COMP2054-ADE

Lecturer: Andrew Parkes
http://www.cs.nott.ac.uk/~pszajp/

Comparison Based Sorting

Lower Bound on Efficiency

1

This lecture looks at sorting algorithms whose only knowledge of the elements it is sorting is based on comparing them.

Then looks at what might be the best possible complexity.

The lecture is fairly short – but conceptually fairly difficult.

Comparison sorting

- A sorting algorithm is a comparison sorting algorithm if it uses comparisons between elements in the sequence to determine in which order to place them
- Examples of comparison sorts: bubble sort, selection sort, insertion sort, heap sort, merge sort, quicksort.
- Example of a sort that is ***not*** comparison-based: bucket sort
  - Runs in O(n), but relies on knowing the range of values in the sequence (e.g. "integers between 1 and 1000").

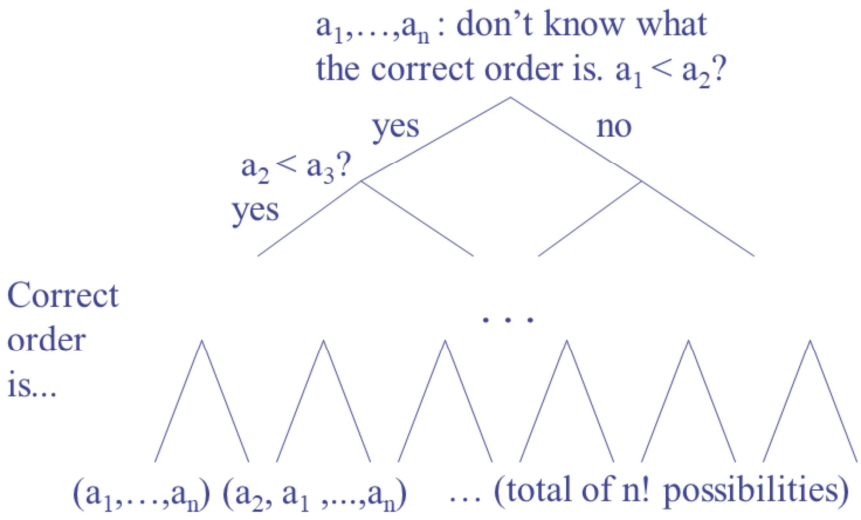
2

Lower bound for comparison sort

- We can model sorting which depends on comparisons between elements as a binary decision tree.
- At each node, a comparison between two elements is made; there are two possible outcomes and we find out a bit more about the correct order of items in the array.
- Finally arrive at full information about the correct order of the items in the array.

3

Comparison sorting



4

We do not know anything about the sequence of elements unless we ask questions to compare them.

We have to ask enough questions to be able to distinguish between all the n! possible orderings.

How many comparisons?

- If a binary tree has n! leaves, than the minimal number of levels (assuming the tree is perfect) is ($\log_2$ n!) +1.
- This shows that O(n log n) sorting algorithms are essentially optimal
  - $\log_2$n! is not equal to n $\log_2$n, but has the same growth rate
- **Comparison-based sorting cannot do better than O(n log n)**
  - Technically, it uses Stirling's approximation https://en.wikipedia.org/wiki/Stirling%27s_approximation that

    $\log_2(n!) = n \log_2(n) +$ "smaller terms"

    - Note: you should know the this approximation, but (obviously) do not need to know the proof.

5

Each time we ask a comparison question we half the number of possible remaining orderings in the original elements.

Once, we have asked enough comparisons then we would "know" the ordering in the original elements and so could just move them to where they should be.

If we ask q questions, then there are 2^q possible answers, and we need this to be at least enough to cover n! possible orderings.

Hence, we need 2^q > n!

i.e. q > log_2 (n!) which is basically q > n log(n)

Questions to ask about sorting algorithms

- Big-Oh complexity (both time and space)?
  - Best case inputs? Worst case inputs?
- Extra workspace needed?
  Or is it `in-place'?
- Stable sorts?
- "Dynamic sorting" – how well does it do if the data is already "nearly sorted"
- Data access patterns?
  - Sequential? Random Access?
- Relevant and appropriate assertions

Aim to understand these issues for various sorting algorithms.

6

Minimum Expectations

- Know the meaning of 'comparison-based sorting' and the resulting O(n log n) lower bound on complexity.

7