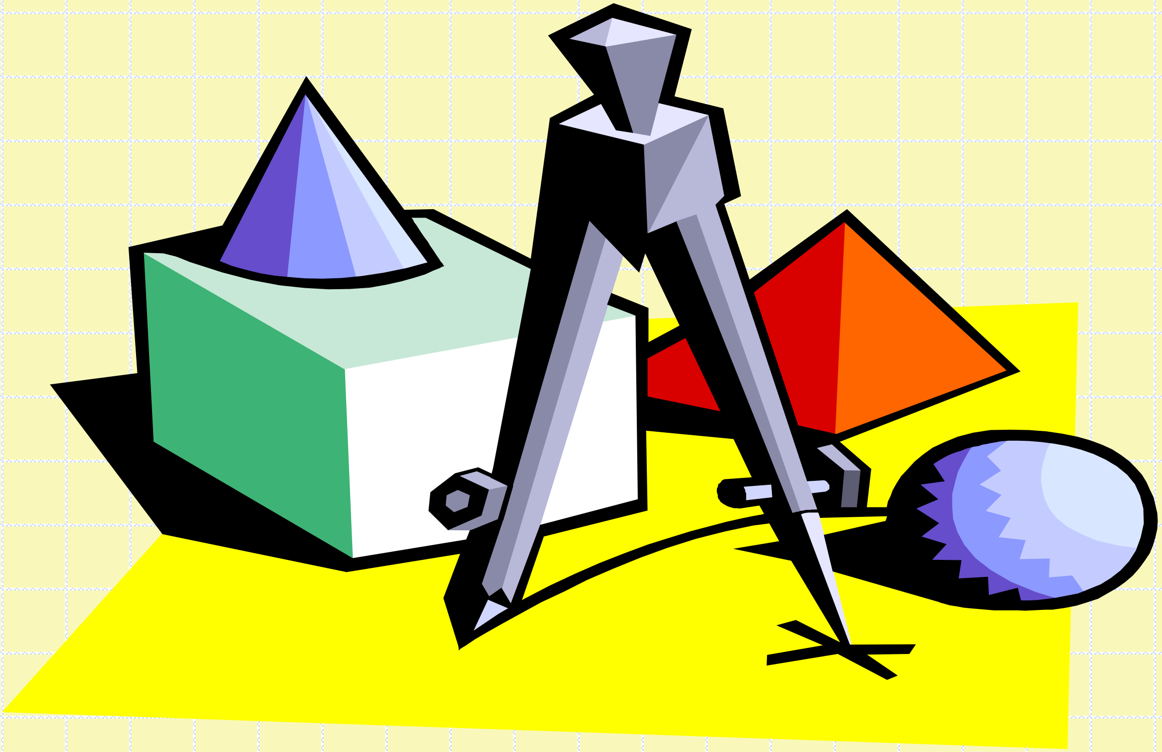
Sorting Lower Bound

Dr. Aiman Hanna
Department of Computer Science & Software Engineering
Concordia University, Montreal, Canada

These slides have been extracted, modified and updated from original slides of :
Data Structures and Algorithms in Java, 5th edition. John Wiley& Sons, 2010. ISBN 978-0-470-38326-1.
Data Structures and the Java Collections Framework by William J. Collins, 3rdedition, ISBN 978-0-470-48267-4.
Both books are published by Wiley.

Coverage

- Sorting Lower Bound

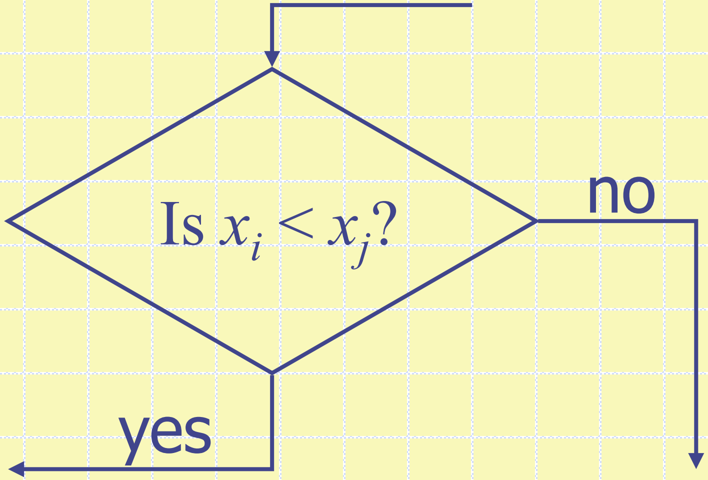


Comparison-Based Sorting

- Many sorting algorithms are *comparison* based.
  - They sort by making comparisons between pairs of objects
  - Examples: bubble-sort, selection-sort, insertion-sort, heap-sort, merge-sort, quick-sort, ...
- Some of these algorithms were able to provide a performance of $O(n \log n)$.
- This question can then be asked**: Can we sort any faster than $O(n \log n)$**?
- In other words, what is the lower bound (best case) that we can achieve when sorting (that is actually $\Omega()$)?

Comparison-Based Sorting

- Let us start by deriving a lower bound on the running time of any algorithm that uses comparisons to sort n elements, $x_1, x_2, \ldots, x_n$.
- Let us then count only the number of comparisons for the sake of finding the lower bound.

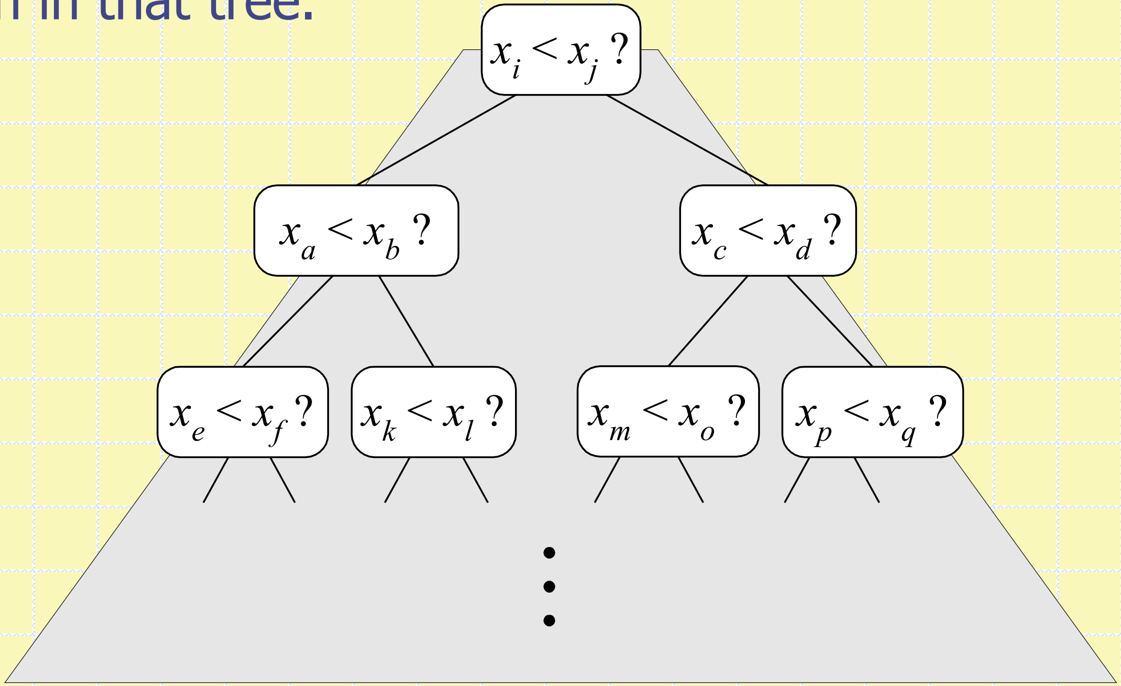


Counting Comparisons

- Suppose, we are given a sequence $S = \{x_1, x_2, \ldots, x_n\}$ that we wish to sort.
- Whether the sequence is implemented using an arrays or a list is irrelevant since we are counting comparisons.
- Each time we compare two elements, $x_i < x_j$, the result is either yes or no.
- Based on this answer, some internal computation may be performed (which we ignore here), then another comparison is conducted, which will again have one of two possible outcomes.

Counting Comparisons

- Consequently, we can represent a comparison-based sorting with a *decision tree*.
- Each possible run of the algorithm corresponds to a root-to-leaf path in that tree.

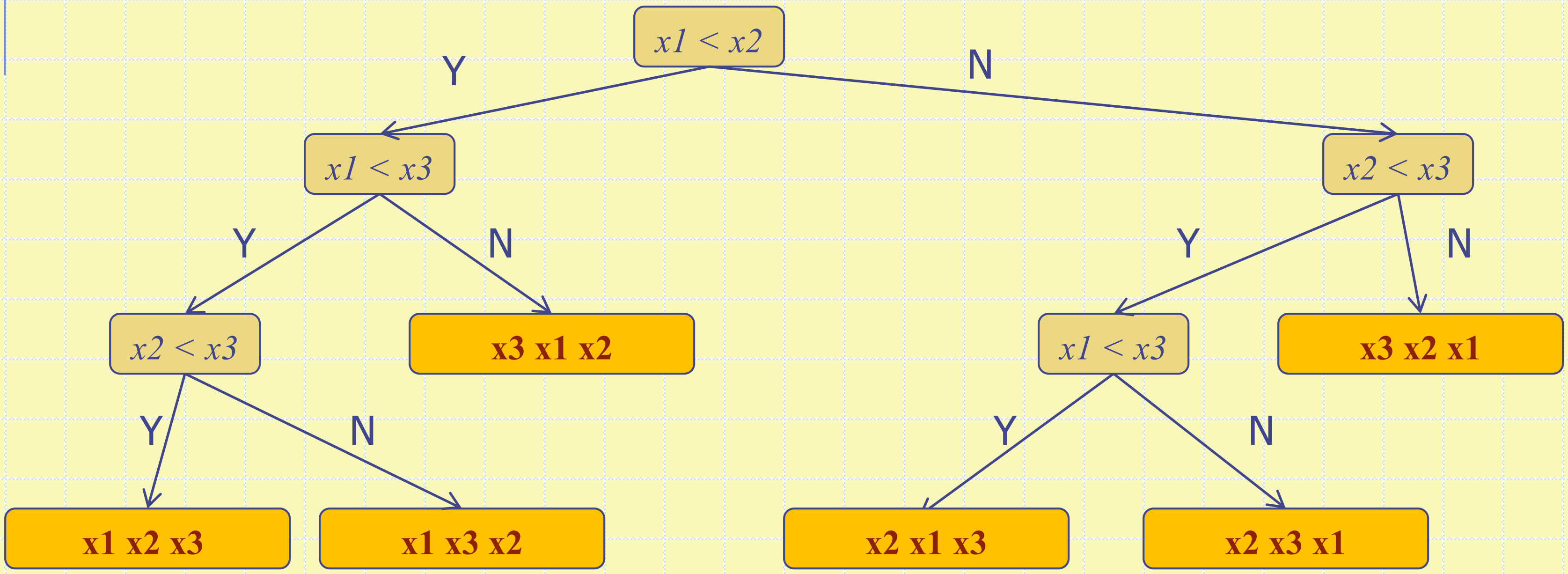


Counting Comparisons

- It is important to note that the algorithm may have no explicit knowledge of the tree; rather, it simply represent all the possible sequences of comparisons that the application might make.
- Each leaf hence represents one possible permutation (sorted sequence) of the elements to be compared.
- The number of leaves can then conclude the height of the tree.
- Given n values to compare, That number of leaves (possible permutations) is actually $n!$

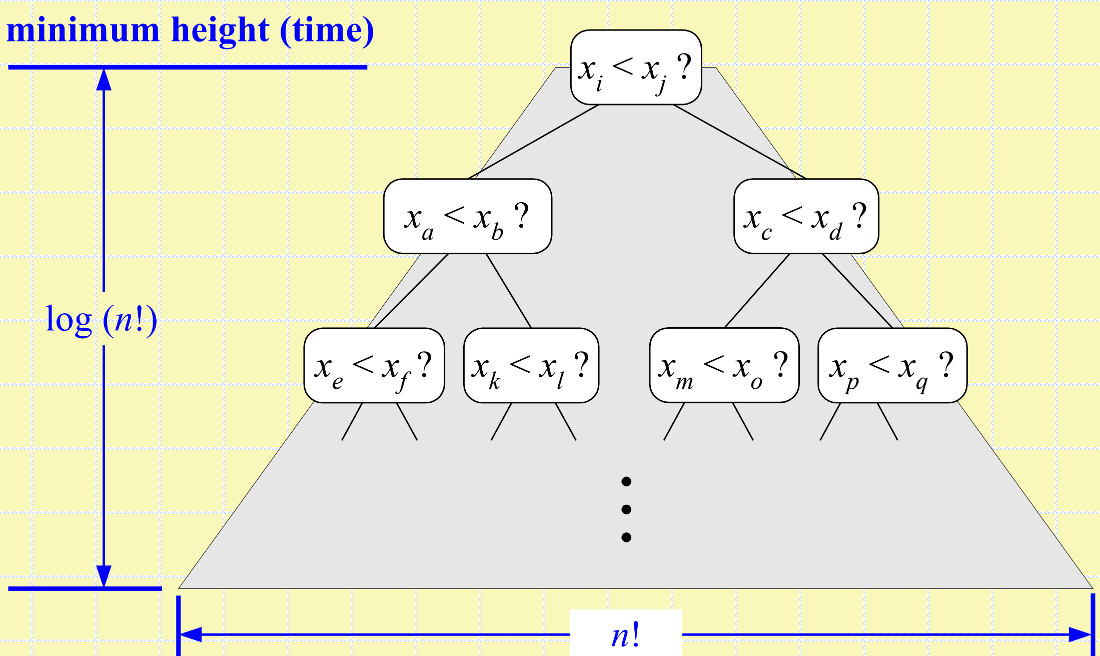
Counting Comparisons

- Example: Assume a sequence *S* with elements {x1, x2, x3} is to be sorted. There are *3!* possible sorted sequences as follows: (x1 x2 x3), (x1 x3 x2), (x2 x1 x3), (x2 x3 x1), (x3 x1 x2), and (x3 x2 x1).
- The following tree illustrates the possible comparisons:

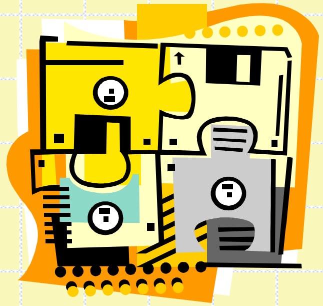


Decision Tree Height

- The height of the decision tree is a lower bound on the running time.
- Since there are $n!$ leaves, the height of the tree is at least $\log(n!)$.



The Lower Bound



- Any comparison-based sorting algorithms takes at least *log (n!)* time

- Therefore, any such algorithm takes, at least, time

$$\log\,(n!) \geq \log\left(\frac{n}{2}\right)^{\frac{n}{2}} = (n/2)\log(n/2).$$

- That is, any comparison-based sorting algorithm must run in at least *Ω(n log n*) time. In other words, we comparison-based algorithms, we cannot achieve any better performance than ***n log n.***
- ***However, can we do any better if the algorithm is not a comparison-based then? Are such algorithms possible?***