Sorting Lower Bound
(CLRS 8.1)

We have seen several sorting algorithms. All have worst-case running time at least $\Omega(n \lg n)$. The natural question to ask is: **Can we do better than $\Theta(n \lg n)$ in the worst-case?** In other words, is there a sorting algorithm whose worst-case running time is asymptotically better than $\Theta(n \lg n)$?

This is a hard question. If we did not find a faster algorithm, it does not mean that there isn't one. Can we say that it is impossible to sort faster than $\Omega(n \lg n)$ in the worst case? If we could, then this would be what's called a *lower bound*.

> A *lower bound* for a problem is the worst-case running time of the best possible algorithm for that problem.

To prove a lower bound of $\Omega(n \lg n)$ for sorting, we would have to prove that no algorithm, however smart, could possibly be faster, in the worst-case, then $\Theta(n \lg n)$.

BUT, how to prove a lower bound, without going through all possible algorithms?!?!

To approach this type of problem we have to go back to what is an algorithm. The power of an algorithm depends on the primitive instructions we allow, which in turn depend on the machine model.

$\downarrow$

How fast we can solve a problem depends on the model of computation. The more powerful the machine model, the smaller the lower bound.

(Ultimately, if our model allowed *sort* as a primitive operation, then we could sort in one instruction, hence constant time!!) Now back to sorting:

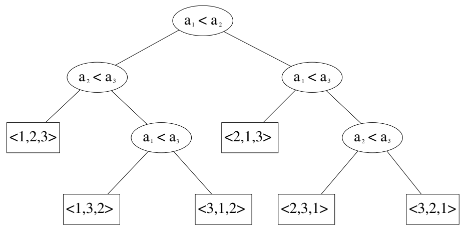
The Comparison Model

- All sorting algorithms we have seen so far use only comparisons to gain information about the input.
- We will prove that such algorithms have to perform $\Omega(n \log n)$ comparisons.
- To prove bound, we need *formal model*:

  > **Comparison (or Decision tree) Model**
  > - Binary tree where each internal node is labeled $a_i \leq a_j$ (a_i is the i'th input element)
  > - Execution corresponds to root-leaf path
  >   - at each internal node a comparison $a_i \leq a_j$ is performed and branching
  > - Leaf contains result of computation

- Decision tree model corresponds to algorithms where only comparisons can be used to gain knowledge about input.
- Any algorithm has a corresponding decision tree (just ignore everything else but the comparisons made by the algorithm); and any decision tree corresponds to an algorithm.

- Example: Decision tree for sorting 3 elements.



- The algorithm must be able to sort any possible input (of n elements, for any n). Put differently, for each input the algorithm must be able to compute the permutation of the input that gives sorted order.
- An execution of the algorithm corresponds to a root-to-leaf path in the tree; it corresponds to identifying the permulation that sorts the input.
- Each leaf of the tree represents a permutation (that corresponds to sorted order for that path).
- Worst case number of comparisons performed corresponds to maximal root-leaf path (=height of tree).
- Therefore lower bound on height $\Rightarrow$ lower bound on sorting.
- If we could prove that any decision tree on n inputs must have height $\Omega(n \lg n) \Rightarrow$ any sorting algorithm (that uses only comparisons) must make at least $\Omega(n \lg n)$ comparisons in the worst case.

Sorting Lower Bound in the Comparison Model

Theorem: Any decision tree sorting n elements has height $\Omega(n \log n)$.

Proof:

- Assume elements are the (distinct) numbers 1 through n
- There must be $n!$ leaves (one for each of the $n!$ permutations of n elements)
- Tree of height h has at most 2^h leaves
- Thus the height must be so that $2^h \geq n!$ and we get

$$\begin{aligned} h \geq \quad & \log(n!) \\ = \quad & \sum_{i=2}^{n} \log i \\ = \quad & \sum_{i=2}^{n/2-1} \log i + \sum_{i=n/2}^{n} \log i \\ \geq \quad & 0 + \sum_{i=n/2}^{n} \log \frac{n}{2} \\ = \quad & \frac{n}{2} \cdot \log \frac{n}{2} \\ = \quad & \Omega(n \log n) \end{aligned}$$

Corollary: Any sorting algorithm that uses only comparisons takes $\Omega(n \log n)$ in the worst case.