605.202 Data Structures Lab 4

This lab assignment requires you to compare the performance of two distinct sorting algorithms to obtain some appreciation for the parameters to be considered in selecting an appropriate sort. Write a **Quicksort** and a **Natural Merge Sort**. They should both be recursive or both be iterative, so that the overhead of recursion will not be a factor in your comparisons. **Be sure to justify your choice.** Also, in your analysis, consider how your code would have differed if you had made the other choice.

A **Quicksort** (or Partition Exchange Sort) divides the data into 2 partitions separated by a pivot. The first partition contains all the items which are smaller than the pivot. The remaining items are in the other partition. You will write four versions of Quicksort:

- Select the first item of the partition as the pivot. Treat partitions of size one and two as stopping cases.
- Select the first item of the partition as the pivot. Treat a partition of size 100 as a stopping case. Use an insertion sort to finish.
- Select the first item of the partition as the pivot. Treat a partition of size 50 as a stopping case. Use an insertion sort to finish.
- Select the median-of-three as the pivot. Treat partitions of size one and two as stopping cases.

Merge Sort is a useful sort to know if you are doing External Sorting. The need for this will increase as data sizes increase. The traditional **Merge** Sort requires double space. To eliminate this issue, you are to implement Natural **Merge** using a linked implementation. In your analysis be sure to compare to the effect of using a straight **Merge Sort** instead.

Create input files of four sizes: 50, 1000, 2000, 5000 and 10000 integers. For each size file make 3 versions. On the first use a randomly ordered data set. On the second use the integers in reverse order. On the third use the integers in normal ascending order. (You may use a random number generator to create the randomly ordered file, but it is important to limit the duplicates to <1%. Alternatively, you may write a shuffle function to randomize one of your ordered files.) **This means you have an input set of 15 files plus whatever you deem necessary and reasonable.** The size 50 files are to be turned in. The other sizes are for timing purposes. Your code needs to print out the sorted values. Files are available at on the course web page if you want to copy them. Your data should be formatted so that each number is on a separate line with no leading blanks. There should be no blank lines in the file.

Each sort must be run against all the input files. With five sorts and 15 input sets, you will have 75 required runs. For grading purposes, for each sort, turn in output only for the size 50 files. You will have 15 sets of output to turn in for the size 50 files.

Your program should access the system clock to get some time values for the different runs. The call to the clock should be placed as close as possible to the beginning and the end of each sort. If other code is included, it may have a large, albeit fixed, cost, which would tend to drown out the differences between the runs, if any. Why take a chance! **If you get too many zero time data values or any negative time values then you must fix the problem.** One way to do this is to use larger, files than those specified. Another solution is to perform the sorting in a loop, N times, and calculate an average value. You would need to be careful to start over with unsorted data, each time through the loop.

Turn in an analysis comparing the two sorts and their performance. Be sure to comment on the relative runtimes of the various runs, the effect of the order of the data, the effect of different size files, and the effect of different partition sizes and pivot selection methods for **Quicksort**. Which factor has the most effect on the efficiency? Be sure to consider both time and space efficiency. Be sure to justify your data structures. As time permits consider examining alternate methods of selecting the pivot for **Quicksort**. Also consider files of size 20,000 or additional random files - perhaps with 15-20% duplicates. Your analysis must include a table of the times obtained.

It is to your advantage to turn the lab in on time to avoid competition with studying for the final. It will not be accepted after the due date except by prior arrangement.