CSE332: Data Abstractions

Lecture 12: Introduction to Sorting

Dan Grossman

Spring 2012

Introduction to Sorting

- Have covered stacks, queues, priority queues, and dictionaries
  - All focused on providing one element at a time

- But often we know we want “all the things” in some order
  - Humans can sort, but computers can sort fast
  - Very common to need data sorted somehow
    - Alphabetical list of people
    - List of countries ordered by population

- Algorithms have different asymptotic and constant-factor trade-offs
  - No single “best” sort for all scenarios
  - Knowing one way to sort just isn’t enough

More Reasons to Sort

General technique in computing:

Preprocess data to make subsequent operations faster

Example: Sort the data so that you can

- Find the **`k`**th largest in constant time for any **`k`**
- Perform binary search to find elements in logarithmic time

Whether the performance of the preprocessing matters depends on

- How often the data will change
- How much data there is

Careful Statement of the Basic Problem

For now, assume we have *n* comparable elements in an array and we want to rearrange them to be in increasing order

Input:

- An array **`A`** of data records
- A key value in each data record
- A comparison function (consistent and total)

Effect:

- Reorganize the elements of **`A`** such that for any **`i`** and **`j`**, if **`i < j`** then **`A[i] ≤ A[j]`**
- (Also, **`A`** must have exactly the same data it started with)

An algorithm doing this is a comparison sort

Variations on the Basic Problem

1. Maybe elements are in a linked list (could convert to array and back in linear time, but some algorithms needn't do so)

2. Maybe ties need to be resolved by "original array position"
   - Sorts that do this naturally are called stable sorts
   - Others could tag each item with its original position and adjust comparisons accordingly (non-trivial constant factors)

3. Maybe we must not use more than $O(1)$ "auxiliary space"
   - Sorts meeting this requirement are called in-place sorts

4. Maybe we can do more with elements than just compare
   - Sometimes leads to faster algorithms

5. Maybe we have too much data to fit in memory
   - Use an "external sorting" algorithm

Sorting: The Big Picture

Surprising amount of juicy computer science over next 2 lectures…

Simple algorithms: $O(n^2)$	Fancier algorithms: $O(n \log n)$	Comparison lower bound: $\Omega(n \log n)$	Specialized algorithms: $O(n)$	Handling huge data sets
Insertion sort Selection sort Shell sort …	Heap sort Merge sort Quick sort (avg) …		Bucket sort Radix sort	External sorting