Desert Oasis

Objective
Give practice with some of the java built in features.

Story
You and a group of “friends” are wandering through a desert at different locations. Every friend has a portable radio communications device, and has been using said radio to periodically update their locations. While searching for a source of water you get a call over the radio of the location of an oasis. You know all of you friends will begin moving in a straight line towards the oasis at the same speed. You want to know based on the last known coordinates of your friends, the order in which people arrive in case any funny business goes down prior to reaching the oasis.

Problem
Given the locations of your friends and the location of an oasis determine the friend order in which the oasis is reached.

Input
Input will begin with a line containing 3 integers, n, x_o, and y_o ($1 \le n \le 100{,}000$; $|x_o|$, $|y_o| \le 100{,}000$), representing the number of friends, the x-coordinate of the oasis, and the y-coordinate of the oasis respectively. The following n lines will each contain 2 integers and a string, the ***i***-th of which are x_i, y_i, and $name_i$ ($|x_i|$, $|y_i| \le 100{,}000$; $1 \le |name_i| \le 20$), which represents for your ***i***-th friend their x-coordinate, y coordinate, and their name, respectively.

All names will be distinct; no two friends will be the same distance from the oasis.

Output
Output should contain n lines. Each line will contain the name of one of your friends, each name occurring once. The order in which the names are output should be the order in which your friends reach the oasis (from first to last).

Sample Input	**Sample Output**

`3 5 7`
`3 2 Jacob`
`5 9 David`
`10 4 Eric`
	`David`
	`Jacob`
	`Eric`

`2 100 0`
`0 101 arup`
`-100 0 travis`
	`arup`
	`travis`

Explanation

Case 1
According to the first line of input there are 3 friends and the oasis is at location (5, 7).

Jacob is 2 units away in the x direction and 5 units away in the y direction. The distance will be the square root of 29 or about 5.3851648.

David is 0 units away in the x direction and 2 units away in the y direction. The distance will be the square root of 4 or exactly 2.

Eric is 5 units away in the x direction and 3 units away in the y direction. The distance will be the square root of 34 or about 5.8309519.

David is closest so he will arrive first. The next to arrive is Jacob, and last to arrive is Eric.

Case 2
According to the first line of input there are 2 friends and the oasis is at location (100, 0).

arup [sic] is 100 units away in the x direction and 101 units away in the y direction. The distance will be the square root of 20201 or about 142.13.

travis [sic] is 200 units away in the x direction and 0 units away in the y direction. The distance will be the square root of 40000 or exactly 200.

arup is closest so he will arrive first. The last to arrive will be travis.

Hints

Distance Formula: Remember the distance formula is the square root of the sum of the squares of the dx and dy.

Custom Sort: It is recommended that programs incorporate a custom sort for arranging the friends in the arrival order. A class could look like the following,

```
public static class Friend {
      int x, y;
      String name;
}
```

It might be worth making a comparable interface. See posted code examples on webcourses. In this case we sort by the distance, and friends that are closest will be printed first.

Doubles in Custom Sorting: DO NOT cast a double when using a custom sort! You should always compare doubles using `Double.compare(double1, double2)`.

Precision Issues: Make sure that if you use ints that you don't overflow the 32 bits of precision when doing math.

Avoid Square Root: The square root function is not required in the program to solve; the friends can be sorted by the square of the distance, since the square root function is strictly increasing. In general if you can avoid doing a square root call it is a good idea.

Grading Criteria

- Read/Write from/to standard input/output
    - 10 points
- Good comments, whitespace, and variable names
    - 15 points
- Read in all the input
    - 5 points
- No extra input output (e.g. input prompts, “Please enter the number of friends:”)
    - 10 points
- Programs will be tested on <u>10 cases</u>
    - 5 points <u>each</u>

No points will be awarded to programs that do not compile using javac.

Sometime a requested technique will be given, and solutions without the requested technique will have their maximum points total reduced. For this problem use a comparable interface (or use a custom comparator in some sorting method) such as Collections.sort, Arrays.sort, TreeSet, TreeMap, or PriorityQueue. ***<u>Without this programs will earn at most 50 points!</u>***

Any case that causes a program to return a non-zero return code will be treated as wrong. Additionally, any case that takes longer than the maximum allowed time (the max of {5 times my solutions time, 10 seconds}) will also be treated as wrong.

<u>No partial credit will be awarded for an incorrect case.</u>