CSE 332: Data Structures and Parallelism

Spring 2022

Richard Anderson

Lecture 16: Sorting IV

Announcements

- Midterm, Friday, November 4
- Next week
  - Graph Theory
  - Graph Theory
  - Veterans Day

Sorting: *The Big Picture*

Simple algorithms: $O(n^2)$
- Insertion sort
- Selection sort
- …

Fancier algorithms: $O(n \log n)$
- Heap sort
- Merge sort
- Quick sort (avg)
- …

Comparison lower bound: $\Omega(n \log n)$

Specialized algorithms: $O(n)$
- Bucket sort
- Radix sort

How fast can we sort?

Heapsort and Mergesort have $O(N \log N)$ **worst** case running time.

These algorithms, along with Quicksort, also have $O(N \log N)$ **average** case running time.

Can we do any better?

Permutations

- How many possible orderings can you get?
  - Example: a, b, c ($N = 3$)
  - (a b c), (a c b), (b a c), (b c a), (c a b), (c b a)
  - 6 orderings = 3•2•1 = 3! (i.e., "3 factorial")
- For N elements
  - N choices for the first position, (N-1) choices for the second position, …, (2) choices, 1 choice
  - $N(N-1)(N-2)\cdots(2)(1)$= <u>N! possible orderings</u>

Sorting Model

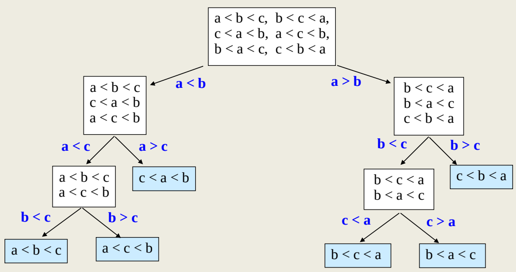
Recall our basic sorting assumption:

We can only compare two elements at a time.

These comparisons prune the space of possible orderings.

We can represent these concepts in a…

Decision Tree



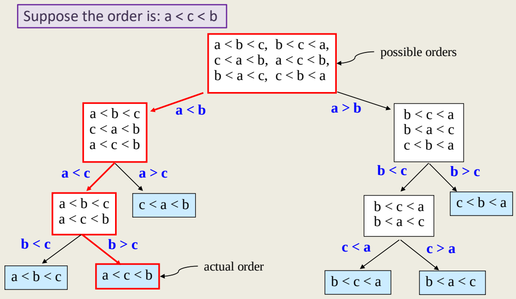
The leaves contain all the possible orderings of a, b, c.

Decision Trees

- A Decision Tree is a Binary Tree such that:
  - Each node = a set of orderings
    - i.e., the remaining solution space
  - Each edge = 1 comparison
  - Each leaf = 1 unique ordering
  - How many leaves for *N* distinct elements?
- Conceptual Tool
- Only 1 leaf has the ordering that is the desired correctly sorted arrangement

Decision Tree Example

Suppose the order is: a < c < b

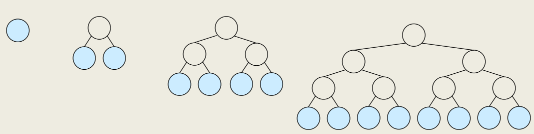


Decision Trees and Sorting

- Every comparison based sorting algorithm corresponds to a decision tree
  - Finds correct leaf by choosing edges to follow
    - i.e., by making comparisons
- We will focus on worst case run time
- Observations:
  - Worst case run time ≥ max number of comparisons
  - Max number of comparisons
    - = length of the longest path in the decision tree
    - = tree height

How many leaves on a tree?

Suppose you have a binary tree of height *h*. How many leaves in a perfect tree?



We can prune a perfect tree to make any binary tree of same height. Can # of leaves increase?

Lower bound on Height

- A binary tree of height h has at most 2^h leaves
- A decision tree has *N*! leaves. What is its minimum height?

Lower bound on log(n!)

$$
\begin{aligned}
n! &= n \cdot (n-1) \cdot (n-2) \cdots 4 \cdot 3 \cdot 2 \cdot 1 \\
&\geq n \cdot (n-1) \cdot (n-2) \cdots \frac{n}{2} \\
&\geq \frac{n}{2} \cdot \frac{n}{2} \cdot \frac{n}{2} \cdots \frac{n}{2} \\
&\geq \left(\frac{n}{2}\right)^{n/2}
\end{aligned}
$$

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

$\Omega(N \log N)$

Worst case run time of any comparison-based sorting algorithm is **$\Omega(N \log N)$** .

Can also show that **average case** run time is also **$\Omega(N \log N)$** .

Can we do better if we don't use comparisons?

Can we sort in O(n)?

- Suppose keys are integers between 0 and 1000

BucketSort (aka BinSort)

If all values to be sorted are integers between 1 and *B*, create an array **`count`** of size *B*, **increment** counts while traversing the input, and finally output the result.

Example *B*=5. Input = (5,1,3,4,3,2,1,1,5,4,5)

count array	
1	
2	
3	
4	
5	

Running time to sort n items?

What about our Ω(n log n) bound?

Dependence on *B*

What if *B* is very large (e.g., 2^{64})?

Definition

- A sort is said to be stable if the order of elements with equal key is preserved.

Fixing impracticality: RadixSort

- RadixSort: generalization of BucketSort for large integer keys
- Origins go back to the 1890 census.
- Radix = "The base of a number system"
  - We'll use 10 for convenience, but could be anything
- Idea:
  - BucketSort on one digit at a time
  - After k^{th} sort, the last k digits are sorted
  - Set number of buckets: B = radix.

Radix Sort Example

Input: 478, 537, 9, 721, 3, 38, 123, 67

BucketSort on 1's

0	1	2	3	4	5	6	7	8	9

BucketSort on 10's

0	1	2	3	4	5	6	7	8	9

BucketSort on 100's

0	1	2	3	4	5	6	7	8	9

Output:

Radix Sort Example (1^{st} pass)

Input data

478
537
9
721
3
38
123
67

Bucket sort by 1's digit

0	1	2	3	4	5	6	7	8	9
	721		3 123				537 67	478 38	9

After 1^{st} pass

721
3
123
537
67
478
38
9

This example uses B=10 and base 10 digits for simplicity of demonstration. Larger bucket counts should be used in an actual implementation.

Radix Sort Example (2^{nd} pass)

After 1^{st} pass

721
3
123
537
67
478
38
9

Bucket sort by 10's digit

0	1	2	3	4	5	6	7	8	9
03 09		721 123	537 38			67	478		

After 2^{nd} pass

3
9
721
123
537
38
67
478

Radix Sort Example (3^{rd} pass)

After 2^{nd} pass

3
9
721
123
537
38
67
478

Bucket sort by 100's digit

0	1	2	3	4	5	6	7	8	9
003 009 038 067	123			478	537		721		

After 3^{rd} pass

3
9
38
67
123
478
537
721

Invariant: after k passes the low order k digits are sorted.

Radix Sort Complexity

In our examples, we had:
- Input size, N
- Number of buckets, B = 10
- Maximum value, M < 10^3
- Number of passes, P =

How much work per pass?

Total time?

Sorting Summary

$O(N^2)$ average, worst case:
- **Selection Sort**, **Bubblesort**, **Insertion Sort**

$O(N \log N)$ average case:
- **Heapsort**: In-place, not stable.
- **BST Sort**: $O(N)$ extra space (including tree pointers, possibly poor memory locality), stable.
- **Mergesort**: $O(N)$ extra space, stable.
- **Quicksort**: claimed fastest in practice, but $O(N^2)$ worst case. Recursion/stack requirement. Not stable.

$\Omega(N \log N)$ worst and average case:
- **Any comparison-based sorting algorithm**

$O(N)$
- **Radix Sort**: fast and stable. Not comparison based. Not in-place.