COP 3502C Programming Assignment # 3
Sorting
Read all the pages before starting to write your code

Introduction: For this assignment you have to write a c program that will use several sorting algorithms with various options.

What should you submit?
You will submit a C file and a pdf file for this assignment

Please include the following commented lines in the beginning of your code to declare your authorship of the code:

/* COP 3502C Assignment 3
This program is written by: Your Full Name */

Compliance with Rules: UCF Golden rules apply towards this assignment and submission. Assignment rules mentioned in syllabus, are also applied in this submission. The TA and Instructor can call any students for explaining any part of the code in order to better assess your authorship and for further clarification if needed.
Caution!!!
Sharing this assignment description (fully or partly) as well as your code (fully or partly) to anyone/anywhere is a violation of the policy. I may report to office of student conduct and an investigation can easily trace the student who shared/posted it. Also, getting a part of code from anywhere will be considered as cheating.

Deadline:
See the deadline in Webcourses. The assignment will accept late submission up to 24 hours after the due date time with 10% penalty. After that the assignment submission will be locked. An assignment submitted by email will not be graded and such emails will not be replied according to the course policy.
What to do if you need clarification on the problem?
Write an email to the TA and put the course teacher in the cc for clarification on the requirements. I will also create a discussion thread in webcourses, and I highly encourage you to ask your question in the discussion board. Maybe many students might have same question as you. Also, other students can reply, and you might get your answer faster.

How to get help if you are stuck?
According to the course policy, all the helps should be taken during office hours. There Occasionally, we might reply in email.

Problem: Let's experiment sorting algorithms by sorting monsters

This assignment is intended to make you implement several different sorting algorithms, and the means to analyze their performance. **This assignment is in two parts: a program, and an analysis.**

You've been given a scattered list of small monsters present in the Knightrola region. You need to evaluate six means of comparison-based sort to get these monsters in order.
You need to implement six sorting algorithms for sorting the monsters:

- Selection sort
- Bubble sort
- Insertion sort
- Quick sort
- Merge sort
- Merge sort, switching to insertion sort at $n \leq 25$ [you need to change the base case of merge sort so that your mergeSort function will call insertion sort with that part of the array whenever your array size decreased to the provided threshold 25]

You will need to sort them based on the following criteria. Note that you will use the same criteria number in our code:

1. Sort monsters by name
2. Sort monsters by weight
3. Sort monsters by name and weight (it should be sorted by name first. However, if multiple monsters have same name, they should be sorted based on weight. For example consider the following names and weights:

   abc 8.0

   bcd 5.0

   abc 5.5

   should be sorted into

   abc 5.5

   abc 8.0

   bcd 5.0)

In order to store the monsters and there finding from sorting, you have to use the following structures:

```
typedef struct monster {
	int id;
	char name[64];
	char element[64];
	int population;
	double weight;
} monster;

typedef struct {
	long long int compares;
	long long int copies;
} sort_results;
```

Input/Output:

Input:

There are *6* input files provided with the assignment with the different number of monsters. Your code should read each of the files and process them. The file name indicates the number of monsters available in the file. *You can assume the file names and the number of monsters are fixed in your code and you can hardcode the names and the sizes if you wish.* Each line of a file represents a monster where the first string is the monster name (Max length 10), next string is element (Max length 10), next integer is population (>=50) and then the last float number represents the weight.

For each criterion, you need to load the data of each file into a monster array and then sort the array using each of the mentioned sorting algorithms based on the criterion (specified by criteria number).

You should write a wrapper function for each algorithm that takes the array, reference of a sort_result structure, and other necessary parameters. While sorting, you should keep track the number of comparison and number of assignment operation happening within the specific algorithm.

Make sure to copy the original array before passing to a sorting algorithm as you will need the original array for another sorting algorithm.

Output:

Your code should produce some console output and also some files.

In the console, you should print whether your array is sorted or not by checking it and then display which sorting algorithm you are calling with which criterion. After coming back, display how many seconds it took to process, status whether your array is sorted or not and then number of comparison and number of copy operation.

Example Console output (I have uploaded a sample console output file. Please check):

```
========
Processing Criteria 1 and file 10K.txt
========
Array status: not sorted by name before calling selection sort
Array status: sorted by name after calling selection sort
Total time taken time_taken second
Total number of comparisons 49995000
Total number of copy operations 29997
Array status: not sorted by name before calling bubble sort
Array status: sorted by name after calling bubble sort
Total time taken time_taken second
Total number of comparisons 49995000
Total number of copy operations 74790471
Array status: not sorted by name before calling insertion sort
Array status: sorted by name after calling insertion sort
Total time taken time_taken second
Total number of comparisons 24940147
Total number of copy operations 24940156
Array status: not sorted by name before calling merge sort
Array status: sorted by name after calling merge sort
Total time taken time_taken second
Total number of comparisons 120572
Total number of copy operations 267232
Array status: not sorted by name before calling merge/insertion sort
Array status: sorted by name after calling merge/insertion sort
Total time taken time_taken second
Total number of comparisons 143846
Total number of copy operations 236067
Array status: not sorted by name before calling quick sort
Array status: sorted by name after calling quick sort
Total time taken time_taken second
Total number of comparisons 156070
Total number of copy operations 113814

========
Processing Criteria 1 and file 20K.txt
========
Array status: not sorted by name before calling selection sort
Array status: sorted by name after calling selection sort
Total time taken time_taken second
Total number of comparisons 199990000
Total number of copy operations 59997
Array status: not sorted by name before calling bubble sort
Array status: sorted by name after calling bubble sort
Total time taken time_taken second
Total number of comparisons 199990000
```

```
Total number of copy operations 301237767
Array status: not sorted by name before calling insertion sort
Array status: sorted by name after calling insertion sort
Total time taken time_taken second
Total number of comparisons 100432578
Total number of copy operations 100432588
Array status: not sorted by name before calling merge sort
Array status: sorted by name after calling merge sort
Total time taken time_taken second
Total number of comparisons 261065
Total number of copy operations 574464
Array status: not sorted by name before calling merge/insertion sort
Array status: sorted by name after calling merge/insertion sort
Total time taken time_taken second
Total number of comparisons 307095
Total number of copy operations 511635
Array status: not sorted by name before calling quick sort
Array status: sorted by name after calling quick sort
Total time taken time_taken second
Total number of comparisons 354562
Total number of copy operations 240213


……
……..
========
Processing Criteria 3 and file 60K.txt
========
Array status: not sorted by name and weight before calling selection sort
Array status: sorted by name and weight after calling selection sort
Total time taken time_taken second
Total number of comparisons 1799970000
Total number of copy operations 179997
Array status: not sorted by name and weight before calling bubble sort
Array status: sorted by name and weight after calling bubble sort
Total time taken time_taken second
Total number of comparisons 1799970000
Total number of copy operations 4837545
Array status: not sorted by name and weight before calling insertion sort
Array status: sorted by name and weight after calling insertion sort
Total time taken time_taken second
Total number of comparisons 1672513
Total number of copy operations 1672514
Array status: not sorted by name and weight before calling merge sort
Array status: sorted by name and weight after calling merge sort
Total time taken time_taken second
Total number of comparisons 784894
Total number of copy operations 1908928
Array status: not sorted by name and weight before calling merge/insertion
sort
Array status: sorted by name and weight after calling merge/insertion sort
Total time taken time_taken second
Total number of comparisons 828722
Total number of copy operations 1655400
Array status: not sorted by name and weight before calling quick sort
Array status: sorted by name and weight after calling quick sort
Total time taken time_taken second
```

```
Total number of comparisons 124042080
Total number of copy operations 527901
```

Your code should produce 3 csv files for three criteria of sorting. ***The filename should be criteria_x.csv where x*** is the criteria number. The csv file should contain the following information in the following format ***(3 csv files available in webcourses for comparison)***:

DataSize, SelectionSortCompare, SelectionSortCopy, SelectionSortTIme, BubbleSortCompare, BubbleSortCopy, BubbleSortTime, InsertionSortCompare, InsertionSortCopy, InsertionSortTime, MergeSortCompare, MergeSortCopy, MergeSortTime, Merge_InsertionSortCompare, Merge_InsertionSortCopy, Merge_InsertionSortTime, QuickSortCompare, QuickSortCopy, QuickSortTime

100000, values of each of the above column separated by comma

200000, values of each of the above column separated by comma

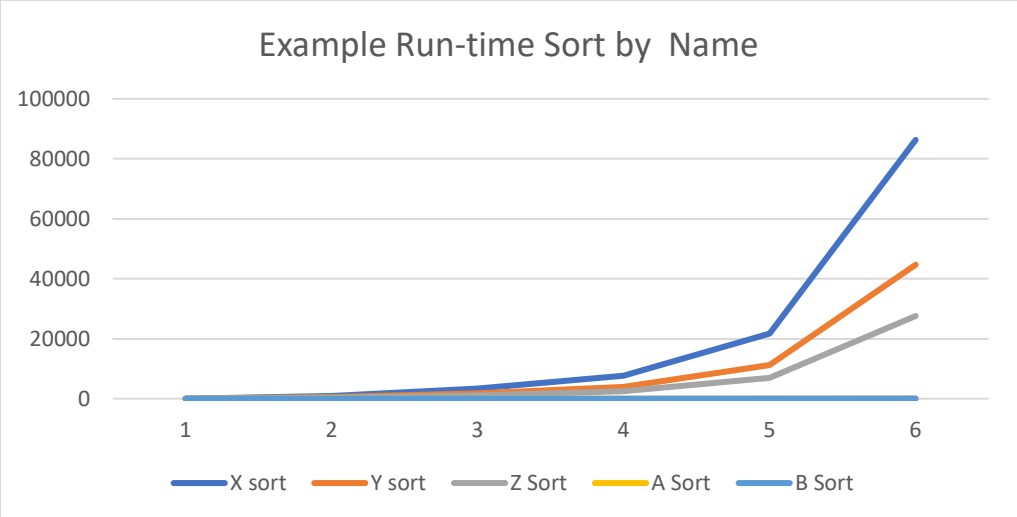
…….

700000, values of each of the above column separated by comma

Additional Submission (pdf file):

You also need to submit a pdf file with a brief analysis report based on the experiment and comparison. Your report should contain the tables you have generated in each of the csv files. Next, you mainly need to show 9 plots (3 criteria and each criterion has 3 matrices to compare [#compare, #copy, execution time]) in the report and add explanation of the plots about the performance of different sorting algorithms. In each plot, you should combine all the sorting algorithms so that you can easily compare them. ***[Additional notes: If you see the lines of multiple algorithms are overlapping in x-axis due to scaling, you can add additional figures showing only those algorithms so that you can see those lines and better compare them. These figures will get 5% extra credit!!!]***

An example plot for run-time could be like this:



Additional Requirements for the program:

- You must have to implement a ***compareTo(monster * m1, monster* m2, int criteria)*** function. Depending on the criterion, the function should return negative if m1 is smaller than m2, positive if m1 is greater than m2, and 0 if both are same. The passed criteria number is very important in this decision. You can add any other parameter if you need.
    - Also, for all comparison during the sorting, you must have to call this compareTo() function
- You must have to implement ***isSorted(monster *m, int length, int criteria)***, function that checks whether the passed array of monsters is sorted or not based on the criterion. Take help from compareTo() function while comparing during this process. The function returns 1 if it is sorted and return 0 if the array is not sorted.

a) You also have to use memory leak detector in your code like past assignments.

Your code must compile in EUSTIS server. If it does not compile in Eustis server, we conclude that your code does not compile even if it works in your computer.

Hints:

- *Please read the last couple of announcements for a lot of hints*
- Running all the algorithms at once will take time. So, for testing purpose, test with each criterion and each file and each algorithm.
- Gradually add other algorithms
- At the end it should be like: for each criterion->for each file-> process each algorithm. So, you will get a csv file for each criterion at the end
- When you are satisfied, then combine them.
- ***For execution time, use the following approach:***

```
clock_t start_cpu, end_cpu;
start_cpu = clock();
//call your sorting algorithm
end_cpu = clock();
print_clocks(end_cpu - start_cpu); //this is a user defined function

void print_clocks(clock_t clocks) {

  printf("%lfs CPU time used\n", ((double) clocks) / CLOCKS_PER_SEC);

}
```

There will be penalty for:

- Not freeing up memory (5%) and not using memory leak detector (-5%)
- Not writing required function (-20%)
- Badly indented code (-15%) and not putting comments in important places (-5%)

Tentative Rubric (subject to change):

1. Do not hardcode any numbers (copy, comparison, time taken, and status of the array). Otherwise there will be penalty -150%
2. Code with all the requirements and correct output: 60
    a. Console output status: 30 (make sure the sorted and not sorted status is correct for each criterion->for each file->for each algorithm)
    b. csv file with proper results and tables: 30
3. Report and plots in pdf: 40 points

FAQ:

1. **What should be counted as a comparison?**
    a. **Whenever you are calling the compareTo function from your sorting algorithm, that should be counted as one comparison**
2. **What should be counted as a copy operation?**
    a. **Whenever you are copying a monster to another monster within your sorting algorithm, that should be counted as a copy operation.**
        i. **For example, a swapping should be counted as 3 copy operation**
3. **Can I hard code the file name and the number of monsters in each file?**
    a. **Yes.**
4. **What should be the content of the report?**
    a. **Please read Additional Submission (pdf file) section again.**
    b. **You don't have to explain several pages. Mainly explain the figures and what you find in the pattern and your conclusion on each algorithm. If you see overlap explain, why and if you add additional figure for that, explain your conclusion further from there.**
5. **What would be the content of a wrapper function?**
    a. **Each algorithm should have a wrapper function and you should call the main sorting algorithm from there.**
    b. **The wrapper function can be used for copying array, initializing counters, printing, etc.**
6. **Can I use the uploaded sorting algorithms?**
    a. **Yes of-course. It is highly encouraged to use those uploaded codes and the codes we have written during the lecture to get consistent numbers.**
7. **Please use the discussion board if you have more questions!!!**