Data Structure

Lecture#19: Internal Sorting 4 (Chapter 7)

U Kang
Seoul National University

In This Lecture

- Empirical comparison of sorting algorithms, and observations from it
- Lower bounds for sorting (using Big- Ω notation)

Empirical Comparison

Sort	10	100	1K	10K	100K	1M	Up	Down
Insertion	.00023	.007	0.66	64.98	7281.0	674420	0.04	129.05
Bubble	.00035	.020	2.25	277.94	27691.0	2820680	70.64	108.69
Selection	.00039	.012	0.69	72.47	7356.0	780000	69.76	69.58
Shell	.00034	.008	0.14	1.99	30.2	554	0.44	0.79
Shell/O	.00034	.008	0.12	1.91	29.0	530	0.36	0.64
Merge	.00050	.010	0.12	1.61	19.3	219	0.83	0.79
Merge/O	.00024	.007	0.10	1.31	17.2	197	0.47	0.66
Quick	.00048	.008	0.11	1.37	15.7	162	0.37	0.40
Quick/O	.00031	.006	0.09	1.14	13.6	143	0.32	0.36
Heap	.00050	.011	0.16	2.08	26.7	391	1.57	1.56
Heap/O	.00033	.007	0.11	1.61	20.8	334	1.01	1.04
Radix/4	.00838	.081	0.79	7.99	79.9	808	7.97	7.97
Radix/8	.00799	.044	0.40	3.99	40.0	404	4.00	3.99

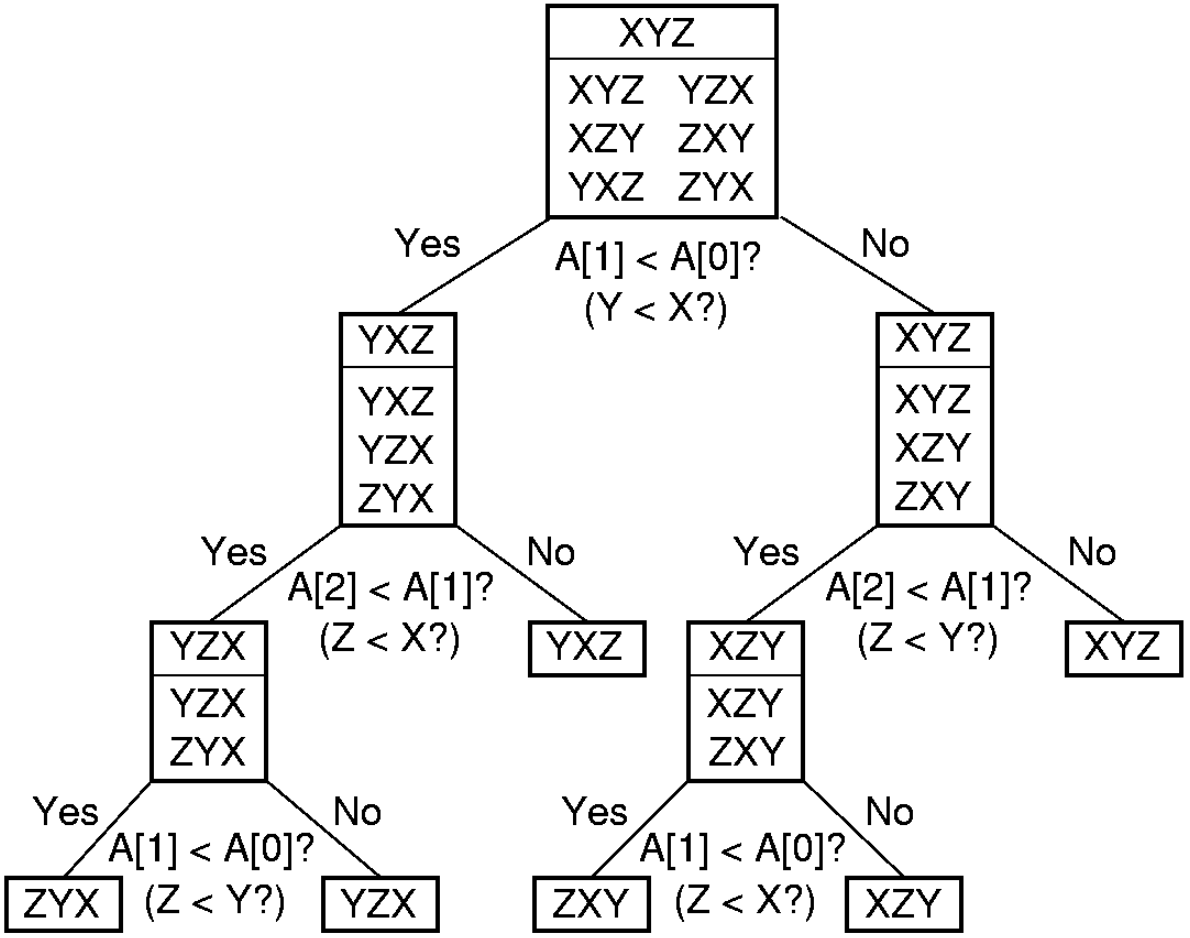
Empirical Comparison: Observation

- Exchange sorts ($O(n^2)$): the slowest at larger inputs.
  - For small elements, they perform well
- Insertion sort: best when “UP” ($O(n)$)
- Quicksort and merge sort provide the best performance
- Radix sort
  - Slower than other $O(n \log n)$ algorithms: $k(\#\text{ of digits}) > \log_r n$
  - Radix/8 is faster than Radix/4 (but, uses more space)

Sorting Lower Bound

- We would like to know a lower bound for all possible *comparison-based sorting* algorithms.
- Sorting is O(n log n) (average, worst cases) because we know of algorithms with this upper bound.
- Sorting I/O takes $\Omega(n)$ time.
- We will now prove $\Omega(n \log n)$ lower bound for sorting in the worst case.

Decision Trees



Lower Bound Proof (1)

- There are $n!$ permutations.
- A sorting algorithm can be viewed as determining which unique permutation of the input corresponds to the sorted list.
- Each leaf node of the decision tree corresponds to one permutation.
- Any comparison-based sorting algorithm must have $n!$ leaves

- What is the minimum depth of nodes in a tree with n nodes?

Lower Bound Proof (2)

- A binary tree with n nodes has $\Omega(\log n)$ levels, so the binary tree with n! leaves has $\Omega(\log n!) = \Omega(n \log n)$ levels.

From Stirling's approximation: n! $\approx \sqrt{2\pi n}(\frac{n}{e})^n$

- Which node in the decision tree corresponds to the worst case?
  - Node in the lowest level
  - Thus, any algorithm would require $\Omega(n \log n)$ in the worst case

What you need to know

- Sorting: puts elements in a certain order
  - Evaluation: # of swaps, # of comparisons
- Sorting algorithms
  - Best: O(n log n) algorithms (e.g., quicksort, merge sort)
  - Some algorithms better than the best ones for special cases
    - Heapsort, Binsort, radix sort
  - Cost analysis
- Lower bound analysis

Questions?