Week 3 - Sorting Algorithms

AD 325 - 2022

Contents

Reading & Videos

- Carrano & Henry, Chapters 15 - 16, Interlude 5 Generics
- https://www.coursera.org/learn/algorithms-part1/home/week/2 - Elementary Sorts
- https://www.coursera.org/learn/algorithms-part1/home/week/3
- https://algs4.cs.princeton.edu/20sorting/ (review)
- https://www.geeksforgeeks.org/sorting-algorithms/ (review)
-

Learning Outcomes

- Elementary sorting algorithms
- Comparable interfaces
- Advanced sorting algorithms
- Performance considerations for sorting

Key Concepts

- **comparable** - data to be sorted must be comparable - e.g. implement Java's *Comparable* interface
- **recursion** - The process in which a function calls itself directly or indirectly. Can achieve results similar to loop, but with different memory usage.
- **cost model** - number of compares and exchanges (or array accesses).
- **best, worst, average cases** - a sorting algorithm's performance can vary in different cases - e.g. whether input data is already sorted.
- **memory usage** - in-place sorting algorithms use constant extra space by modifying elements in the existing array or list. Others use extra memory to hold another copy of the array to be sorted.

Key Concepts, cont.

- **stability** - output from a stable sorting algorithm has objects with equal values in the same order as in the original input. https://www.geeksforgeeks.org/stability-in-sorting-algorithms/ (Links to an external site.)Links to an external site.
- **divide-and-conquer** - is an algorithmic paradigm that solves a problem using following three steps:
  - Divide: Break the given problem into subproblems of same type.
  - Conquer: Recursively solve these subproblems
  - Combine: Appropriately combine the answers

Key Sorting Algorithms

Elementary

- Selection sort
- Insertion sort
- Shell sort

Advanced

- Mergesort
- Quicksort

Selection Sort

Select the smallest item in the array, and exchange it with the first entry. Then, find the next smallest item and exchange it with the second entry. Continue in this way until the entire array is sorted.

- average case: uses ~N2/2 compares and N exchanges

Insertion Sort

Consider items one at a time, swapping the current item with larger items to the left.

- average case: for randomly ordered arrays, uses $\sim N^2/4$ compares and $\sim N^2/4$ exchanges.
- worst case is $\sim N^2/2$ compares and $\sim N^2/2$ exchanges
- best case is N-1 compares and 0 exchanges
- works well for certain types of non-random arrays such as partially sorted arrays

Shell Sort

A simple extension of insertion sort that gains speed by exchanging entries that are far apart (*h-sorting*), to produce partially sorted arrays that can be efficiently sorted, eventually by insertion sort. H-sorting is repeated with smaller values of *h* until the array is fully sorted.

- The number of compares used by shellsort is ~ $O(N^{3/2})$

Mergesort

- A simple recursive method to sort an array by dividing it into two halves, sorting the two halves (recursively), and then merging the results
- Mergesort guarantees to sort an array of N items in time proportional to NlogN, no matter what the input.
- It uses extra space proportional to N
- Mergesort has too much overhead for tiny subarrays, so can be sped up using Insertion Sort if num of items < 7

Quicksort

- Works well for a variety of different kinds of input data, and is substantially faster than any other sorting method in typical applications.
- Quicksort is a divide-and-conquer method for sorting. It works by partitioning an array into two parts, then sorting the parts independently.
- It uses in-place swaps and requires time proportional to NlogN on the average to sort N items
- Quicksort uses ~2NlnN compares (and one-sixth that many exchanges) on the average to sort an array of length N with distinct keys.
- Quicksort uses $\sim N^2/2$ compares in the worst case, but random shuffling protects against this case.