CS302 -- Lab 6 -- Sorting

- **CS302 -- Data Structures and Algorithms II**
- **Spring, 2022**
- **James S. Plank**
- **This file: http://web.eecs.utk.edu/~jplank/plank/classes/cs302/Labs/Lab6**
- **Lab Directory: /home/jplank/cs302/Labs/Lab6**

What you Hand In

As with labs 2 and 5, this lab is split into two labs on Canvas:

- Lab 6A: You submit **merge_1_sort.cpp**
- Lab 6B: You submit **quick_2_sort.cpp**

Each of these is graded independently as a 50-point lab.

Quicksort and Merge Sort

Your job is clear -- write **quick_2_sort.cpp** and **merge_1_sort.cpp** from the sorting lecture notes.

In **quick_2_sort.cpp**, you implement Quicksort, choosing the partition as the median of the first, middle and last element. When the size is two, you do not partition, but just sort the two elements by hand. See the lecture notes for how I calculate the middle element.

To compile and test, copy the files **sort_driver_lab.cpp**, **sorting.hpp** and **makefile** from the sorting lecture note directory. On the EECS machines, these are in the directory:

```
/home/jplank/cs302/Notes/Sorting
```

If you type **"make clean ; make lab"**, it will compile your **merge_1_sort.cpp** and **quick_2_sort.cpp** with **sort_driver_lab.cpp**, and create the executables **merge_1_lab_sort** and **quick_2_lab_sort**. You can use the executables in the sorting lecture note directory to test yourself. Those are the ones against which your programs will be graded.

Make your output match mine *exactly*. See the lecture notes for detail on the output of the two programs.

Whenever you use a seed that is a multiple of 2000, the driver program will produce vectors with lots of duplicate entries.

A note on quicksort and duplicate entries

You'll note that there are six cases of duplicate entries for which I haven't specified what you should do with the pivots. These are when two of the three candidates for the pivot equal each other, and the third doesn't. This has caused students consternation in the past. To figure this out, you need to generate the six cases, and then see what my program does with each of them. Then, you do the same thing.

To help with this, there is a program called **quick_2_stdin**, which runs on standard input. That way, you can see how to handle the six cases. For example:

```
UNIX> echo 0 2 3 4 1 6 7 8 1 | quick_2_stdin | head -n 2
S:     0     9        0.00 2.00 3.00 4.00 1.00 6.00 7.00 8.00 1.00
M:     0     9   1.00 1.00 2.00 3.00 4.00 1.00 6.00 7.00 8.00 0.00
UNIX>
```