UNIT 5: Decision Trees

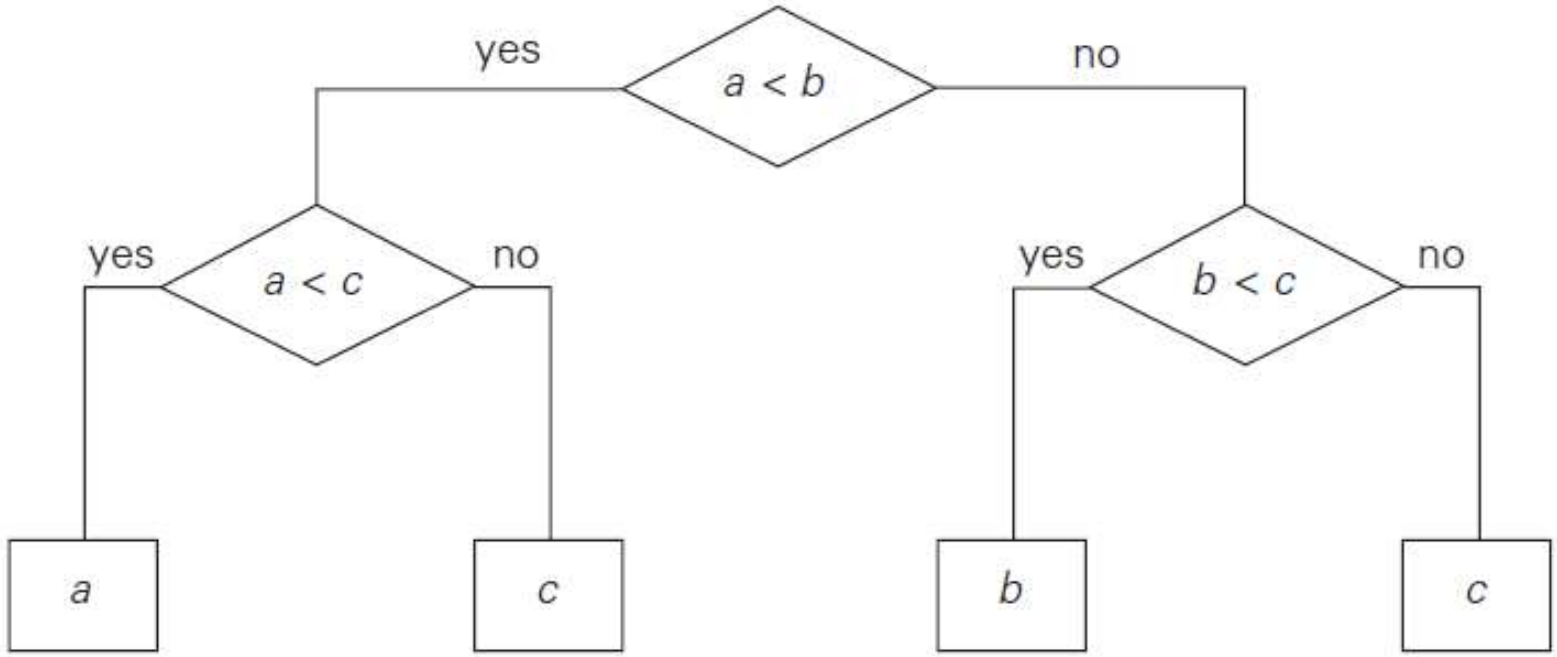
Decision Trees for Sorting

Many important **sorting and searching algorithms**, work by comparing items of their inputs.

Decision tree can be used as device to study the performance of such algorithms

By studying properties of decision trees for such algorithms, we can derive important **lower bounds** on their time efficiencies.

Decision tree for finding a minimum of three numbers



Each leaf represents a possible outcome of the algorithm's run on some input of size n.

Decision tree

- The number of leaves must be at least as large as the number of possible outcomes.
- The algorithm's work on a particular input of size n can be traced by a path from the root to a leaf in its decision tree, and the **number of comparisons made by the algorithm on such a run is equal to the length of this path**.
- The number of comparisons in the worst case is equal to the **height of the algorithm's decision tree**.

i.e., any binary tree with l leaves and height h,

$$h \geq \lceil \log_2 l \rceil$$

Idea: a tree with a given number of leaves, which is dictated by the number of possible outcomes, has to be tall enough to have that many leaves.

Revisiting selection sort

$$\underbrace{A_0 \le A_1 \le \cdots \le A_{i-1}}_{\text{in their final positions}} \;\Big|\; \underbrace{A_i, \ldots, A_{min}, \ldots, A_{n-1}}_{\text{the last } n-i \text{ elements}}$$

ALGORITHM *SelectionSort*$(A[0..n-1])$

//Sorts a given array by selection sort
//Input: An array $A[0..n-1]$ of orderable elements
//Output: Array $A[0..n-1]$ sorted in nondecreasing order
for $i \leftarrow 0$ **to** $n-2$ **do**
    $min \leftarrow i$
    for $j \leftarrow i+1$ **to** $n-1$ **do**
        if $A[j] < A[min]$ $min \leftarrow j$
    swap $A[i]$ and $A[min]$

89	45	68	90	29	34	**17**
17	| 45	68	90	**29**	34	89
17	29	| 68	90	45	**34**	89
17	29	34	| 90	**45**	68	89
17	29	34	45	| 90	**68**	89
17	29	34	45	68	| 90	**89**
17	29	34	45	68	89	| 90

$$C(n) = \sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} 1 = \sum_{i=0}^{n-2} [(n-1) - (i+1) + 1] = \sum_{i=0}^{n-2} (n-1-i)$$

NOTE

- An outcome of a sorting algorithm can be interpreted as finding a permutation of the element indices of an input list that puts the list's elements in ascending order.

Example:

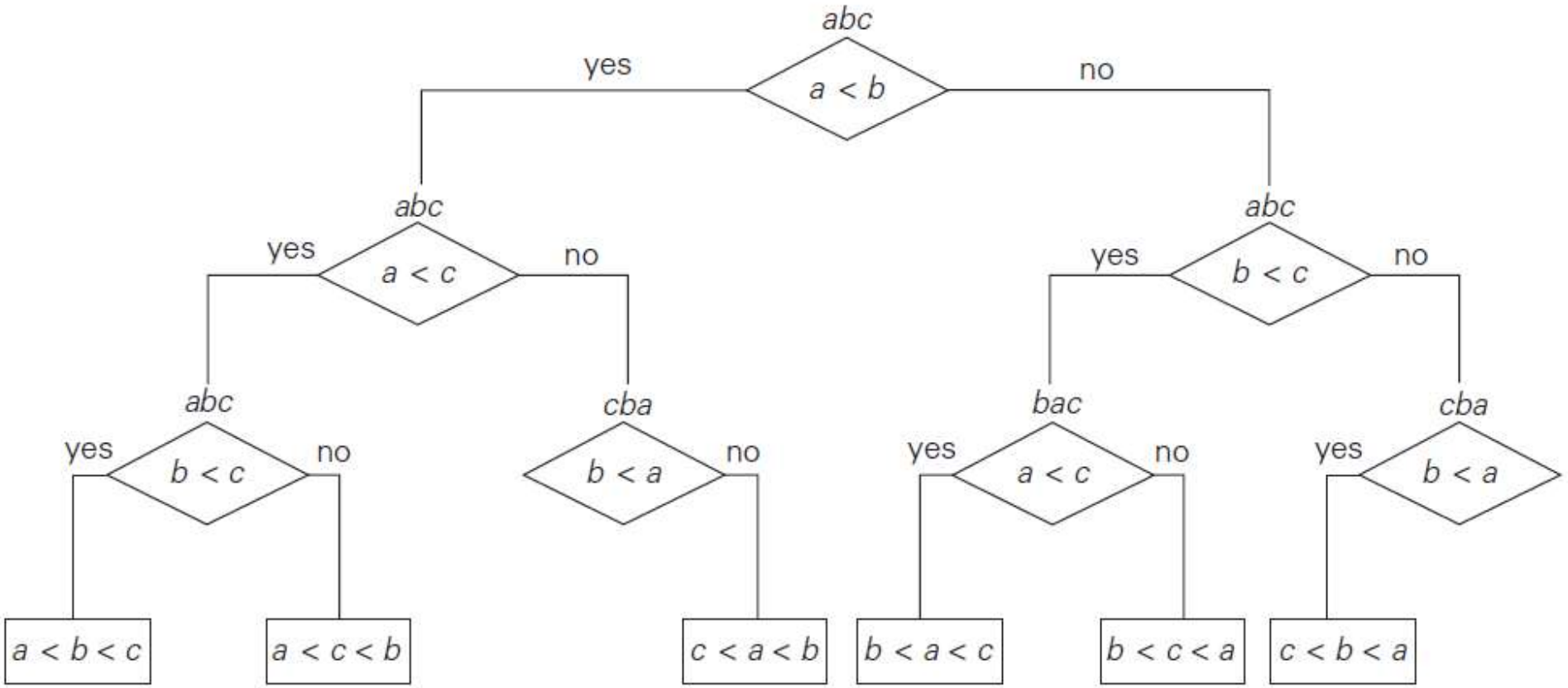
consider a list with three elements to be sorted: { a, b, c}

1. a<b<c
2. a<c<b
3. b<a<c
4. b<c<a
5. c<a<b
6. c<b<a

Hence we have 6 possible permutations.

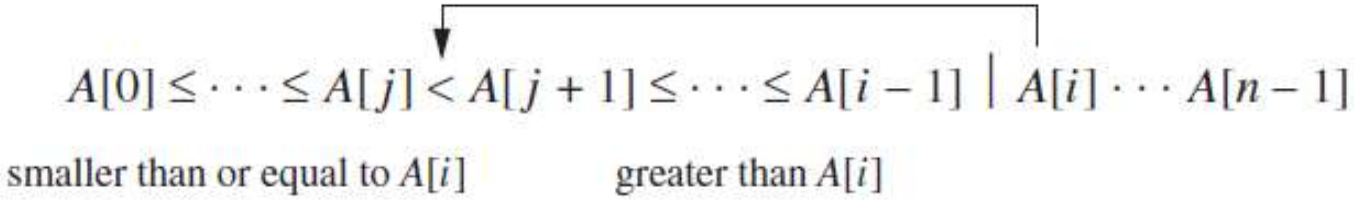
In general, the number of possible outcomes for sorting an arbitrary n-element list is equal to n!

Decision Trees for Sorting (selection sort)



Decision tree for the tree-element selection sort. A triple above a node indicates the state of the array being sorted. Note two redundant comparisons $b < a$ with a single possible outcome because of the results of some previously made comparisons.

Revisiting insertion sort



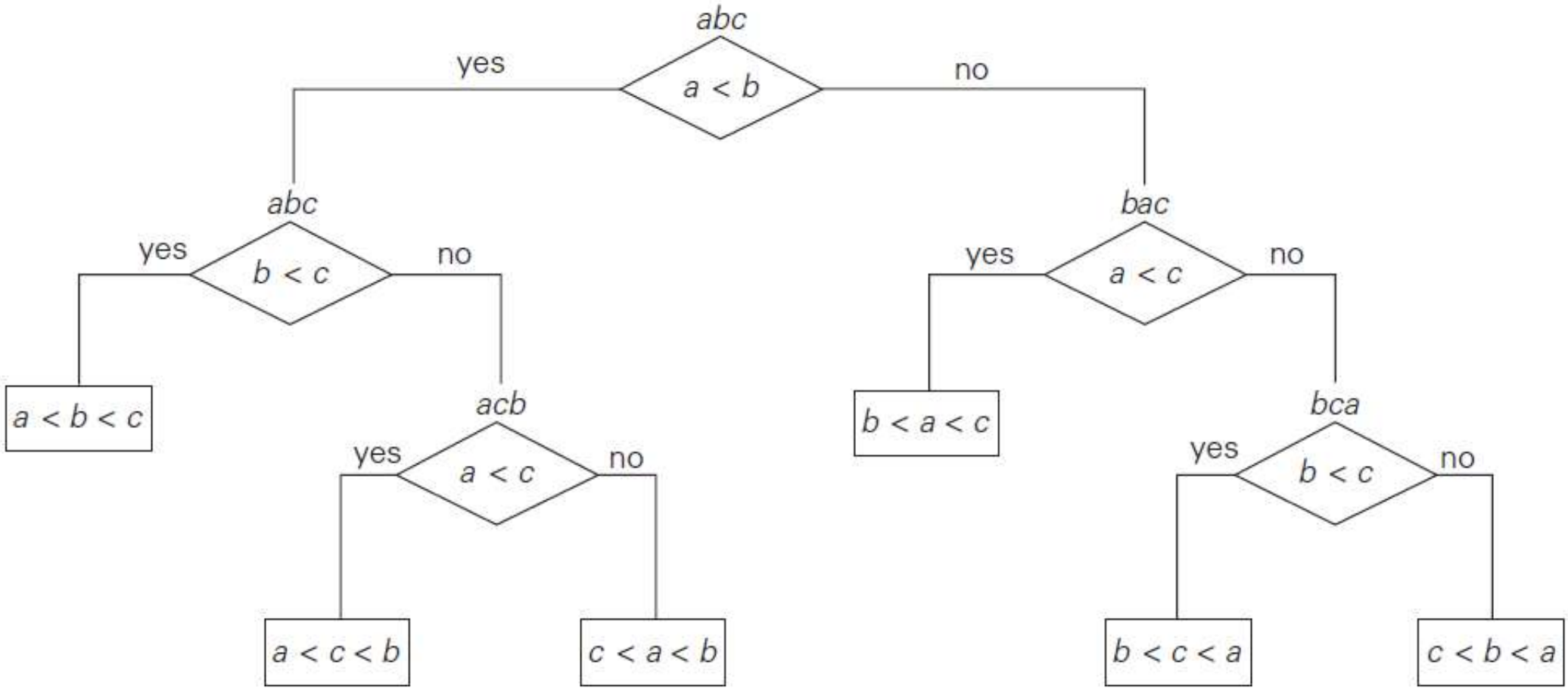
ALGORITHM *InsertionSort*$(A[0..n-1])$

//Sorts a given array by insertion sort
//Input: An array $A[0..n-1]$ of n orderable elements
//Output: Array $A[0..n-1]$ sorted in nondecreasing order
for $i \leftarrow 1$ **to** $n-1$ **do**
    $v \leftarrow A[i]$
    $j \leftarrow i-1$
    while $j \ge 0$ **and** $A[j] > v$ **do**
        $A[j+1] \leftarrow A[j]$
        $j \leftarrow j-1$
    $A[j+1] \leftarrow v$

89 | **45** 68 90 29 34 17
45 89 | **68** 90 29 34 17
45 68 89 | **90** 29 34 17
45 68 89 90 | **29** 34 17
29 45 68 89 90 | **34** 17
29 34 45 68 89 90 | **17**
17 29 34 45 68 89 90

$$C_{worst}(n) = \sum_{i=1}^{n-1} \sum_{j=0}^{i-1} 1 = \sum_{i=1}^{n-1} i = \frac{(n-1)n}{2} \in \Theta(n^2)$$

Decision tree for the three element insertion sort



Decision trees can be used for analyzing the worst and average-case efficiencies of comparison-based sorting algorithms

Worst case number of comparisons: height of a binary decision tree

- the worst-case number of comparisons made by such an algorithm cannot be less than **log n!**

$$C_{worst}(n) \geq \lceil \log_2 n! \rceil .$$

Using Stirling's formula for $n!$, we get

$$\lceil \log_2 n! \rceil \approx \log_2 \sqrt{2\pi n}(n/e)^n = n \log_2 n - n \log_2 e + \frac{\log_2 n}{2} + \frac{\log_2 2\pi}{2} \approx n \log_2 n$$

Conclusion: about n $\log_2$ n comparisons are necessary in the worst case to sort an arbitrary n-element list by any comparison-based sorting algorithm

Average case number of comparisons:

Average path length from root to the leaves

- We can compute the average number of comparisons for a particular algorithm as the average depth of its decision tree's leaves

the three-element insertion sort is

$=(2 + 3 + 3 + 2 + 3 + 3)/6 = 2\frac{2}{3}$

Conclusion: Under the standard assumption that all n! outcomes of sorting are equally likely, the lower bound on the average number of comparisons is

$$C_{avg}(n) \geq \log_2 n!$$

$$C_{avg}(n) \geq \log_2 n!$$

- Using Stirling's formula, the lower bound is about $n\log_2 n$.
- the lower bounds for the average and worst cases are almost identical.
- however, these bounds are obtained by maximizing the number of comparisons made in the average and worst cases, respectively.

- **For a particular sorting algorithm, the average-case efficiency can, of course, be significantly better than their worst-case efficiency.**