Michael Beaver

CS 355

Assignment 10

November 13, 2012

Proposed Sorting Algorithms System Design

All sorting algorithms have one quality in common: they sort data. Of course, some algorithms may share other similarities. For example, Bubble Sort and Selection Sort both have O(N2) runtimes; likewise, Merge Sort and Quick Sort have O(N logN) runtimes. Other sorting algorithms may be grouped similarly, either by runtime or by some other identifying trait. All of these identifying traits follow a hierarchy of groups of related sorting algorithms that all inherit from a general, abstract sorting class. By definition, hierarchies are mutable; hence, a user could easily include sorting algorithms in the system. Therefore, a class hierarchy of sorting algorithms grouped by runtimes, or other identifying traits, is an optimal system design for running and comparing different sorting algorithms.

Sorting is an ambiguous idea; data may be sorted in many different ways. Hence, the base of the class hierarchy should be an abstract class, possibly named AbstractSort. To make this base class abstract, at least one method would need to be made purely virtual. The obvious decision is to make a general “sort” method purely virtual. To accommodate the algorithms that swap or merge data, virtual methods “swap” and “merge” may be declared in the class header. Of course, any number of general statistical functions may be declared and implemented in the AbstractSort class. Because each sorting algorithm is its own self-contained class and reports its own statistical data, there is no real need for AbstractSort to have its own data members. However, a user may require data members; for example, a user may wish to know the number of instantiated sorting objects.

The sorting algorithms in this system design are grouped by runtime (e.g., O(N2)). Hence, it makes sense that “dummy” classes that represent these groups should inherit from the base class AbstractSort. For example, NSquaredSorts would inherit from AbstractSort, and Bubble Sort, Selection Sort, and other N2 sorting algorithms would inherit from NSquaredSorts. Likewise, Quick Sort and Merge Sort would inherit from NLogNSorts, which would inherit from AbstractSort.

Each sorting algorithm is its own self-contained class that inherits from its respective parent class. Each sorting algorithm class also has its own respective data members for statistical purposes or as needed for actual sorting. Each class also overrides as necessary the virtual “sort,” “swap,” and “merge” methods inherited from AbstractSort. Overriding one method allows for consistent method calls in the driver program, which increases code readability and consistency. Because each sorting algorithm is its own class, each instance of a sorting algorithm object will maintain its own data, which is useful for statistical purposes.

Because the system consists of a class hierarchy, user-defined sorting algorithms may be added into the system with ease. For example, a user-defined optimized version of Bubble Sort may inherit from the original Bubble Sort. See the attached example class hierarchy for an illustration.

Using the system in a driver program is relatively easy and consistent among the sorting algorithms. For example:

BubbleSort myBS(\*arrPtr);[[1]](#footnote-1)

QuickSort myQS;

myBS.sort();

myQS.sort(\*arrPtr);

1. The system should be implemented to allow array pointers to be passed into the constructors and/or directly to the sort methods as necessary. [↑](#footnote-ref-1)