Sorting Basics

During the first week, we discussed two different sorting algorithms. This week, we will examine the sorting problem in more detail. We begin by formally defining the problem to be solved.

Definition: The *sorting problem* is defined as follows:

Input: A sequence of n objects or numbers $\langle a_1, \ldots, a_n \rangle$.

Output: A permutation (or reordering) of the sequence as $\langle b_1, \ldots, b_n \rangle$ such that $b_1 \leq \cdots \leq b_n$.

As an added requirement, we will be interested in considering several sorting algorithms that sort in place. By in place, we mean that sorting can be performed with only a constant amount of memory in addition to the memory required to store the input sequence.

Given this focus, we briefly review four sorting algorithms that should be familiar to the student:

- *Insertion sort*: This is an $O(n^2)$ algorithm that involves incrementally inserting new elements into an array in order. The sort is in place in that only an additional swap variable is required.
- *Merge sort*: This is an efficient sorting algorithm in that it can be completed in $O(n \lg n)$ time. As we will see later in this lecture, merge sort is considered to be an asymptotically optimal sort. Unfortunately, because of the recursive nature of the algorithm merge sort is not an in place sort.
- *Selection sort*: The selection sort algorithm is extremely simple to implement but runs in $O(n^2)$ time. The reason for the poor performance is that selection sort requires every element to be compared with every other element. Selection sort, however, is an in place sort.
- *Bubble sort*: Perhaps a favorite sort algorithm to implement, bubble sort is very similar to selection sort in that, in the worst case, every element must be compared with every other element. Thus bubble sort has complexity $O(n^2)$ and is an in place sort. Note that the bound is not a theta bound because bubble sort terminates as soon as no swaps are performed.

To summarize, of these four sorts, only one can be regarded as “efficient”—merge sort. The other three sorts are very inefficient in that they perform brute force sorting on the input sequence. Their advantage, when compared to merge sort however, is that they are all in place sorting algorithms. In the following, we will consider two more “efficient” sorting algorithms and discuss whether or not they can be done in place.