Intro

A comparison-based sorting algorithm is one in which all swaps are based on direct or indirect comparisons. For indirect comparisons, it may be known from earlier comparisons that two elements are out of order (e.g. in quicksort when you compare things to the pivot rather than each other).

Formal Definition

This is definition 33 from the lecturer's notes.

Relevance

Searching and sorting algorithms can take up to 25% of the total computation time of applications.

Binary Comparison Paths

Comparison-based algorithms can be organised into a binary tree, based on all possible outcomes. A comparison path is one branch in this tree – one set of decisions/outcomes.

The decision trees have one path for each input, and since there are n! inputs for a (distinct) list of size n, then there are n! paths, and n! leaves.

From this you can show that the paths have a minimum length of nlogn (we will see the proof later).

Exercise 42

Draw the binary tree representation for the binary comparison paths of insertion sort for the 6 permutations of three distinct inputs.

Four Main Properties

1. Completeness
    - For a comparison-based algorithm to properly sort a list, every pair of elements must be compared at least once.

    - Note that these don't have to be explicit comparisons, as the transitive closure can provide some of the comparisons.
    - Algorithm needs to reach a conclusion about the relative order of each pair at least once.
2. Faithfulness
    - Input lists in the same relative order are executed on in the same way.
3. Separation
    - Input lists not in the same relative order must cause the algorithm to diverge at some point in its computation.
4. Reduction: The 0-1 Principle
    - To check that a comparison-based algorithm is a sorting algorithm, it's enough to show that it sorts all binary sequences (e.g. [0, 1, 1, 0, 1]).
    - This brings us from n! cases down to 2^n cases, which is much more manageable.

Loose Ends

Rules:

1. Boolean conditions are entirely based on comparisons between list elements only.
2. All assignments are based on prior comparisons. [rough wording]
3. All swaps are based on prior comparisons. [rough wording]

Four Main Properties (recap)

- Need to know the contexts in which they are useful
- Need to know proofs for completeness and faithfulness
    - Need to understand proof for reduction
- Need to know the properties and be able to explain and justify them

Completeness Proof

[...]

Faithfulness Proof

- follows your intuition

Separation Proof

- too complicated, won't cover it in this course

Reduction Proof

Depends on faithfulness – since f is an increasing function and we've proven faithfulness, then the relative order is the same before and after application of f. Then the comparison-based algorithm will operate in the same way on both. (Monotonicity lemma)

Then proof by contradiction. Assume all binary lists are sorted, but there exists a non-binary list that isn't sorted. Use an increasing function f to transform that non-binary list into a binary list that isn't sorted.