O COMP 215 Algorithms Θ

Lab #9

This work should be submitted electronically before 11:59pm Sunday (I'll give you a little more time since this is our first empirical analysis). Submit a .zip file that contains all the file with your code (preferably a single file lab9.cpp, but you can make it multiple files if you prefer).

In this lab, you will be implementing the MergeSort, QuickSort and we will also take this opportunity to do a little empirical analysis (which we did not really have a chance to do before).

First, let's create the data structure to hold the records from file (yay!)

1. Implement a class `myItem`. This class should have two data members, a public integer `serialNumber` and a public string `name`.
2. Implement a default constructor (that takes no input and does nothing) as well as a constructor that takes an integer `sn` and a string `s` and does the obvious initialization of the object.
3. Override the operators $==$, $<$, $>$, $\leq$ and $\geq$ so that an object of type `myItem` is equal to another, less than another, greater than another, etc, if its serial number is equal, less than, greater than that of the other.

Then, we need to read the input file (YAY!!!!!). You may assume that the files will always contain 30000 (thirty thousand) lines

4. Ask the user for the name of a file.
5. Read the file assuming that each line contains exactly one integer and one string, in that order.
6. Store all that data in an array of thirty thousand `myItem`s.

We next implement the sorting algorithms.

7. Implement functions `Merge` and `MergeSort`.
8. Implement functions `LomutoPartition` and `LomutoQuickSort`.
9. Implement functions `HoarePartition` and `HoareQuickSort`.

In order to do the empirical analysis, we will also need equivalent functions that count the number of comparisons executed by each sorting algorithm.

10. Implement functions `cLomutoPartition`, `cLomutoQuickSort`, `cHoarePartition`, `cMerge` and `cMergeSort`, `cHoareQuickSort` that do exactly the same as the originals, but that increase a global variable `count` every time a comparison is executed by one of the algorithms.

And now, we can test the functions you implemented.

11. Run mergesort, lomutoquicksort and Hoarequicksort on each input file given by the user, and for each, your program should print on screen the time taken to run the algorithm (using the algorithm that does not do the comparison count – for this, you will need the timing functions we used before) and the number of comparisons made by each algorithm for each input file entered by the user (remember to reset the global variable `count` to zero between each experiment). You should stop asking the user for new files when the user enters an empty line as name of the next file.