CSC 225

Algorithms and Data Structures: I

Rich Little
rlittle@uvic.ca
ECS 516

Comparison-Based Sorting

- sorting algorithm that sorts based only on comparisons
- elements to be sorted must satisfy total order properties
- https://www.youtube.com/watch?v=ZZuD6iUe3Pc

	Type of Sorting Algorithm	Worst Case Time	Best Case Performance	Average Case Performance	Properties
Insertion Sort	Comparison Based Sorting	$O(n^2)$	$O(n)$	$O(n^2)$	adaptive, in place, stable, online
Bubblesort	Comparison Based Sorting	$O(n^2)$	$O(n)$	$O(n^2)$	in place
Selection Sort	Comparison Based Sorting	$O(n^2)$	$O(n^2)$	$O(n^2)$	in place
Binary Insertion	Comparison Based Sorting	$O(n^2)$	$O(n)$	$O(n^2)$	adaptive, in place
Shakersort	Comparison Based Sorting	$O(n^2)$	$O(n)$	$O(n^2)$	stable, in place
Shellsort	Comparison Based Sorting	$O(n^2)$	$O(n \log n)$		in place
Quicksort	Comparison Based Sorting	$O(n^2)$	$O(n \log n)$	$O(n \log n)$	in place
Heapsort	Comparison Based Sorting	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$	in place
Mergesort	Comparison Based Sorting	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$	not in place

How fast can we sort? A lower bound for comparison based sorting

Theorem: The running time of any comparison-based algorithm for sorting an n-element sequence is $\boldsymbol{\Omega}(\boldsymbol{n}\ \boldsymbol{log}(\boldsymbol{n}))$ in the worst-case.

Proof:

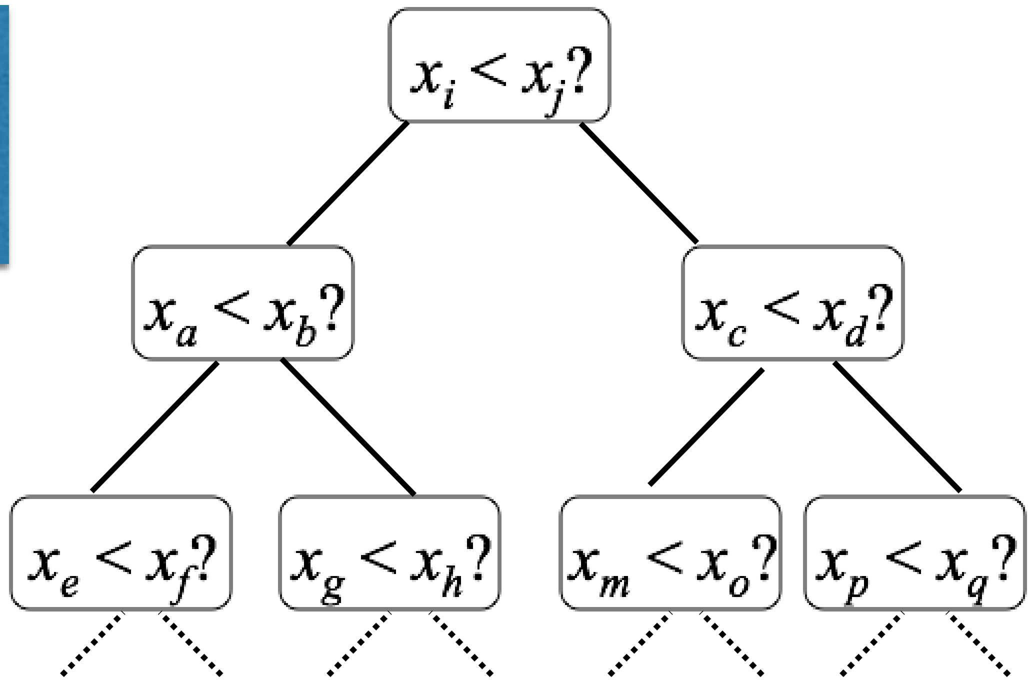
- Consider a sequence S containing n distinct elements, say $[x_0, x_1, \dots, x_{n-1}]$
- To decide the order of elements, a comparison-based algorithm compares elements pairwise—a sufficient number of times
- In particular, to decide which element of x_i and x_j is smaller, it answers "is $x_i < x_j$?"
- Depending on the outcome—i.e., "yes" or "no"—the algorithm performs either no further comparisons or it continues with more comparisons

Proof (continued)

- We want to know: how good is the best of all comparison-based sorting algorithms? (Let’s call it the *optimal* algorithm)
- This optimal sorting algorithm requires a certain number of comparisons (at least) to sort *any* sequence (not just the easiest input)
- We ask: How many comparisons are required for an optimal sorting algorithm to sort n elements?
- This can be depicted in a decision tree.

Decision tree of an optimal sorting algorithm that is sorting a general sequence of elements

- The decision tree contains every possible path the optimum algorithm might take to sort sequence *S*.

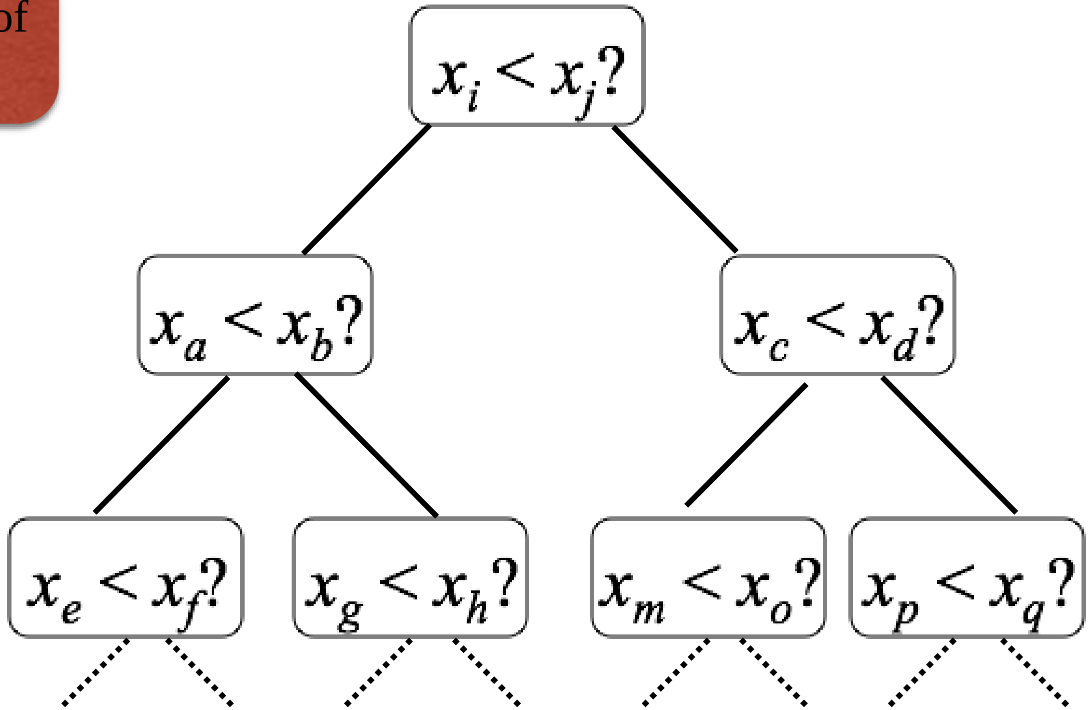


Since we don't know what *S* looks like, any permutation of *S* could be the sorted one. Thus, every permutation of *S* has to be represented by a path from the root to a leaf in the decision tree.

Example: Let $S = [x_0, x_1, x_2]$, where each distinct $x_i \in \{1,2,3\}$, and draw the corresponding decision tree.

Decision tree of an optimal sorting algorithm sorting a general sequence of elements

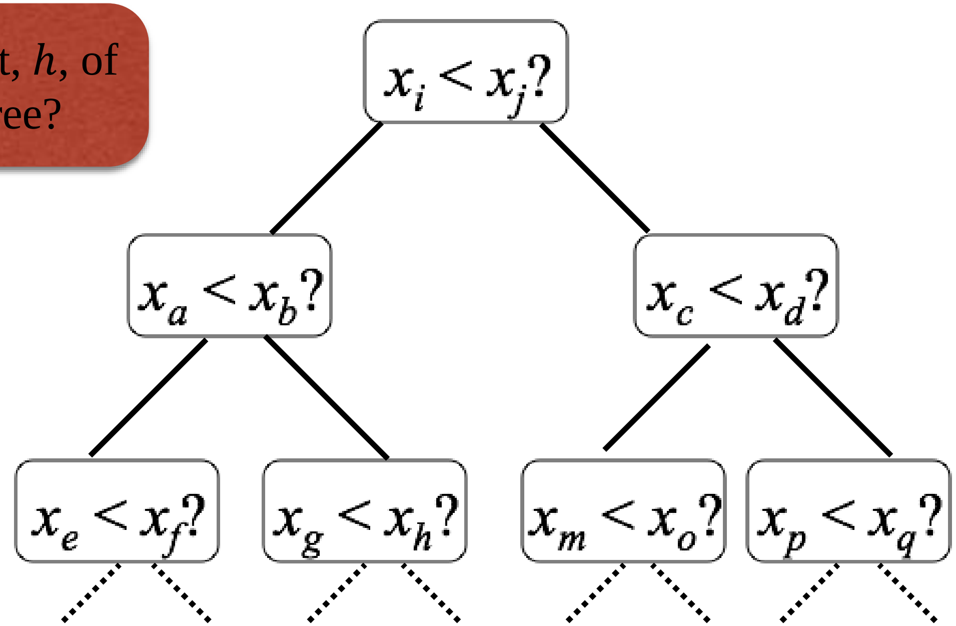
What is the height, *h*, of this decision tree?



Lemma: Each external node v in the decision tree T represents the sequence of comparisons for at most one permutation of S.

Decision tree of an optimal sorting algorithm sorting a general sequence of elements

What is the height, h, of this decision tree?



- # leaves = $2^h \geq n!$
- Thus, $h \geq \log n!$

Proof (continued)

- Since the height of the tree is at least log(n!), we know that
  - at least log(n!) worst case comparisons are required by an optimal comparison based sorting algorithm
  - at least log(n!) worst case comparisons are required by any comparison based algorithm

Proof (continued)

What is $\Omega(\log(n!))$?

Stirling's Formula

- Another useful formula for ordering functions by growth rate is Stirling's Formula (1730)

$$n! \approx \sqrt{2\pi n}\left[\frac{n}{e}\right]^{n}$$

- Can also be expressed as the following:

$$\sqrt{2\pi}n^{n+\frac{1}{2}}e^{-n} \leq n! \leq en^{n+\frac{1}{2}}e^{-n}$$