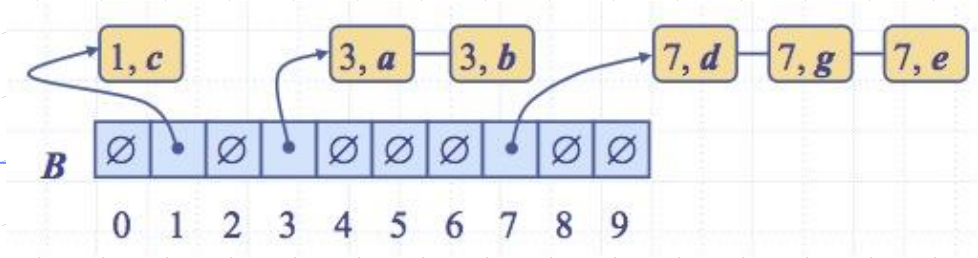
Lesson 7

Lower Bound on Comparison-Based Algorithms: *Discovering the Range of Natural Law*



Wholeness of the Lesson

Using the technique of decision trees, one establishes the following lower bound on comparison-based sorting algorithms: Every comparison-based sorting algorithm has at least one worst case for which running time is $\Omega(n \log n)$. Bucket Sort and its relatives, under suitable conditions, run in linear time in the worst case, but are not comparison-based algorithms. Each level of existence has its own laws of nature. The laws of nature that operate at one level of existence may not apply to other levels of existence.

Road Map for Lower Bound on Comparison Based Sorting

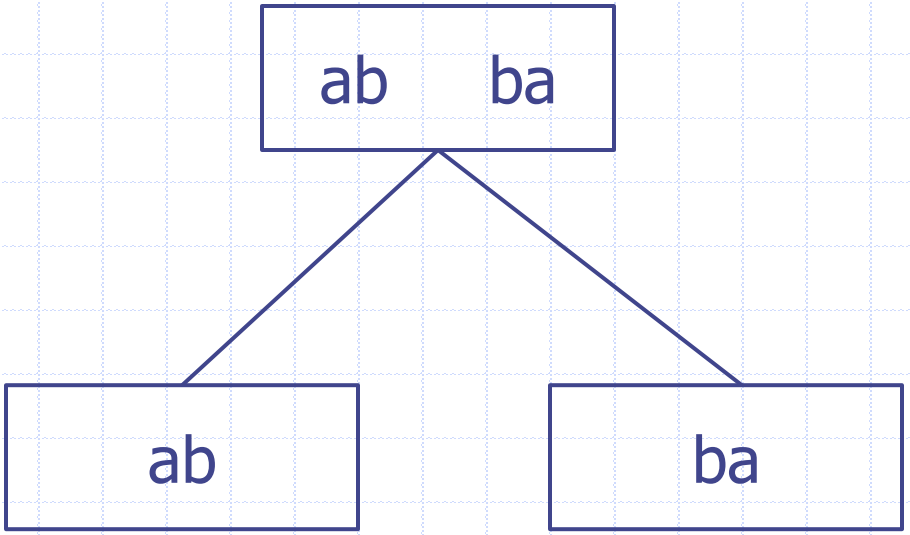
1. Given n, there are n! permutations.
2. Therefore any decision tree to sort n items will have n! leaves.
3. Since the decision tree is a binary tree, it must at least have log (n!) height.
4. Since the height of the decision tree is the same as the number of comparisons, any comparison based sorting algorithm must perform log (n!) comparisons.
5. log (n!) is theta(n log n).
6. Thus lower bound for the comparison based sorting algorithms is big-Omega(n log n)

Sample Decision Tree

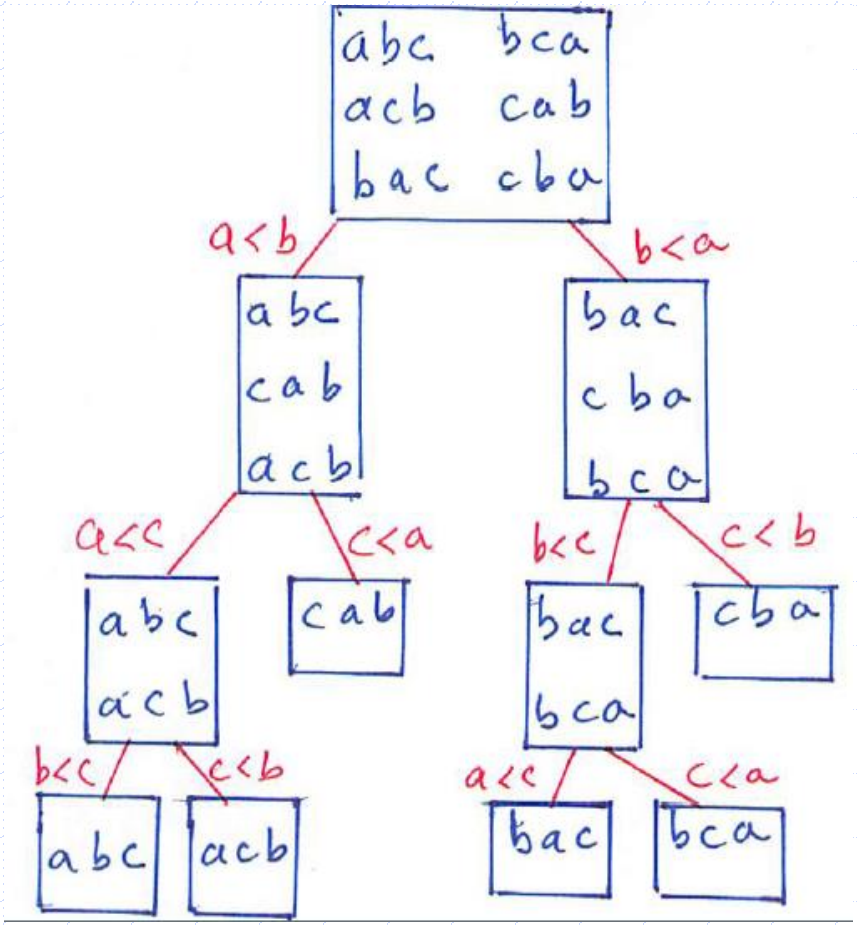
- Every comparison-based sorting algorithm, applied to a given array, can be represented by a decision tree
- To illustrate the technique, we use a simple but typical example: sorting 2, 3 and 4 element array of distinct values *a,b,c,d.*

Decision Tree for Sorting 2 Elements

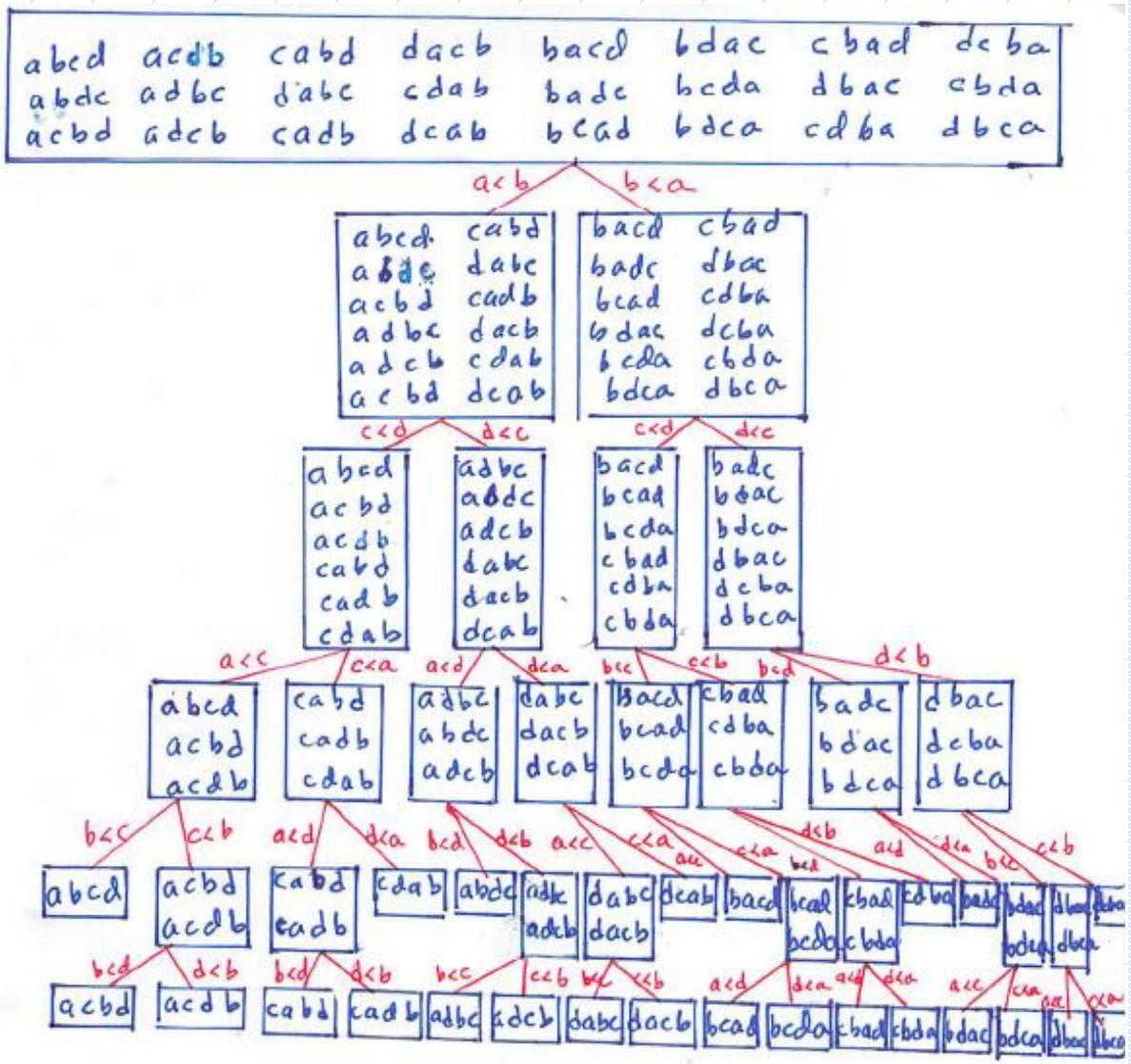
```
Read a, b
if (a < b)
    print(a, b)
else
    print(b, a)
```



Decision Tree for Sorting 3 Elements



Decision Tree for Sorting 4 Elements



Anatomy of the Decision Tree

- Each node of the tree represents all possible sorting outcomes that have not been eliminated by the comparisons done so far.
- The labels on the links of the tree represent comparison steps as the algorithm runs. Different algorithms will perform some of the comparison steps in a different sequence.
- A leaf in the decision tree represents a possible sorting outcome. The different paths to the leaves represent all possible ways three (as in slide 4 or four as in slide 5) distinct items could be put in sorted order; therefore, the leaves represent all possible arrangements of three (or four) distinct elements.

Strategy for Computing # Comparisons

- *Observation:* The number of comparisons performed in order to arrive at an arrangement found in a leaf node equals the depth of that leaf node in the decision tree.

- Therefore, to count # comparisons performed in the worst case, we determine the depth of the deepest node in the decision tree

Mathematical Observations

Suppose T is a binary tree with L leaves and height h.

- Lemma 1. $L \leq 2^h$.

Use Lemma 1 to prove the next important fact:

- Lemma 2. $h \geq \lceil \log L \rceil$.
- Lemma 3. A decision tree T representing a comparison-based sorting algorithm for an n-element array has n! leaves. [Proof: Follows because leaves represent all possible permutations of the n elements of the array]

Establishing the Lower Bound

Summary: Suppose we have a decision tree T for a sorting algorithm running on input of size n. Then T has n! leaves. So the height of T is at least $\lceil \log n! \rceil$ so some leaf in T has depth at least $\lceil \log n! \rceil$. Therefore, in the worst case, at least $\lceil \log n! \rceil$ comparisons are performed, so running time is $\Omega(\log n!)$. We determine the complexity class of log n!

The Computation of log n!

$$
\begin{aligned}
\log n! &= \log(1 \cdot 2 \cdot 3 \cdot \ldots \cdot n) \\
&= \log n + \log(n-1) + \log(n-2) + \ldots + \log 2 + \log 1 \\
&\geq \log n + \log(n-1) + \ldots + \log \frac{n}{2} \\
&\geq \sum_{i=1}^{n/2} \log \frac{n}{2} \\
&= \frac{n}{2} \cdot \log \frac{n}{2} \\
&= \frac{n}{2} \cdot (\log n - 1) \\
&= \frac{n}{2} \cdot \log n - \frac{n}{2} \\
&= \Omega(n \log n)
\end{aligned}
$$