Lab Assignment 2

CSCI 311 Data Structures and Algorithms in C++

Recall that in our first lab we used a map to count the number of words in a file by incrementing the value, of an associated key, where the key represents one word from our input stream. By the fact that we were using a map, elements were automatically sorted by keys, so it was relatively simple to print out words and their counts in sorted order with respect to the words.

In this second assignment you need to print out words and their counts sorted by the keys or values without using a map. You are required to implement quicksort, merge sort, heap sort, and insertion sort. Each of these sort methods can either use the keys (words) or values (word counts) as the item to be compared. You will take the contents of your map and place it into a vector and sort the vector based on VALUES or KEYS.

Take the following psuedocode as an example:

```
cout << “QuickSort by VALUES” << endl;
WordCounts.quicksort(VALUES);
cout << “QuickSort by KEYS” << endl;
WordCounts.quicksort(KEYS);
```

Which would have the following output:

```
QuickSort by VALUES
does 1
jump 1
jumps 1
so 1
very 1
why 1
brown 2
fox 2
high 2
quick 2
the 2
Quicksort by KEYS
brown 2
does 1
fox 2
```

```
high 2
jump 1
jumps 1
quick 2
so 1
the 2
very 1
why 1
```

You will then repeat this process for Merge Sort, Heap Sort, and Insertion Sort. Be sure to use the following delimiters between each sort run. Remember that your output must match my output exactly, so you have to use these exact cout statements in your program (Your methods and objects may have any name you choose. I'm also assuming that each sort method will print out the result, but you should have a method that will print out the vector and use this between each sort run.)

```
cout << "QuickSort by VALUES" << endl;
WordCounts.quicksort(VALUES);
cout << "QuickSort by KEYS" << endl;
WordCounts.quicksort(KEYS);
cout << "MergeSort by VALUES" << endl;
WordCounts.mergesort(VALUES);
cout << "MergeSort by KEYS" << endl;
WordCounts.mergesort(KEYS);
cout << "HeapSort by VALUES" << endl;
WordCounts.heapsort(VALUES);
cout << "HeapSort by KEYS" << endl;
WordCounts.heapsort(KEYS);
cout << "InsertionSort by VALUES" << endl;
WordCounts.insertionsort(VALUES);
cout << "InsertionSort by KEYS" << endl;
WordCounts.insertionsort(KEYS);
```

I also want you to wrap timers around your initial calls to the sort functions. Your timing method must be able to achieve microsecond accuracy.

```
cout << "QuickSort by VALUES" << endl;
timer_start(t1);
WordCounts.quicksort(VALUES);
timer_end(t2); // Repeat this process for calls to sort methods
```

Do not include any timing information in the output of your program that you submit into the turnin system! The calls may be present in your source code, but make sure that you are not printing out the timing information.

In addition to submitting your code online through the turnin system, you will have to turn in a lab2writeup.pdf file. In this file you will include the timing information I want you to collect. You must also describe how you extended your code from lab1 to implement lab2. Describe any classes you were able to reuse and if your original design was easily extended to implement lab2 functionality.

If you have to redesign your OO design, please describe your new design in lab2writeup.pdf. In addition explain your timing results using the properties of each sort algorithm (for example heapSort is an O(nlgn) algorithm). Also include a discussion of the memory usage of each sort method in terms of input size n.

Here is some code to help you with timing (Assumes you are coding on a Linux machine!):

http://www.guyrutenberg.com/2007/09/22/profiling-code-using-clock_gettime/

Grading:

Test Cases	70%
Write Up	15%
OO Design	10%
Comments	5%

Due: October 5, by 11:59 pm.