USA  Tutor

Rectangular Pasture

Analysis by David Yang

Statement Summary

You are given N<=2500 cells and you want to construct rectangles.

Find the number of distinct rectangles that you can build.

Each cell is denoted by a distinct X and Y coordinate <=1e9

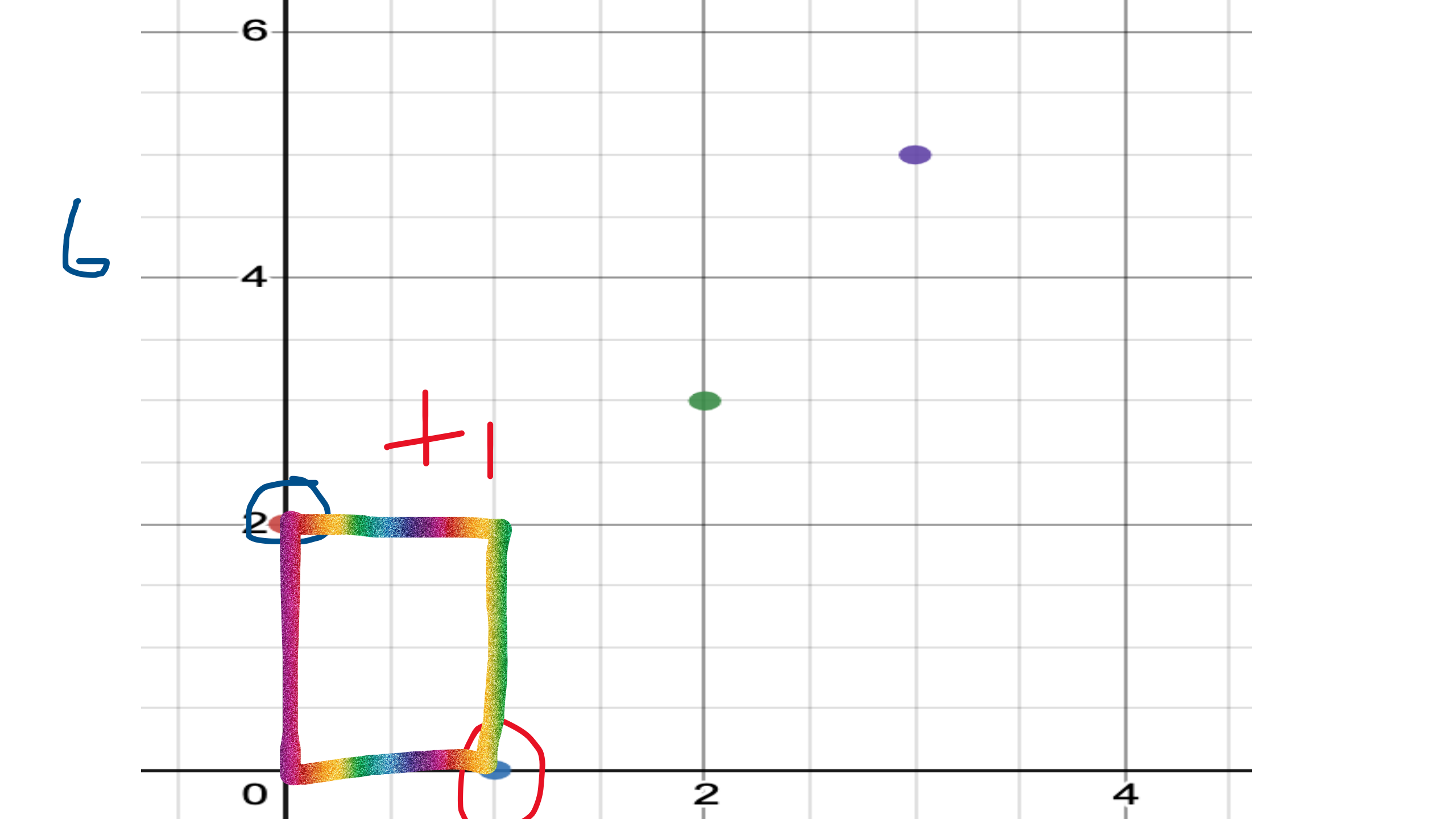
Statement Summary

You are given N<=2500 cells and you want to construct rectangles.

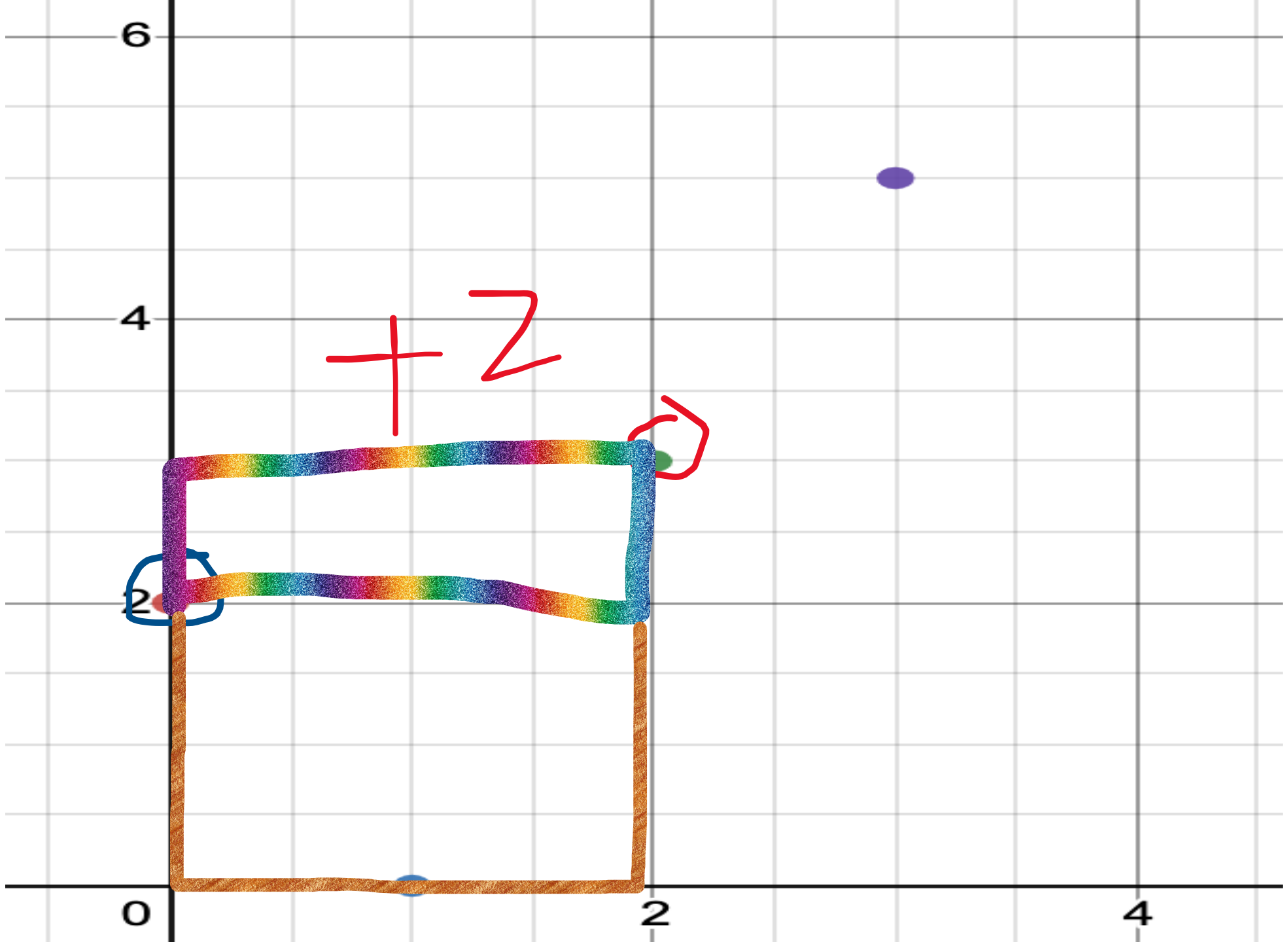
Find the number of distinct rectangles that you can build.

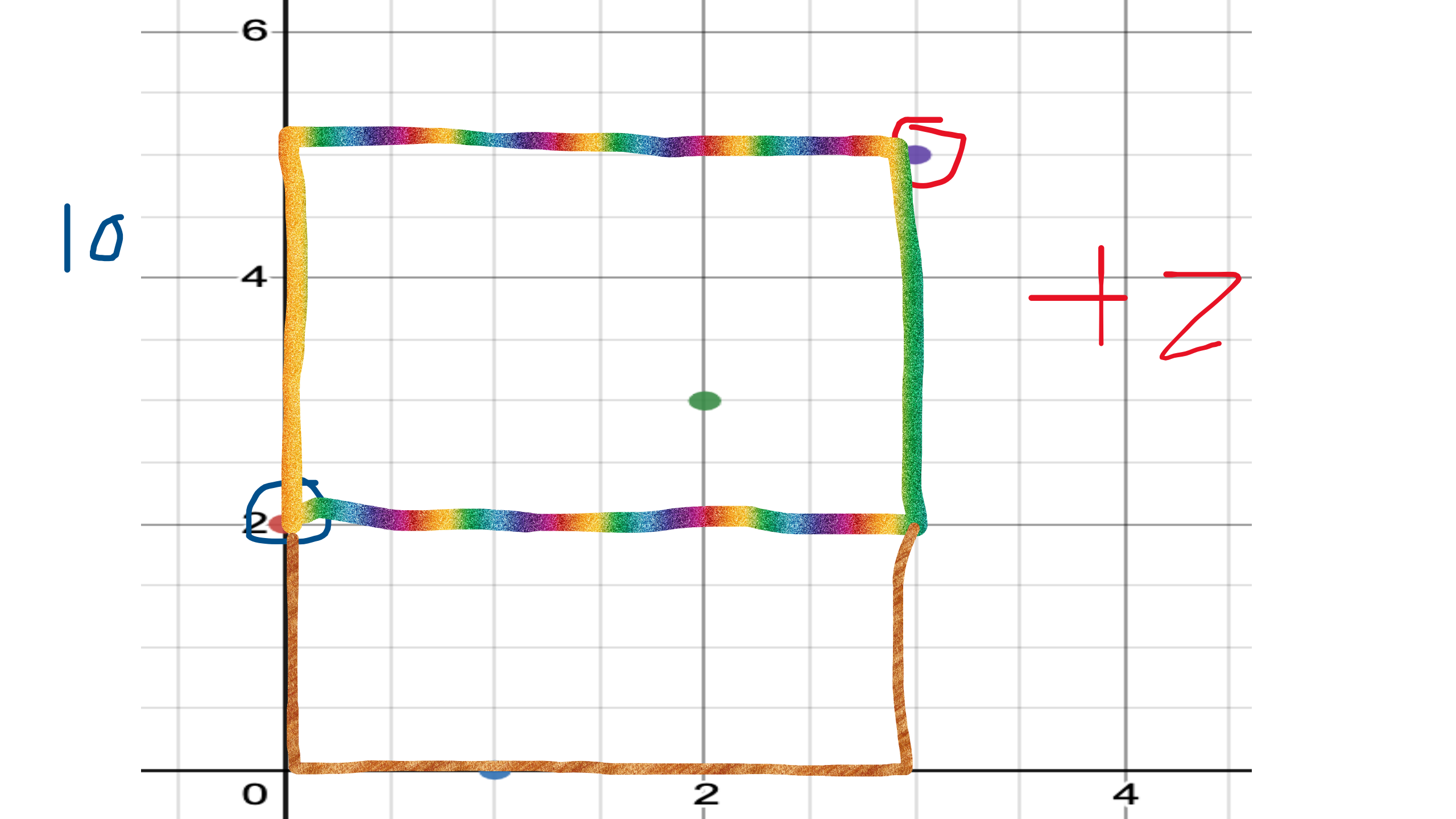
Each cell is denoted by a distinct X and Y coordinate <=1e9

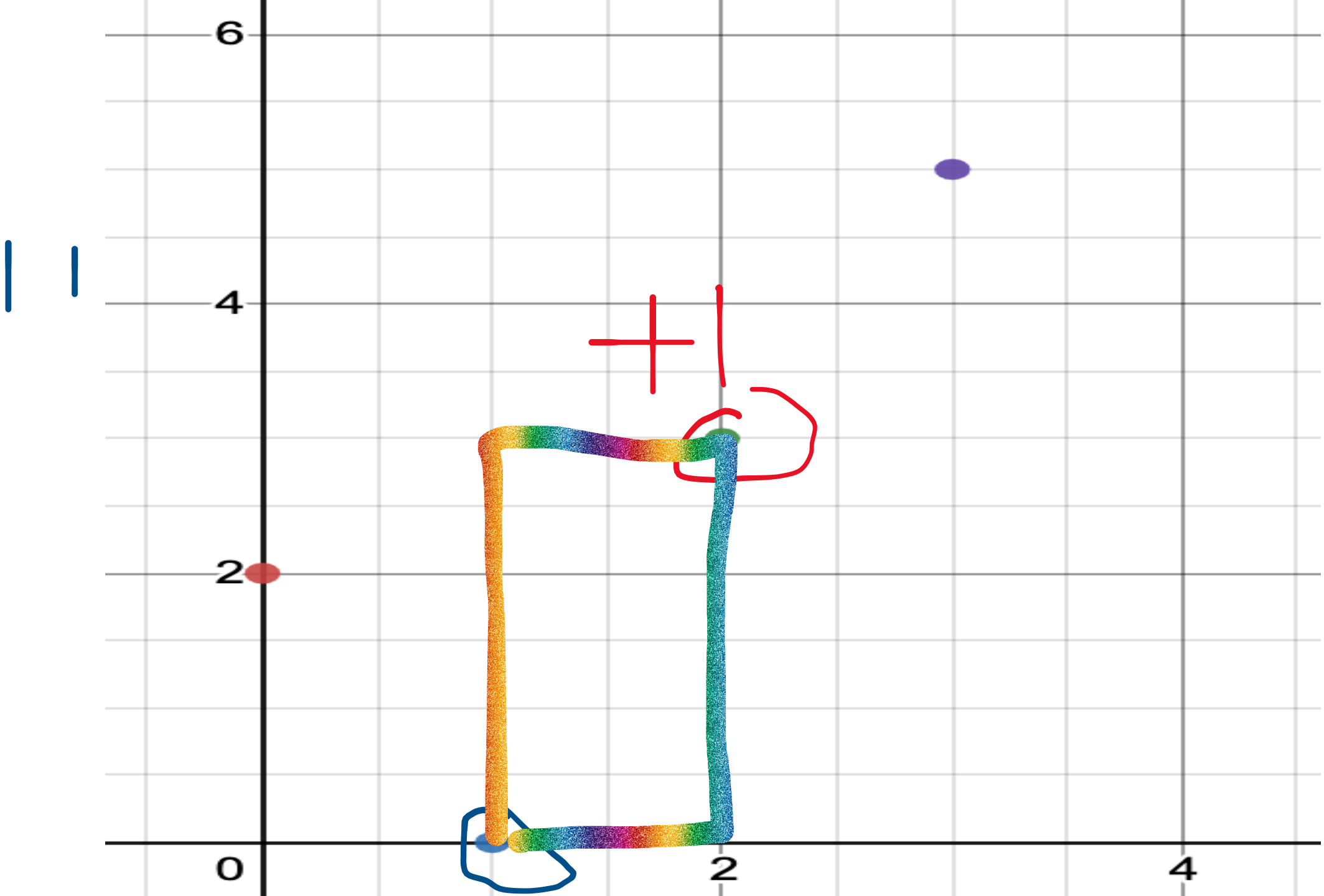
6
4
2
0
2
4


6
4
2
0
2
4
+1

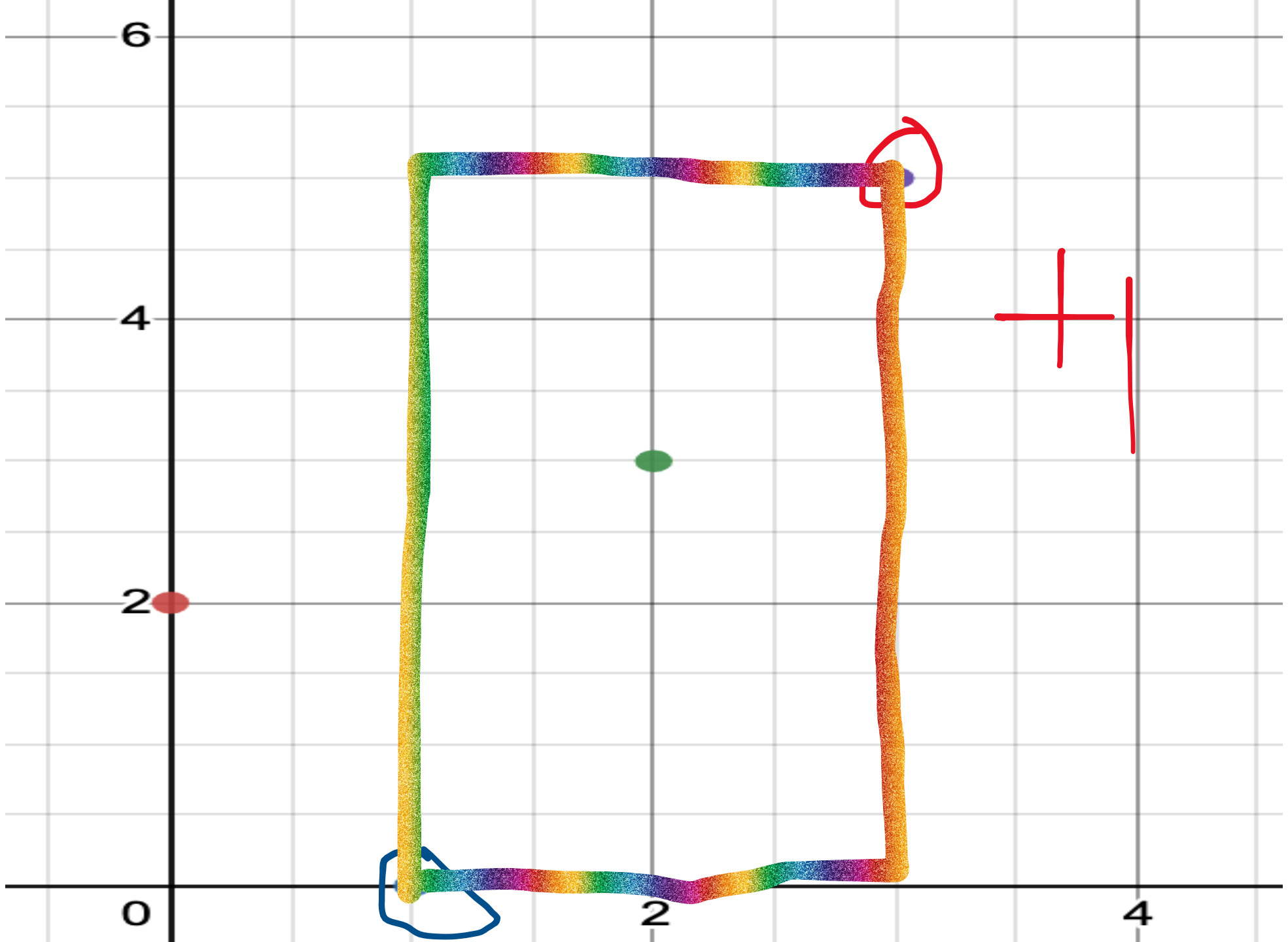
8



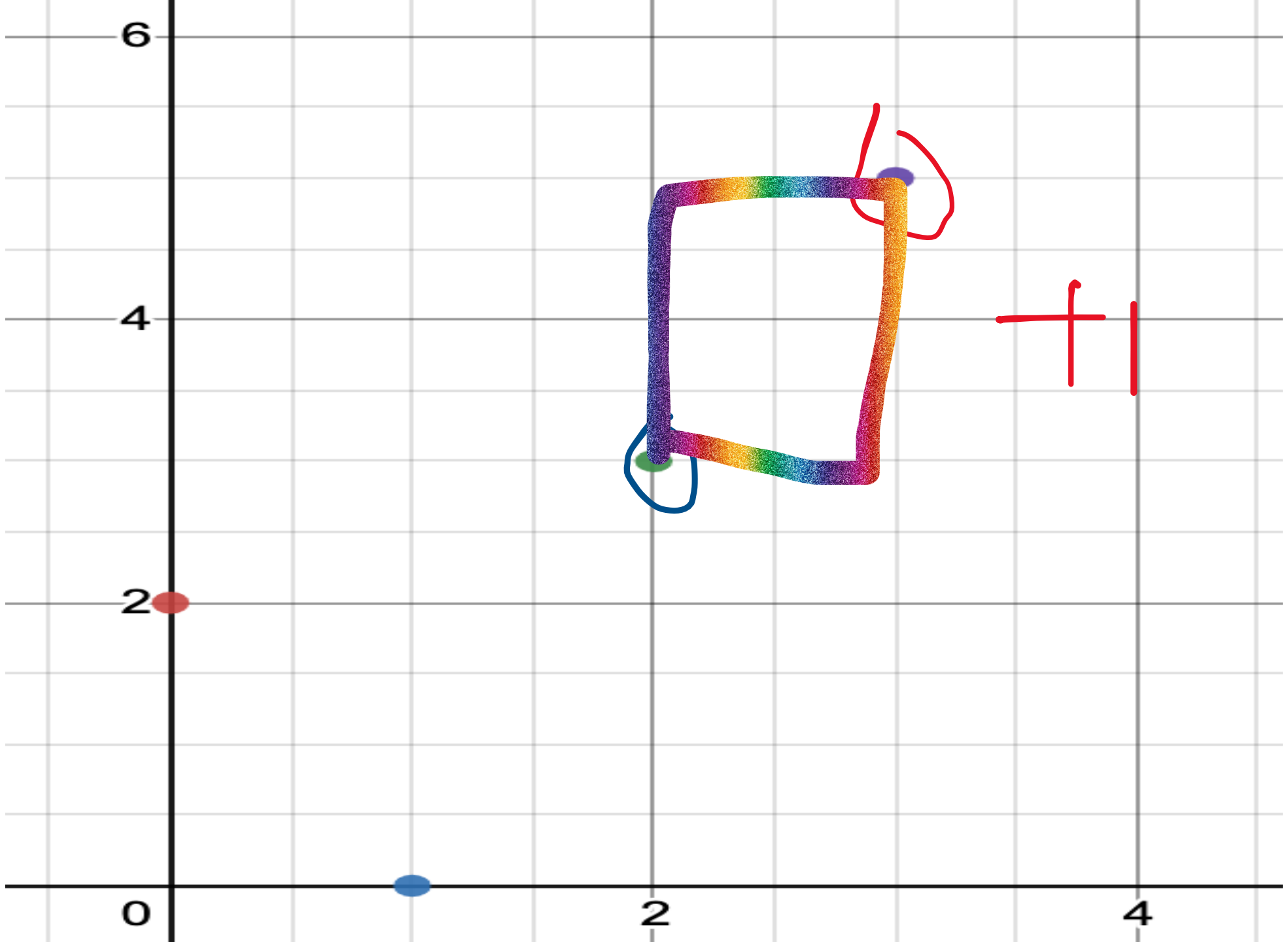

6
4
2
0
2
4
10
+2


6
4
2
0
2
4
+1

12



13



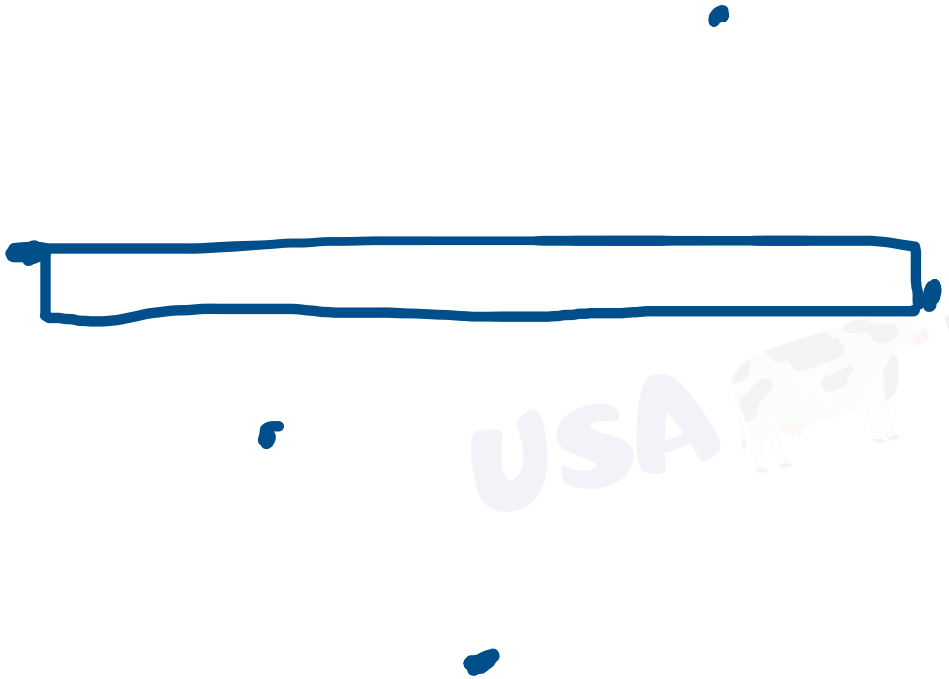
+1

Final Solution

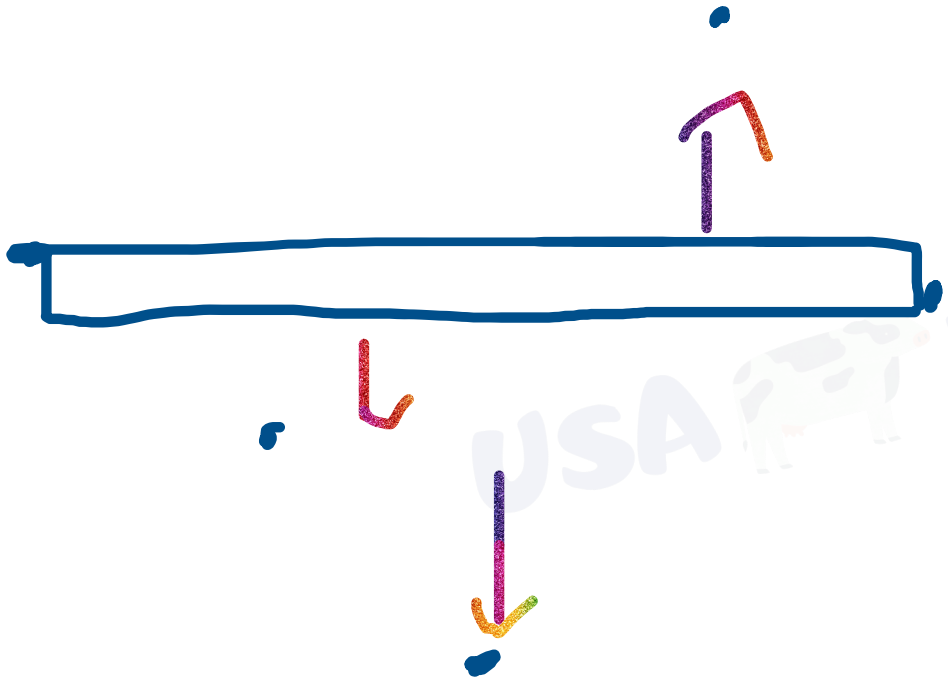
We built 13 distinct rectangles! Yay! We're correct!

But how did we build them?

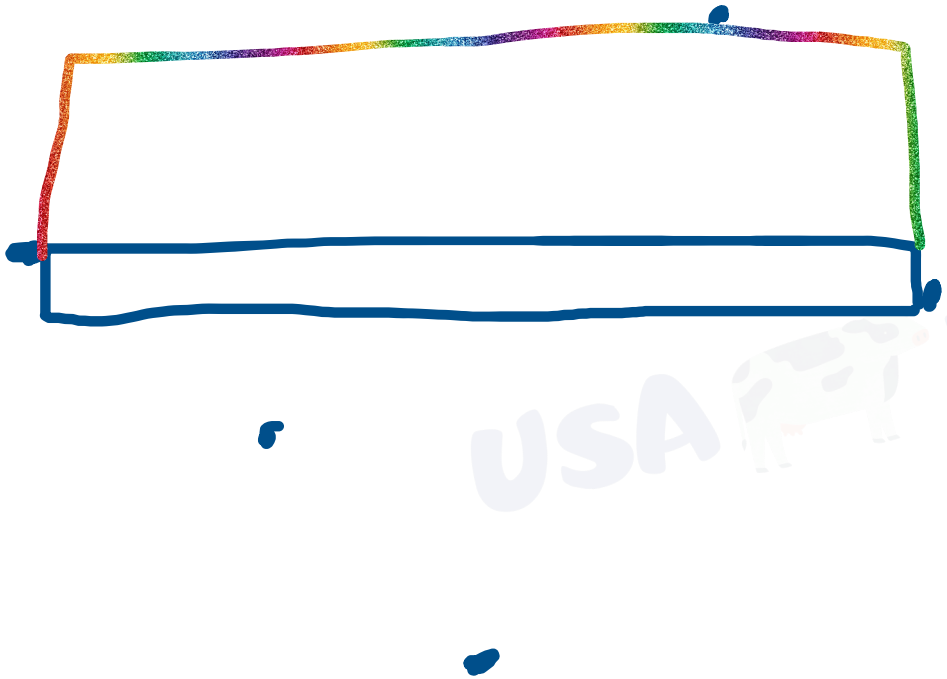
Deceiving Sample



Deceiving Sample

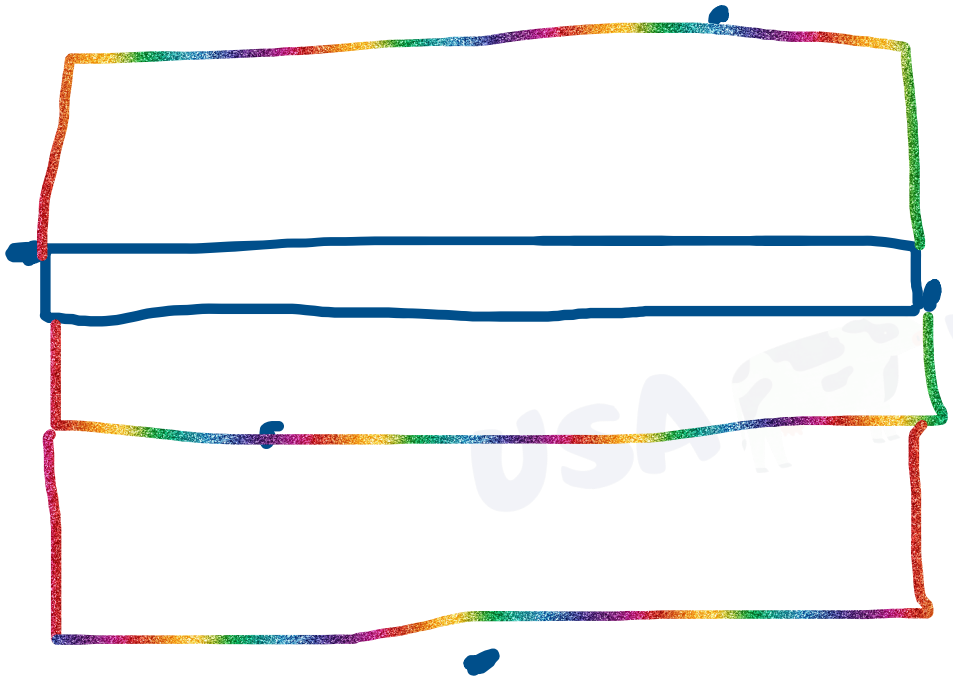


Deceiving Sample

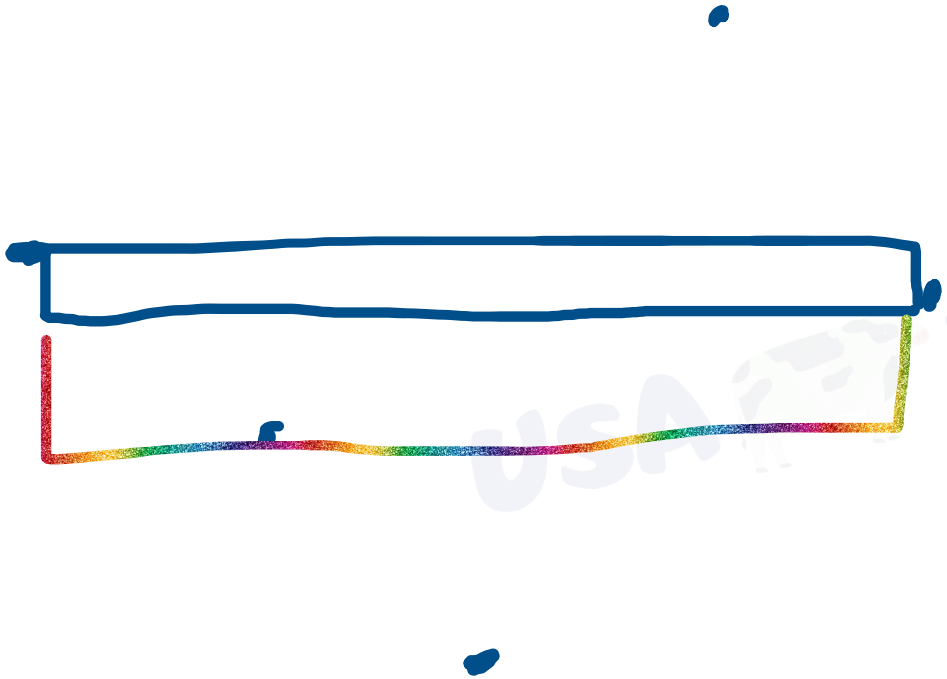


Deceiving Sample

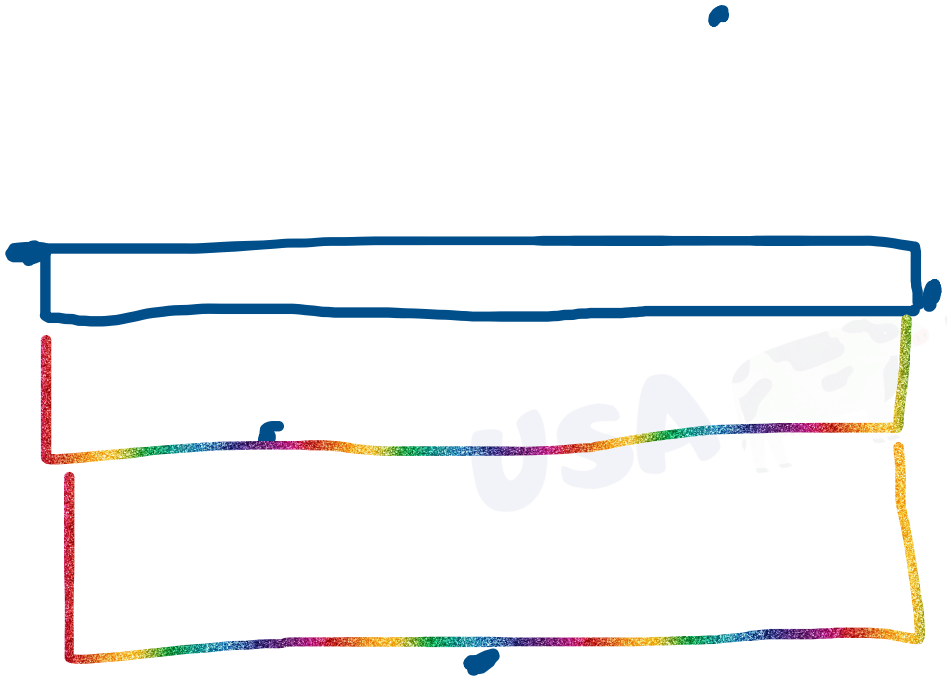
Deceiving Sample



Deceiving Sample



Deceiving Sample



Why did we get 6?

Well, how did we even get 6?

In a way, this sample tells us more than the given one.

We can see that below+above is wrong.

Simplifying this “rectangle” thing

Each point above our “base rectangle” contributes the number of points below it.

Each point below our “base rectangle” contributes the number of points above it.

We must also include the base rectangle as a point that contributes as well.

(below+1) * (above+1)

Coding in N^3

For loop for left, for loop for right, then for loop for sweep

In sweep:

count the number of points below min(y[left],y[right])

and max(y[left],y[right])

Works in N^3, gets half credit ☺

Coding in N^3

```
for(int i=0; i<N; i++){
        for(int j=i+1; j<N; j++){
                long above=0; long below=0;
                int minY = Math.min(arr[i].second, arr[j].second);
                int maxY = Math.max(arr[i].second, arr[j].second);
                for(int k=i+1; k<j; k++){
                        if(arr[k].second>maxY) above++;
                        if(arr[k].second<minY) below++;
                }
                answer += (below+1) * (above+1);
        }
}
```

How can we do better?

Well we obviously can't have N^3 because that is too slow, so we need some way to simplify this query of "find points below" and "find points above".

An interesting way to think about "find points below" is:

what is the prefix sum value of the values up to i?

the issue is that i is <=1e9

Coordinate Compression

We need the number of points below and above.

Instead of the absolute value (no not the math one) of the Y coordinate, we only need to find the relative position of it.

Compresses 1e9 coordinates into 2500 relative values. Big help.

CC Code Part 1: Sorting by X

```
// we sort by x because this is how you process the elements anyways
Pair arr[] = new Pair[N];
for(int i=0; i<N; i++){
        int x = sc.nextInt();
        int y = sc.nextInt();
        arr[i] = new Pair(x,y);
}
Arrays.sort(arr);
```

CC Code Part 2: Sorting by Y

```
// we sort by y now for the relative values
Pair tempY[] = new Pair[N];
for(int i=0; i<N; i++){
        int y = temp[i].second;
        tempY[i] = new Pair(y, i);
}
Arrays.sort(tempY);
```

CC Code Part 3: Actually compressing

```
int compressY[] = new int[N];
// I check for duplicates here for edu purposes, there are none
for(int i=0; i<N;
        int j=i;
        int indexReal= tempY[i].second;
        compressY[indexReal] = i;
        while(tempY[i].first == tempY[j].first){
                compressY[tempY[j].second] =i;
                j++;
        }
        i=j;
}
```

How it looks on a graph

Next Steps

We’re done with the coordinate compression code, now what?

We need to query the range [0,minY) and [maxY+1, N)

Let x= Find the number of elements in the [0,minY) range

Let y =Find the number of elements in the [maxY+1, N) range

Ans+=(x+1)* (y+1)

Using Segment Tree

We need to support 2 operations:

Update(index, value)

Sum(left, right)

How to do N^2 (log(N) / log(2))

```
for(int i=0; i<N; i++){
        for(int j=i+1; j<N; j++){
                int minY = Math.min(compressY[i],compressY[j]);
                int maxY = Math.max(compressY[i], compressY[j]);
                long above= segtree.sum(maxY, N);
                long below=   segtree.sum(0, minY);
                answer += (below+1) * (above+1);
                segtree.update(compressY[j], 1);
        }
        segtree = new SegmentTree(N);
}
```