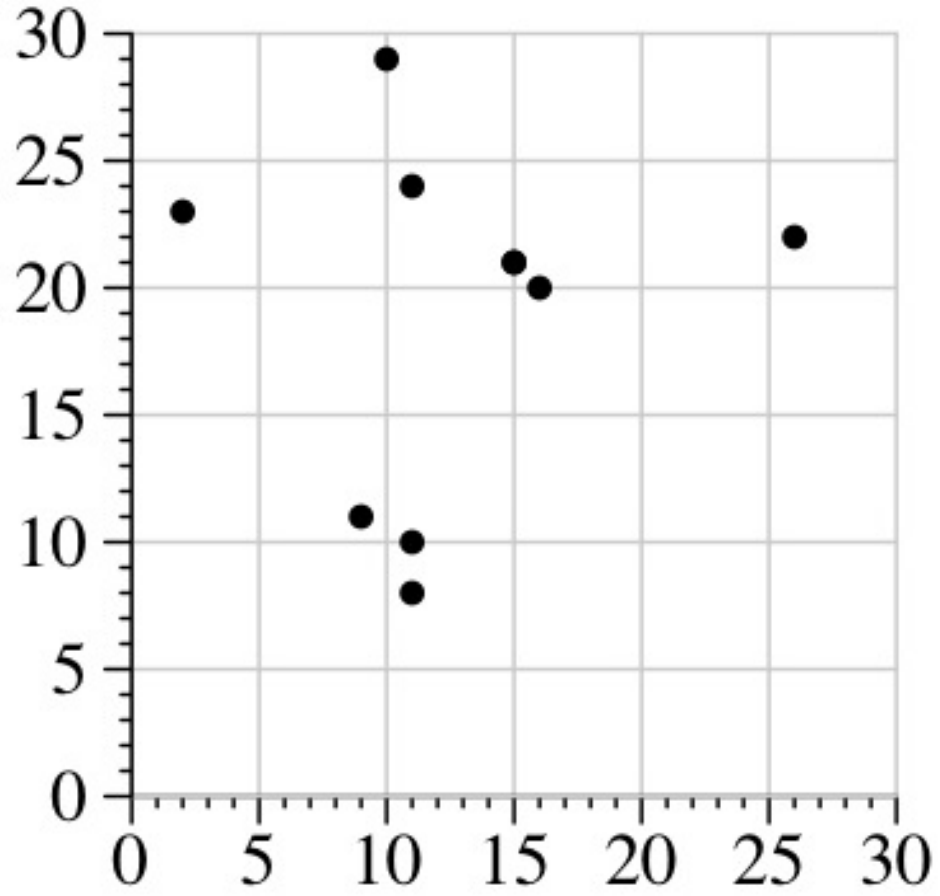
Topcoder SRM 614, D1, 250-Pointer "MinimumSquare"

James S. Plank
EECS Department
University of Tennessee

CS494 Class
February 15, 2016

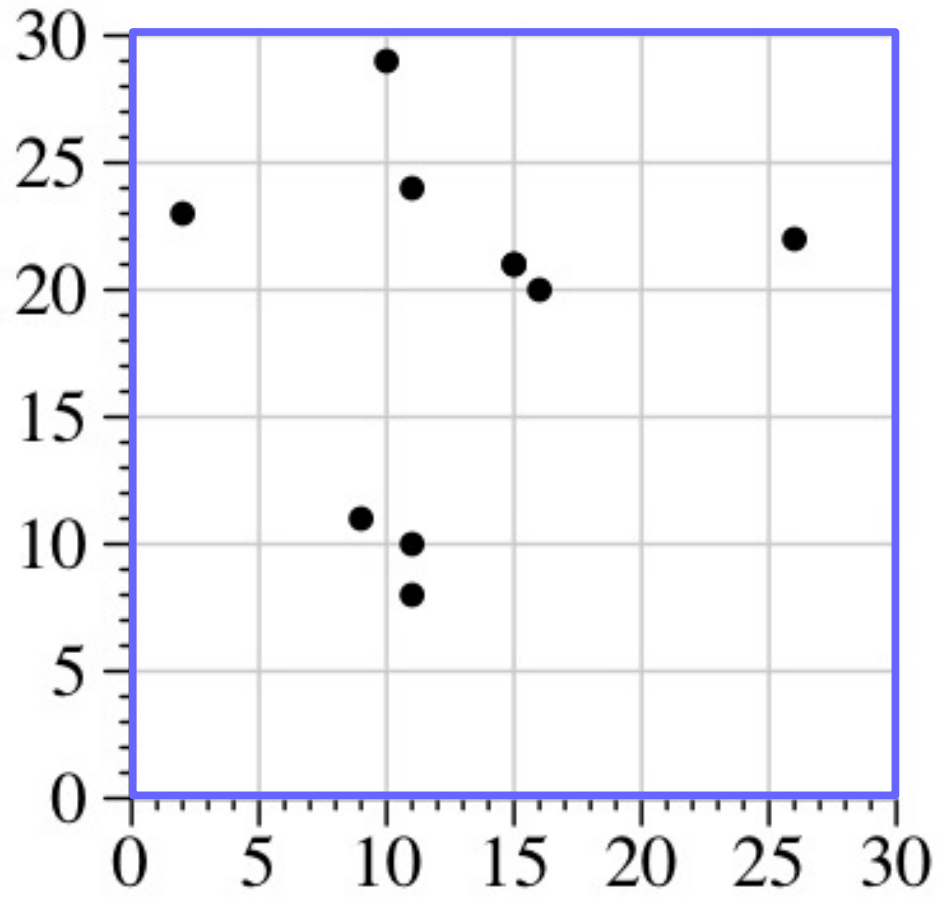
The problem

- You are given *(x,y)* points on a plane, and a value *K*.
- What's the smallest square that contains *K* points?



The problem

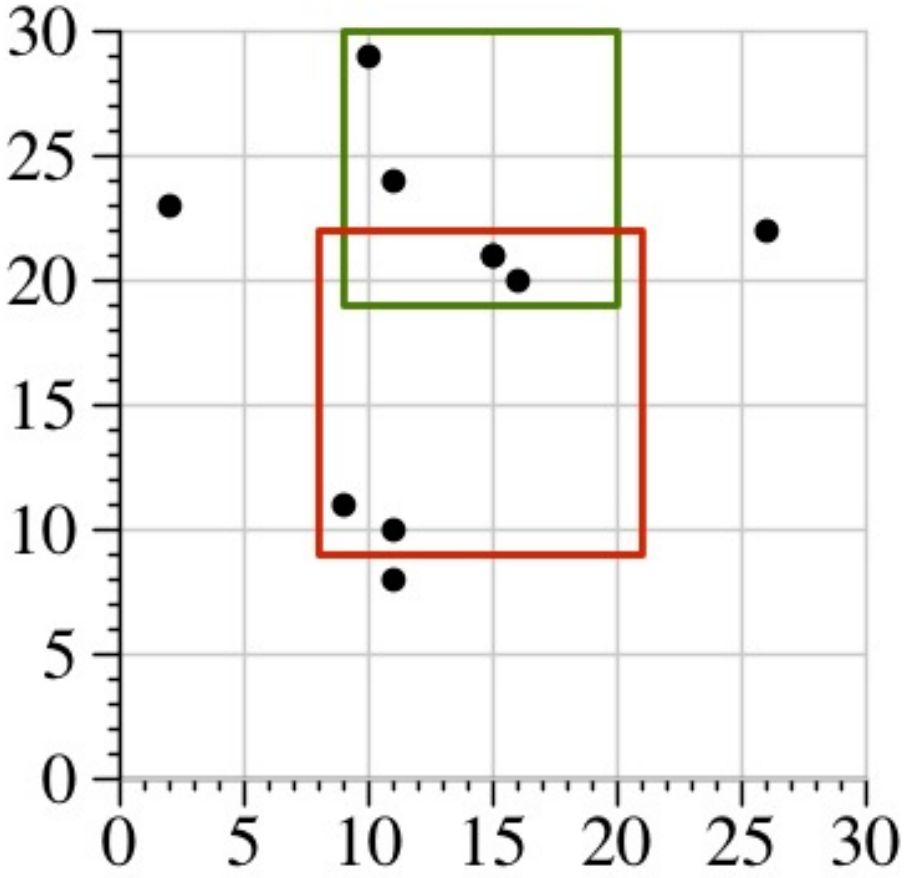
- You are given *(x,y)* points on a plane, and a value *K*.
- What's the smallest square that contains *K* points?



Obviously, we can use a big square to encompass all of the points.

The problem

- You are given *(x,y)* points on a plane, and a value *K*.
- What's the smallest square that contains *K* points?



Width = 11

Width = 13

However, finding that minimum square seems like a pretty challenging problem!

Prototype

- **Class name**: `MinimumSquare`
- **Method**: `minArea()`
- **Parameters**:

x	`vector <int>`	The *x* coordinates
y	`vector <int>`	The y coordinates
K	`int`	The number of points

- **Return Value**: `long long`
- You need to return the area.
- The square's coordinates must be integers.
- The points must be *inside* the square.

Constraints

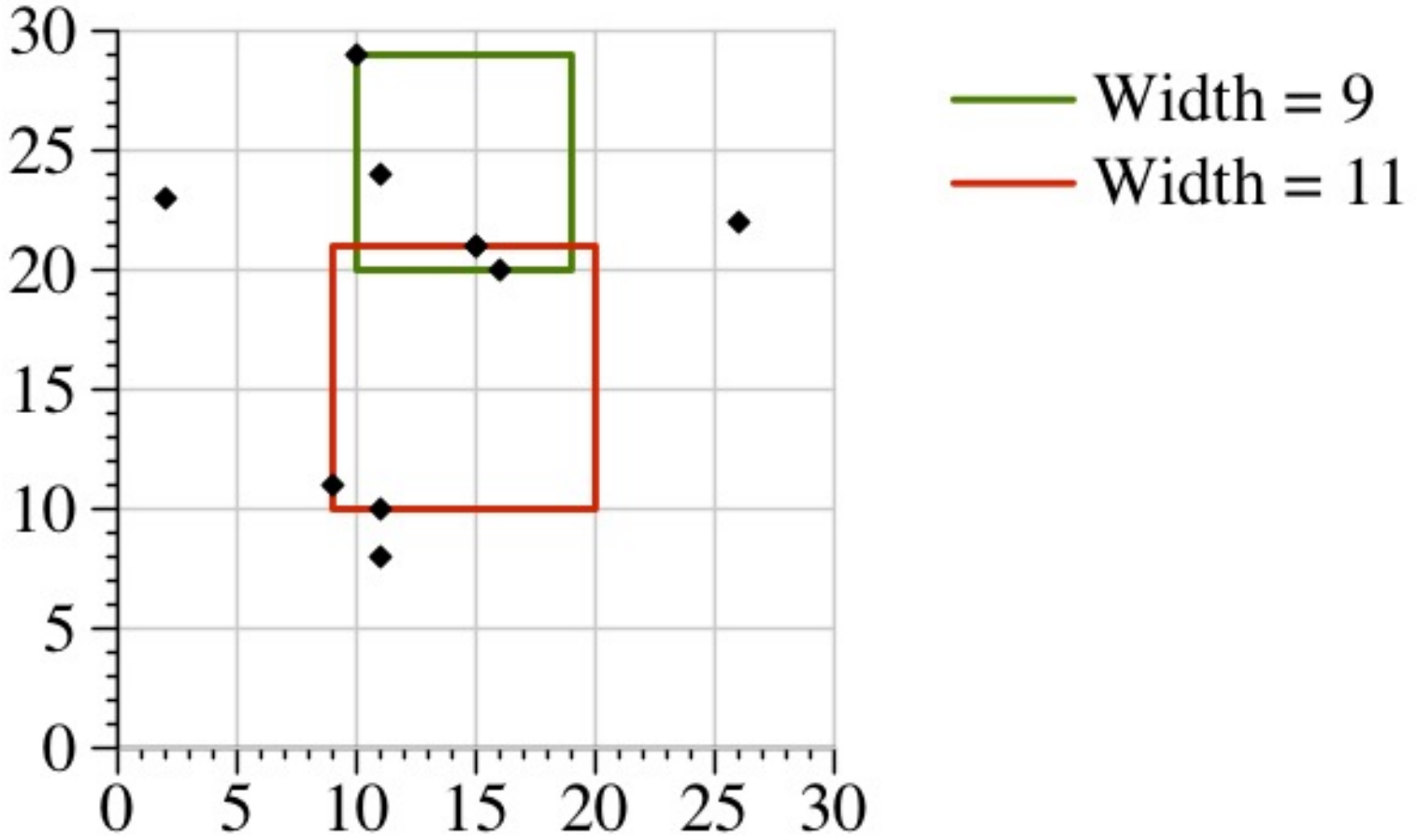
- The points are between

  -1,000,000,000 and 1,000,000,000.

- The maximum number of points is 100.

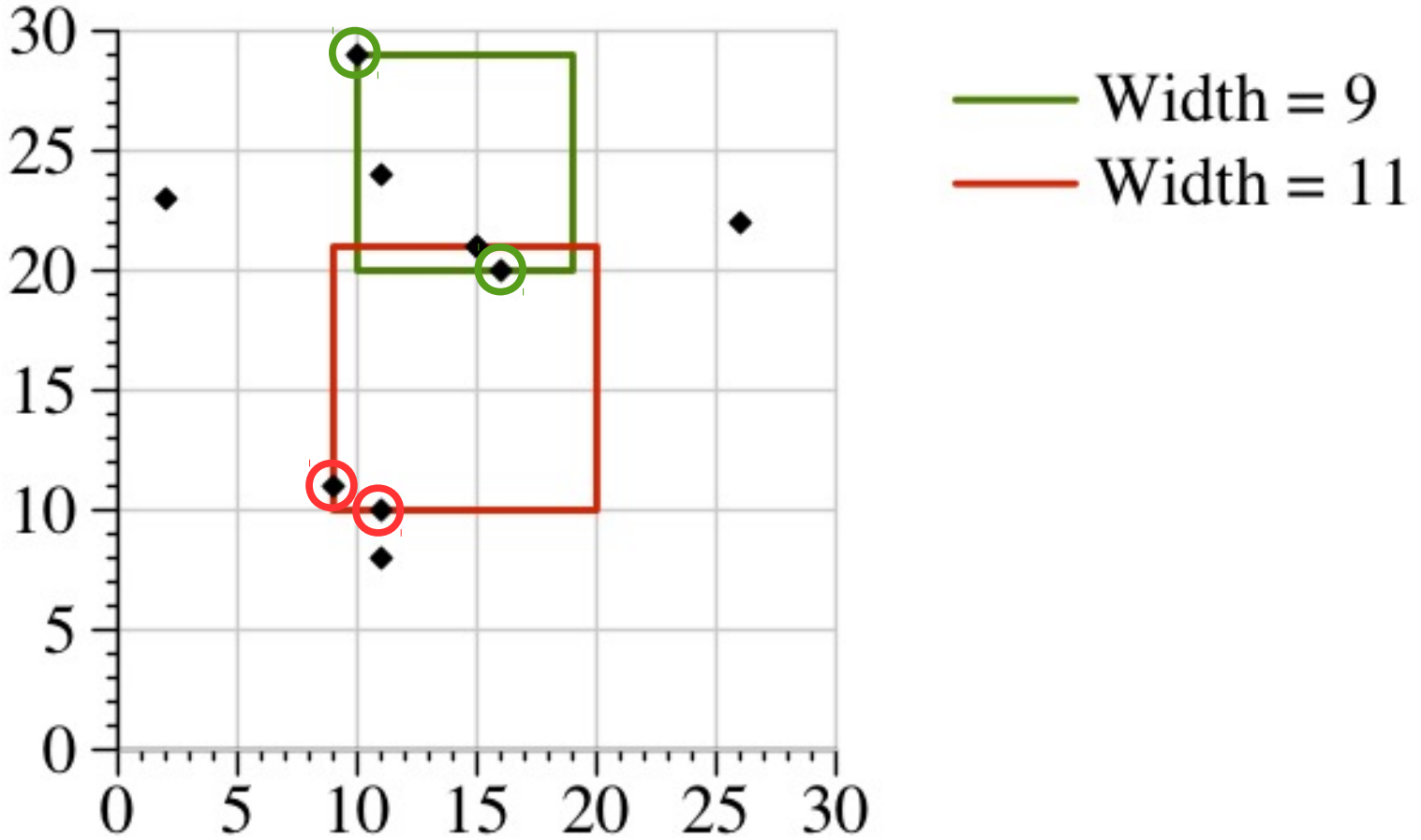
Observation #1

- It is going to be easier if we allow points to touch the square. Then we add two to the width:



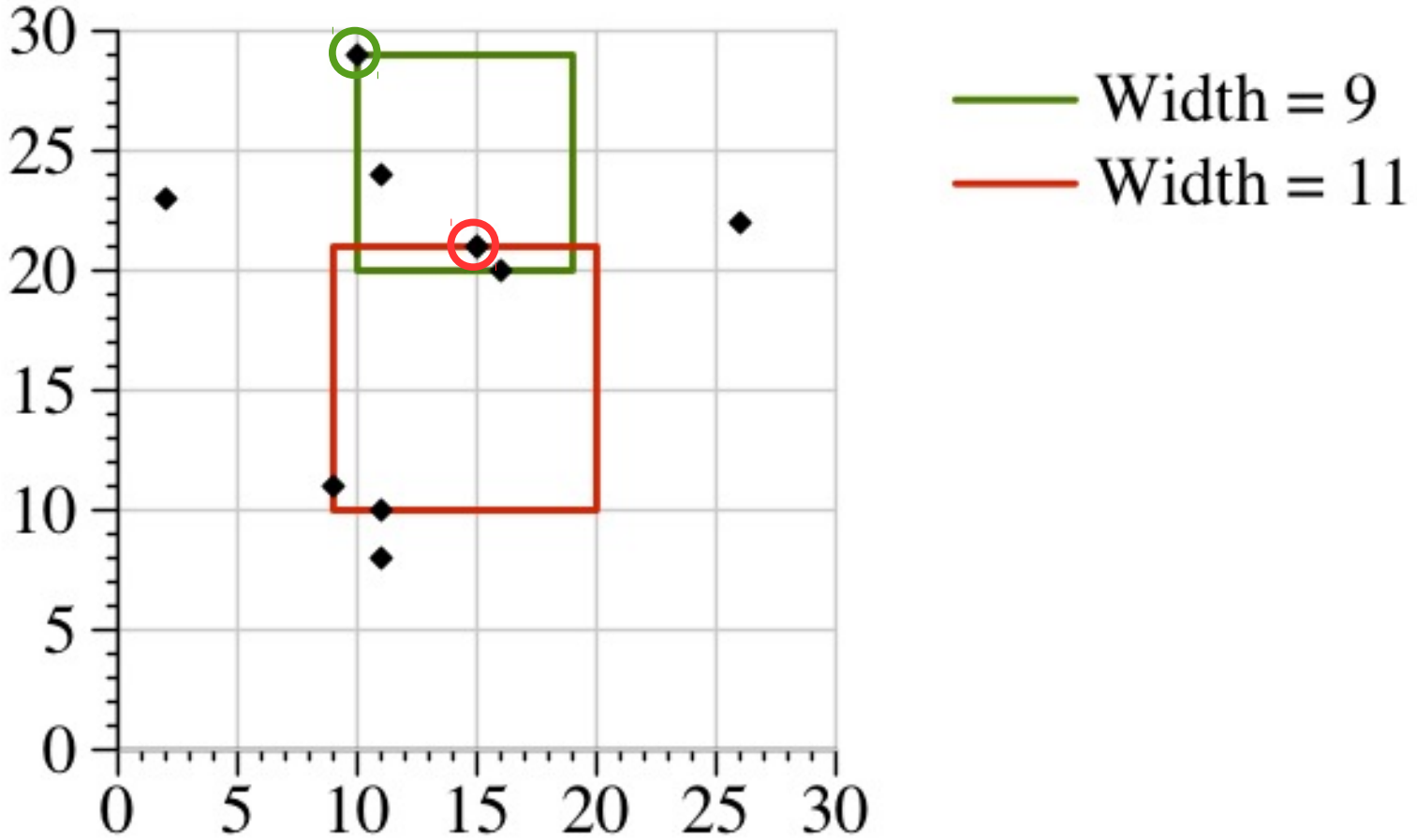
Observation #2

- The bottom-left corner of the square has to correspond to the x and y coordinates of points:



Observation #3

- And the width will correspond to another point, either *w* units higher or to the right.



A Brain-Dead Enumeration

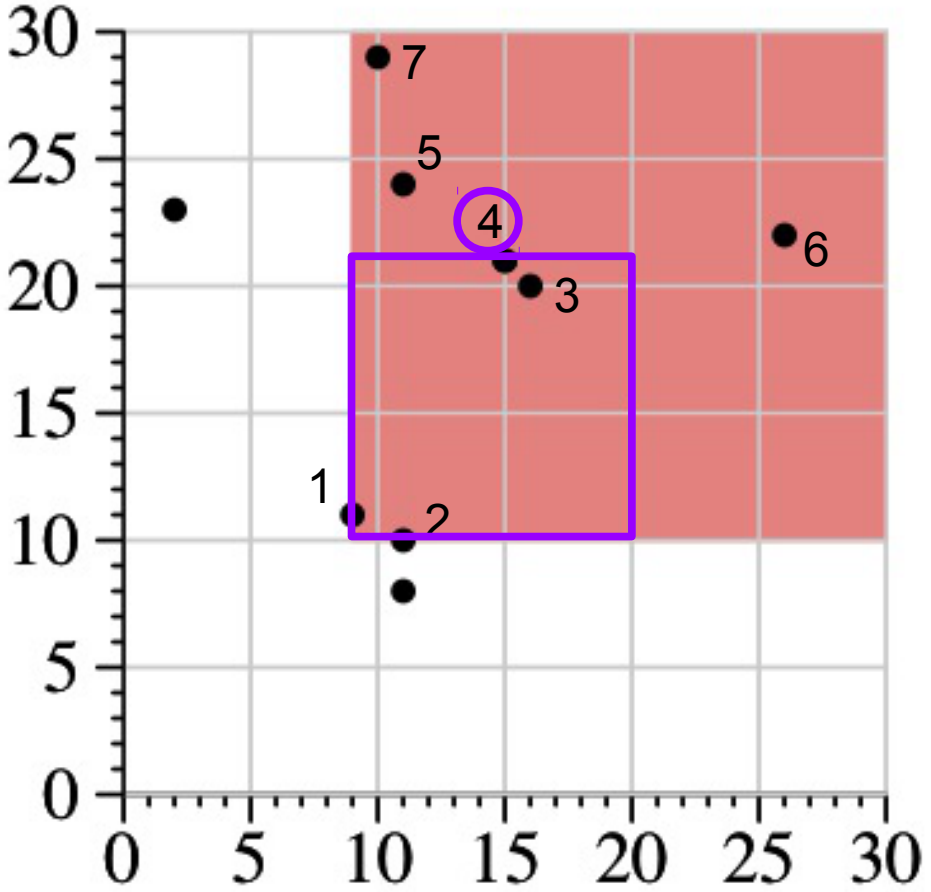
- *O(n)* Enumerate l, all possible left positions.
- *O(n)* Enumerate b, all possible bottom positions.
- *O(n)* Enumerate w, all possible widths.
- *O(n)* Count the points in the square

That's $O(n^4)$

But with $n \leq 100$, it may be fast enough!

A Faster Approach – Tianli's Solution

- If you sort the points that are ≥ *(l,b)* by the width of the square they they produce, then the K'th point will define the square:



A Faster Approach – Tianli's Solution

O(n) • Enumerate l, all possible left positions.

O(n) • Enumerate b, all possible bottom positions.

O(n log(n)) • Sort all points $\geq (l,b)$.

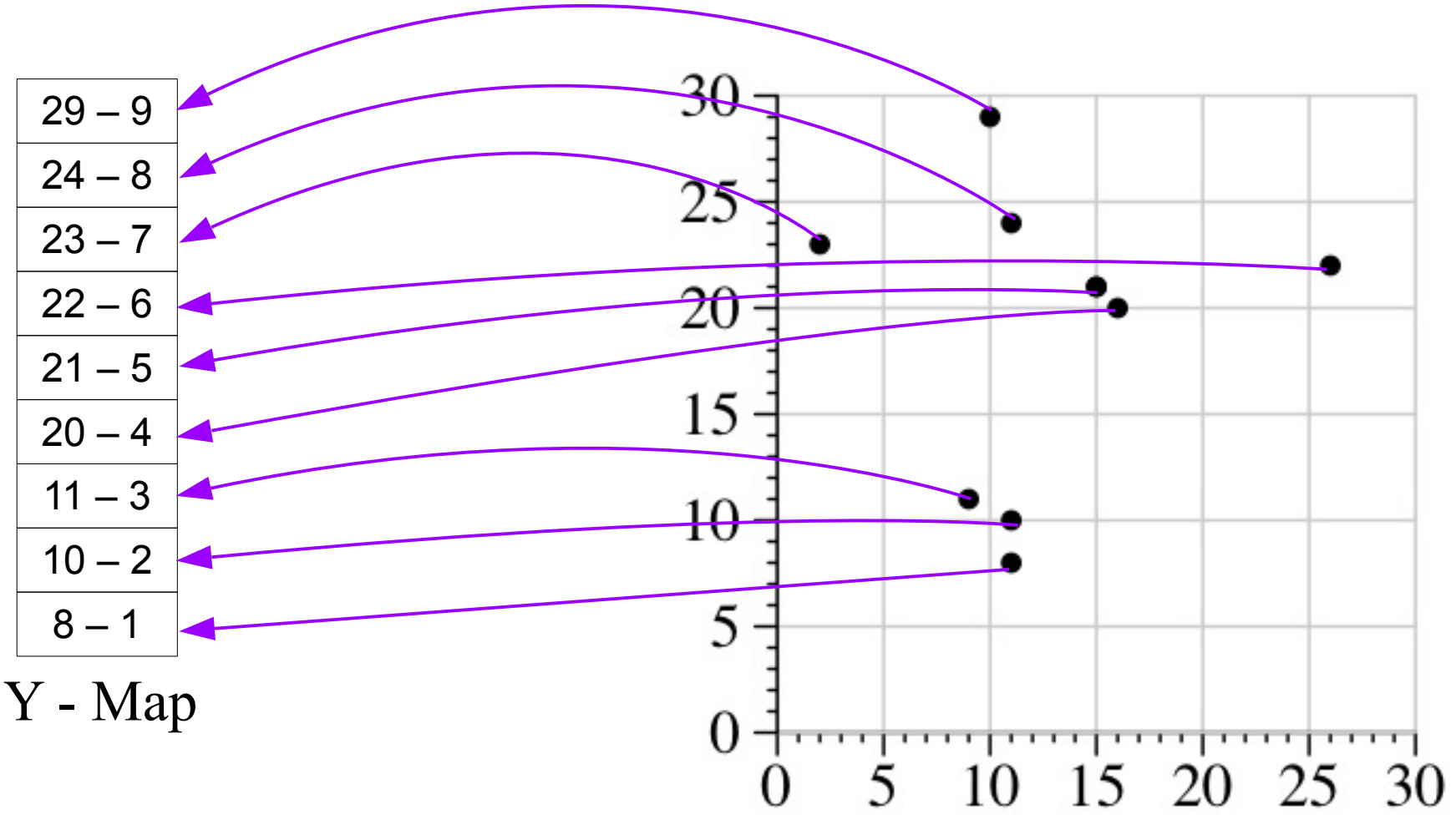
That's $O(n^3 log(n))$

How about faster still?

- Given *l, b* and *w*, can you determine the number of points in the square in $O(log(n))$ time?
- If so, then you can perform a binary search on *w*, and reduce the time to $O(n^2 log(n)^2)$.

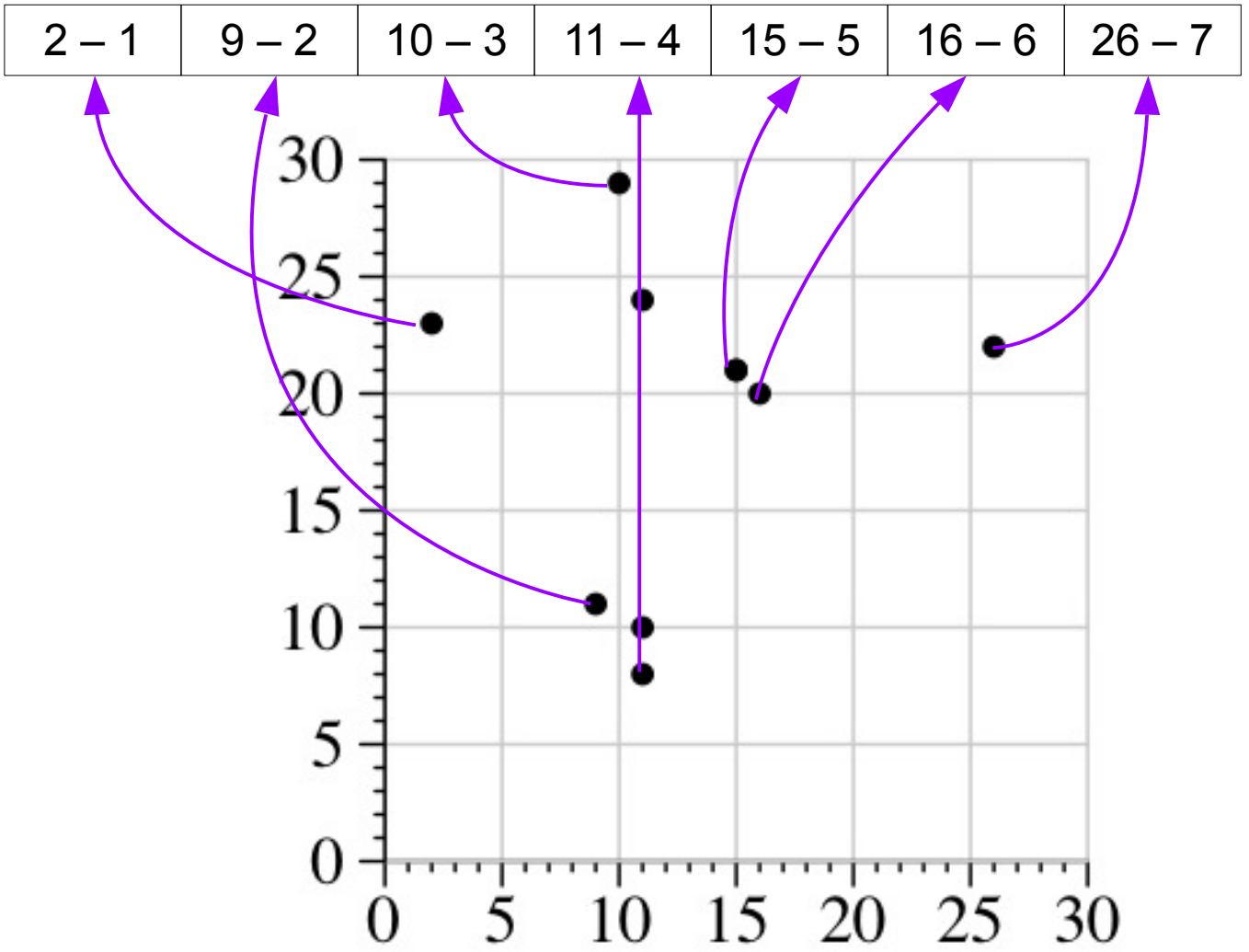
The $O(n^2 log(n)^2)$ Solution

- Create two maps that have all of the potential *x* and *y* values. The "val" is their number in sorted order.



The $O(n^2 log(n)^2)$ Solution

- Create two maps that have all of the potential *x* and *y* values. The "val" is their number in sorted order.

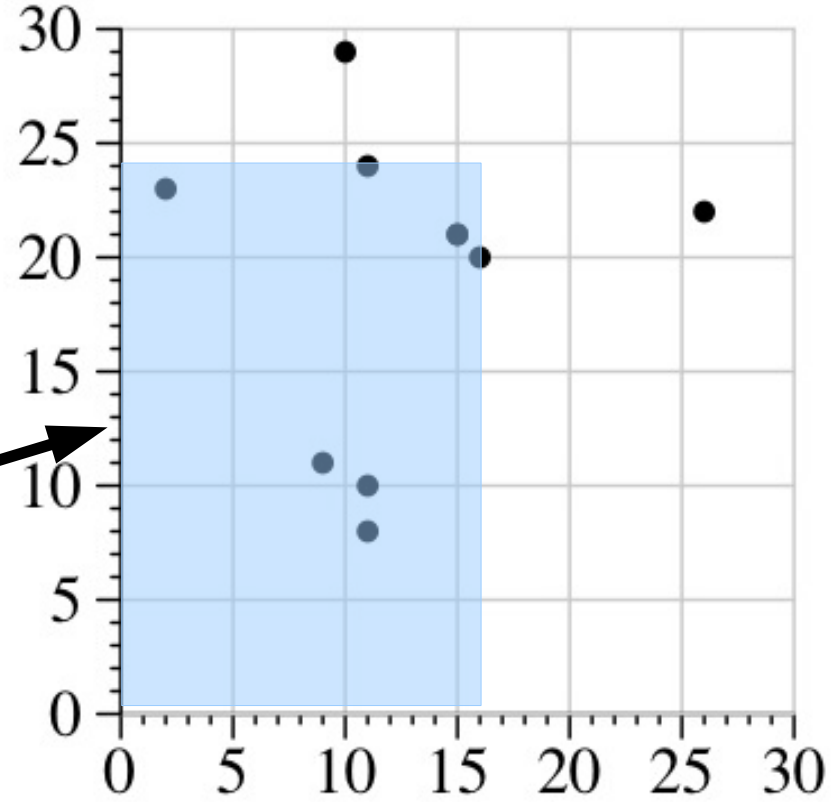


X - Map

The $O(n^2 log(n)^2)$ Solution

Create a table that for every (x,y) pair, has the number of points in the rectangle from $(-\infty,-\infty)$ to (x,y).

	-∞	2	9	10	11	15	16	26	∞
-∞	0	0	0	0	0	0	0	0	0
8	0	0	0	0	1	1	1	1	1
10	0	0	0	0	2	2	2	2	2
11	0	0	1	1	3	3	3	3	3
20	0	0	1	1	3	3	4	4	4
21	0	0	1	1	3	4	5	5	5
22	0	0	1	1	3	4	5	6	6
23	0	1	2	2	4	5	6	7	7
24	0	1	2	2	5	6	7	8	8
29	0	1	2	3	6	7	8	9	9
∞	0	1	2	3	6	7	8	9	9

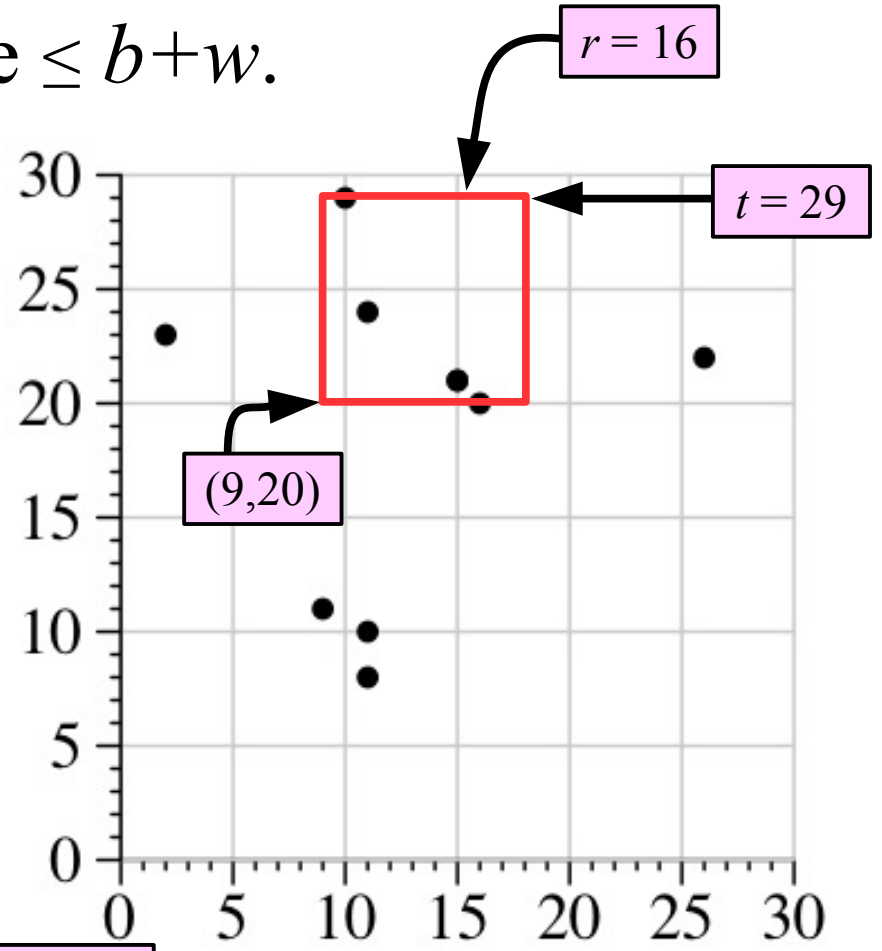


(creating this table is a little subtle)

The $O(n^2 log(n)^2)$ Solution

Given (l,b) and w, use upper-bound to find (r,t) such that r is the greatest value $\leq l+w$, and t is the greatest value $\leq b+w$.

	-∞	2	9	10	11	15	16	26	∞
-∞	0	0	0	0	0	0	0	0	0
8	0	0	0	0	1	1	1	1	1
10	0	0	0	0	2	2	2	2	2
11	0	0	1	1	3	3	3	3	3
20	0	0	1	1	3	3	4	4	4
21	0	0	1	1	3	4	5	5	5
22	0	0	1	1	3	4	5	6	6
23	0	1	2	2	4	5	6	7	7
24	0	1	2	2	5	6	7	8	8
29	0	1	2	3	6	7	8	9	9
∞	0	1	2	3	6	7	8	9	9



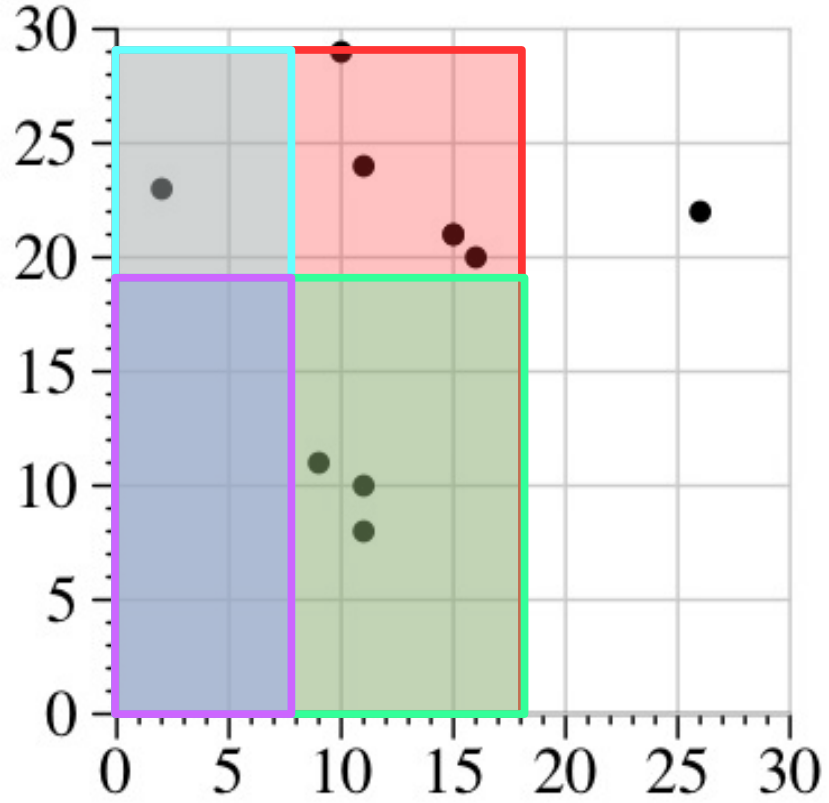
E.g.: (l,b) = (9,20) & w=9, then (r,t) = (16,29)

The $O(n^2 log(n)^2)$ Solution

The number of points in the square defined by (l,b) and w,is equal to:

`M[r,t]` − `M[l-1,t]` − `M[r,b-1]` + `M[l-1,b-1]`

	-∞	2	9	10	11	15	16	26
-∞	0	0	0	0	0	0	0	0
8	0	0	0	0	1	1	1	1
10	0	0	0	0	2	2	2	2
11	0	0	1	1	3	3	3	3
20	0	0	1	1	3	3	4	4
21	0	0	1	1	3	4	5	5
22	0	0	1	1	3	4	5	6
23	0	1	2	2	4	5	6	7
24	0	1	2	2	5	6	7	8
29	0	1	2	3	6	7	8	9



8-1-3+0 = 4

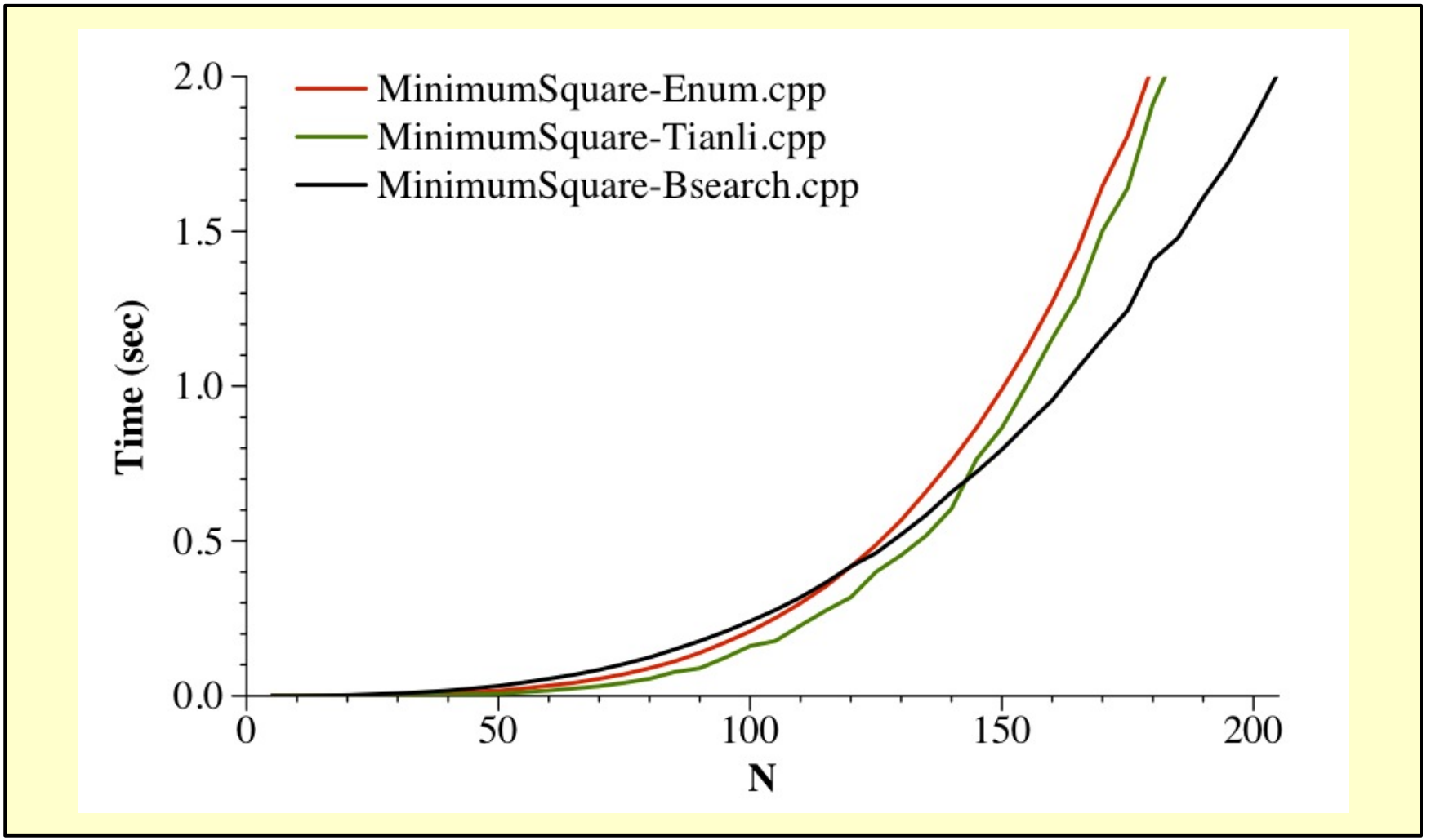
The $O(n^2 log(n)^2)$ Solution

$O(n\ log(n))$	• Create the maps
$O(n^2)$	• Create the matrix
$O(n)$	• Enumerate l, all possible left positions.
$O(n)$	• Enumerate b, all possible bottom positions.
$O(log(n))$	• Do a binary search on w.
$O(log(n))$	• For each l,b,w, count the points.

That's $O(n^2 log(n)^2)$

Performance

- MacBook Pro, Core i5, 2.4 GHz, -O2.



How did the Topcoders Do?

- This one was tough:
  - 748 Topcoders opened the problem.
  - 647 (87%) submitted a solution.
  - 362 (55%) of the submissions were correct.
  - Best time was 3:03
  - Average correct time was 25:24.

Topcoder SRM 614, D1, 250-Pointer "MinimumSquare"

James S. Plank
EECS Department
University of Tennessee

CS494 Class
February 15, 2016