Site Exploration

Objective

Give practice divide and conquer / dynamic programming.
Give practice with Fenwick Trees.
Give practice with counting problems.
Give practice with Sweeps.

Story

You managed to escape unsathed from the collapsing temple with a fair bit of artifacts in tow, yet your work is not done. One last thing you recovered from the scrolls is a piece of information about the location of three more ancient sites (each possibly filled with historically significant items). You will head out with your teams to these locations. The only problem is you don't know exactly which spot these sites might be.

The scrolls mentioned that the sites were along a nearby river, and that the sites were built in the direction of the river flowing over time. The scrolls did not provide which spots they might be located. Luckily there are several well-known spots on the river that have some ancient markers. You also know roughly when each marker was built. Now all you need to do is check out the ones that are the most likely candidates.

To ease your mind you decided to count the number of possible triples of markers that would represent a valid set of sites according to the scrolls.

Problem

Given the location of possible ancient sites as a distance from the river source (d_i), and the times at which they were built (t_i), determine the number of triples that satisfy the scroll's property (e.g. (x, y, z) such that $d_x < d_y < d_z$ and $t_x < t_y < t_z$).

Input

Input will begin with a line containing a positive integer, ***n*** (***n*** ≤ 200,000), representing the number of possible sites built along the river. The following ***n*** lines will each contain 2 positive integers, the ***i***-th of which are $\boldsymbol{d_i}$ and $\boldsymbol{t_i}$ ($\boldsymbol{d_i}$, $\boldsymbol{t_i}$ ≤ 1,000,000), which represents for the ***i***-th possible site the distance from the source of the river and the time of the possible site's construction.

All distances and times will be distinct.

Output

Output should contain 1 line containing the number of possible triplets of sites that form a valid construction sequence.

Sample Input	**Sample Output**

5
8 6
7 5
30 9
6 8
1 1
	5

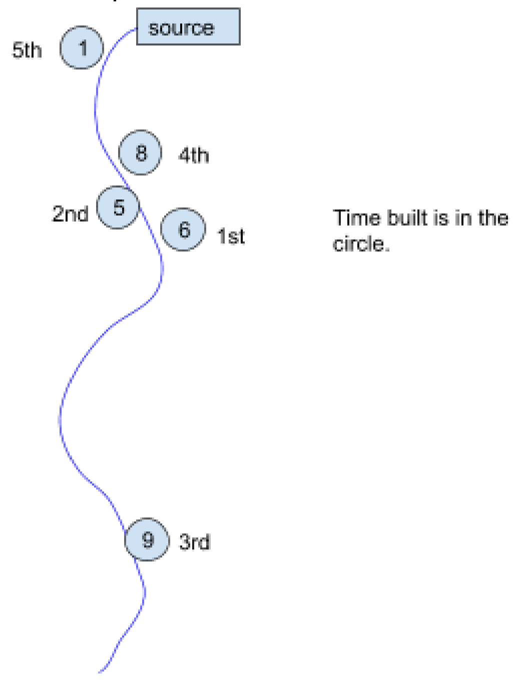
4
3 1
2 2
1 3
4 4
	0

Explanation

Case 1

In the first case we have 5 locations.

The first point is 8 units from the river source; it was built at time 6.
The second point is 7 units from the river source; it was built at time 5.
The third point is 30 units from the river source; it was built at time 9.
The fourth point is 6 units from the river source; it was built at time 8.
The fifth point is 1 unit from the river source; it was built at time 1.



The triplets that work is (5th, 2nd, 3rd), (5th, 2nd, 1st), (5th, 1st, 3rd), (5th, 4th, 3rd), and (2nd, 1st, 3rd)

Case 2

There are 24 total triplets, all of which are invalid.

Hints

THIS IS NOT FLOW: Do not try to solve this using flow. The runtime would be crazy. The source is only a story element.

Yet Another Sorting Problem (YASP): I recommend sorting your points by the time in which they were built (or the distance they are from the source; only sort by one of these values please). Why? Because, if the points are sorted by their time, then we need to find triplets of points that have strictly increasing distances in the order in which they occur in the sorted list. It makes the problem a lot easier (especially when using the Fenwick Tree).

Consider 2D Point Visualization: Try approaching this problem where the time is an x-component and the distance is a y-component. This allows mapping all the points into a 2D Cartesian grid.

Dividing to Conquer: The trick to the problem (to avoid using a triple for loop) is to work from two angles.

The first subproblem you want to solve is for all points, how many points are built before me AND closer to the source than me. With the 2D visualization, we would count the number of points in the lower left quadrant of every point.

The second subproblem you want to solve is for all points, how many points are built after me AND further from the source than me. With the 2D visualization, we would count the number of points in the upper right quadrant of every point.

Using these two subproblems we could count the number of triplets where a point is the middle value. This is done by considering all the ways we can select a lower left TIMES all the ways we can select an upper right point.

Counting the Points In a Quadrant: You can use a Fenwick Tree (BIT) to count the number of points in a Quadrant for all points quickly. How? By sweeping!

For simplicity consider counting the lower left quadrant for now. We should use the BIT to store the number of points with a y-value less than or equal a given point. We want to ensure that we only count the points that have a smaller y value, IF the x value is also smaller. That is we don't want to add values with a larger x value before processing the smaller x values. That could cause us to say that more points are in the lower left quadrant than actually exist.

Remember that a BIT sums the values that are at some or before some given index. We can if we count the number of points by incrementing the bit at the index that represents its y-value. To add the y-value to the BIT we increment the index of the BIT by 1 at the position of the given y-value.

We can use a similar approach for the upper right quadrant with an EMPTY BIT. The differences would be that we need to loop in the reverse direction of the time and we need to know the values that are greater (current total - the values less than or equal).

Pit Falls

- **DO NOT** use flow
- **DO NOT** use a triple for loop
    - You should only use it for validation in a separate program that you should not upload
- **DO NOT** use the points strictly in the order given (sort first)
- Within your sweep **DO NOT** add the point to the BIT BEFORE getting the sum of points.
- **DO NOT** use ints to store your answers
- **DO** use Fenwick Trees (BITs)
- **DO** use longs for your BITs, your partial answer arrays, and your final answer
- **DO** store your points as objects with a custom comparator
- **DO** sort your data by some coordinate and use the other coordinate to index

Grading Criteria

- Good comments, whitespace, and variable names
    - 15 points
- No extra input output (e.g. input prompts, “Please enter the number of friends:”)
    - 10 points
- Sort the spots by one of the given values
    - 10 points
- Break problem into subproblems
    - 10 points
- Use subproblems to create the answer
    - 5 points
- Programs will be tested on 10 cases
    - 5 points each

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 Fenwick Tree. ***Without this programs will earn at most 50 points!***

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.

No partial credit will be awarded for an incorrect case.