Sorting: The Final Chapter

Treesort
Radix sort

Other ways to do sorting

- Treesort: build a binary tree, then read it
  - Can be done in $O(n \log n)$ time
  - Doesn’t require extra arrays
  - Need not be recursive
- Radix sort: a different kind of sorting
  - Group items by a subset of the sort key
  - Repeat for each subset “in order”
  - Now, everything is sorted…
Treesort

- We know how to build a binary tree
  - Can be read back in sorted order
  - Each item takes $O(\log n)$ time to insert
- Sorting algorithm:
  - Build the binary tree: $n$ items at $O(\log n)$ each = $O(n \log n)$
  - Read the binary tree back using in-order traversal
- Issues
  - Keep the tree balanced!
  - May need recursion to traverse the tree in-order

Other $O(n \log n)$ sorting algorithms

- Several data structures can be used to do sorting
  - Binary trees (treesort)
  - Heaps / priority queues: data structure that makes accessing the “least” element fast
    - Build a heap with all of the elements
    - Repeatedly pull off the “least” element
- Hash tables aren’t so good for sorting
  - Hash functions tend to distribute data randomly
  - Is there a way to use something like a hash function for sorting?
One more sort: radix sort

- Most sorting algorithms only require the ability to compare two elements
  - Result is $a < b$, $a = b$, or $a > b$
- Radix sort requires the ability to examine *parts* of each element
  - Example: sorting strings by grouping them letter by letter
  - Example: sort numbers by ordering them by rightmost digit, then digit 1, then digit 2, etc.
- Radix sort is $O(n \times d)$, where $d$ is the number of digits (or characters) that must be compared
  - Note that $d$ is independent of $n$…

Radix sort: how it works

- Repeatedly group the items by parts of the key
  - Maintain order within ties when doing successive groupings
  - In example, unused buckets aren’t always shown
- At the end, items will be sorted
- This is one of the oldest sorting algorithms
  - Originally used for punched cards in the early 1900’s
Radix sort: pseudocode

Radixsort (int theArray[], int n, int d)
{
    for (j = d; d >= 1; d--)
    {
        Initialize 10 groups to empty
        Initialize a counter for each group to 0
        for (i = 0; i < n; i++)
        {
            k = j\text{th digit of theArray}\[i]\]
            Put theArray[i] at the end of group k
            Increment k\text{th counter}
        }
        for (i = 0; i < 10; i++)
        {
            copy all of the items in group i to the “end”
            of theArray in order
        }
    }
}

More on radix sort

- Radix sort doesn’t require comparisons
  - It does require that each element have “pieces” that can be grouped
  - It also requires enough space to do the grouping
- Time is proportional to
  - Number of elements: O(n)
  - Number of “pieces” being grouped: O(d)
- May be faster than other sorts if keys are relatively small: running time is O(nd)
- Has limited use because the algorithm has to know a lot about the makeup of each key
  - Works for numeric keys
  - May work for (short) strings
  - Long strings can take too long!
    - If \(d > \log n\), “standard” sorts are better…
Sorting: wrapping it up

- Generally, the best sorts are the recursive-style algorithms: $O(n \log n)$ average running time
  - Quicksort is good in general, $O(n^2)$ worst-case
  - Mergesort has better worst-case running time, but uses lots of memory
  - Treesort can also work well
- $O(n^2)$ algorithms like insertion sort can be faster for small sets
  - Longer running time for big sets
  - Actual running time $k_1 n^2$ can be less than $k_2 n \log n$ for small $n$, depending on the value of $k_1$ and $k_2$
- General solution:
  - Use recursive sorts to “break up” a large set to be sorted
  - Use $O(n^2)$ sorts to order small pieces of the large set (stop the recursion when set size is around 5–10)
  - Exact tradeoff depends on many factors: speed of machine, speed of comparison, memory usage, etc.

Next time: interesting topics!

- Computer security
  - Using hash functions to sign documents
  - (Very) brief overview of encryption
- Software engineering (writing real code)
  - How does it actually happen?
  - What can I do to prepare for it?
- Preview of future classes
  - CMPE 12C
  - CMPS 101