Sorting

Manuel Serna-Aguilera

Introduction

The sorting problem takes in the following input and produces the following output

- **Input:** A sequence of n numbers $\langle a_1, a_2, \ldots, a_n \rangle$.
- **Output:** A permutation (reordering) $\langle a_1^{'}, a_2^{'}, \ldots, a_n^{'} \rangle$ of the input sequence such that $a_1^{'} \leq a_2^{'} \leq \ldots \leq a_n^{'}$.

The input sequence is usually an n-element array, although it can take on different forms of lists, such as a linked list.

Why sorting?

You may be asking, "why sorting?" Well, many scientists consider sorting to be the most fundamental problem in the study of algorithms for a multitude of reasons.

- An application's function is to sort.
- Sorting may be a subroutine in a much bigger solution to a more complex problem.
- Not all sorting algorithms are the same, many employ a diverse set of techniques developed over years of research. It is the historical value that matters too.
- Optimizing algorithms' upper and lower bounds to match asymptotically means the running time is optimized, and this can affect the solutions overall running time.

Sorting Algorithms

Back in *Getting Started*, insertion sort and merge sort were discussed, and their implementations are in the Sorting Algorithms directory. Next heapsort, quicksort, counting sort, radix sort, and bucket sort will be discussed and implemented.

Do note that although heapsort's best-case running time is $\Theta(n \cdot \log_2(n))$, quicksort is more popular. Insertion sort, merge sort, heapsort, and quicksort are all comparison sorts, meaning they need to compare values in order to gather information on the data given. Algorithms with running times like these will have the absolute worst-case of $\Omega(n \cdot \log_2(n))$, in other words, this is the best that they can do. Counting sort, radix sort, and bucket sort are linear-time sorting algorithms. Each require some assumptions about the given data and do not need to compare. Again, what algorithm is best to use depends on the application.

You will also hear the phrase "sorting to place" or something like that, a sorting algorithm sorts **in place** if only a constant number of elements of the input array are ever stored outside the array.

Summary:

Table 1: Sorting algorithms

Algorithm	Worst-case running time	Average-case/expected running time	Sorts in place
Insertion sort	$\Theta(n^2)$	$\Theta(n^2)$	yes
Merge sort	$\Theta(n \cdot \log_2(n))$	$\Theta(n \cdot \log_2(n))$	no
Heapsort	$O(n \cdot \log_2(n))$	–	yes
Quicksort	$\Theta(n^2)$	$\Theta(n \cdot \log_2(n))$ (expected)	yes
Counting sort	$\Theta(n+k)$	$\Theta(n+k)$	yes
Radix sort	$\Theta(d(n+k))$	$\Theta(d(n+k))$	yes
Bucket sort	$\Theta(n^2)$	$\Theta(n)$ (average-case)	no

Counting sort assumes there are k integers, radix sort assumes every number has d digits, and bucket sort assumes all numbers are uniformly distrbuted across $[0, 1)$. Of course there are n numbers, things, etc. to sort.

Order Statistics

The ith order statistic of a set of n numbers is the ith smallest number in the set.