Sorting

Maunendra Sankar Desarkar
IIT Hyderabad

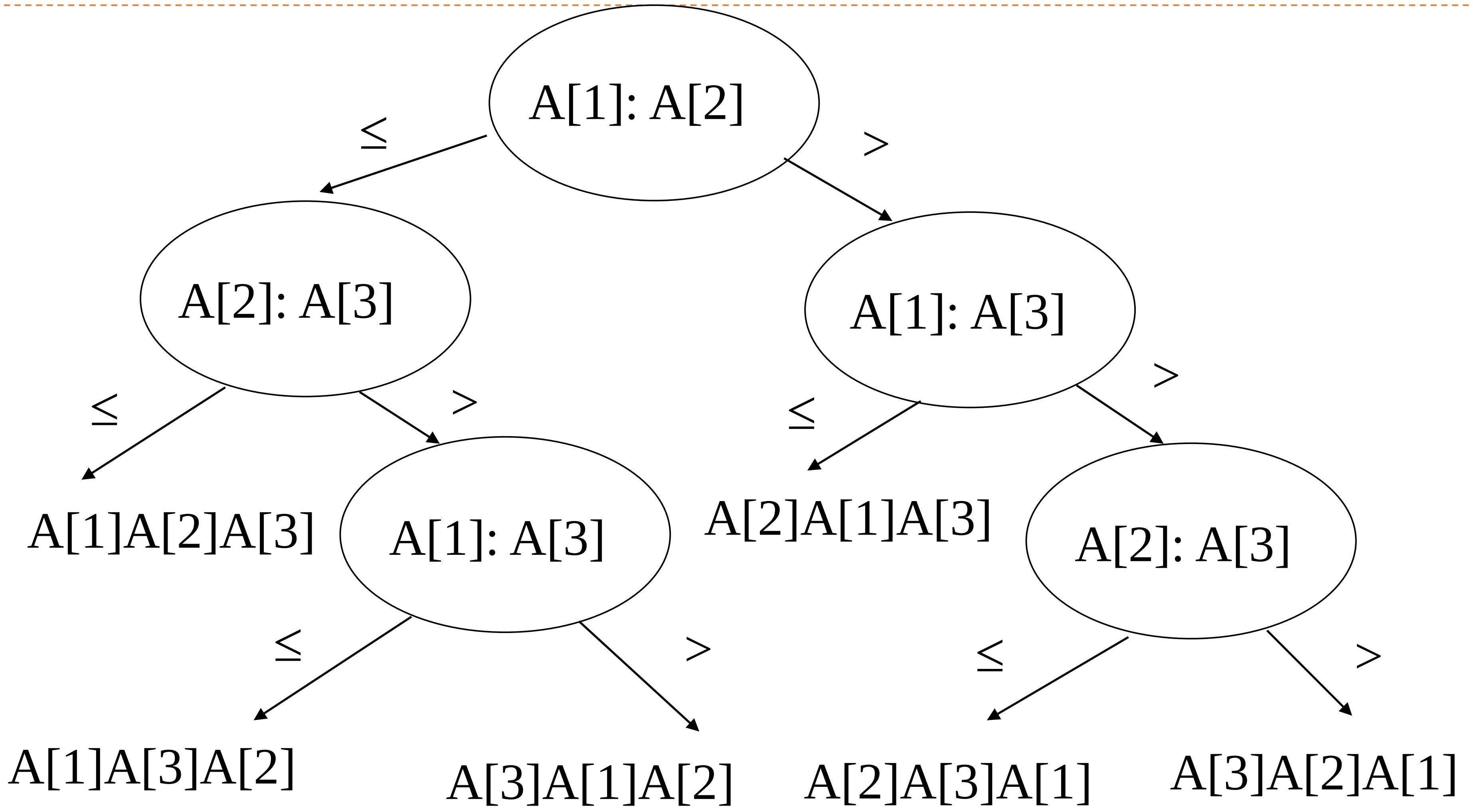
Lower bounds for comparison-based sorting

- Trivial: $\Omega(n)$ – every element must take part in a comparison.
- Best possible result – $\Omega(n \log n)$ comparisons, since we already know several O(n log n) sorting algorithms.
- Proof is non-trivial: how do we reason about all possible comparison-based sorting algorithms?

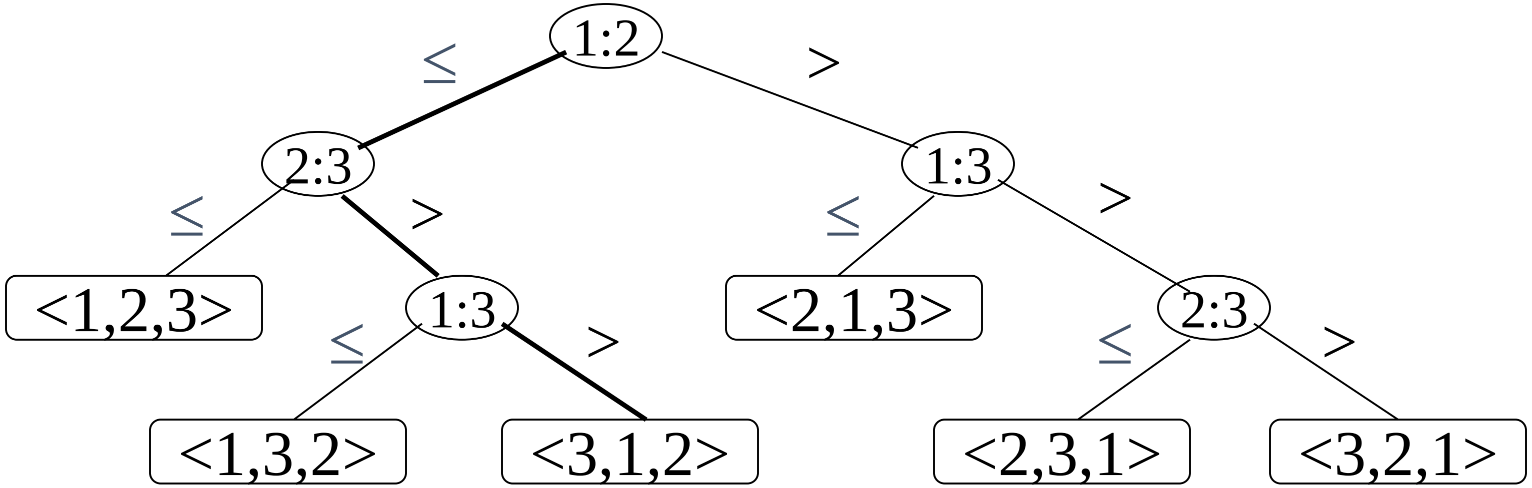
The Decision Tree Model

- Assumptions:
  - All numbers are distinct (so no use for $a_i = a_j$)
  - All comparisons have form $a_i \leq a_j$ (since $a_i \leq a_j$, $a_i \geq a_j$, $a_i < a_j$, $a_i > a_j$ are equivalent).
- Decision tree model
  - Full binary tree
  - Ignore control, movement, and all other operations, just use comparisons.
  - suppose three elements $< a_1, a_2, a_3 >$ with instance $<6,8,5>$.

Example: insertion sort (n=3)



The Decision Tree Model



Internal node i:j indicates comparison between a_i and a_j.

Leaf node $<\pi(1), \pi(2), \pi(3)>$ indicates ordering $a_{\pi(1)} \leq a_{\pi(2)} \leq a_{\pi(3)}$.

Path of bold lines indicates sorting path for <6,8,5>.

There are total 3!=6 possible permutations (paths).

Summary

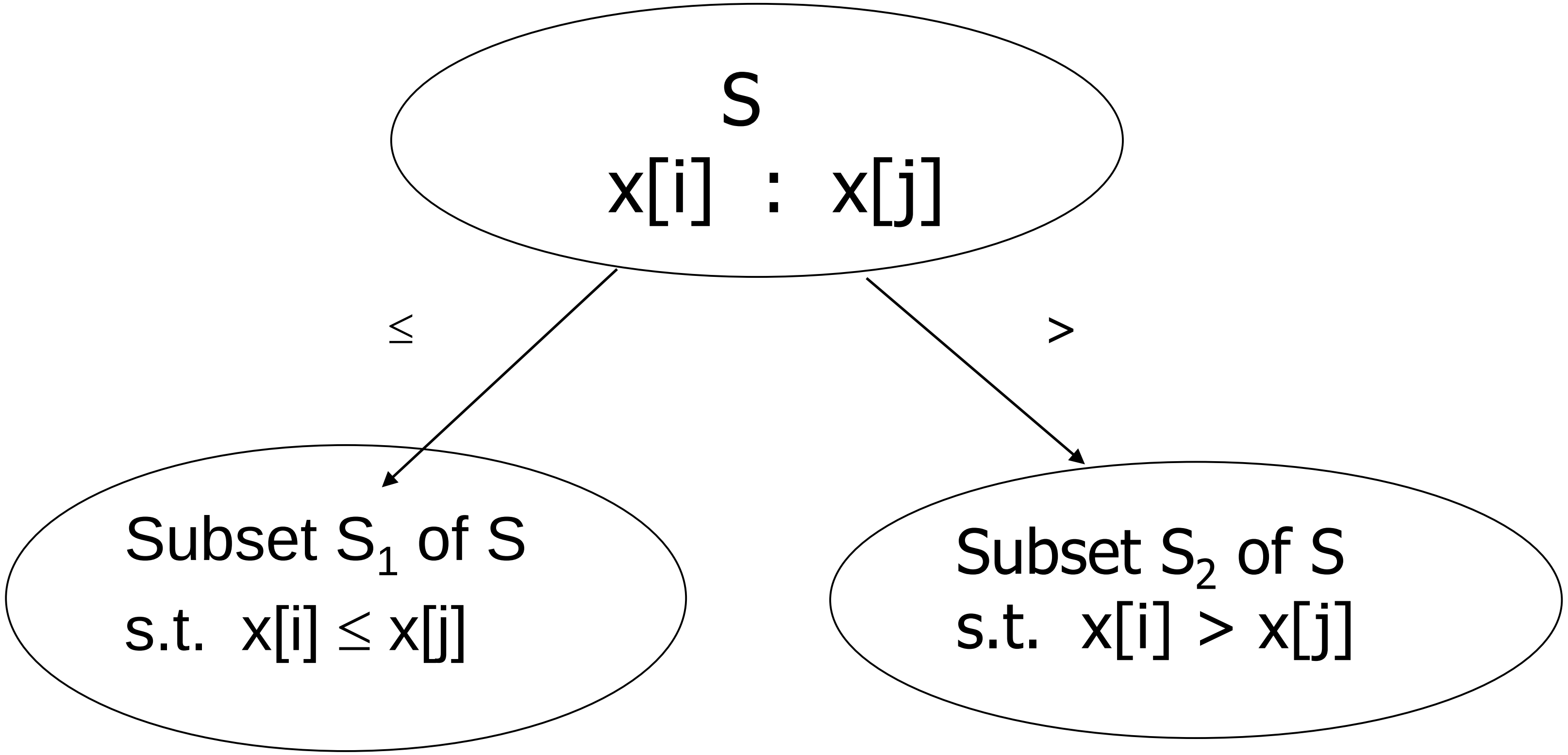
- Only consider comparisons
- Each internal node = 1 comparison
- Start at root, make the first comparison

  - if the outcome is ≤ take the LEFT branch

  - if the outcome is > - take the RIGHT branch
- Repeat at each internal node
- Each LEAF represents ONE correct ordering

Intuitive idea

S is a set of permutations



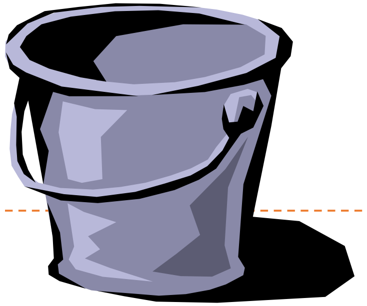
Lower bound for the worst case

- Claim: The decision tree must have at least n! leaves. WHY?
- worst case number of comparisons= the height of the decision tree.
- Claim: Any comparison sort in the worst case needs Ω(n log n) comparisons.
- Suppose height of a decision tree is h, number of paths (i,e,, permutations) is n!.
- Since a binary tree of height h has at most 2^h leaves,

  $n! \leq 2^h$, so $h \geq \lg (n!) \geq \Omega(n \lg n)$

Other sorting techniques??

Bucket-Sort (§ 4.5.1)



- Let be S be a sequence of n (key, element) items with keys in the range $[0, N-1]$
- Bucket-sort uses the keys as indices into an auxiliary array B of sequences (buckets)

  Phase 1: Empty sequence S by moving each item (k, o) into its bucket $B[k]$

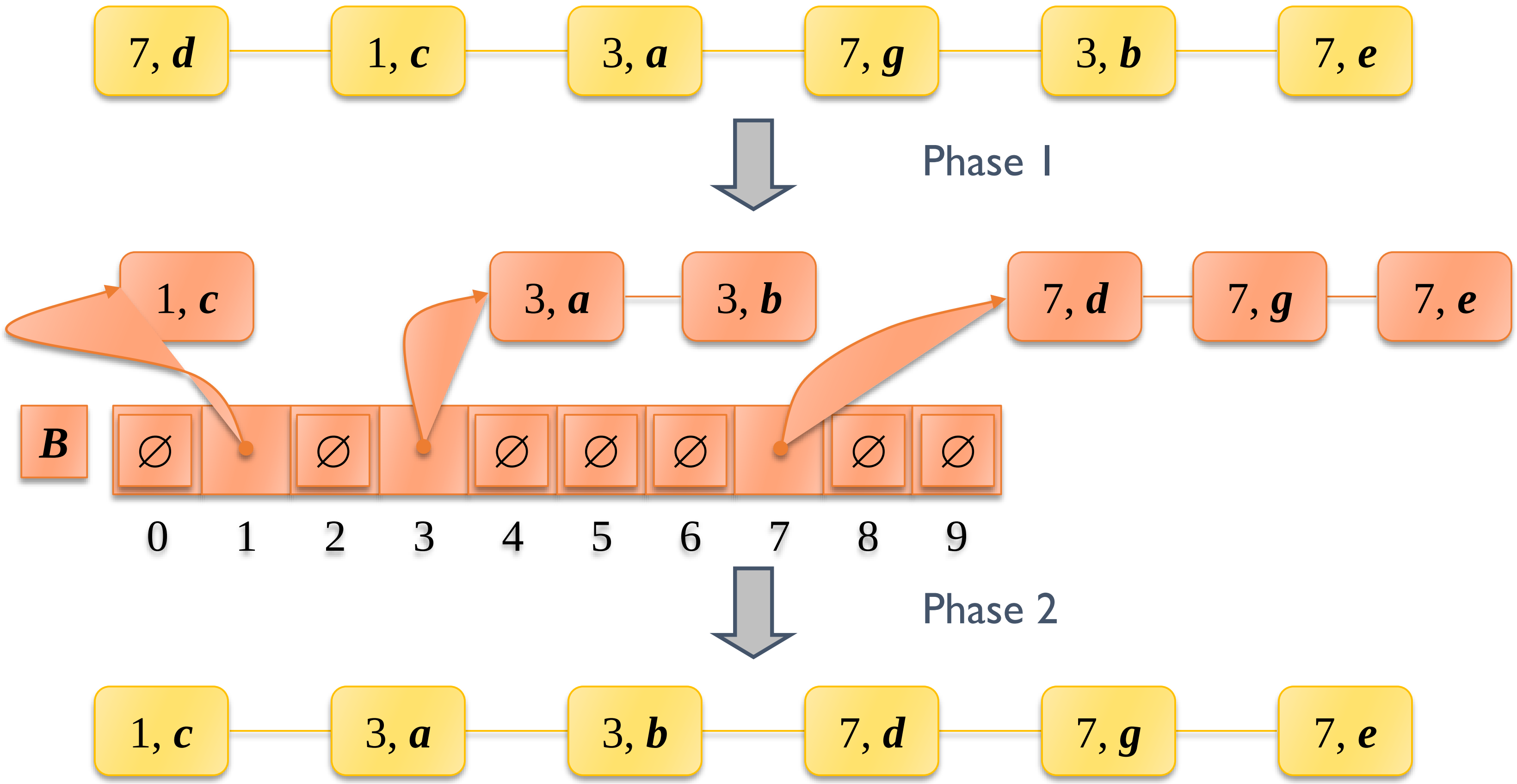
  Phase 2: For $i = 0, \ldots, N-1$, move the items of bucket $B[i]$ to the end of sequence S
- Analysis:
  - Phase 1 takes $O(n)$ time
  - Phase 2 takes $O(n + N)$ time

  Bucket-sort takes $O(n + N)$ time

Example on next slide

Example

- Key range [0, 9]



Properties and Extensions

- Key-type Property
  - The keys are used as indices into an array and cannot be arbitrary objects
  - No external comparator
- **Stable** Sort Property
  - The relative order of any two items with the same key is preserved after the execution of the algorithm

Extensions

- Integer keys in the range $[a, b]$
  - Put item (k, o) into bucket $B[k - a]$
- String keys from a set D of possible strings, where D has constant size (e.g., names of the 50 U.S. states)
  - Sort D and compute the rank $r(k)$ of each string k of D in the sorted sequence
  - Put item (k, o) into bucket $B[r(k)]$

Radix Sort

Sort by keys

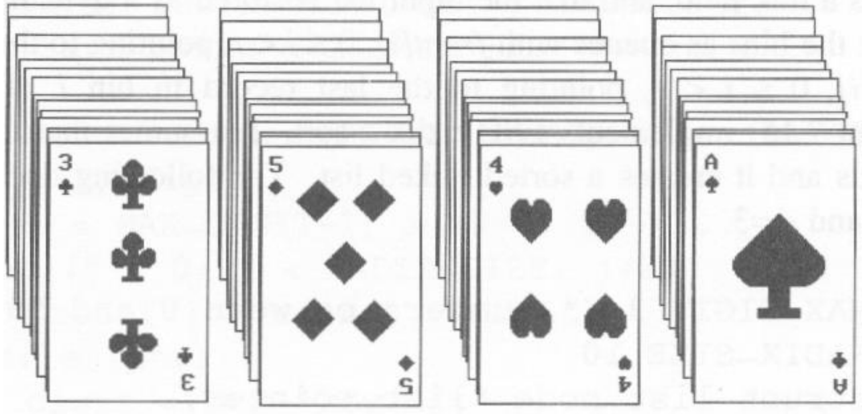
K^0, K^1, …, K^{r-1}

Most significant key

Least significant key

Most significant digit first: sort on K^0, then K^1, ...

Least significant digit first: sort on K^{r-1}, then K^{r-2}, ...



Suits: ♣ < ♦ < ♥ < ♠

Face values: 2 < 3 < 4 < … < J < Q < K < A

(1) MSD sort first, e.g., bin sort, four bins ♣ ♦ ♥ ♠

LSD sort second, e.g., insertion sort

(2) LSD sort first, e.g., bin sort, 13 bins

2, 3, 4, …, 10, J, Q, K, A

MSD sort, e.g., bin sort four bins ♣ ♦ ♥ ♠

RadixSort – magic! It works.

- Input list:
  126, 328, 636, 341, 416, 131, 328
- BinSort on lower digit:
  341, 131, 126, 636, 416, 328, 328
- BinSort result on next-higher digit:
  416, 126, 328, 328, 131, 636, 341
- BinSort that result on highest digit:
  126, 131, 328, 328, 341, 416, 636