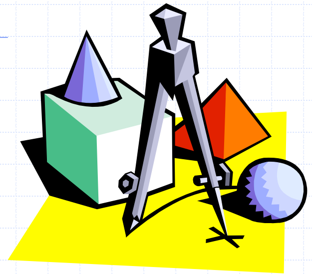
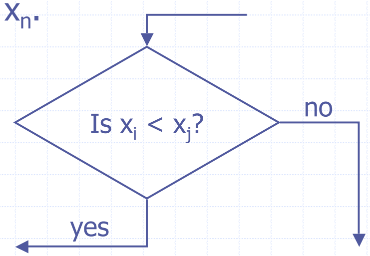
# Sorting Lower Bound





# Comparison-Based Sorting (§ 10.3)

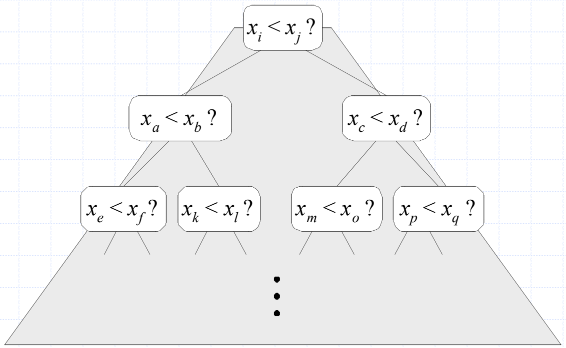
- Many sorting algorithms are comparison based.
  - They sort by making comparisons between pairs of objects
  - Examples: bubble-sort, selection-sort, insertion-sort, heap-sort, merge-sort, quick-sort, ...
- Let us therefore derive a lower bound on the running time of any algorithm that uses comparisons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






# Counting Comparisons

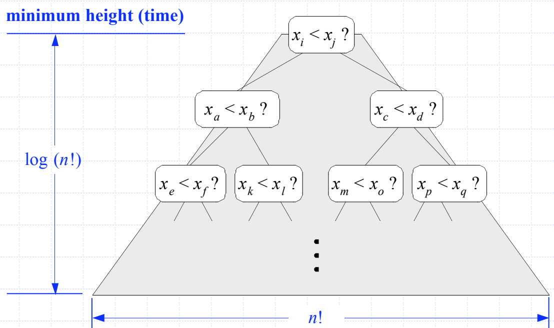
- Let us just count comparisons then.
- Each possible run of the algorithm corresponds to a root-to-leaf path in a **decision tree**



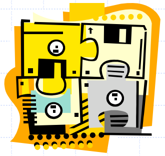



# Decision Tree Height

- The height of this decision tree is a lower bound on the running time
- Every possible input permutation must lead to a separate leaf output.
  - If not, some input …4…5… would have same output ordering as …5…4…, which would be wrong.
- Since there are n!=1*2*…*n leaves, 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!) = log 1 + log 2 + … + log n
>= (n log n )/2
(by concavity, or integration)

- That is, any comparison-based sorting algorithm must require at least Ω(n log n) time.



# Summary of Sorting Algorithms

| Algorithm | Time | Notes |
| --- | --- | --- |
| selection-sort | $O(n^2)$ | ◆ in-place<br>◆ slow (good for small inputs) |
| insertion-sort | $O(n^2)$ | ◆ in-place<br>◆ slow (good for small inputs) |
| quick-sort | $O(n \log n)$ expected | ◆ in-place, randomized<br>◆ fastest (good for large inputs) |
| heap-sort | $O(n \log n)$ | ◆ in-place<br>◆ fast (good for large inputs) |
| merge-sort | $O(n \log n)$ | ◆ sequential data access<br>◆ fast (good for huge inputs) |