CSCE350 Programming Assignment 1

--Due Date on dropbox.cse.sc.edu--

Please submit *just* quicksort.h. Do not zip the entire folder. All your code goes in that one file.

Description:

- You will code Quicksort
- This is a shorter than usual *because you may need more time to get your environment setup (you need Linux).Your code must run on the department Linux machines (1D39). If you have a separate Linux installation that you want to use, ssh over to the department machines to make sure nothing breaks or just go visit them. You will need to be on the USC network or else VPN in (takes awhile to setup) to ssh in.*

Notes:

- We will be doing this in C++.
- You will build with CMake and Make.
- You will need to do this in Linux, so use either the department lab or work from an Ubuntu install. There are various ways to use Ubuntu without overwriting your OS.
- Included is a copy of Google Test (gtest). Cmake should automatically compile everything for you so as long as all you change is the quicksort.h file.
- Your code should go inside the given functions – if you change the signature (return type, parameters) it won't even compile.
- If you don't have a favorite text editor for C++ on Linux, popular choices are emacs, vim, and Sublime.
- Don't forget to *save* your file before you **make** it (**make** is what actually compiles and links it). **cmake** is what generates a **Makefile** for **make** to use.

Setup:

- Download the program assignment to a convenient directory (outside of of Downloads probably)
- Run the following commands:
  - **tar -xvzf Program_1_350.tar.gz**
  - **cd Program_1_350**
  - **mkdir build**
  - **cd build**
  - **cmake ..**
  - **make**
  - **./runUnitTests**
- You should see 5 tests fail and one pass (there's one test to make sure that your Quicksort can handle a single item array – it shouldn't change anything).

Hoare Partition:

- Code hoarePartition(T[],int,int) in **quicksort.h**
- Remember that the bounds are inclusive (the last parameter is the index of the last item, not one past it)
- You must use the medianOf3() to pick the pivot. Hoare Partition tests will fail if you do not. It

returns the index of the item to use as pivot, the median of 3. Just swap() the median to the left position and then start the algorithm as in the book.

- You can use std::swap (already included and all). This is overloaded in the library but you can probably just use as **swap(A[i],A[j])**, which is close to the book.
- A do-while loop is a natural replacement for a **repeat**.
- The scans (array accesses) need to have the bounds checked (the book skips this). You may assume that it will only get called on a valid array of at least one item, however (a valid quicksort ensures this).
- The code is templated – it should work on most types. If you need to define a variable to hold the *value* of what's in the array, do it as (the latter only works for **int**s):

  T p = A[l];

  not

  int p =A[l];
- *Indices* are all ints, however.
- Test your code:
  - **make**
  - **./runUnitTests**
  - (occasionally, perhaps run **make clean** first if things are behaving unexpectedly)
- 5 of 6 tests should pass.
- I added what I think are pretty useful error messages that show up above the summary (it dumps the arrays and gives some plain text error messages). Feel free to look at the targetgtest.cpp file to see what's going on; it helps to see how your code is getting called, sometimes.

Quicksort

- Code quicksort(T[],int,int)
- Remember that the bounds are inclusive
- Most of the work was done in hoarePartition()
- Test as above, all six tests should pass.

Submission

- upload **quicksort.h** to dropbox.cse.sc.edu (nothing but this one file for this assignment – we'll do more later).
- that's it