CSE 6010
Workshop 4

Due Date: 11:59pm on Thursday, October 26
Submit 'quicksort.c', 'contrib.txt', and 'howthingswent.txt" to Gradescope through Canvas
48-hour grace period applies to all deadlines

There are several purposes for the workshop assignments:

- To provide some low-stakes programming practice in C where you can get real-time help from the TAs (if you attend the class in person);
- To cover small programming-related subjects that don't have a logical place among the course topics;
- To give you the opportunity to work together with other students, which can expose you to new ways of thinking about and writing C code; and
- To provide the opportunity for extra credit (if you submit more than 4 workshop assignments).

For this assignment, you may work in groups of 2-3 students. If you prefer, you also may choose to work independently.

If you choose to work as a team, you are expected to **work together for the full assignment**. A divide-and-conquer approach, where each team member works independently on some part of the assignment and the team only interacts when they combine their separate codes to submit, is not in the spirit of this assignment. If you don't want to work with anyone, you should submit independently.

The workshop is designed so that we expect most of you will be able to finish most of the assignment during the **75-minute class period**, possibly with a little extra time offline for cleanup and submission. Nevertheless, we know that different individuals and groups will have different levels of comfort/proficiency with C, and that programming does not always follow a timeline. Thus, it is not strictly required that every submission necessarily will include all topics. As part of your submission, please write a few sentences about how things went in a text file named **'howthingswent.txt'**. This is your opportunity to let us know if you got bogged down in a particular area and how you went about resolving any problems you may have encountered.

Contributions

We ask you to identify how each team member participated by filling in and including the contrib.txt file provided. For each problem, please assess the contribution level for each category according to the following scale.

- 3 = >50% (student did the majority of the work for that category – so there can only be a maximum of one "3" per category + problem combination)
- 2 = 20-50% (solid contribution)
- 1 = 1-20% (minor contribution)
- 0 = no contribution

The categories are:

- Ideas / planning
- Detailed code design
- Writing code / implementation
- Debugging / testing
- Documentation / comments

You can find the template 'contrib.txt' file on Canvas under Workshop 1. When submitting to Gradescope, make sure your filename is '**contrib.txt**'. **Submit this file even if you are a "group" with one member;** delete the space for contributions by other participants when not needed.

Your assignment: Write an implementation of Quicksort using the approach discussed in class and use it to sort an array of items. Each item will consist of three double values. You should sort on the second value.

- Before sorting:
  - Define a struct for an item containing three double values.
  - Read in the item information from a file. The first line of the file will specify the number of items (N). Each following line of the file will specify the three double values of the associated item. You may assume the file is formatted properly, with the correct number of lines to specify the number of items given in the first line of the file.
  - Dynamically allocate an array of your structs to store the item information.
  - The name of the file containing the information about the items to be sorted should be read as the ***first command-line argument***. You do not need to perform any validation and may assume that the user will specify a valid filename.
  - The ***second command-line argument*** is optional. If the second command-line argument is specified as `p`, the code should print out the sorted array, one item per line, just before printing the number of times QuickSort is called (see below). If there is no second command-line argument, or if it is any value other than `p`, do not print out the sorted array.
- Sorting:
  - Use the recursive QuickSort implementation presented in class (with the last element chosen as the pivot). Sort on the **second** value of each item.
  - Count the number of times QuickSort is called so that you can output it later. You will test your array with an "average" set of data as well as with a sorted version of the same data, so that you can look at how this implementation performs in the "best case" of the data (in terms of the number of QuickSort calls).
  - You may wish to define a function to swap two array elements in your array of structs.
- After sorting:
  - If the command-line argument `p` was used, your program should output the array of items, with one item per line, using a statement like the following (include a tab character (\t) between the values associated with each item):

```
printf("%.4f\t%.4f\t%.4f\n", array[i].value1,
array[i].value2, array[i].value3);
```

  - As the final output (the only output unless the command-line argument `p` was used), your program should print out the total number of times QuickSort was called as a single integer followed by the newline character (\n) (no text or fancy formatting).
  - All dynamically allocated memory should be freed (consider using valgrind to test for memory leaks).
- Submission:
  - When submitting to Gradescope, make sure your code is in a single file named '**quicksort.c**'. For this assignment, please keep all your code including function prototypes and definitions in this one file.

Group submissions

Submit your code file named '**quicksort.c**' along with '**contrib.txt**' and '**howthingswent.txt**' on **Gradescope** under **Workshop 2**. If you don't know how to submit on Gradescope as a group, you can follow this tutorial on YouTube.