

COP 3502C Programming Assignment#3

Topics covered: Sorting and Searching

Please check Webcourses for the Due Date

Read all the pages before starting to write your code

Introduction: For this assignment you have to write a c program that will utilize the merge sort, insertion sort, and binary search algorithm.

What should you submit?

Write all the code in a single main.c file.

Please include the following lines in the beginning of your code to declare that the code was written by you:

```
/* COP 3502C Programming 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 such incidence to the 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. An assignment submitted by email will not be graded and such emails will not be replied according to the course policy.

Additional notes: The TAs and course instructor can call any students to explain their code for grading.

Problem Scenario: Monster City Tour

You are in a monster city to capture monsters. It is of most importance for you to be aware of the locations of the nearby monsters. Keeping track of this information is known as "monster tracing." You need to write a small program for monster tracing to show your coding skills!

The monster city can be modeled on the Cartesian plane. You are located at the point (x, y) . In addition, you have the Cartesian coordinates of all monsters around you in the city. What you would like to do is write a program that sorts these locations based on their distance from you, followed by handling queries. The queries are of the form of a point you are thinking of visiting. Your program should identify if there is a monster at that location, and if so, what their rank is on the sorted list of the monsters. If no monster is at that location, you should correctly identify this.

Note: There are many important implementation restrictions for this assignment, so to make sure everyone reads these, the section on implementation restrictions will be next, changing the order of the sections as compared to other assignments.

Implementation Restrictions

1. You must use a specified combination of Merge Sort and Insertion Sort to sort the point data. Specifically, for each input case, a threshold value, t , will be given. If the subsection of the array to sort has t or fewer values to sort, Insertion Sort should be used (The reason and this strategy was/will be briefly discussed in the class). Otherwise, Merge Sort should be used. Further details about the comparison used for the sorting are below.
2. You must store your coordinates in a struct that contains two integer fields. You can add more fields if needed.
3. You must implement a `ReadData()` function that reads the required inputs and return the points to be sorted. (note that it does not read the search queries)
4. You must write a function `compareTo` which takes in two pointers, `ptrPt1` and `ptrPt2`, to coordinate structs and returns a negative integer if the point pointed to by `ptrPt1` is closer to you than the point pointed to by `ptrPt2`, 0 if the two locations pointed to by both are identical locations, and a positive integer if the point pointed to by `ptrPt1` is farther from you than the point pointed to by `ptrPt2`. Exceptions to this will be when the two pointers are pointing to points that are the same distance from you, but are distinct points. In these cases, if `ptrPt1`'s x coordinate is lower than `ptrPt2`'s x coordinate, a negative integer must be returned. Alternatively, if `ptrPt1`'s x coordinate is greater than `ptrPt2`'s x coordinate a positive integer must be returned. Finally, if the x coordinate of both points is the same, if `ptrPt1`'s y coordinate is lower than `ptrPt2`'s y coordinate, a negative integer must be returned. If `ptrPt1`'s y coordinate is greater than `ptrPt2`'s y coordinate, a positive integer must be returned.
5. Since your location must be used for sorting, please make the variable that stores your **x and y coordinates global**. Your program should have **no other** global variables.
6. A Binary Search function must be used when answering queries.
7. Your sort function should take in the array to be sorted, the length of the array as well as the threshold value, t , previously mentioned. This function should NOT be recursive. It should be a wrapper function. It means it will call necessary sorting function from here.
8. The recursive sort function (you can call this `mergeSort`) should take in the array, a starting index into the array, an ending index into the array and the threshold value t . In this function, either recursive calls should be made OR a call to an insertion sort function should be made.

The Problem

Given your location, and the location of each monster, sort the list by distance from you from shortest to longest, breaking ties by x-coordinate (lower comes first), and then breaking those ties by y coordinate (lower comes first).

After sorting, answer several queries about points in the coordinate plane. Specifically, determine if a query point contains a monster or not. If so, determine that monster's ranking on the sorted list in distance from you.

The Input (to be read from standard input using scanf(No file I/O. Using file I/O will get 0 in the assignment) - Your Program Will Be Later Tested on Multiple test cases

The first line of the input contains 5 integers separated by spaces. The first integer is n ($2 \leq n \leq 10^6$), representing the number of monsters. The second integer is s ($1 \leq s \leq 2 \times 10^5$), representing the number of points to search for. The third and fourth values are x and y ($|x|, |y| \leq 10000$), representing your location. The last integer, t ($1 \leq t \leq 30$), represents the threshold to be used for determining whether you run Merge Sort or Insertion Sort.

The next n lines of the input contain x and y coordinate values, respectively, separated by spaces, representing the locations of monsters. Each of these values will be integers and the points will be distinct (and also different from your location) and the absolute value of x and y for all of these coordinates will not exceed 10,000.

Then the next s lines of the file contain x and y coordinate values for searching. Both values on each line will be integers with an absolute value less than or equal to 10,000.

The Output (to be printed to console as well as to out.txt file (it means you hav to use both printf and fprintf. You can declare the output file pointer as a global variable as well)

The first n lines of output should contain the coordinates of the monsters, sorted as previously mentioned. These lines should have the x -coordinate, followed by a space, followed by the y -coordinate.

The last s lines of output will contain the answers to each of the s queries in the input. The answer for a single query will be on a line by itself. If the point queried contains a monster, output a line with the following format:

```
x y found at position R
```

where (x, y) is the query point, and R is the one-based rank or position of that monster in the sorted list. (Thus, R will be 1 more than the array index in which (x, y) is located, after sorting.)

If the point queried does NOT contain a monster, output a line with the following format:

```
x y not found
```

Sample Input data(Note: Query points in blue for clarity.)

```
14 5 0 0 5
3 1
-6 -2
-4 3
4 -4
2 4
-1 3
2 2
0 -5
-4 -2
-6 6
4 4
-2 4
0 5
-4 6
2 -1
3 1
0 -5
0 5
-6 7
```

Sample Output (out.txt as well as in standard console output)

```
2 2
-1 3
3 1
-4 -2
-2 4
2 4
-4 3
0 -5
0 5
4 -4
4 4
-6 -2
-4 6
-6 6
2 -1 not found
3 1 found at position 3
0 -5 found at position 8
0 5 found at position 9
-6 7 not found
```

Additional Requirement:

You must have to read data from standard input using scanf and write the result to console and out.txt file. You have to use Merge sort, insertion sort, and binary search for your solution based on the requirements mentioned above. Without using them, you will get zero. The output must have to match with the sample output format. Do not add additional space, extra characters or words with the output as we will use diff command to check whether your result matches with our result. Next, you must have to write a well structure

code. *There will be a deduction of 10% points if the code is not well indented and 5% for not commenting important blocks. Missing header comment may get zero*

- As always, all the programming submission will be graded base on the result from **Codegrade platform**. If your code does not work on Codegrade, we will conclude that your code has an error even if it works in your computer.
- Your code should contain the memory leak detector like programming assignment 1
- Note that you can use `<math.h>` library if you need. In that case you have to use `-lm` option while compiling your code.

For example: `$gcc filename.c -lm`

Some Steps (if needed) to check your output AUTOMATICALLY in a command line in repl.it or other compiler:

You can run the following commands to check whether your output is exactly matching with the sample output or not.

Step1: Copy the sample output to `sample_out.txt` file and move it to the server

Step2: compile your code using typical gcc and other commands.

//if you use `math.h` library, use the `-lm` option with the gcc command. Also, note that `scanf` function returns a value depending on the number of inputs. If you do not use the returned value of the `scanf`, gcc command may show warning to all of the `scanf`. In that case you can use “-Wno-unused-result” option with the gcc command to ignore those warning. So the command for compiling your code would be:

`# gcc main.c leak_detector.c.c -Wno-unused-result -lm`

Step3: Execute your code and pass the sample input file as a input and generate the output into another file with the following command

`$./a.out < sample_in.txt > out.txt`

Step4: Run the following command to compare your `out.txt` file with the sample output file

`$cmp out.txt sample_out.txt`

The command will not produce any output if the files contain exactly same data. Otherwise, it will tell you the first mismatched byte with the line number.

Step4(Alternative): Run the following command to compare your `out.txt` file with the sample output file

`$diff -y out.txt sample_out.txt`

The command will not produce any output if the files contain exactly same data. Otherwise, it will tell you the all the mismatches with more details compared to `cmp` command.

`# diff -c myout1.txt sample_out1.txt` //this command will show ! symbol to the unmatched lines.

Rubric (Subject to change):

- 1) A code not compiling or creating seg fault without producing any result can get **zero**. There may or may not be any partial credit. But at least they will not get more than 50% even if it is for a small

reason. Because, we cannot test those code at all against our test cases to see whether it produces the correct result or not.

- 2) There is no grade for just writing the required functions. However, there will be 20% penalty for not writing and using CompareTo() function, 10% penalty for not writing and using ReadData() function,
- 3) Applying Sorting properly (including merge sort and insertion sort) and writing the correct results in exactly same as required format into out.txt file: 60%
 - There will be various types of simple to complex test cases within the assignment criteria
- 4) Performing Binary search properly and output on console and into the out.txt file after sorted data: 40%
 - There will be various types of simple to complex test cases within the assignment criteria
- 5) There is no point for well structured, commented and well indented code. ***There will be a deduction of 10% points if the code is not well indented and 5% for not commenting important blocks.***

Please see the lecture slides/recording and uploaded codes and recording for learning merge sort and binary search.

You can take help from the codes that I have uploaded. However, they should be typed by you and should be modified as needed

For any clarification on the problem, don't hesitate to use the discussion board or meet us during office hours! Any coding help/debugging help should be taken during office hours.