CS61BL Tutoring

Sorting algorithms

Hongli (Bob) Zhao

Agenda

- Sorting algorithms
  - Comparison sort vs. Counting sort
  - Runtime analysis
  - Stability
  - Practice problems

Sorting Algorithms

- Main ideas
- Stability
- Practice test problems

Notations

- Stability:

A sorting algorithm is stable if it preserves the original ordering of already sorted items

Ex. {2, 3, 1, 4a, 9, 4c, 7, 4b}

{1, 2, 3, 4a, 4c, 4b, 7, 9}

{1, 2, 3, 4a, 4b, 4c, 7, 9}

- Inversions:

Measures how disordered a sequence of items is

- Number of inversions:
    - The minimum number of pair swaps required to sort the list
    - Ex. {1, 3, 4, 2} has 2 inversions

Comparison sorts

- Does not rely on structure of data; only assumes an order exists
  - Arranges elements in order such that `arr[i] <= arr[i+1]` is true for all `i`
- Cannot perform better than O(NlogN)
  - Proof by Stirling's approximation
- Important sorts: Insertion sort, Selection sort, Quick sort, Heap sort, Merge sort

Insertion Sort

- In each loop, start from the leftmost unsorted entry, and compare with the entry immediately to its left; Swap the two entries if arr[i+1] < arr[i]
- Repeat the first step until the entry to the left is not larger than this entry, or this entry has reached the left end of the array
- Repeat the previous two steps until all entries are sorted
- **Runtime:** O(N), O(N^2)
  - Best case achieved when list is nearly sorted
  - Worst case when list is in reverse order

{ 53, 26, 94, 18, 70 }

{ 26, 53, 94, 18, 70 }

{ 26, 53, 94, 18, 70 }

{ 26, 53, 18, 94, 70 }

{ 26, 18, 53, 94, 70 }

{ 18, 26, 53, 94, 70 }

{ 18, 26, 53, 70, 94 }

{ 18, 26, 53, 70, 94 }

Briefly on Tree Sort

- Moving elements in an array takes linear time, how can we make it faster?
- Insert all elements into a Binary Search Tree, and perform traversal
  - Recall in-order traversal of a BST would produce a sorted sequence.
- Runtime to sort N elements depends on the runtime to create the tree:
  - Best case: NlogN
  - Worst case: N^2

Selection Sort

- Starting from the unsorted array, find the minimum value, and swap with the first value
- Starting from the unsorted (N-1) values, find the minimum value, and swap with the second value
- Repeat the process until all values are sorted
- **Runtime**: O(N^2) in all cases:
    - O(N) for finding minimum value
    - O(N) for running selection sort on each entry

{ 53, 26, 94, 18, 70 }

{ 18, 26, 94, 53, 70 }

{ 18, 26, 94, 53, 70 }

{ 18, 26, 53, 94, 70 }

{ 18, 26, 53, 70, 94 }

{ 18, 26, 53, 70, 94 }

Heap Sort

Better than selection sort

- Assuming we are using a max-heap, starting from the unsorted array, heapify the array
- Swap smallest value with the root of the heap, pop the largest value and put at the back of the array
- Re-heapify the (N-1) unsorted array
- Repeat the previous steps until all values have been popped
- **Runtime**: O(NlogN):
  - Heapifying: O(NlogN)
  - Swap: O(1) * N = O(N)
- Question: How to implement Heap Sort using a min-heap?

{ 53, 26, 94, 18, 70 }

{ 94, 70, 53, 26, 18 }

{ 18, 70, 53, 26, 94 }

{ 18, 70, 53, 26 } {94}

{ 70, 26, 53, 18 } {94}

{ 18, 26, 53, 70 } {94}

{ 18, 26, 53 } {70, 94}

{ 53, 26, 18 } {70, 94}

{ 18, 26, 53 } {70, 94}

{18, 26} { 53, 70, 94 }

Merge Sort

- Split: Recursively splits array into halves until further partitioning is impossible (singleton lists)
- Merge: From the bottom level, recursively build up the original sorted list
- **Runtime**: always Omega(NlogN)!
  - O(N): merging back every level
  - O(logN): number of levels

{ 53, 26, 94, 18, 70 }

{53, 26, 94} {18, 70}

{53, 26} {94} {18} {70}

{53} {26} {94} {18} {70}

{26, 53} {94} {18, 70}

{ 26, 53, 94 } {18, 70 }

{ 18, 26, 53, 70, 94 }

Quick Sort

- Select a pivot to partition the array (usually the middle element)
- Smaller value goes left, large value goes right
- Repeat the first two steps until all sub-arrays cannot be partitioned anymore or have met a certain limit
- **Runtime**: Omega(NlogN), O(N^2)
  - Depends on specific choice of pivoting!

{ 53, 26, 94, 18, 70 }

{ 53, 26 } 94 { 18, 70 }

{ 53, 26, 18, 70 } 94

{ 53, 26 } 18 { 70 } 94

18 { 53, 26, 70 } 94

18 { 53 } 26 { 70 } 94

18 26 { 53, 70 } 94

{ 18, 26, 53, 70, 94 }

Counting sorts

- Further takes advantage of structure of data
  - range of data is limited (ex. Alphabet, bits, range of integers)
- Groups objects according to a certain criteria, and use the array structure to indicate ordering
- Can outperform Omega(NlogN)
  - Has linear dependence on size of “dictionary”
  - Runtime: O(N+K), where N is number of items, K is number of digits
- Important sorts: LSD sort, MSD sort

LSD / MSD Radix Sort

- Starting from the most significant / least significant digit, perform counting sort on the digit
- Repeat counting sort on the rest of the digits
  - LSD: from right to left
  - MSD: from left to right
- **Runtime**:
  - LSD: O(D(N+K)), D is the maximum number of digits
  - MSD: Best case O(N+K), Worst case O(D(N+K))

{1219, 2523, 1311, 4215, 3132}

{1311, 3132, 2523, 4215, 1219}

{1311, 4215, 1219, 2523, 3132}

{3132, 4215, 1219, 1311, 2523}

{1219, 1311, 2523, 3132, 4215}

Summary

Algorithm	Best case	Worst case	Stability	Note	
Insertion	N	N^2	Yes	Performance depends on number of inversions	
Selection	N^2	N^2	No	Has constant space. Can be made stable if use linked lists	
Heap	N	NlogN	No	Can achieve linear time if didn’t start from scratch	
Quick	NlogN	N^2	Depends	Runtime depends on choice of pivoting	
Merge	NlogN	NlogN	Yes	Can be highly paralellized	
LSD Radix	D(N+R)	D(N+K)	Yes		
MSD Radix	N+K	D(N+K)	Yes	Can possibly stop early since we sort by the most significant digit (when N < K) the size of our radix (or alphabet)	

Tips for Exam Problems

- Pattern matching
  - Heap sort has the greatest “shuffling” when starting out
  - Merge sort does not start sorting until all splitting has finished
  - Look for growing sorted sequence in selection and heap sort
  - insertion sort moves sequentially to the right
- Algorithm comparisons / Choosing algorithms
  - Example facts:
    - When would we prefer insertion sort over merge sort?
    - What algorithm should we use to sort a linked list?
- Details about algorithms:
  - How many inversions are there in {10,9,7,6,1}?
  - Will {1123, 1830, 1960, 1110, 1210, 1390} ever appear in the process of a LSD radix sort?

Thank you!