Sorting Lower Bounds

- Often can’t control data which needs to be sorted
- Need to plan for worst-case scenario
- Big-Omega, $\Omega$
  - lower bound, the “best worst-case scenario”
- Merge sort had best, $\Omega(n \log n)$
- Can we do better?
Lower Bound on Comparison Sorts

• comparing elements to each other
• ignore other operations for sake of argument
  • (can do this because trying to find best lower bound)
• Sequence S
• compare $s[i]$ with $s[j]$, either T or F
• worst case for work is
  • T – more comparisons necessary
  • F – also more comparisons necessary
  • (i.e., no early exit as in insertion sort)
Lower Bound, cont’d

• For all $i, j$ comparisons we have a *binary decision tree*
  • signifies all possible comparisons for any ordering of $S$
  • internal nodes are the comparisons
  • edges are all the other work (splitting, merging, etc.)
  • path from root to external node is the result of the particular comparisons given a particular permutation of $S$
• Therefore $n!$ external nodes
Lower Bound, cont’d, again

• For a proper binary tree
  • \( \log(n+1) - 1 \leq h \leq (n-1)/2 \)
• \( n_e < n \) (external nodes less than total nodes)
• \( \log(n_e) - 1 \leq \log(n+1) \leq h \)
• Therefore the decision tree \( h \) is at least \( \log(n!) \)
• \( n! \geq (n/2)^{(n/2)} \)
• \( \log(n!) \geq \log((n/2)^{(n/2)}) = (n/2)\log(n/2) \)
• So the \( h \) is \( n \log n \)
Getting Around $\Omega(n \log n)$

If instead of comparing elements to each other, we can compare to some outside standard

- Numerical ordering
- Alphabetical ordering

Then we can “mindlessly” throw things into buckets or categories and avoid even having to worry about the size of the comparison decision tree.
Bucket Sort

• n items
• N integer keys of 0 to N-1
• N buckets
  • bucket[0], bucket[1], ... bucket[N-1]
• loop over keys of S
  • throw current item into bucket for that key
• loop over buckets
  • append bucket items together
Bucket Sort, cont’d

• similar mechanics to hashing with separate chaining
  • don’t care about duplicate keys!
• First loop is $O(n)$ – size of input sequence
• Second loop is $O(N)$ – number of buckets
• total $O(n + N)$ -> $O(n)$
• efficient as long as $N$ isn’t $>> n$
Counting Sort

• If working with actual integer data, just count the number of appearances of each
  • “2 1’s, 15 2’s, 3 4’s, ...”
  • Then expand according to counts
  • $1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 4 4 4 ...$
Stable sorting

- **stable sort** - for equal keys, the sort doesn’t affect the order the data was in in the original input
- **unstable sort** – for equal keys, the sort might flip the order
- Bucket sort will be stable if the buckets act like queues
Lexicographic ordering

• a.k.a. alphabetical ordering
  • elements sorted according to some standard
  • different alphabets give different results!
• \((a, b) < (c, d)\) is True if \(a \leq c\) and \(b < d\)
• \([1, 1, 1] < [1, 2, 0]\)
• Project 3 tuple priorities
• ‘aaa’ < ‘ab’
• ‘11’ < ‘5’ – numeral strings, not numbers.
Radix Sort

• Type of bucket sort
• multiple passes
• for stable version go right to left
• compare just the rightmost part of key
  • then next rightmost, etc
• if keys are integers, start with least significant digit
• $O(k \times (n + N))$
  • $k$ is the number of keys (size of the alphabet)

Sorting Algorithm Review

• Many more algos than we covered

• Real world requires
  • knowing about the input data (is it already ordered?)
  • profiling for real performance
When to Choose Which Sort

• Default – what comes with the language/library

• Insertion
  • small sequences
  • nearly sorted sequences
  • part of hybrid quicksort

• Selection
  • never

• Heap
  • small to medium sequences that fit in RAM, not stable
When to Choose, pt 2

• Quicksort
  • random data
  • large sequences

• Merge sort
  • very large sequence (too big to fit in RAM)
  • GNU command line sort
  • Python/Java use variant called Tim sort

• Bucket/radix
  • small integer keys, strings, tuple keys
Python sorts

• `.sort` and `sorted()`

• Work with any iterable container
  • `__iter__()` defined

• can change default key with `key=`
  • decorate-sort-undecorated pattern
  • `sorted(wordlist, key=len)`
Selection

• Want to find a particular item
• Sort first?
  • best case $O(n)$ bucket/radix
  • then binary search $O(\log n)$
  • total $O(n + \log n)$
• But wait, plain linear search is $O(n)$...
• Can alter quicksort to quickselect
  • prune and search instead of divide and conquer
  • early exit if item found
  • could still be $O(n^{**2})$ but can optimize to be $O(n)$ with acceptable probability
• quick_select.py