Problem Scenario: Garage Locator

Our expressway project to connect the garages was not approved due to cost concerns, despite our efforts to minimize the total distance in our PA3. The monster campus, however, is vast, and there is a strong desire to develop a more efficient garage search system to help users locate garages in their vicinity. As you code this system, you should approach it from the user's perspective, structuring your program accordingly.

The monster campus can be represented on a Cartesian plane, with your location at the point (x, y). You also have the coordinates of all garages on campus. Your objective is to create a program that sorts these garage locations by their distance from your position. Additionally, your program must handle queries about specific points that users may wish to visit. For each query, it should determine whether a garage exists at that location and, if it does, identify its position on the sorted list of garages. If no garage is present at the queried location, the program should clearly indicate this.

Note: There are many critical implementation restrictions for this assignment. To ensure that everyone is aware of them, we will present the section on implementation restrictions first, altering the usual order of sections compared to previous assignments.

Implementation Restrictions

1. You are required to implement a specific combination of Merge Sort and Insertion Sort to sort the point data. For each input case, a threshold value, *t*, will be provided. If the subsection of the array to be sorted contains *t* or fewer values, Insertion Sort should be utilized within the mergeSort function (this approach will be discussed briefly in class). For larger arrays, you should apply standard Merge Sort. The rationale behind this is that we want to optimize the sorting process while doing merge sort; instead of waiting for the array to reduce to a size of one, we can leverage Insertion Sort as soon as the array size falls below the threshold. Additional details regarding the sorting comparison method are provided 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 data from the inputs and return the array of points to be sorted.
4. While comparing points during sorting, you need to use a compareTo function to decide which point should come first out of the two points you are comparing. So, you must write a function `compareTo` which takes in two pointers, ptrPt1 and ptrPt2, to coordinate structs and returns:
    a. a negative integer if the point pointed by ptrPt1 is closer to you than the point pointed to by ptrPt2
    b. 0 if the two locations pointed to by both are identical locations,
    c. a positive integer if the point pointed to by ptrPt1 is farther from you than the point pointed to by ptrPt2.
    d. Exceptions to this will be when the two pointers are pointing to points that are the same distance from you, but are distinct points.
        i. In these cases, if ptrPt1's x coordinate is lower than ptrPt2's x coordinate, a negative integer must be returned.
        ii. Alternatively, if ptrPt1's x coordinate is greater than ptrPt2's x coordinate a positive integer must be returned.
        iii. 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. As you are searching points, using compareTo function would help you duirng this process.
7. Write a wrapper sort function that 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 mergesort function 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.
9. Make sure to write the insertion sort function that takes the array, starting and ending index from the array that you want to sort. Make sure you sort only from the starting and ending index of the array. Do not sort the entire array by mistake in this function, which will increase the run-time a lot instead of optimizing!

The Problem

Given your location, and the location of each garage, 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 garage or not. If so, determine that garage's ranking on the sorted list in distance from you.

The Input (to be read from standard input. Using file i/o on any submission (either latest or history of submissions while grading) will result in zero in this assignment and 0 in another past assignment)

The first line of the input contains 5 integers separated by spaces. The first two of these values are x and y ($|x|$, $|y| \leq 10000$), representing your location. The third integer is n ($2 \leq n \leq 10^6$), representing the number of garages. The fourth integer is s ($1 \leq s \leq 2x10^5$), representing the number of points to search for. The last integer, t ($1 \leq t \leq 30$), represents the threshold to be used for determining whether you run Merge Sort of Insertion Sort.

The next n lines of the input contain x and y coordinate values, respectively, separated by spaces, representing the locations of garages. 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 input 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 standard console output)

The first n lines of output should contain the coordinates of the garages, 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 garage, output a line with the following format:

```
x y garage found at position R in the order
```

where (x, y) is the query point, and R is the one-based position of that garage 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 garage, output a line with the following format: