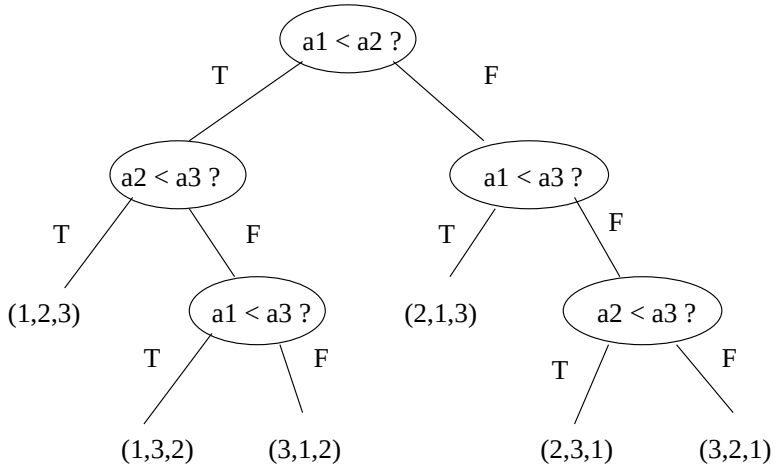
Can we sort in better than $n \lg n$?

Any comparison-based sorting program can be thought of as defining a decision tree of possible executions.

Running the same program twice on the same permutation causes it to do exactly the same thing, but running it on different permutations of the same data causes a different sequence of comparisons to be made on each.



Claim: the height of this decision tree is the worst-case complexity of sorting.

Once you believe this, a lower bound on the time complexity of sorting follows easily.

Since any two different permutations of n elements requires a different sequence of steps to sort, there must be at least $n!$ different paths from the root to leaves in the decision tree, ie. at least $n!$ different leaves in the tree.

Since only binary comparisons (less than or greater than) are used, the decision tree is a binary tree.

Since a binary tree of height h has at most 2^h leaves, we know $n! \leq 2^h$, or $h \geq \lg(n!)$.

By inspection $n! > (n/2)^{n/2}$, since the last $n/2$ terms of the product are each greater than $n/2$. By Sterling's approximation, a better bound is $n! > (n/e)^n$ where $e = 2.718$.

$$h \geq \lg(n/e)^n = n \lg n - n \lg e = \Omega(n \lg n)$$

Why don't CS profs ever stop talking about sorting?!

1. Computers spend more time sorting than anything else, historically 25% on mainframes.

2. Sorting is the best studied problem in computer science, with a variety of different algorithms known.

3. Most of the interesting ideas we will encounter in the course can be taught in the context of sorting, such as divide-and-conquer, randomized algorithms, and lower bounds.

You should have seen most of the algorithms - we will concentrate on the analysis.

Applications of Sorting

One reason why sorting is so important is that once a set of items is sorted, many other problems become easy.

Searching

Binary search lets you test whether an item is in a dictionary in $O(\lg n)$ time.

Speeding up searching is perhaps the most important application of sorting.

Closest pair

Given n numbers, find the pair which are closest to each other.

Once the numbers are sorted, the closest pair will be next to each other in sorted order, so an $O(n)$ linear scan completes the job.

Element uniqueness

Given a set of n items, are they all unique or are there any duplicates?

Sort them and do a linear scan to check all adjacent pairs.

This is a special case of closest pair above.

Frequency distribution – Mode

Given a set of n items, which element occurs the largest number of times?

Sort them and do a linear scan to measure the length of all adjacent runs.

Median and Selection

What is the kth largest item in the set?

Once the keys are placed in sorted order in an array, the kth largest can be found in constant time by simply looking in the kth position of the array.