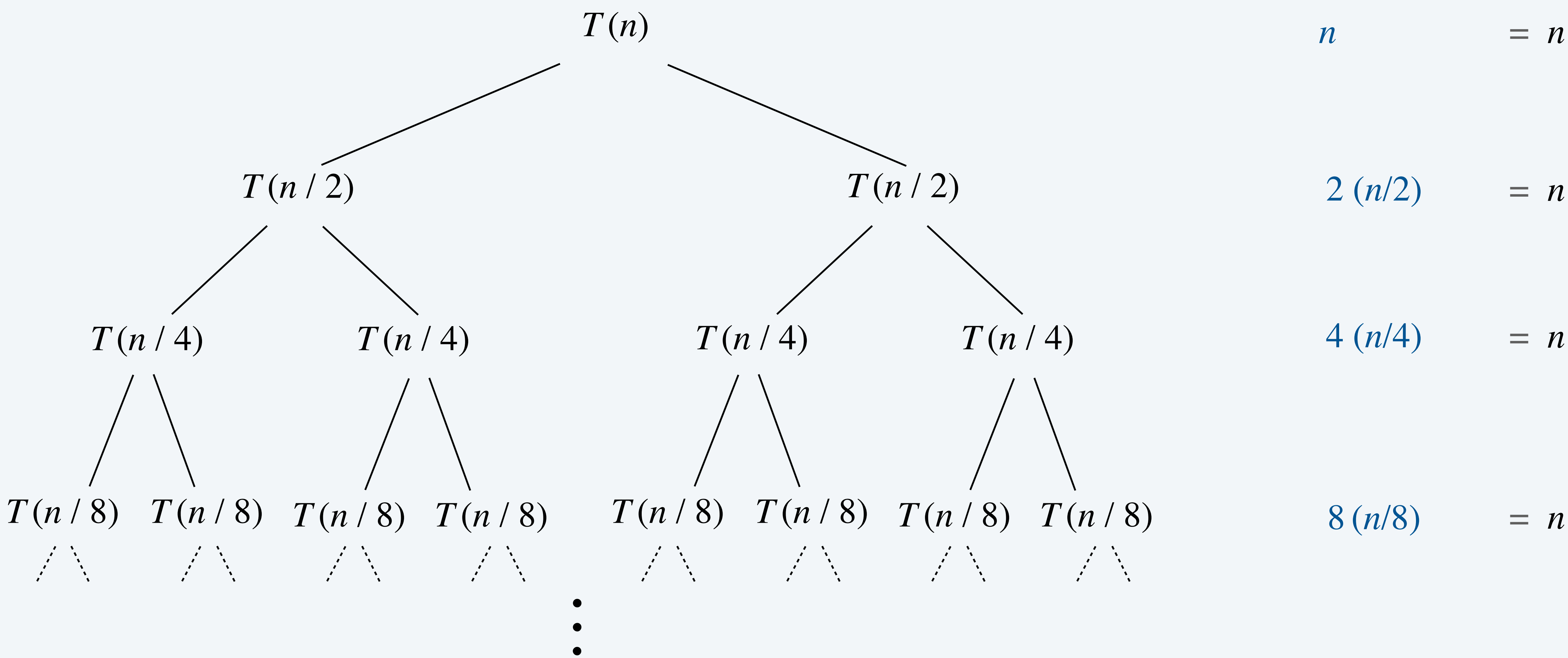


Approach # 1: unroll the recurrence

Proposition. If $T(n)$ satisfies the following recurrence, then $T(n) = n \log_2 n$.

assuming n is a power of 2

$$T(n) = \begin{cases} 0 & \text{if } n = 1 \\ 2\,T(n/2) + n & \text{if } n > 1 \end{cases}$$

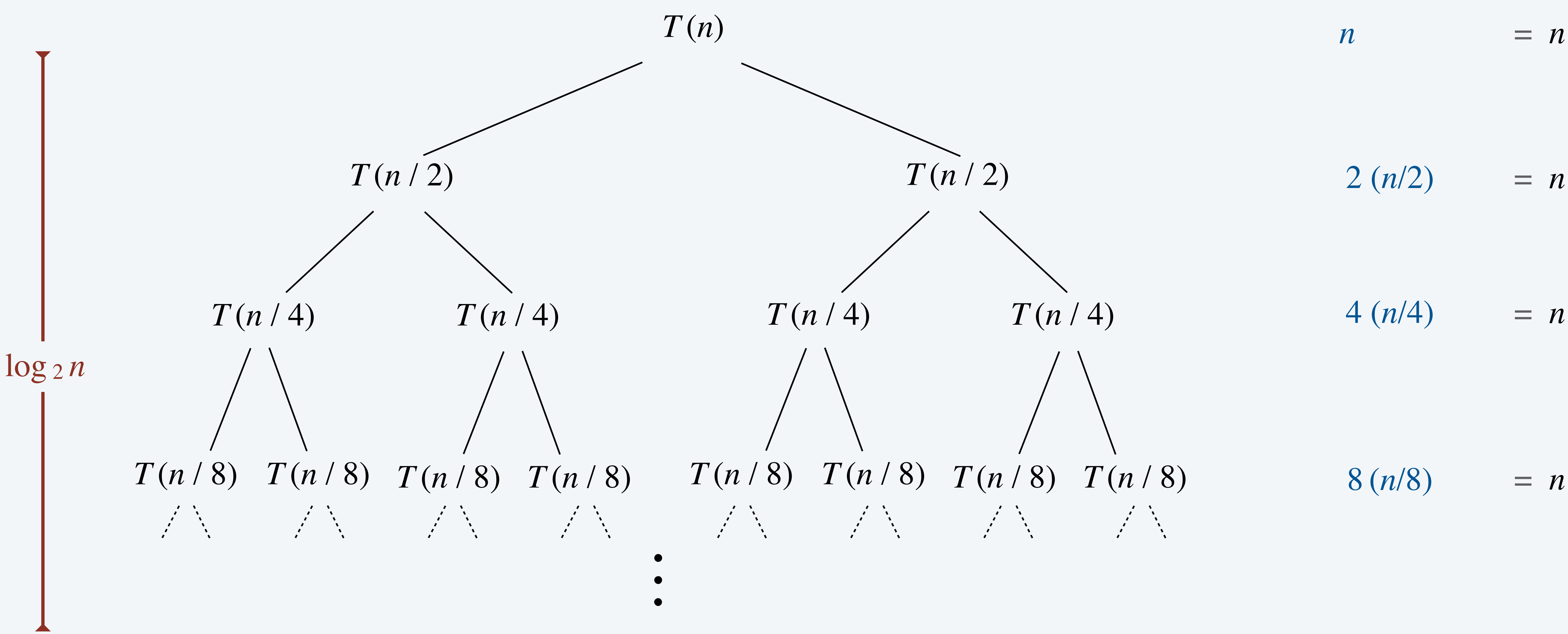


Approach # 1: unroll the recurrence

Proposition. If $T(n)$ satisfies the following recurrence, then $T(n) = n \log_2 n$.

assuming n is a power of 2

$$T(n) = \begin{cases} 0 & \text{if } n = 1 \\ 2\,T(n/2) + n & \text{if } n > 1 \end{cases}$$

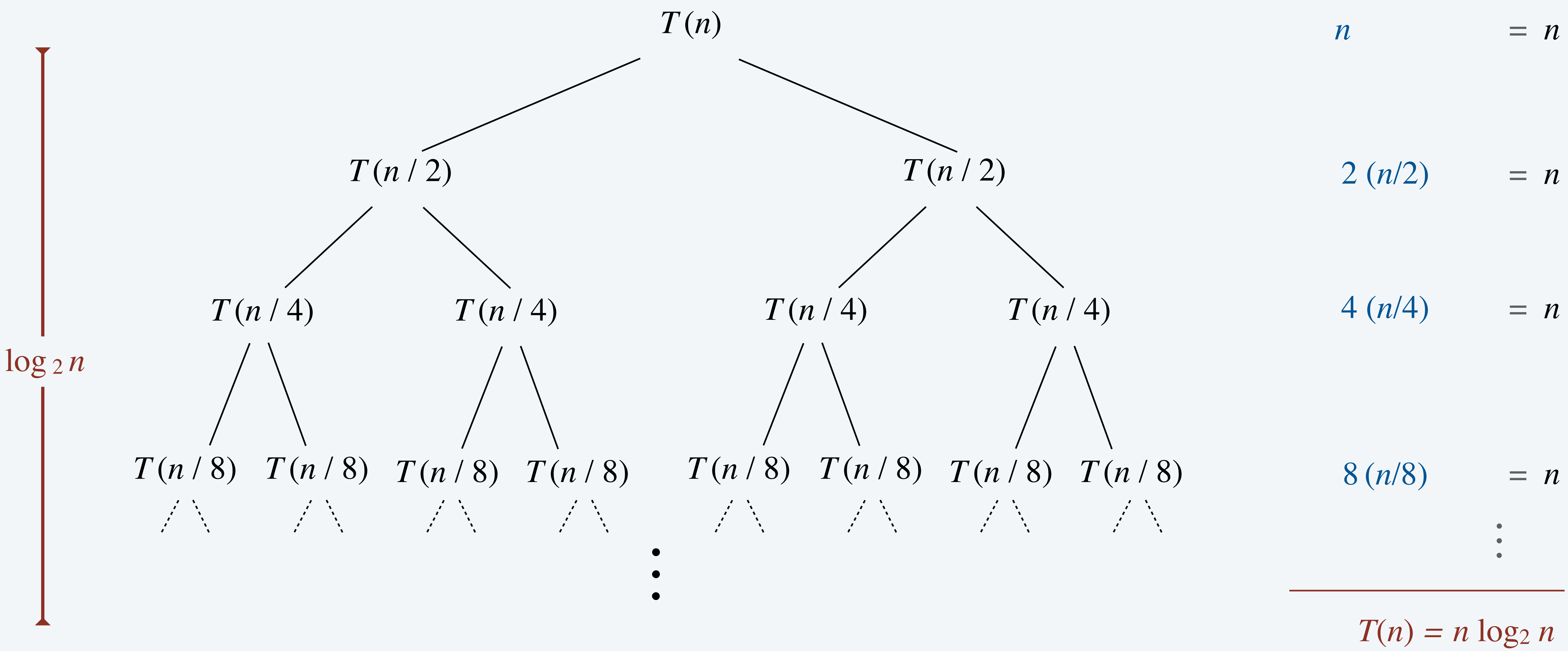


Approach # 1: unroll the recurrence

Proposition. If $T(n)$ satisfies the following recurrence, then $T(n) = n \log_2 n$.

assuming n is a power of 2

$$T(n) = \begin{cases} 0 & \text{if } n = 1 \\ 2\,T(n/2) + n & \text{if } n > 1 \end{cases}$$



Choose an answer

```
fancymergesort(L):
  L1 = first third of L
  L2 = second third of L
  L3 = last third of L
  sorted_L1 = mergesort(L1)
  sorted_L2 = mergesort(L2)
  sorted_L3 = mergesort(L3)
  return merged L1,L2,L3
```

n

n

normal MS

$T(n) = 2T(\frac{n}{2}) + n$

3 calls to T

w/ $\frac{n}{3}$

What is a valid recurrence relation for fancymergesort?

1. $T(n) = n^2$
2. $T(n) = 3T(n/3) + n$
3. $T(n) = cn \log_3 n$
4. $T(n) = nT(n) + 3n$

$T(n) = 3T(\frac{n}{3}) + n$

Proof by induction

$$T(n) = \begin{cases} 0 & \text{if } n = 1 \\ 2\,T(n/2) + n & \text{if } n > 1 \end{cases}$$

assuming n is a power of 2

Proof by induction

Proposition. If $T(n)$ satisfies the following recurrence, then $T(n) = n \log_2 n$.

assuming n is a power of 2

$$T(n) = \begin{cases} 0 & \text{if } n = 1 \\ 2\,T(n/2) + n & \text{if } n > 1 \end{cases}$$

Claim: for $n \geq 1$, if [above] then $T(n) = n \log n$

Proof:

Let n be ≥ 1.

Assume that for all $m < n$, if $T(m) = 2T(m/2) + m$ (for $m > 1$) then $T(m) = m \log m$.

Base case: If $n = 1$, then $T(1) = 0$. And $1 \cdot \log 1 = 0$.

If $n > 1$, then $T(n) = 2T(n/2) + n$

$= 2\left(\frac{n}{2} \log \frac{n}{2}\right) + n$ by IH

$= n \log \frac{n}{2} + n$

$$= n\left(\log \frac{n}{2} + 1\right)$$

$$= n\left(\log_2 \frac{n}{2} + \log_2 2\right)$$

$$= n\left(\log_2 \cancel{2}\left(\frac{n}{\cancel{2}}\right)\right)$$

$$= n \log n$$

Digression: sorting lower bound

Challenge. How to prove a lower bound for all conceivable algorithms?

Digression: sorting lower bound

Challenge. How to prove a lower bound for all conceivable algorithms?

Model of computation. Comparison trees.

Digression: sorting lower bound

Challenge. How to prove a lower bound for all conceivable algorithms?

Model of computation. Comparison trees.

- Can access the elements only through pairwise comparisons.

Digression: sorting lower bound

Challenge. How to prove a lower bound for all conceivable algorithms?

Model of computation. Comparison trees.

- Can access the elements only through pairwise comparisons.
- All other operations (control, data movement, etc.) are free.

Digression: sorting lower bound

Challenge. How to prove a lower bound for all conceivable algorithms?

Model of computation. Comparison trees.

- Can access the elements only through pairwise comparisons.
- All other operations (control, data movement, etc.) are free.

Cost model. Number of compares.

Digression: sorting lower bound

Challenge. How to prove a lower bound for all conceivable algorithms?

Model of computation. Comparison trees.

- Can access the elements only through pairwise comparisons.
- All other operations (control, data movement, etc.) are free.

Cost model. Number of compares.

Q. Realistic model?

Digression: sorting lower bound

Challenge. How to prove a lower bound for all conceivable algorithms?

Model of computation. Comparison trees.

- Can access the elements only through pairwise comparisons.
- All other operations (control, data movement, etc.) are free.

Cost model. Number of compares.

Q. Realistic model?

A1. Yes. Java, Python, C++, …

```
Comparable[] a = ...;
...
Arrays.sort(a);
```

← can access elements only via calls to compareTo()

Digression: sorting lower bound

Challenge. How to prove a lower bound for all conceivable algorithms?

Model of computation. Comparison trees.

- Can access the elements only through pairwise comparisons.
- All other operations (control, data movement, etc.) are free.

Cost model. Number of compares.

Q. Realistic model?

A1. Yes. Java, Python, C++, …

`sort`(***, *key=None*, *reverse=False*)

This method sorts the list in place, using only `<` comparisons between items. Exceptions are not suppressed – if any comparison operations fail, the entire sort operation will fail (and the list will likely be left in a partially modified state).

Digression: sorting lower bound

Challenge. How to prove a lower bound for all conceivable algorithms?

Model of computation. Comparison trees.

- Can access the elements only through pairwise comparisons.
- All other operations (control, data movement, etc.) are free.

Cost model. Number of compares.

Q. Realistic model?

A1. Yes. Java, Python, C++, …

A2. Yes. Mergesort, insertion sort, quicksort, heapsort, …

Digression: sorting lower bound

Challenge. How to prove a lower bound for all conceivable algorithms?

Model of computation. Comparison trees.

- Can access the elements only through pairwise comparisons.
- All other operations (control, data movement, etc.) are free.

Cost model. Number of compares.

Q. Realistic model?

A1. Yes. Java, Python, C++, …

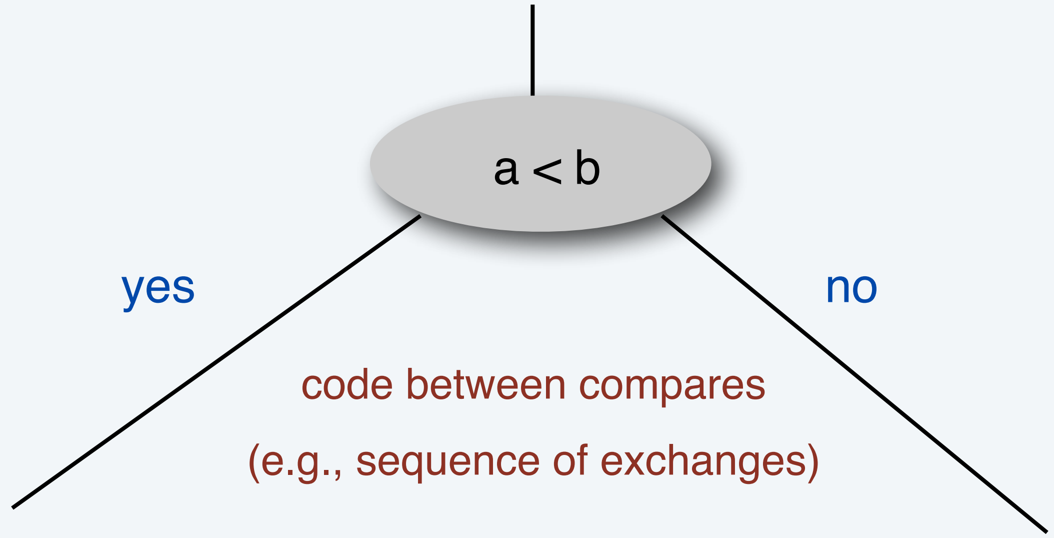
A2. Yes. Mergesort, insertion sort, quicksort, heapsort, …

A3. No. Bucket sort, radix sorts, …

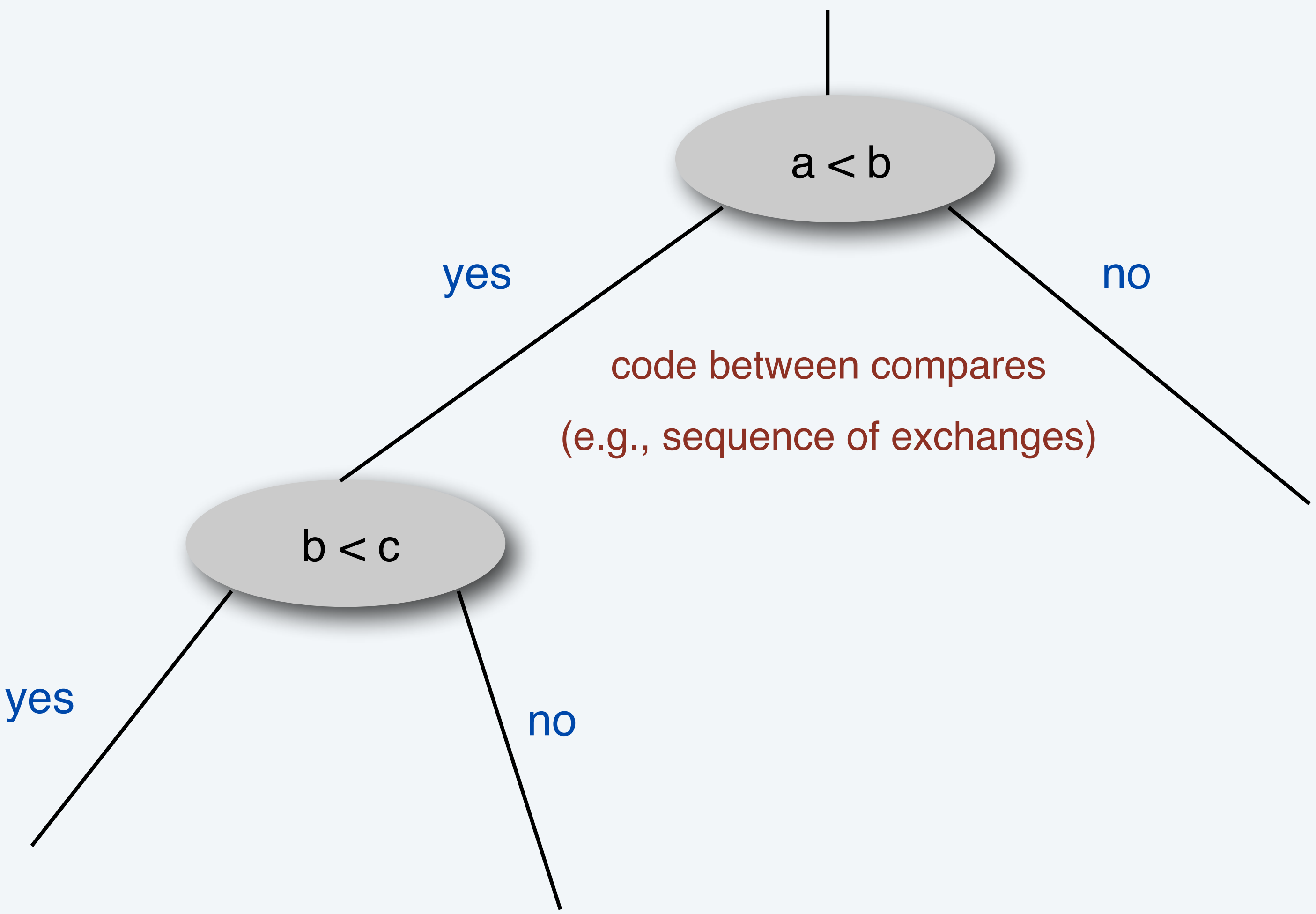
Comparison tree (for 3 distinct keys a, b, and c)

Comparison tree (for 3 distinct keys a, b, and c)

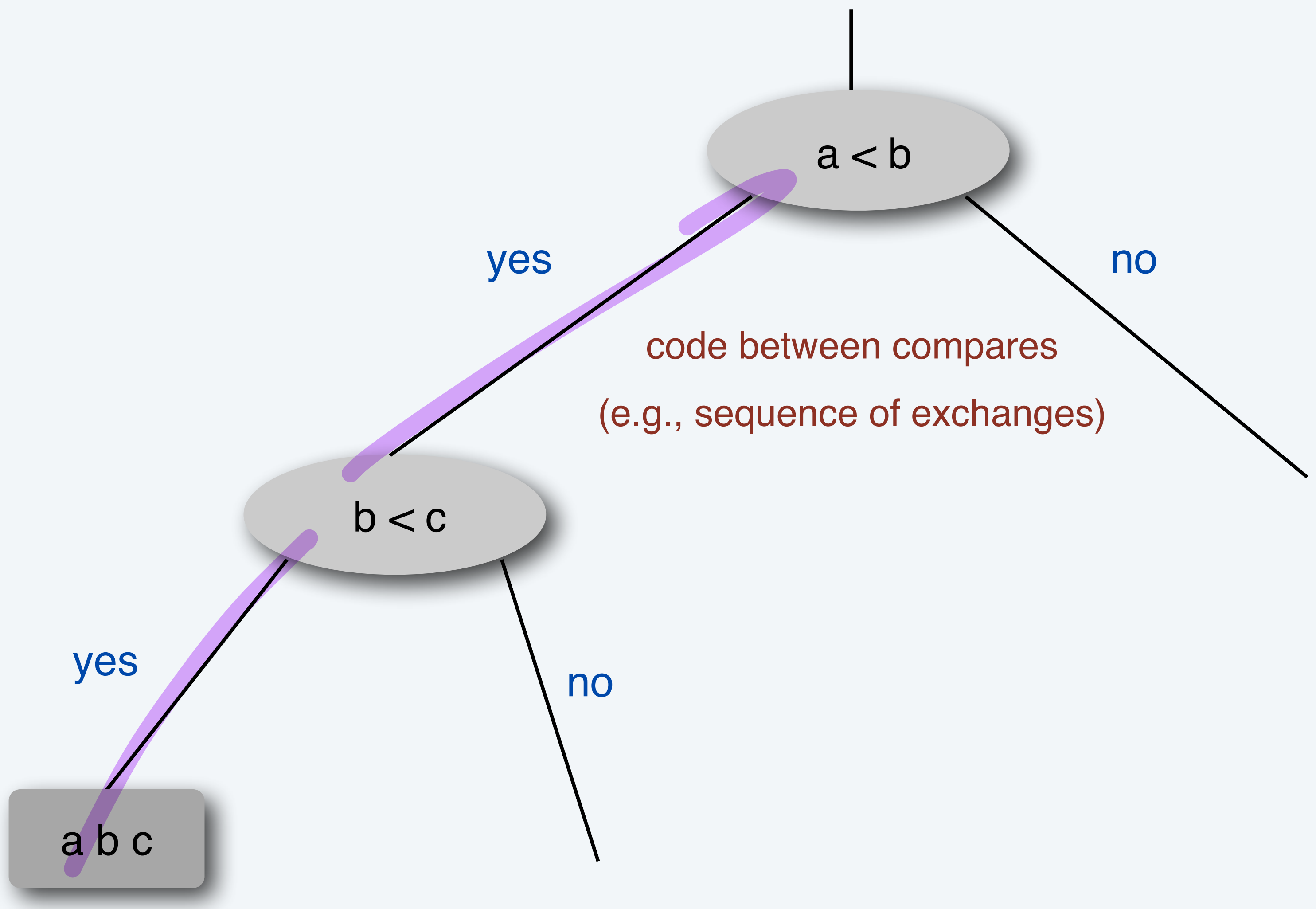
Comparison tree (for 3 distinct keys a, b, and c)



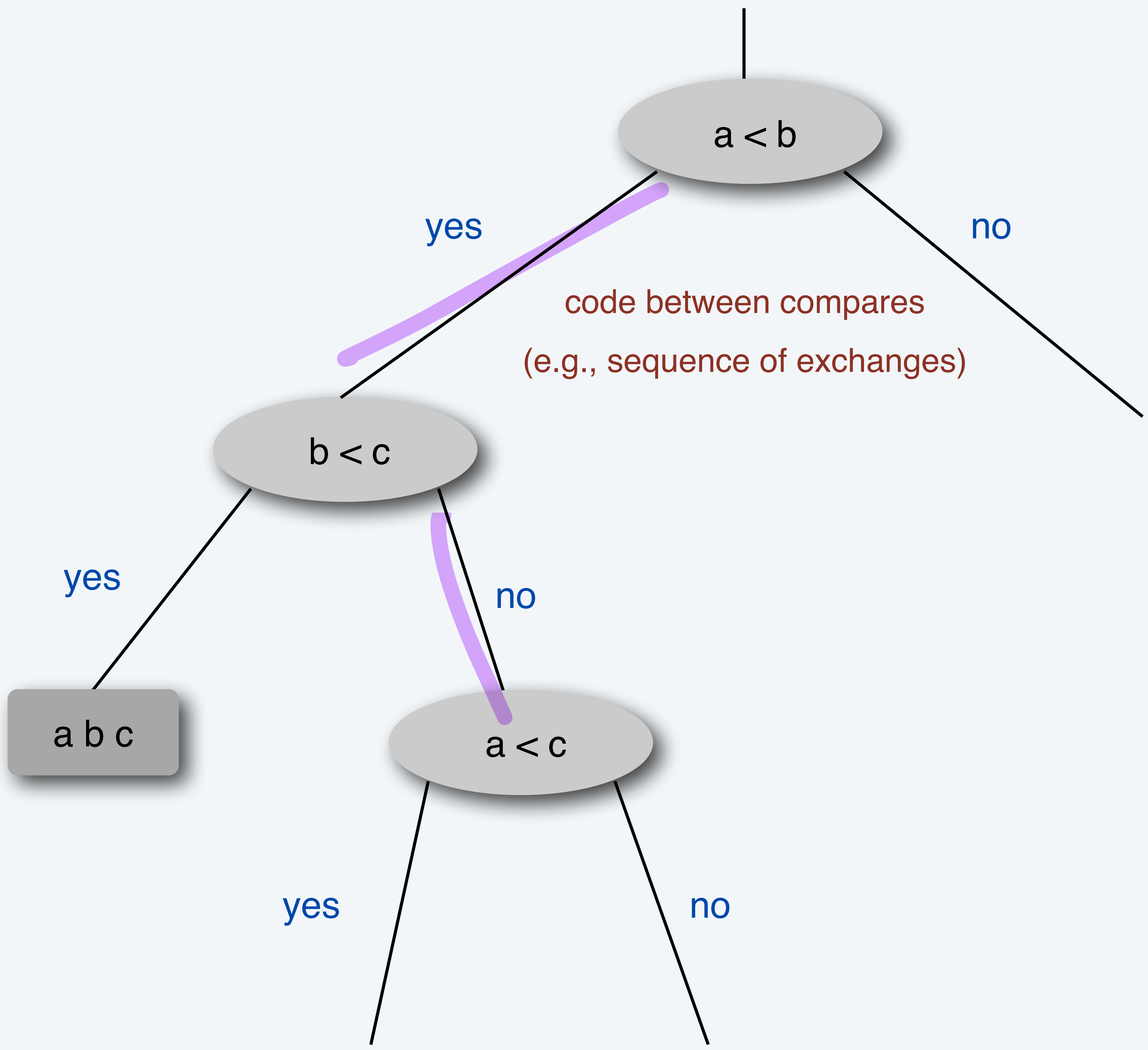
Comparison tree (for 3 distinct keys a, b, and c)



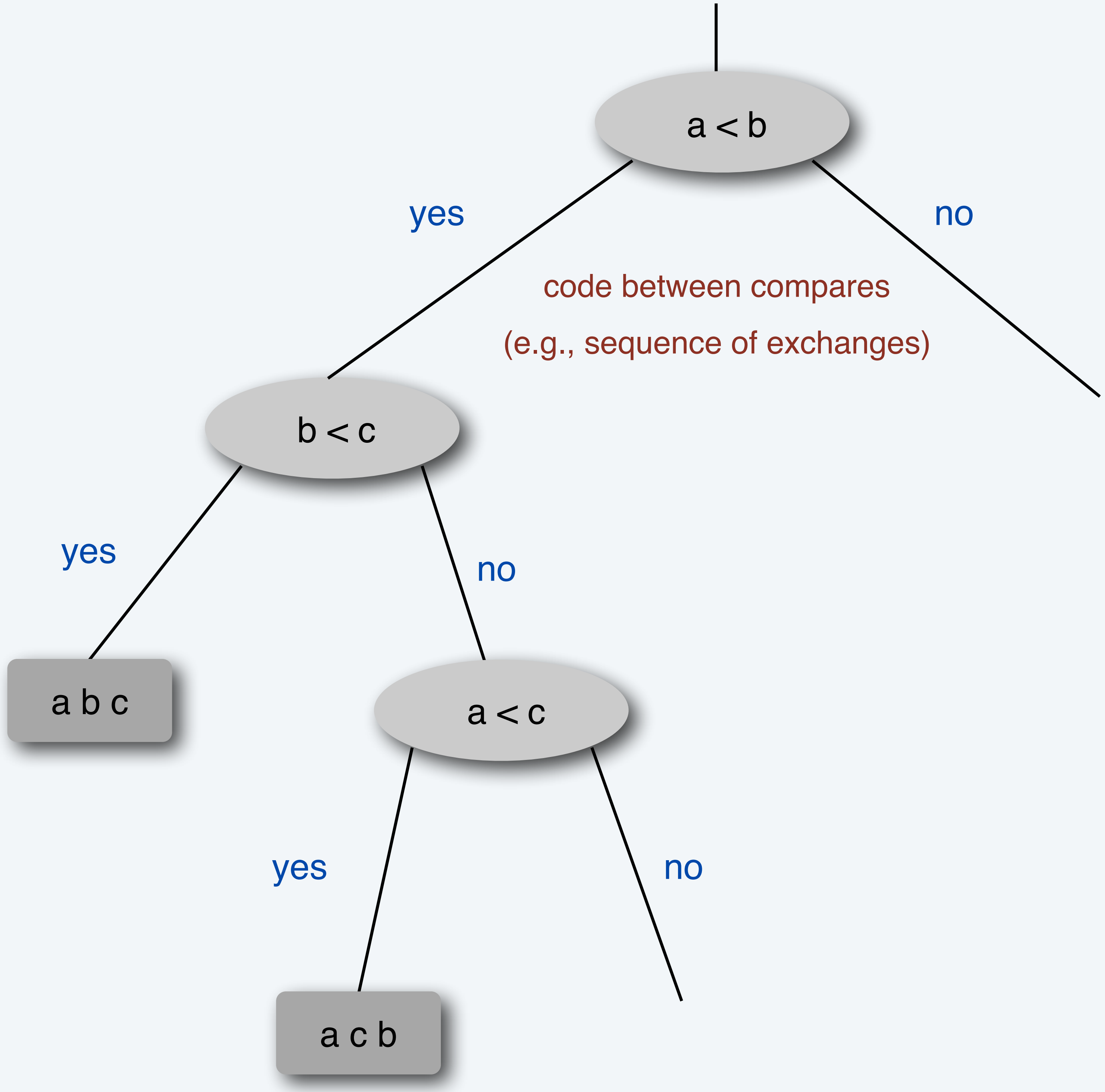
Comparison tree (for 3 distinct keys a, b, and c)



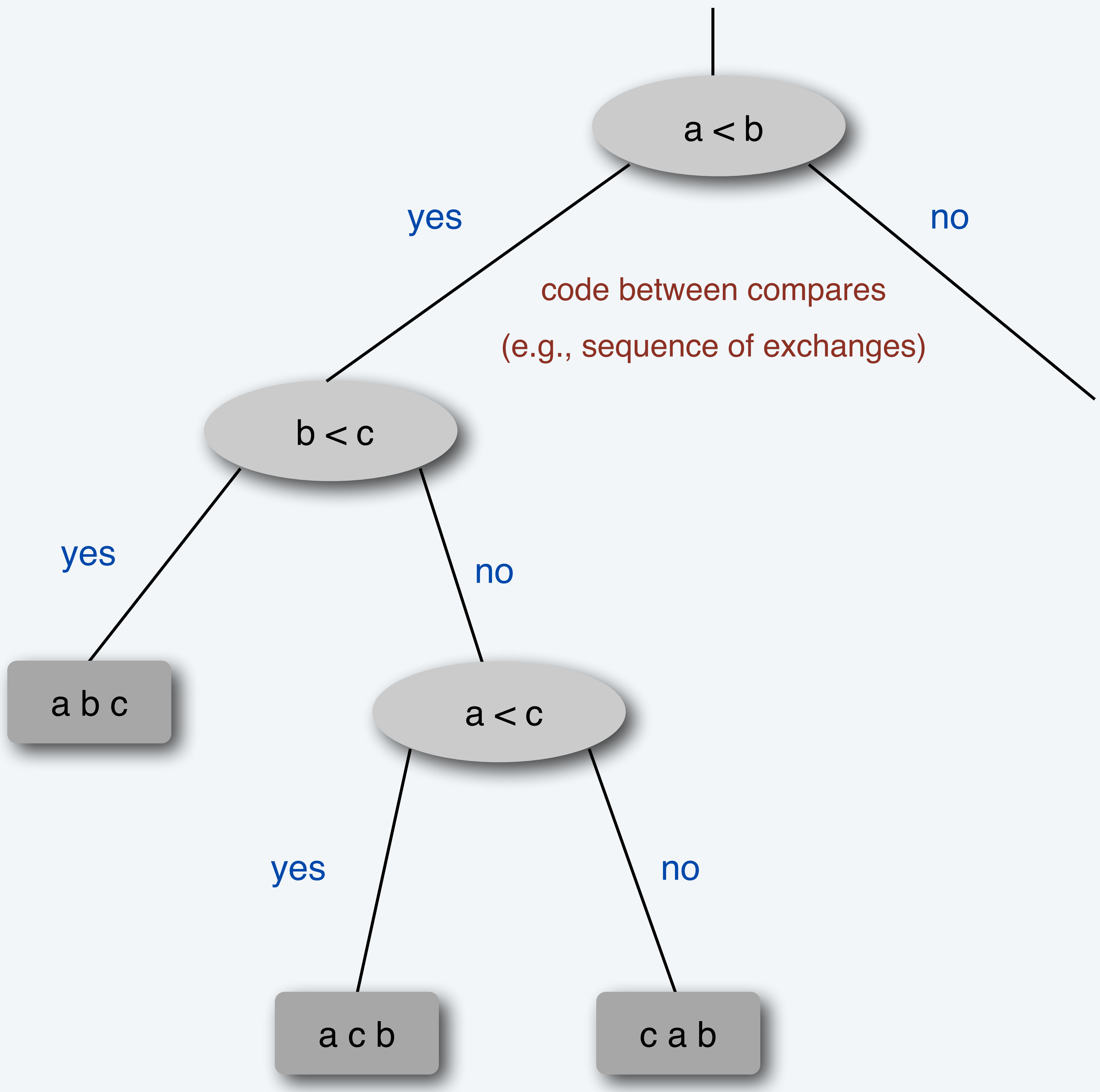
Comparison tree (for 3 distinct keys a, b, and c)



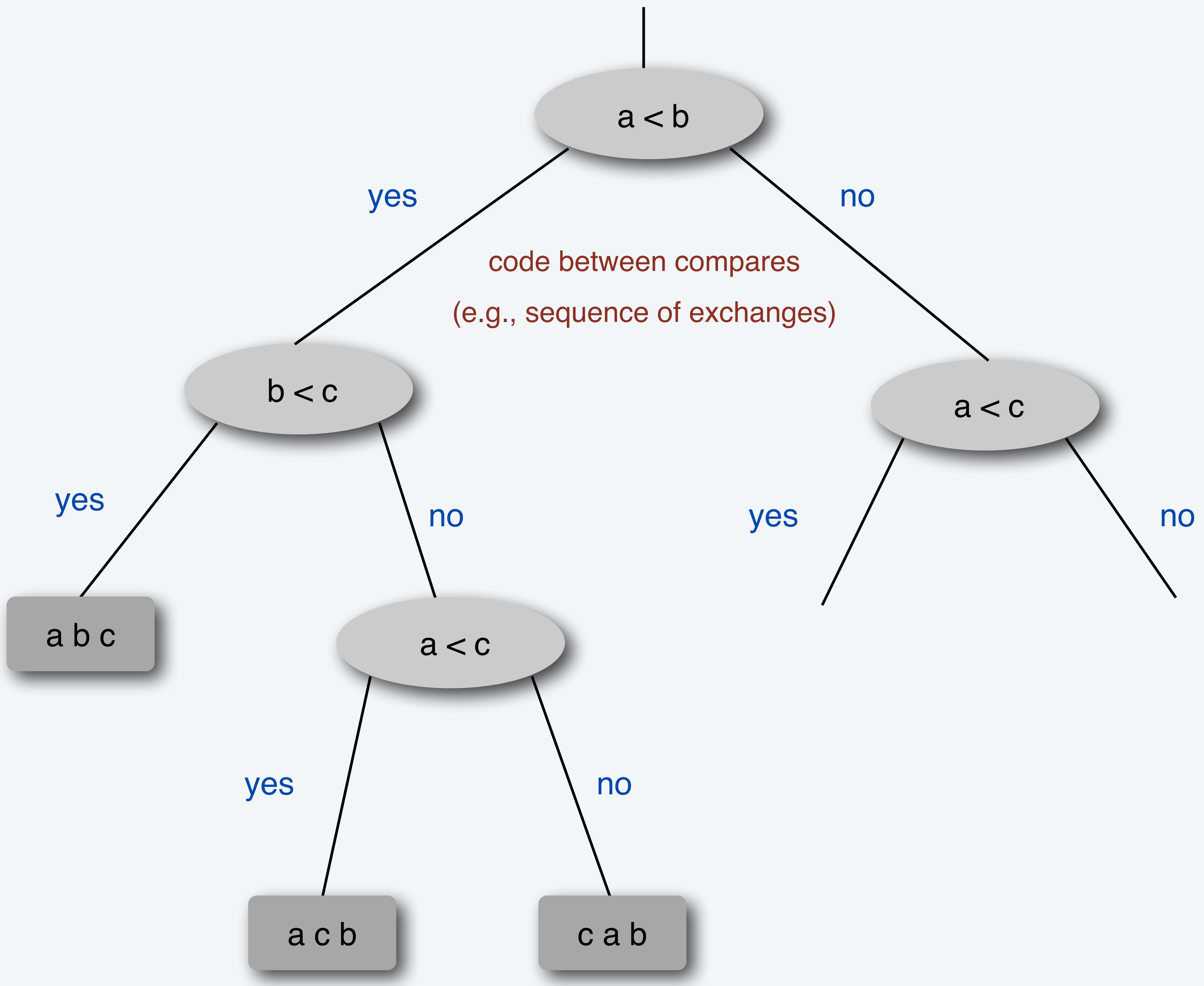
Comparison tree (for 3 distinct keys a, b, and c)



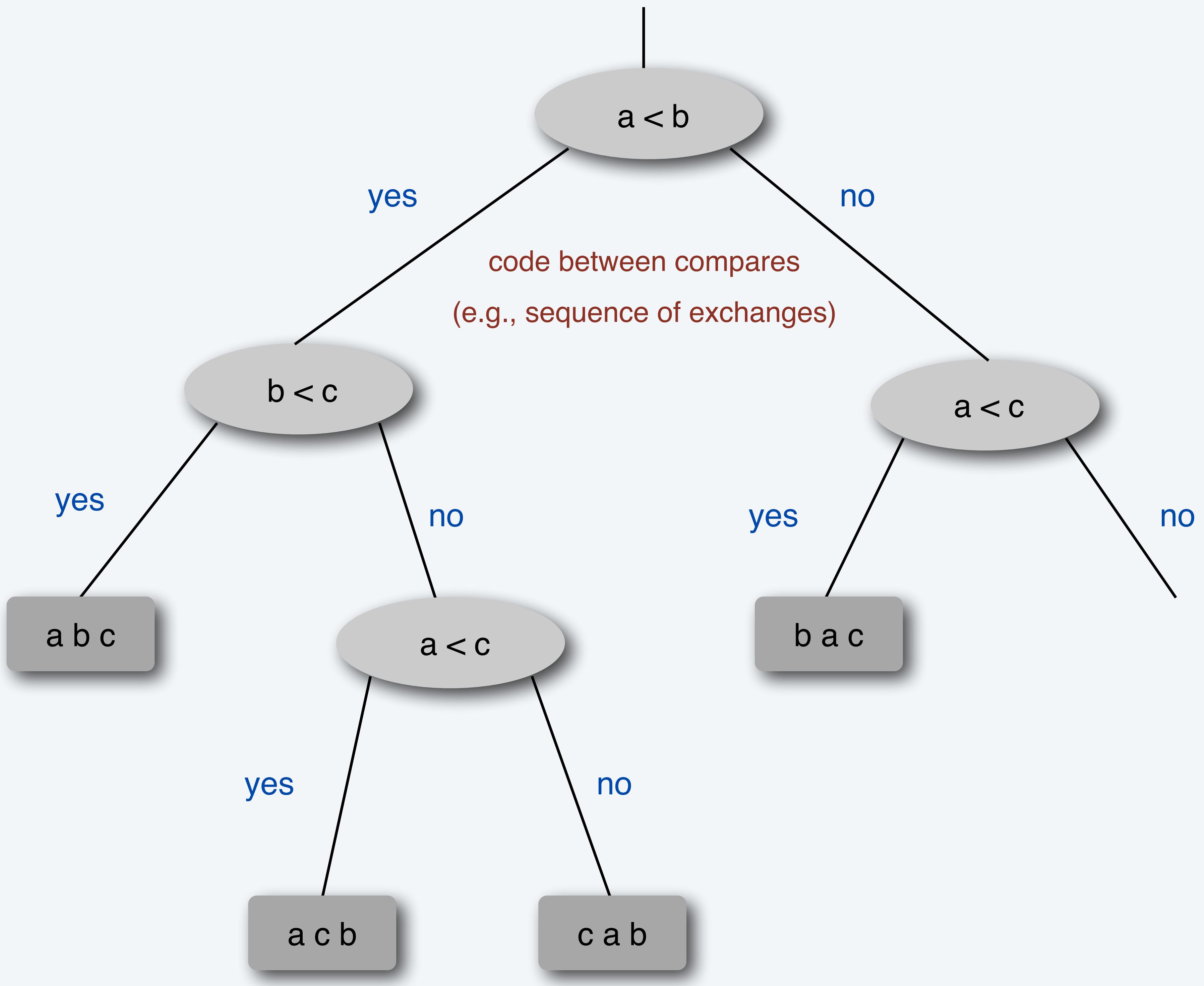
Comparison tree (for 3 distinct keys a, b, and c)



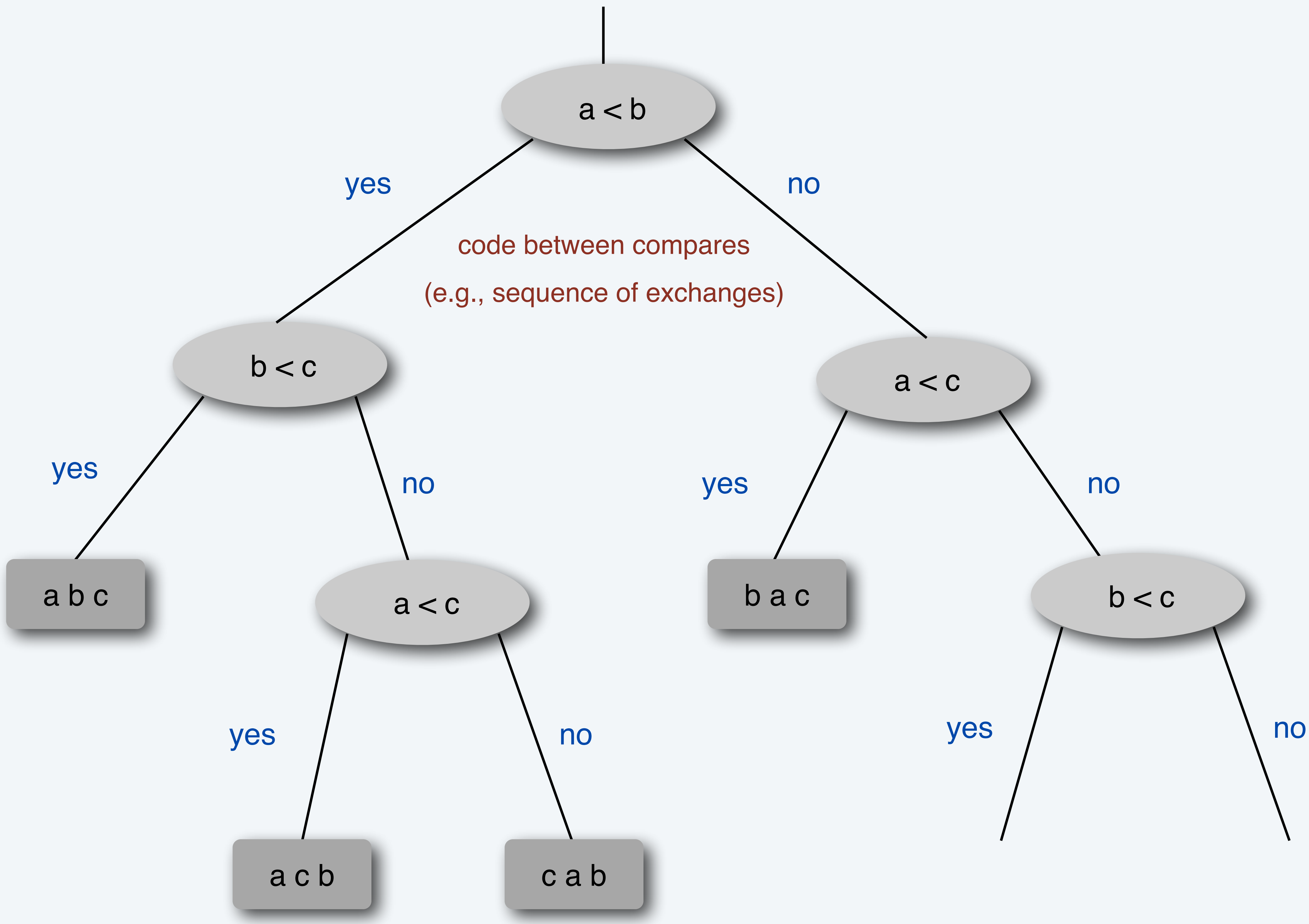
Comparison tree (for 3 distinct keys a, b, and c)



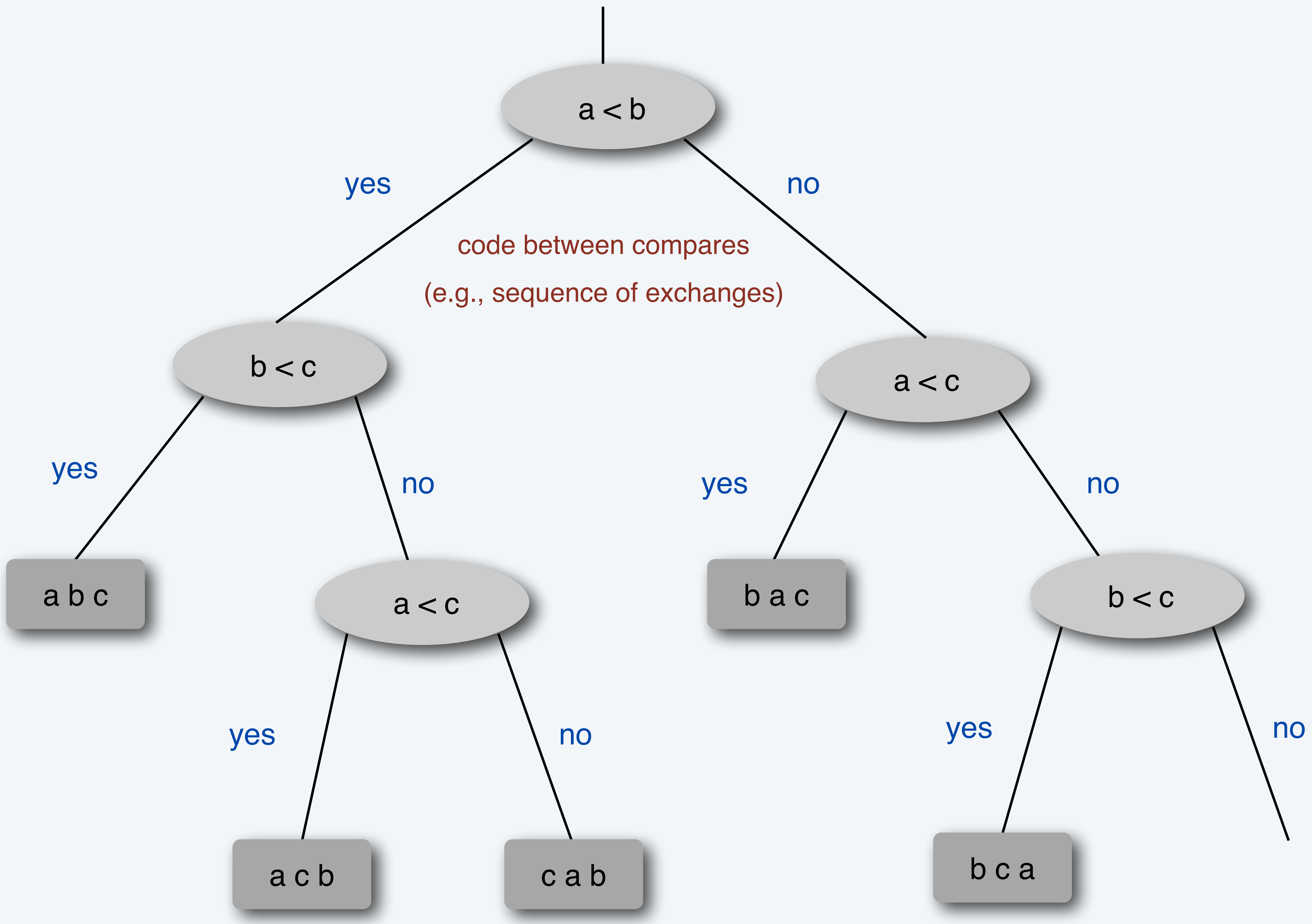
Comparison tree (for 3 distinct keys a, b, and c)



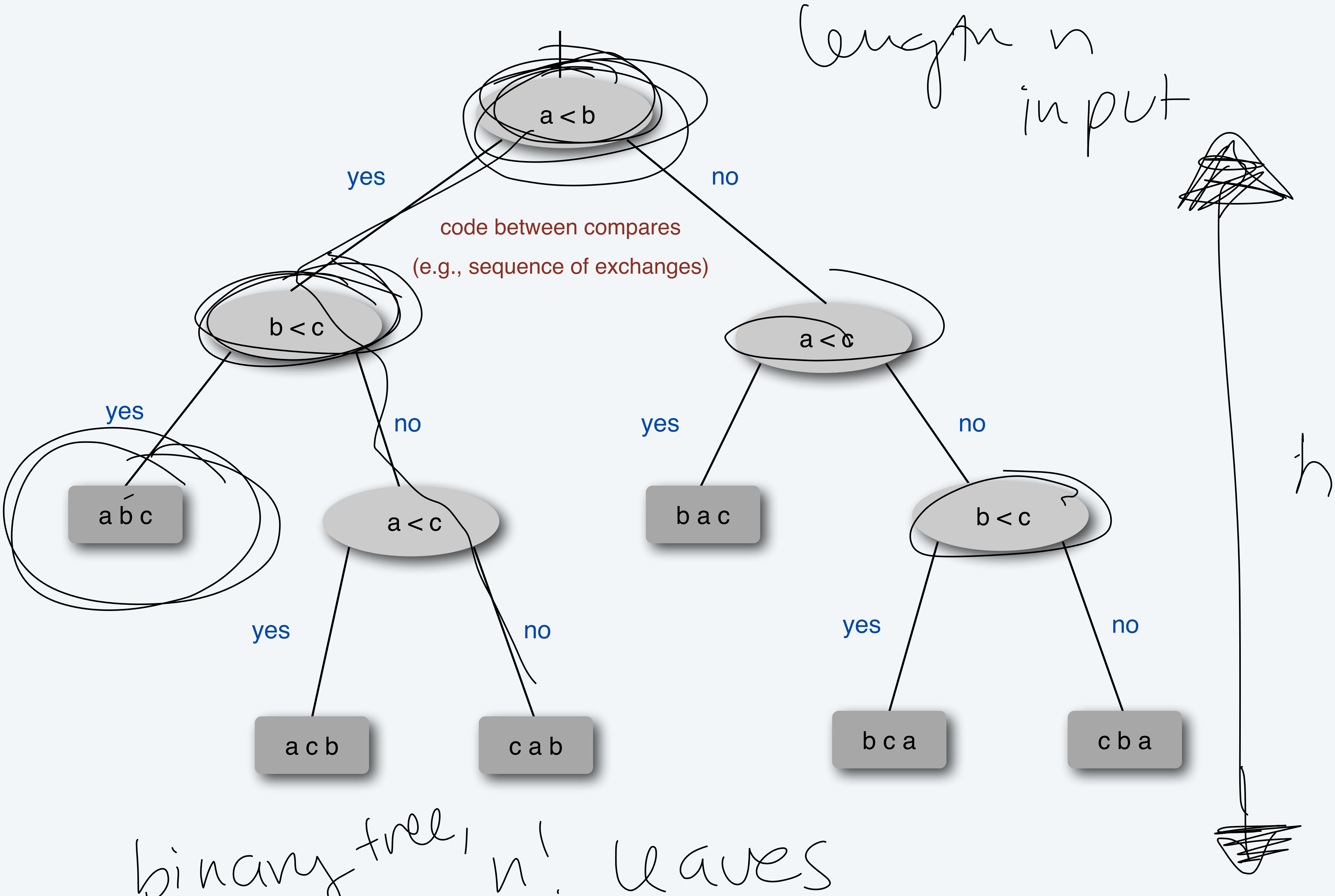
Comparison tree (for 3 distinct keys a, b, and c)



Comparison tree (for 3 distinct keys a, b, and c)



Comparison tree (for 3 distinct keys a, b, and c)



Sorting lower bound

Theorem. Any deterministic compare-based sorting algorithm must make $\Omega(n \log n)$ compares in the worst-case.

Sorting lower bound

Theorem. Any deterministic compare-based sorting algorithm must make $\Omega(n \log n)$ compares in the worst-case.

Pf. [information theoretic]

Sorting lower bound

Theorem. Any deterministic compare-based sorting algorithm must make $\Omega(n \log n)$ compares in the worst-case.

Pf. [information theoretic]

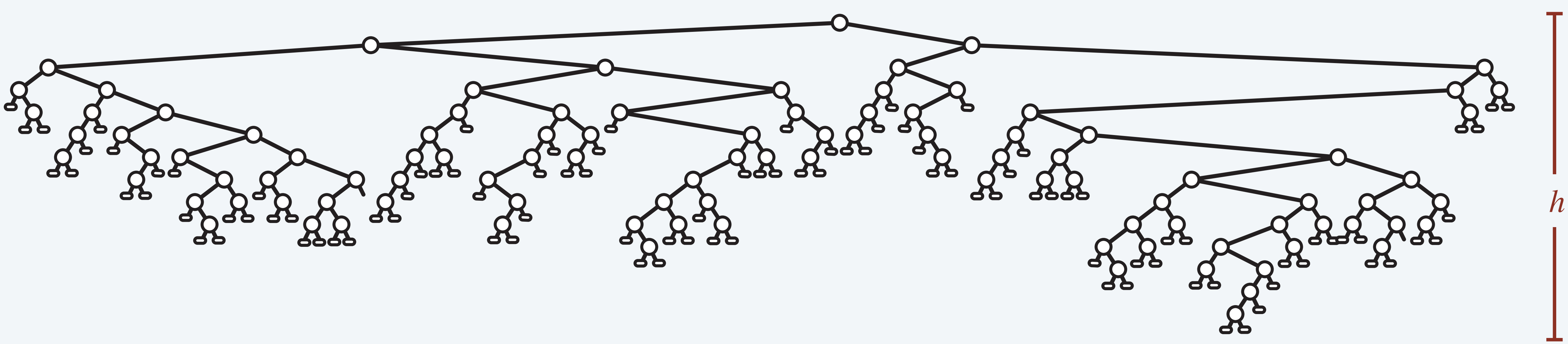
- Assume array consists of n distinct values a_1 through a_n.

Sorting lower bound

Theorem. Any deterministic compare-based sorting algorithm must make $\Omega(n \log n)$ compares in the worst-case.

Pf. [information theoretic]

- Assume array consists of n distinct values a_1 through a_n.
- Worst-case number of compares = height h of pruned comparison tree.

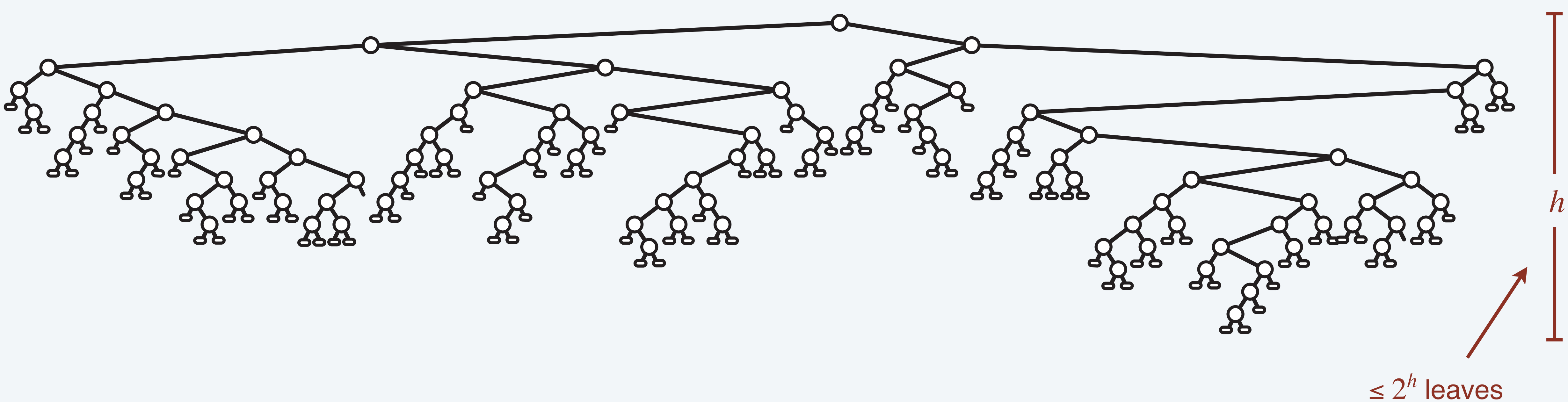


Sorting lower bound

Theorem. Any deterministic compare-based sorting algorithm must make $\Omega(n \log n)$ compares in the worst-case.

Pf. [information theoretic]

- Assume array consists of n distinct values a_1 through a_n.
- Worst-case number of compares = height h of pruned comparison tree.
- Binary tree of height h has ≤ 2^h leaves.

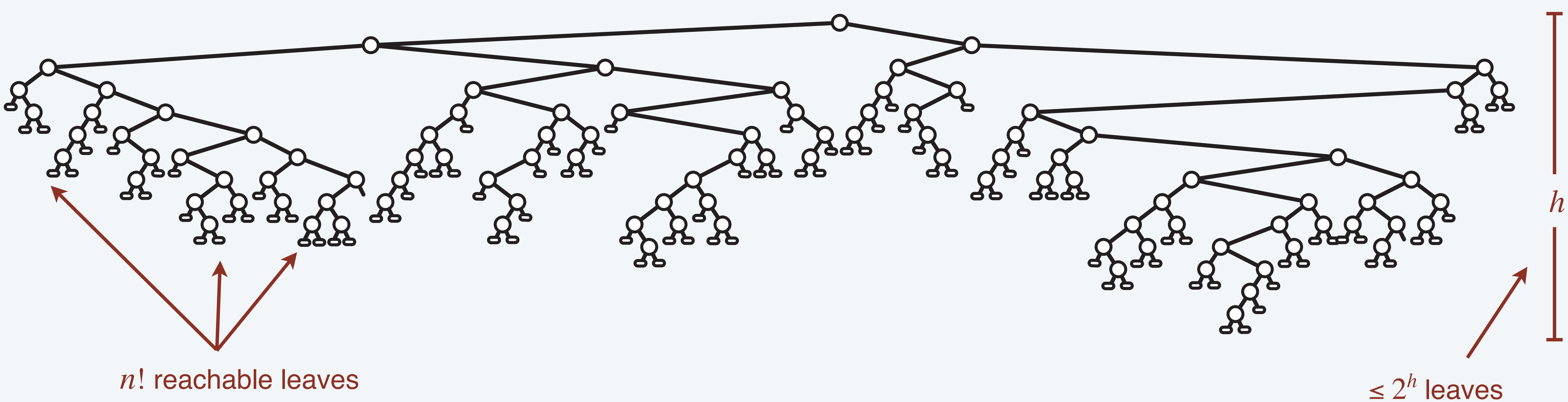


Sorting lower bound

Theorem. Any deterministic compare-based sorting algorithm must make $\Omega(n \log n)$ compares in the worst-case.

Pf. [information theoretic]

- Assume array consists of n distinct values a_1 through a_n.
- Worst-case number of compares = height h of pruned comparison tree.
- Binary tree of height h has ≤ 2^h leaves.
- $n!$ different orderings $\Rightarrow$ $n!$ reachable leaves.



Sorting lower bound

Theorem. Any deterministic compare-based sorting algorithm must make $\Omega(n \log n)$ compares in the worst-case.

Pf. [information theoretic]

- Assume array consists of n distinct values a_1 through a_n.
- Worst-case number of compares = height h of pruned comparison tree.
- Binary tree of height h has ≤ 2^h leaves.
- $n\,!$ different orderings $\Rightarrow$ $n\,!$ reachable leaves.

$$2^h \;\geq\; \#\text{ reachable leaves } \;=\; n\,!$$

Sorting lower bound

Theorem. Any deterministic compare-based sorting algorithm must make $\Omega(n \log n)$ compares in the worst-case.

Pf. [information theoretic]

- Assume array consists of n distinct values a_1 through a_n.
- Worst-case number of compares = height h of pruned comparison tree.
- Binary tree of height h has $\leq 2^h$ leaves.
- $n\,!$ different orderings $\Rightarrow$ $n\,!$ reachable leaves.

$$2^h \;\geq\; \#\text{ reachable leaves } \;=\; n\,!$$

$$\Rightarrow \;\; h \;\geq\; \log_2 (n!)$$

Sorting lower bound

Theorem. Any deterministic compare-based sorting algorithm must make $\Omega(n \log n)$ compares in the worst-case.

Pf. [information theoretic]

- Assume array consists of n distinct values a_1 through a_n.
- Worst-case number of compares = height h of pruned comparison tree.
- Binary tree of height h has $\leq 2^h$ leaves.
- $n!$ different orderings $\Rightarrow$ $n!$ reachable leaves.

$$2^h \geq \text{\# reachable leaves} = n!$$

$$\Rightarrow h \geq \log_2(n!)$$

$$\geq n \log_2 n - n / \ln 2 \quad \blacksquare$$

Stirling's formula

Sorting lower bound

Theorem. Any deterministic compare-based sorting algorithm must make $\Omega(n \log n)$ compares in the worst-case.

Pf. [information theoretic]

- Assume array consists of n distinct values a_1 through a_n.
- Worst-case number of compares = height h of pruned comparison tree.
- Binary tree of height h has $\leq 2^h$ leaves.
- $n!$ different orderings $\Rightarrow$ $n!$ reachable leaves.

$$2^h \geq \text{\# reachable leaves} = n!$$

$$\Rightarrow h \geq \log_2(n!)$$

$$\geq n \log_2 n - n / \ln 2 \quad ▪$$

Stirling's formula

Note. Lower bound can be extended to include randomized algorithms.

Counting inversions

Counting inversions

Music site tries to match your song preferences with others.

Counting inversions

Music site tries to match your song preferences with others.

- You rank n songs.

Counting inversions

Music site tries to match your song preferences with others.

- You rank n songs.
- Music site consults database to find people with similar tastes.

Counting inversions

Music site tries to match your song preferences with others.

- You rank n songs.
- Music site consults database to find people with similar tastes.

Similarity metric: number of inversions between two rankings.

Counting inversions

Music site tries to match your song preferences with others.

- You rank n songs.
- Music site consults database to find people with similar tastes.

Similarity metric: number of inversions between two rankings.

- My rank: $1, 2, \ldots, n$.

	A	B	C	D	E
me	1	2	3	4	5

Counting inversions

Music site tries to match your song preferences with others.

- You rank n songs.
- Music site consults database to find people with similar tastes.

Similarity metric: number of inversions between two rankings.

- My rank: $1, 2, \ldots, n$.
- Your rank: $a_1, a_2, \ldots, a_n$.

	A	B	C	D	E
me	1	2	3	4	5
you	1	3	4	2	5

Counting inversions

Music site tries to match your song preferences with others.

- You rank n songs.
- Music site consults database to find people with similar tastes.

Similarity metric: number of inversions between two rankings.

- My rank: $1, 2, \ldots, n$.
- Your rank: $a_1, a_2, \ldots, a_n$.
- Songs i and j are inverted if $i < j$, but $a_i > a_j$.

	A	B	C	D	E
me	1	2	3	4	5
you	1	3	4	2	5

2 inversions: 3-2, 4-2