Design & Analysis of Algorithms

CS11313 - Fall 2023

Sorting Lower Bound

Sorting Lower Bound

Proposition. Any comparison-based sorting algorithm performs at least $\sim n \log n$ compares in the worst case.

Sorting Lower Bound

Proposition. Any comparison-based sorting algorithm performs at least $\sim n \log n$ compares in the worst case.

are there sorting algorithms that
are not comparison-based?
Yes! (e.g. Radix Sort)

Sorting Lower Bound

Proposition. Any comparison-based sorting algorithm performs at least $\sim n \log n$ compares in the worst case.

proposition holds in the worst case only. E.g. Insertion sort does $\Theta(n)$ comparisons in the best case.

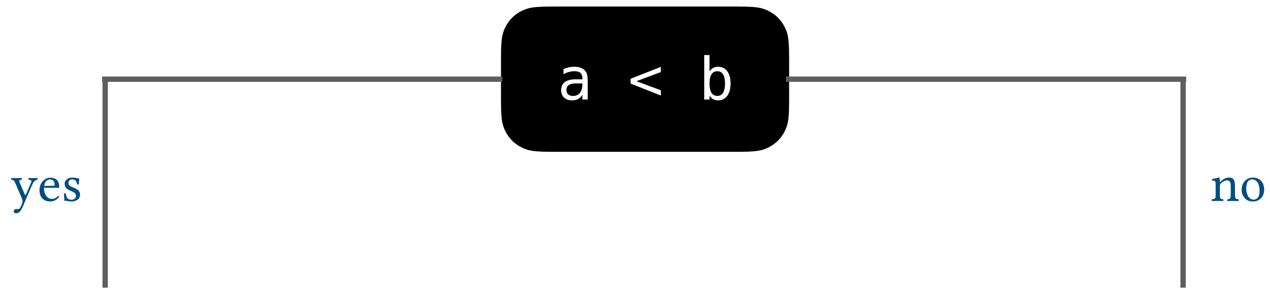
Sorting Lower Bound

Proposition. Any comparison-based sorting algorithm performs at least $\sim n \log n$ compares in the worst case.

Put another way. For any comparison-based sorting algorithm, there must be at least one sequence of elements for which the sorting algorithm needs $\sim n \log n$ comparisons to sort.

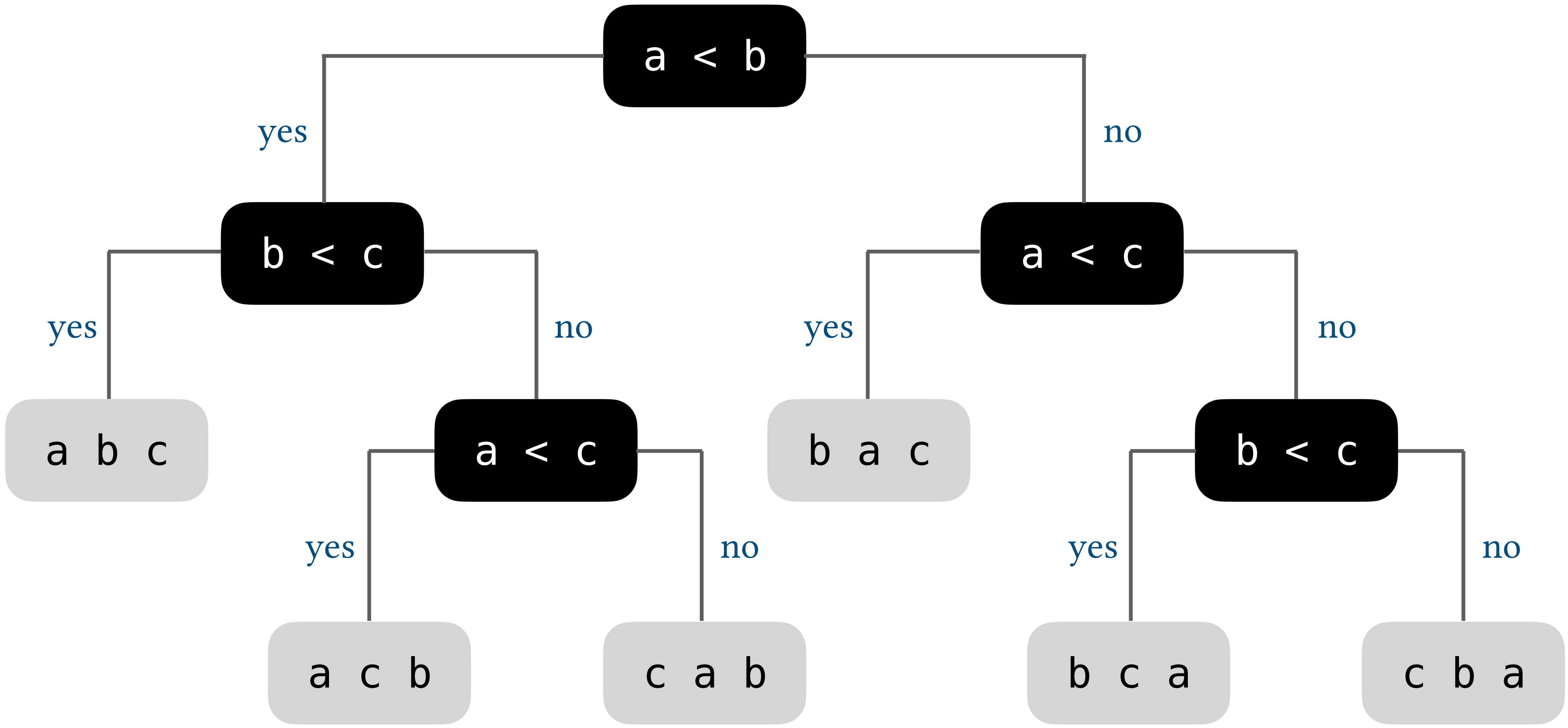
Sorting Lower Bound

A comparison tree for three distinct keys (a, b and c)



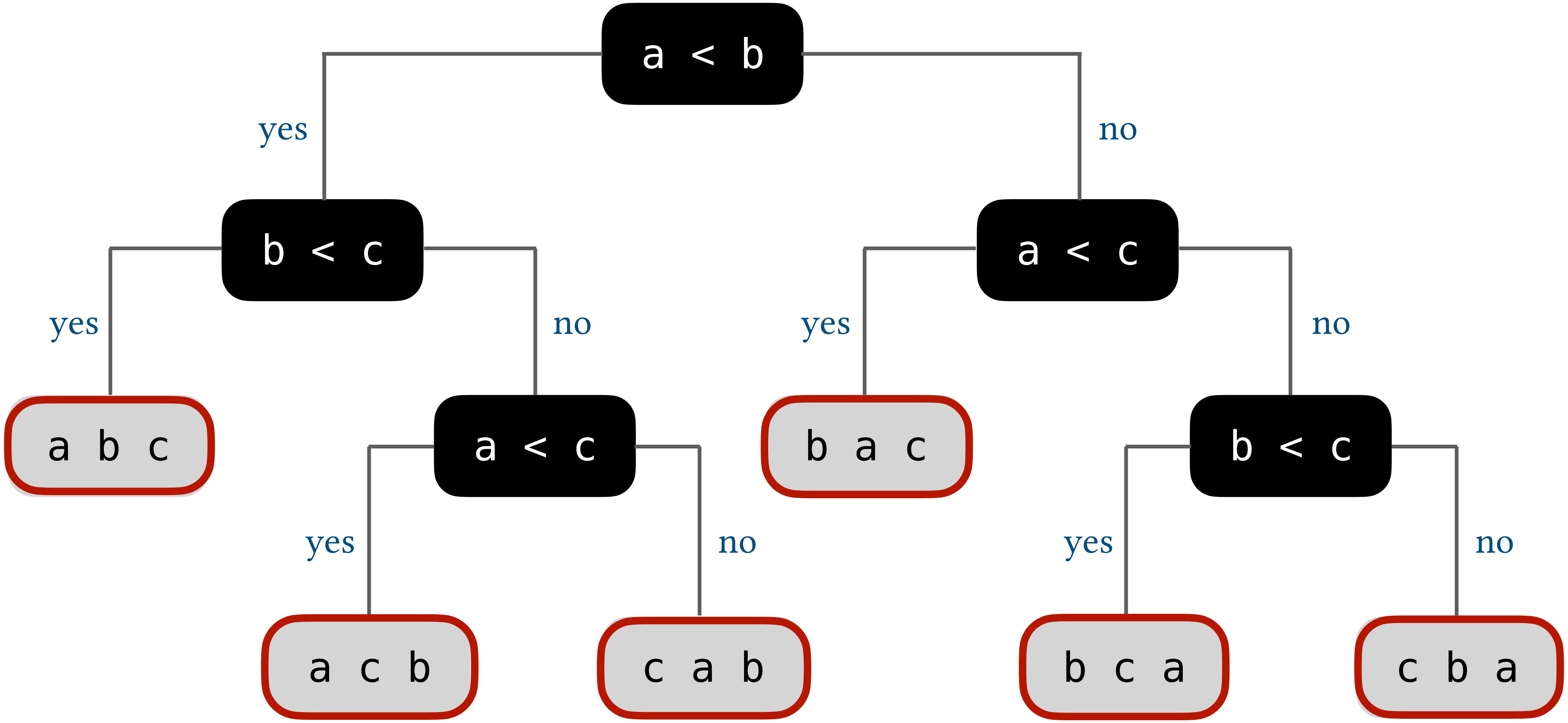
Sorting Lower Bound

A comparison tree for three distinct keys (a, b and c)



Sorting Lower Bound

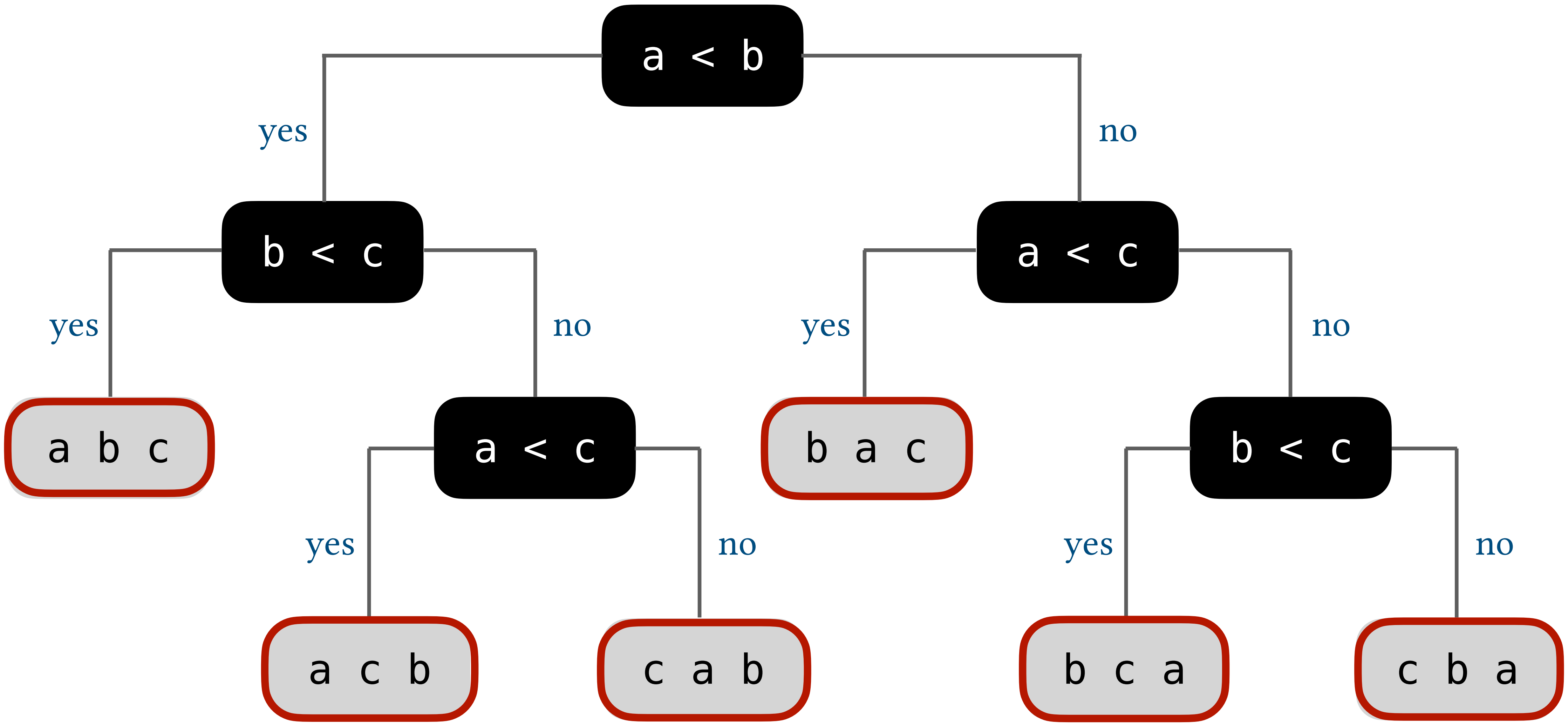
A comparison tree for three distinct keys (a, b and c)



There are $n!$ unique orderings making $n!$ leaves

Sorting Lower Bound

A comparison tree for three distinct keys (a, b and c)

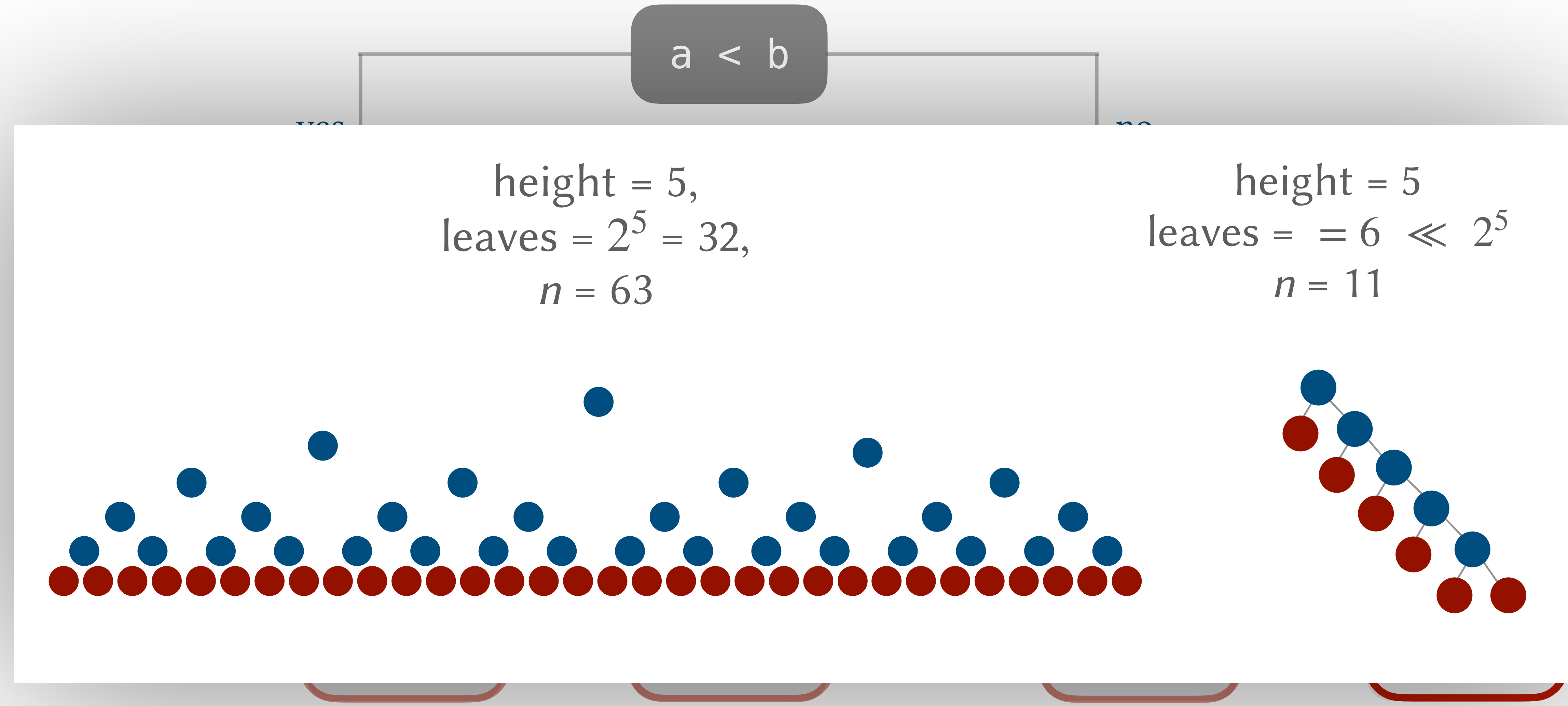


There are $n!$ unique orderings making $n!$ leaves

$$\# \text{ of leaves} \quad \leq \quad 2^{\text{height}}$$

Sorting Lower Bound

A comparison tree for three distinct keys (a, b and c)

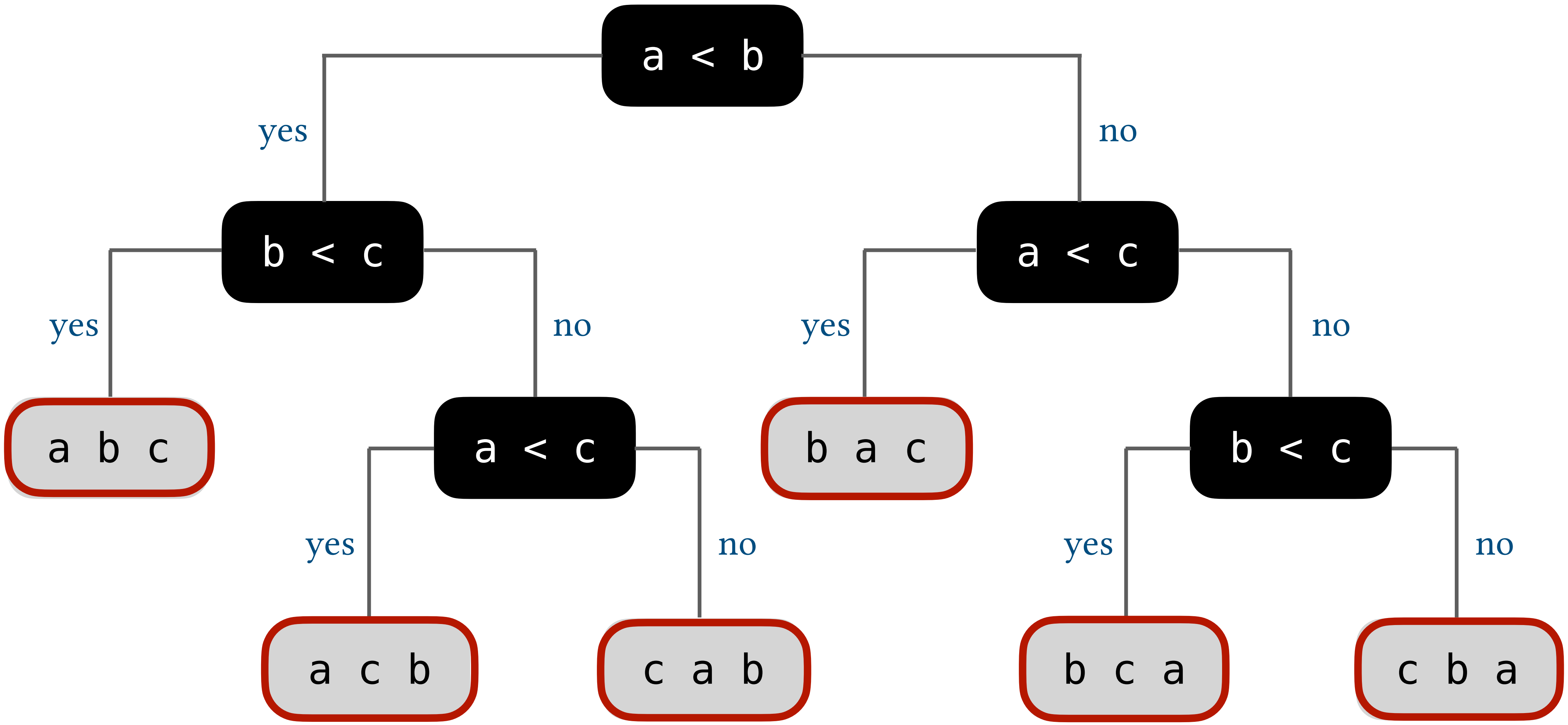


There are $n!$ unique orderings making $n!$ leaves

\# of leaves $\leq$ 2^{height}

Sorting Lower Bound

A comparison tree for three distinct keys (a, b and c)

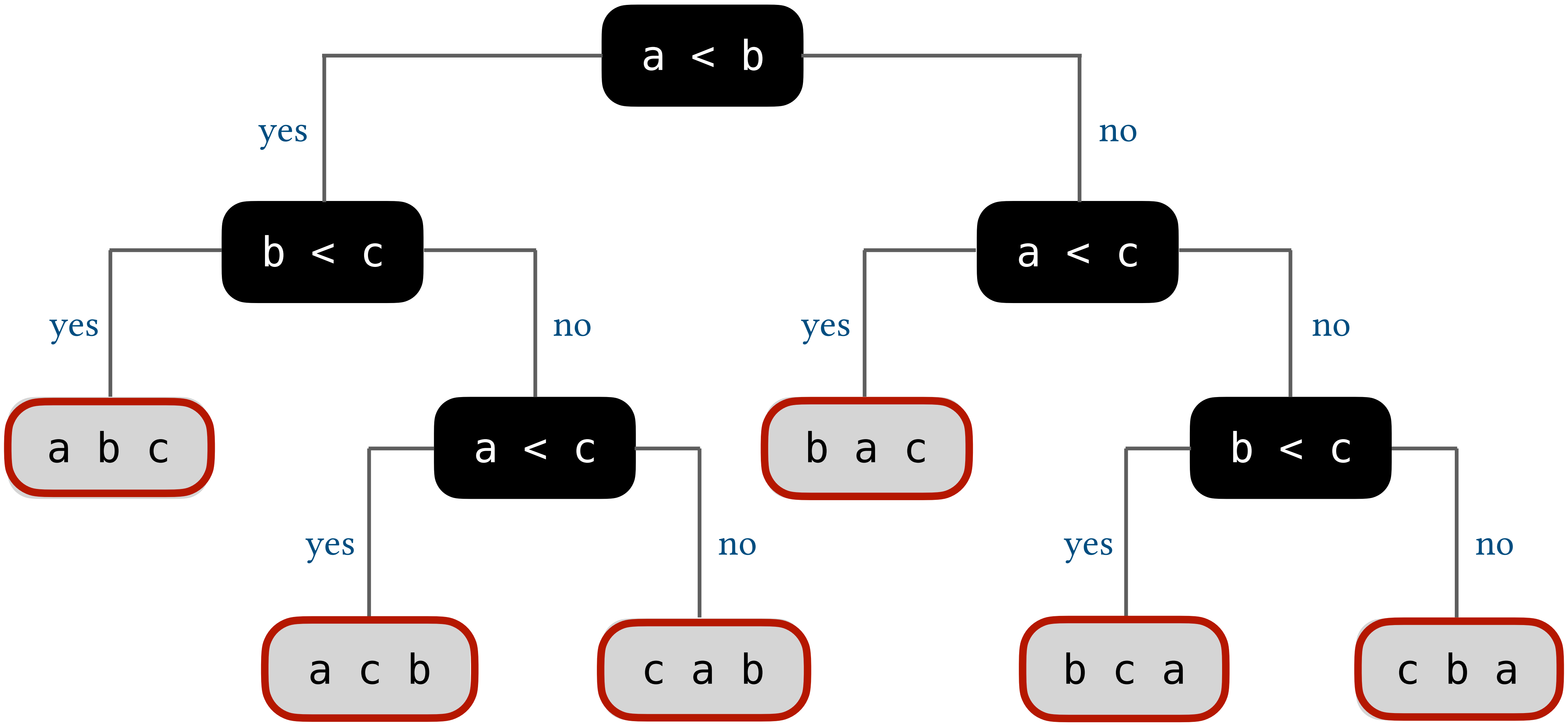


There are $n!$ unique orderings making $n!$ leaves

$$\begin{aligned} \#\text{ of leaves} &\leq 2^{\text{height}} \\ n! &\leq 2^{\text{height}} \end{aligned}$$

Sorting Lower Bound

A comparison tree for three distinct keys (a, b and c)

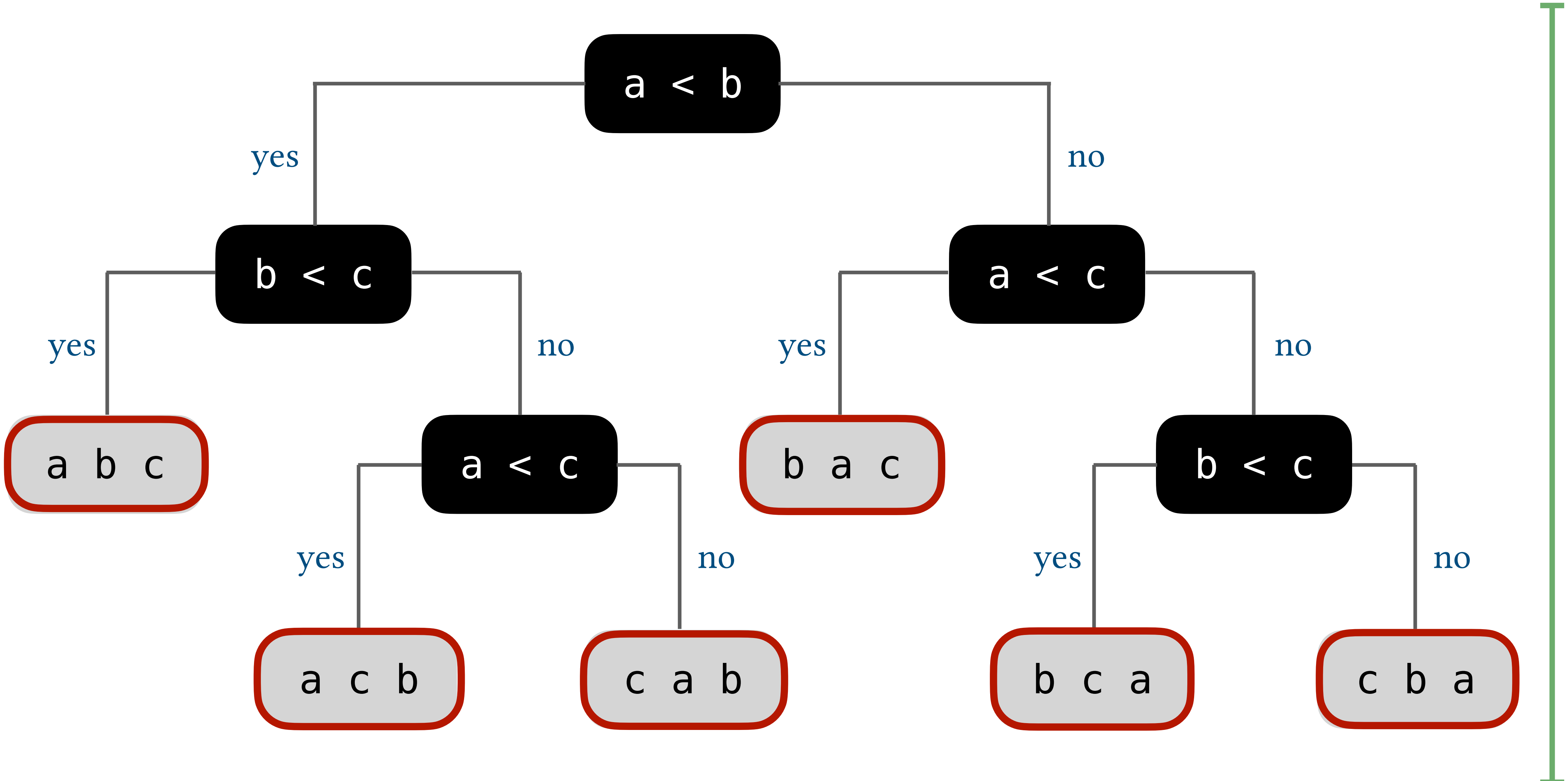


There are $n!$ unique orderings making $n!$ leaves

$$
\begin{aligned}
\text{\# of leaves} &\leq 2^{\text{height}} \\
n! &\leq 2^{\text{height}} \\
\log(n!) &\leq \log(2^{\text{height}})
\end{aligned}
$$

Sorting Lower Bound

A comparison tree for three distinct keys (a, b and c)



$h \geq \log_2(n!)$

There are $n!$ unique orderings making $n!$ leaves

$$
\begin{aligned}
\text{\# of leaves} &\leq 2^{\text{height}} \\
n! &\leq 2^{\text{height}} \\
\log(n!) &\leq \log(2^{\text{height}}) \\
\sim n\log(n) &\leq \text{height}
\end{aligned}
$$

height represents # of comparisons

Sorting Lower Bound

Proposition. Any comparison-based sorting algorithm performs $\Omega(n \log n)$ compares in the worst case to sort an arbitrary array of size n.

Proof Sketch.

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

Sorting Lower Bound

Proposition. Any comparison-based sorting algorithm performs $\Omega(n \log n)$ compares in the worst case to sort an arbitrary array of size n.

Proof Sketch.

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

- There are $n!$ unique orderings for this array.
  (any sorting algorithm must be able to distinguish between these $n!$ permutations).

Sorting Lower Bound

Proposition. Any comparison-based sorting algorithm performs $\Omega(n \log n)$ compares in the worst case to sort an arbitrary array of size n.

Proof Sketch.

- Assume the array consists of n distinct values a_1 through a_n .
- There are $n!$ unique orderings for this array.
  (any sorting algorithm must be able to distinguish between these $n!$ permutations).
- Consider a binary decision tree, where each node is labeled with a comparison between two elements ($a_i < a_j$) and each leaf is a possible ordering for the array.
  (path from the root to a leaf represents a run of a sorting algorithm).

Sorting Lower Bound

Proposition. Any comparison-based sorting algorithm performs $\Omega(n \log n)$ compares in the worst case to sort an arbitrary array of size n.

Proof Sketch.

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

- There are $n!$ unique orderings for this array.
  (any sorting algorithm must be able to distinguish between these $n!$ permutations).

- Consider a binary decision tree, where each node is labeled with a comparison between two elements ($a_i < a_j$) and each leaf is a possible ordering for the array.
  (path from the root to a leaf represents a run of a sorting algorithm).

- The tree has $n!$ leaves.

Sorting Lower Bound

Proposition. Any comparison-based sorting algorithm performs $\Omega(n \log n)$ compares in the worst case to sort an arbitrary array of size n.

Proof Sketch.

- Assume the array consists of n distinct values a_1 through a_n .
- There are $n!$ unique orderings for this array.
  (any sorting algorithm must be able to distinguish between these $n!$ permutations).
- Consider a binary decision tree, where each node is labeled with a comparison between two elements ($a_i < a_j$) and each leaf is a possible ordering for the array.
  (path from the root to a leaf represents a run of a sorting algorithm).
- The tree has $n!$ leaves.
- The height of a binary tree with $n!$ leaves is $\geq \log_2(n!)$.
  (the height of the tree is $\log_2(n!)$ if it is a complete tree and possibly more if it is not).

Sorting Lower Bound

Proposition. Any comparison-based sorting algorithm performs $\Omega(n \log n)$ compares in the worst case to sort an arbitrary array of size n.

Proof Sketch.

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

- There are $n!$ unique orderings for this array.
  (any sorting algorithm must be able to distinguish between these $n!$ permutations).

- Consider a binary decision tree, where each node is labeled with a comparison between two elements ($a_i < a_j$) and each leaf is a possible ordering for the array.
  (path from the root to a leaf represents a run of a sorting algorithm).

- The tree has $n!$ leaves.

- The height of a binary tree with $n!$ leaves is $\geq \log_2(n!)$.
  (the height of the tree is $\log_2(n!)$ if it is a complete tree and possibly more if it is not).

- If the longest path in the tree is $\geq \log_2(n!)$ then there must always be a sequence of input that requires $\log_2(n!)$ comparisons to be sorted.
  (the height of a binary tree is the length of the longest path from the root to a leaf).