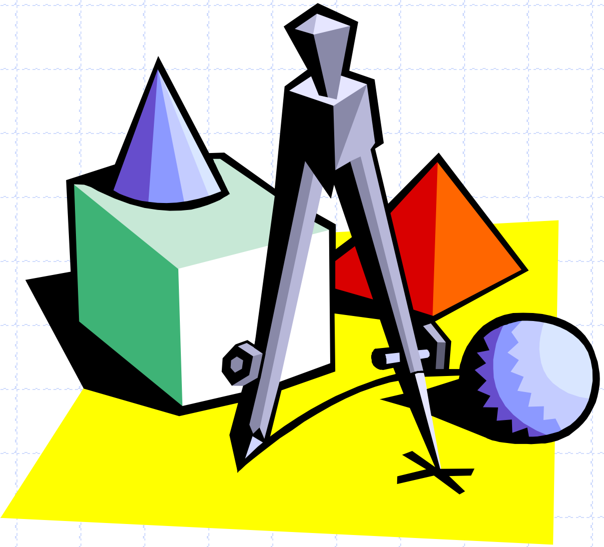
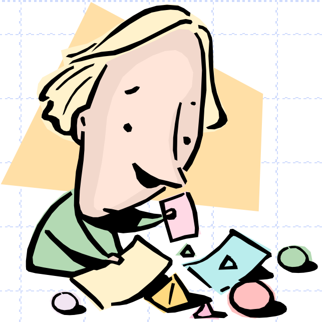
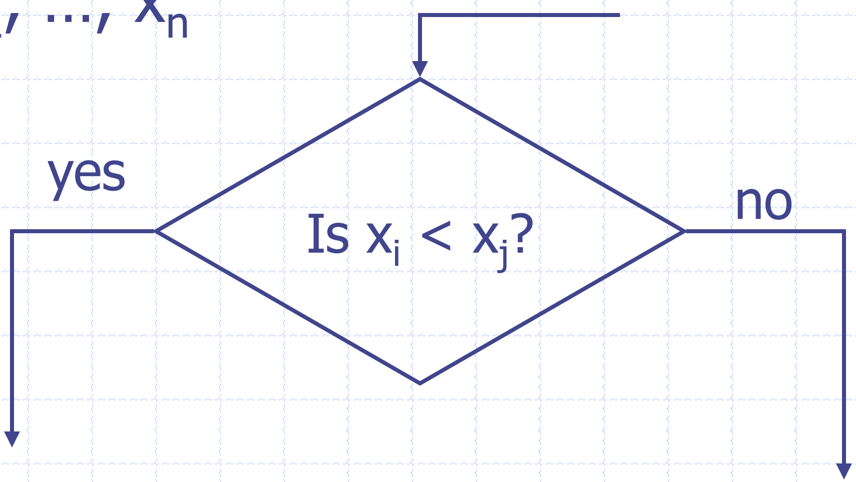
Sorting Lower Bound



Comparison-Based Sorting

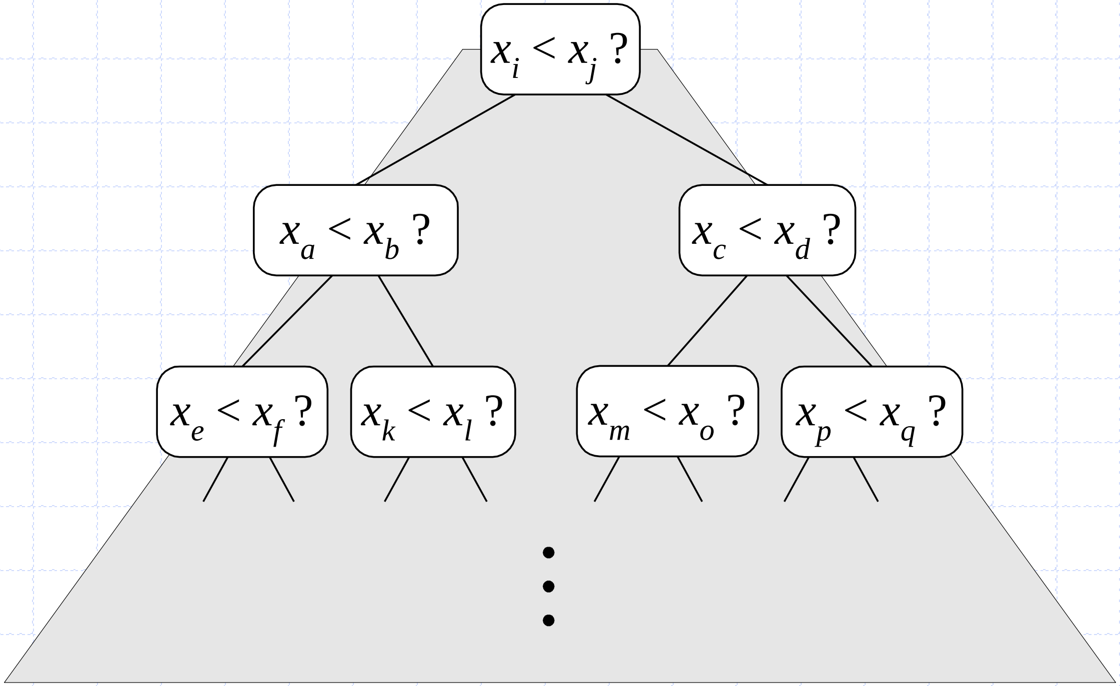


- Many sorting algorithms are comparison based
  - They sort by making comparisons between pairs of objects
  - Examples: bubble-sort, selection-sort, insertion-sort, heap-sort, merge-sort, quick-sort, ...
- Let us therefore derive a lower bound on the running time of any algorithm that uses comparisons to sort n elements, $x_1, x_2, \ldots, x_n$



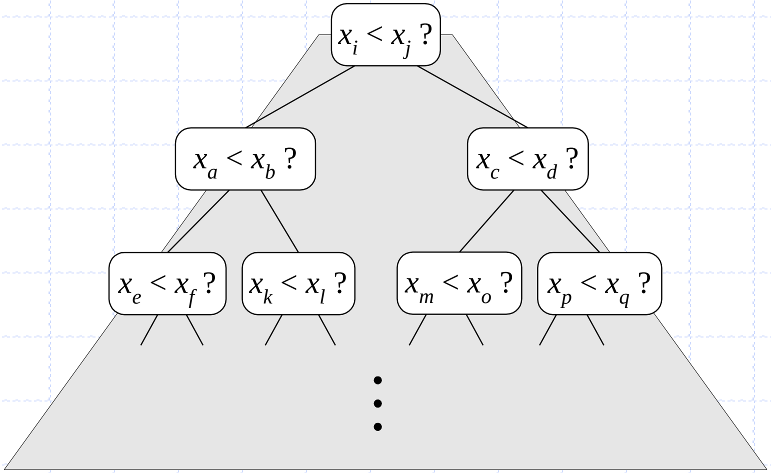
Counting Comparisons

- Let us just count comparisons then
- Each possible run of the algorithm corresponds to a root-to-leaf path in a **decision tree**



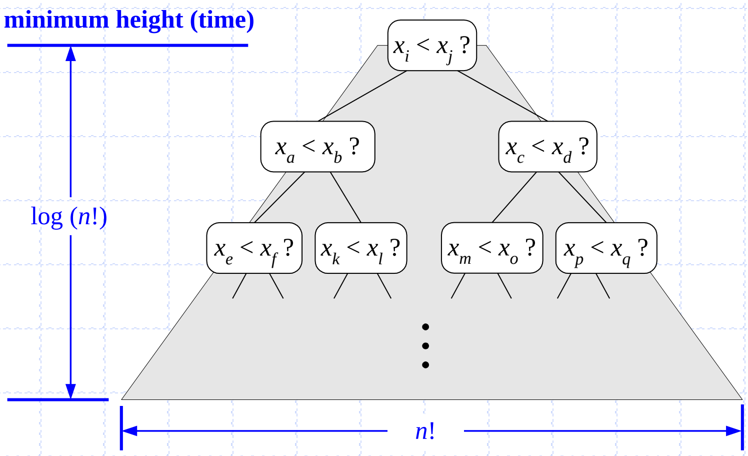
Decision Tree Height

- Height of this decision tree is a lower bound on the running time
- Every possible input permutation leads to a separate leaf output
  - If not, some input …4…5… would have same output ordering as …5…4…, which would be wrong.
- How many leaves are there?
  - There are n!=1*2*…*n leaves
- What is the height of the tree?
  - The height is at least log (n!)

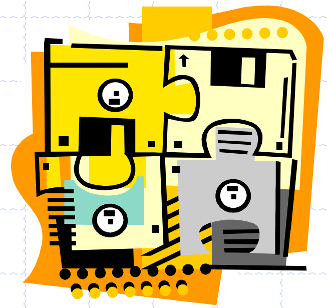


Decision Tree Height

- Height of this decision tree is a lower bound on the running time
- Every possible input permutation leads to a separate leaf output
  - If not, some input …4…5… would have same output ordering as …5…4…, which would be wrong.
- How many leaves are there?
  - There are n!=1*2*…*n leaves
- What is the height of the tree?
  - The height is at least log (n!)



The Lower Bound



- Any comparison-based sorting algorithms takes at least log (n!) time
- Therefore, any such algorithm takes time at least

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

  - Why?
  - (because there are at least (n/2) terms greater than (n/2))

- That is, any comparison-based sorting algorithm must run in Ω(n log n) time

Is there a way to break the O(nlogn) barrier?

- Yes!
  - Don’t use comparisons
    - That is, a non-comparison based sort