

Senior CCC Prep

By Shahrukh Qureshi



Taking a look at S1 from 2020 CCC

Problem Description

Trick E. Dingo is trying, as usual, to catch his nemesis the Street Sprinter. His past attempts using magnets, traps and explosives have failed miserably, so he's catching his breath to gather observational data and learn more about how fast Street Sprinter is.

Trick E. Dingo and Street Sprinter both inhabit a single straight west-east road with a particularly famous rock on it known affectionately as The Origin. Positions on this straight road are measured numerically according to the distance from The Origin, and using negative numbers for positions west of The Origin and positive numbers for positions east of The Origin.

The observations by Trick E. Dingo each contain two numbers: a time, and the value of Street Sprinter's position on the road at that time. Given this information, what speed must Street Sprinter must be capable of?

Input Specification

The first line contains a number $2 \leq N \leq 100\,000$, the number of observations that follow. The next N lines each contain an integer $0 \leq T \leq 1\,000\,000\,000$ indicating the time, in seconds, of when a measurement was made, and an integer $-1\,000\,000\,000 \leq X \leq 1\,000\,000\,000$ indicating the position, in metres, of the Street Sprinter at that time. No two lines will have the same value of T .

For 7 of the 15 available marks, $N \leq 1000$.

Output Specification

Output a single number X , such that we can conclude that Street Sprinter's speed was at least X metres/second at some point in time, and such that X is as large as possible. If the correct answer is C , the grader will view X as correct if $|X - C|/C < 10^{-5}$.

Sample Input 1

```
3
0 100
20 50
10 120
```

Output for Sample Input 1

```
7.0
```

Explanation of Output for Sample Input 1

Since the Street Sprinter ran from position 100 to position 120 between time 0 and time 10, we know its speed must have been at least 2 at some point in time: if it was always less than 2, then the distance of 20 could not be covered in 10 seconds. Likewise, the speed must have been at least 7 in order to travel between position 120 and 50 in 10 seconds.

Sample Input 2

```
5
20 -5
0 -17
10 31
5 -3
30 11
```

Output for Sample Input 2

```
6.8
```

The slide features a light blue grid background. The corners are decorated with various hand-drawn blue icons representing science and technology. Top-left icons include a molecular model, an atom, a beaker with bubbles, the chemical formula H2O, a lightbulb, a brain, a rocket, a globe, and the mathematical symbol sqrt(2). Top-right icons include a calculator, a hexagonal molecule, a globe, a plug, a book, a microorganism, a magnet, a star, and a test tube with a plant. Bottom-left icons include a lightbulb, a brain, a rocket, a globe, a graph, a test tube, and the equation E=mc^2. Bottom-right icons include a magnet, a hexagonal molecule, a globe, a rocket, a DNA helix, a planet, a star, the mathematical symbol sqrt(2), and the chemical formula H2O.

Break it down into smaller problems

Sub-Problems

1. Get input
 - a. How do we store it?
 - i. What data type do we store it as?
2. Sort the data based on the TIME (first index)
3. Calculate SPEED
 - a. How do we calculate it?
 - i. What formula do we use?
 - ii. What does the algorithm look like?
4. Output the SPEED

1. GET INPUT

Scanner is not efficient

- Low buffer memory
 - Low physical memory storage
- Scanner parses the input data

```
Scanner input = new Scanner(System.in);  
String name = input.nextLine();
```

InputStreamReader and BufferedReader

- Very high buffer memory
 - Has a lot of physical memory storage
- Reads the sequence rather than parsing

Method of Declaration #1

```
BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));  
int num = Integer.parseInt(reader.readLine());
```

Method of Declaration #2

```
InputStreamReader inputStreamReader = new InputStreamReader(System.in);  
BufferedReader myReader = new BufferedReader(inputStreamReader);  
String name = myReader.readLine();
```


Input Specification

The first line contains a number $2 \leq N \leq 100\,000$, the number of observations that follow. The next N lines each contain an integer $0 \leq T \leq 1\,000\,000\,000$ indicating the time, in seconds, of when a measurement was made, and an integer $-1\,000\,000\,000 \leq X \leq 1\,000\,000\,000$ indicating the position, in metres, of the Street Sprinter at that time. No two lines will have the same value of T .

1. Get the first number
2. Store N Numbers within a Matrix (2D Array)

1. Get the first number

```
BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));  
int num = Integer.parseInt(reader.readLine());
```

reader.readLine() returns a String so we need to convert it to an Integer

2. Store n Numbers within a Matrix (2D Array)

```
0_100  
20_50  
10_120
```

The space between the numbers makes the dataset a STRING not INTEGER

1. Use a delimiter to separate the numbers by the space
 - a. Delimiter = Sequence of characters that represents separation
2. The separation returns tokens which need to be stored
 - a. We will store these tokens in a List
 - i. You can use an array but it's a bit more complicated to do so here

3. Store each individual token as an element in the matrix

1. Use a delimiter to separate the numbers by the space
split() method is very slow so we will use the StringTokenizer Class

The StringTokenizer Class breaks the String into tokens using a delimiter passed in upon declaration

```
private static List<Integer> getTokens(String str) {  
    StringTokenizer tokenizer = new StringTokenizer(str, " ");  
}
```

We pass in the String and the Delimiter

2. The separation returns tokens which need to be stored

1. Declare the List of type Integer
2. Store the tokens in the list using the add() method


```
private static List<Integer> getTokens(String str) {  
    StringTokenizer tokenizer = new StringTokenizer(str, " ");  
    List<Integer> tokens = new ArrayList<Integer>();  
    while (tokenizer.hasMoreElements()) {  
        tokens.add(Integer.parseInt(tokenizer.nextToken()));  
    }  
    return tokens;  
}
```

As long as there are elements in the StringTokenizer object we keep adding those tokens to the list

Add it to the tokens List using the add() method and convert the tokens from type String to Integer



3. Store each individual token as an element in the matrix

1. Declare the matrix of type Int
 2. Pass in the line entered to the `getTokens()` method we just made
 - a. Storing the tokens in another List
 3. Copy the data from the List to the Matrix
- 

1. Declare the matrix of type Int

Input Specification

The first line contains a number $2 \leq N \leq 100\,000$, the number of observations that follow. The next N lines each contain an integer $0 \leq T \leq 1\,000\,000\,000$ indicating the time, in seconds, of when a measurement was made, and an integer $-1\,000\,000\,000 \leq X \leq 1\,000\,000\,000$ indicating the position, in metres, of the Street Sprinter at that time. No two lines will have the same value of T .

```
int[][] elements = new int[num][2];
```

The problem tells us that there will only be 2 elements in each row

2. Pass in the line entered to the `getTokens()` method we just made
3. Copy the data from the List to the Matrix

```
for (int i = 0; i < num; i++) {  
    List<Integer> tokens = getTokens(reader.readLine());  
    for (int j = 0; j < 2; j++) {  
        elements[i][j] = tokens.get(j);  
    }  
}
```

- Double for loop to traverse the matrix (2D array)
- Declare an Integer List
- Pass in the inputted line
- Enter the second loop and copy the data

2. SORT the data based on time

Java's `Arrays.sort()` method uses quicksort, insertion sort and merge sort

- First Argument: Array (Matrix in this case)
- Second Argument: Comparison
 - We only want to sort the data based on time which is the first index (0) so we have the comparator sort the matrix by the time and adjust the distance accordingly

```
Arrays.sort(elements, (a, b) -> a[0] - b[0]);
```

3. Calculate the SPEED

We will need some variables

int currentIndex -> Representing the current distance

int previousIndex -> Representing the current time

float currentSpeed -> Representing the current calculated speed

float speed

```
int currentIndex = 0;  
int previousIndex = 0;  
float currentSpeed = 0;  
float speed = 0;
```

We will use a for-each loop to let us manipulate the matrix one dimensionally

Within this loop we will:

- Declare 2 temp variables -> 1 for time and 1 for distance
- Check if the current time is 0 (first iteration)
 - If not then we will calculate the speed and check to see if the current calculated speed is > than the previously calculated speed
 - If it is then store the new speed as the currently calculated speed

```
for (int[] data : elements) {  
    int time = data[0];  
    int distance = data[1];  
    if (time == 0) {  
        currentIndex = distance;  
    } else {  
        speed = (float) Math.abs((distance - currentIndex) / (float) (time - previousIndex));  
        previousIndex = time;  
        currentIndex = distance;  
        if (currentSpeed < speed) {  
            currentSpeed = speed;  
        }  
    }  
}
```

We use `Math.abs` since the problem states that its looking for the absolute value

Output Specification

Output a single number X , such that we can conclude that Street Sprinter's speed was at least X metres/second at some point in time, and such that X is as large as possible. If the correct answer is C , the grader will view X as correct if $|X - C|/C < 10^{-5}$.

1. Output the SPEED

```
System.out.println(currentSpeed);
```

The slide features a light blue grid background. The corners are decorated with various hand-drawn blue icons representing science and technology. Top-left icons include a molecular structure, an atom, a beaker with bubbles, the chemical formula H2O, a lightbulb, a brain, a graph, a test tube, and the equation E=mc^2. Top-right icons include a calculator, a hexagonal molecule, a globe, a plug, a book, a microorganism, a magnet, a star, and a test tube with a plant. Bottom-left icons include a rocket, a globe, a lightbulb, a brain, a graph, a test tube, and the equation E=mc^2. Bottom-right icons include a magnet, a hexagonal molecule, a globe, a rocket, a DNA helix, a planet, a test tube, and the chemical formula H2O.

Test your solution

```
3
0 100
20 50
10 120
7.0
Press any key to continue . . .
```

Sample Input 1

```
3
0 100
20 50
10 120
```

Output for Sample Input 1

```
7.0
```

```
5
20 -5
0 -17
10 31
5 -3
30 11
6.8
Press any key to continue . . .
```

Sample Input 2

```
5
20 -5
0 -17
10 31
5 -3
30 11
```

Output for Sample Input 2

```
6.8
```

Grading

Status: Completed [15/15 points]

Subtasks: 1 2 3 4 5 6 7 8

Sample tests [2/2 tests passed]

Test 1	Correct	0.2 seconds
Test 2	Correct	0.2 seconds

Subtask 1 [1/1 point] [1/1 test passed]

Test 3	Correct	0.2 seconds
--------	---------	-------------

Subtask 2 [1/1 point] [1/1 test passed]

Test 4	Correct	0.2 seconds
--------	---------	-------------

Subtask 3 [1/1 point] [1/1 test passed]

Test 5	Correct	0.2 seconds
--------	---------	-------------

Subtask 4 [1/1 point] [1/1 test passed]

Test 6	Correct	0.2 seconds
--------	---------	-------------

Subtask 5 [1/1 point] [1/1 test passed]

Test 7	Correct	0.2 seconds
--------	---------	-------------

Subtask 6 [1/1 point] [1/1 test passed]

Test 8	Correct	0.2 seconds
--------	---------	-------------

Subtask 7 [1/1 point] [1/1 test passed]

Test 9	Correct	0.2 seconds
--------	---------	-------------

Subtask 8 [8/8 points] [8/8 tests passed]

Test 10	Correct	0.7 seconds
---------	---------	-------------

Test 11	Correct	0.7 seconds
---------	---------	-------------

Test 12	Correct	0.7 seconds
---------	---------	-------------

Test 13	Correct	0.6 seconds
---------	---------	-------------

Test 14	Correct	0.7 seconds
---------	---------	-------------

Test 15	Correct	0.6 seconds
---------	---------	-------------

Test 16	Correct	0.6 seconds
---------	---------	-------------

Test 17	Correct	0.7 seconds
---------	---------	-------------


```

import java.io.BufferedReader;
import java.util.ArrayList;
import java.util.List;
import java.util.StringTokenizer;
import java.io.InputStreamReader;
import java.util.Arrays;

public class S1 {
    Run | Debug
    public static void main(String[] args) throws Exception {

        BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));
        int num = Integer.parseInt(reader.readLine());

        int[][] elements = new int[num][2];

        for (int i = 0; i < num; i++) {
            List<Integer> tokens = getTokens(reader.readLine());
            for (int j = 0; j < 2; j++) {
                elements[i][j] = tokens.get(j);
            }
        }

        Arrays.sort(elements, (a, b) -> a[0] - b[0]);

        int currentIndex = 0;
        int previousIndex = 0;
        float currentSpeed = 0;
        float speed = 0;
        for (int[] data : elements) {
            int time = data[0];
            int distance = data[1];
            if (time == 0) {
                currentIndex = distance;
            } else {
                speed = (float) Math.abs((distance - currentIndex) / (float) (time - previousIndex));
                previousIndex = time;
                currentIndex = distance;
                if (currentSpeed < speed) {
                    currentSpeed = speed;
                }
            }
        }
        System.out.println(currentSpeed);
    }

    private static List<Integer> getTokens(String str) {
        StringTokenizer tokenizer = new StringTokenizer(str, " ");
        List<Integer> tokens = new ArrayList<Integer>();
        while (tokenizer.hasMoreElements()) {
            tokens.add(Integer.parseInt(tokenizer.nextToken()));
        }
        return tokens;
    }
}

```