2D Maxima Sweep-Line Algorithm

(Class 5)

From Book's Chapter 2

- We want to improve the efficiency of the algorithm we developed in previous class.
- For example, one algorithm has $O(n^2)$ running time.
- And then we improved the algorithm that has $O(n)$ running time.

- We introduced a brute-force algorithm that ran in $O(n^2)$ time.
- It operated by comparing all pairs of points.
- Is there an approach that is significantly better?
- The problem with the brute-force algorithm is that it uses no intelligence in pruning out decisions.

- For example, once we know that a point p_i is dominated by another point p_j, we do not need to use p_i for eliminating other points.
- For example, if $p_i < p_j$ then there is no possibility that p_i will appear as maximal point.
- So, we should exclude p_i in the future comparisons inside that algorithm.

- This follows from the fact that dominance relation is *transitive*.
- If p_j dominates p_i, and p_i dominates p_h then p_j also dominates p_h; p_i is not needed.

$$p_h < p_i < p_j$$

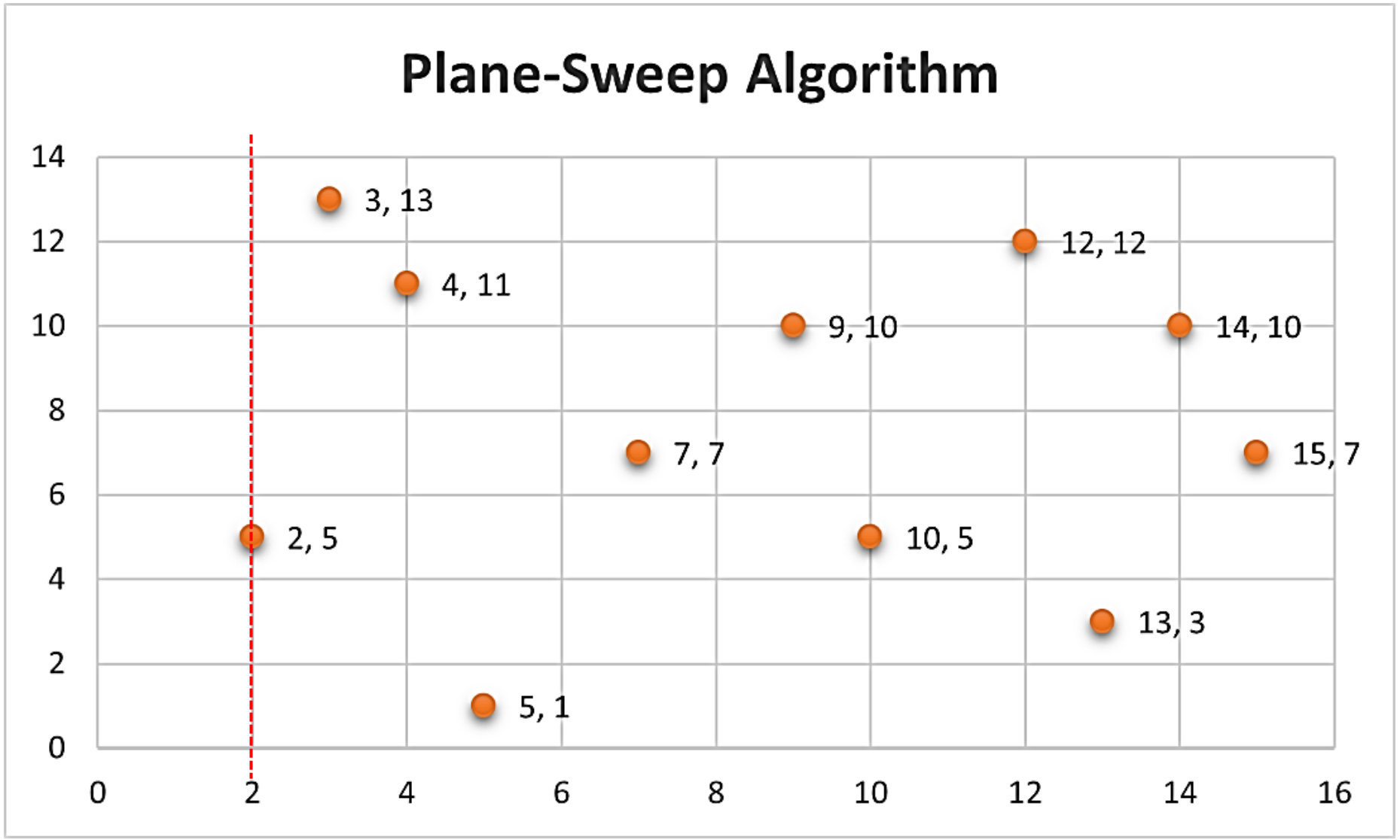
$$p_h < p_j$$

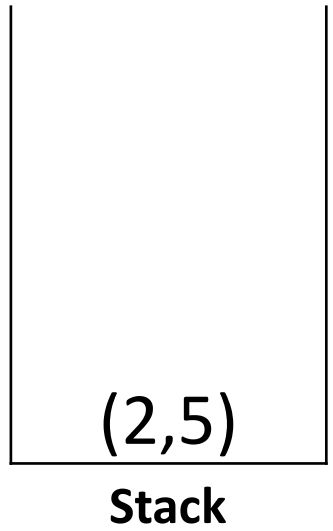
- Hence, both p_h and p_i can be excluded from the future comparisons.

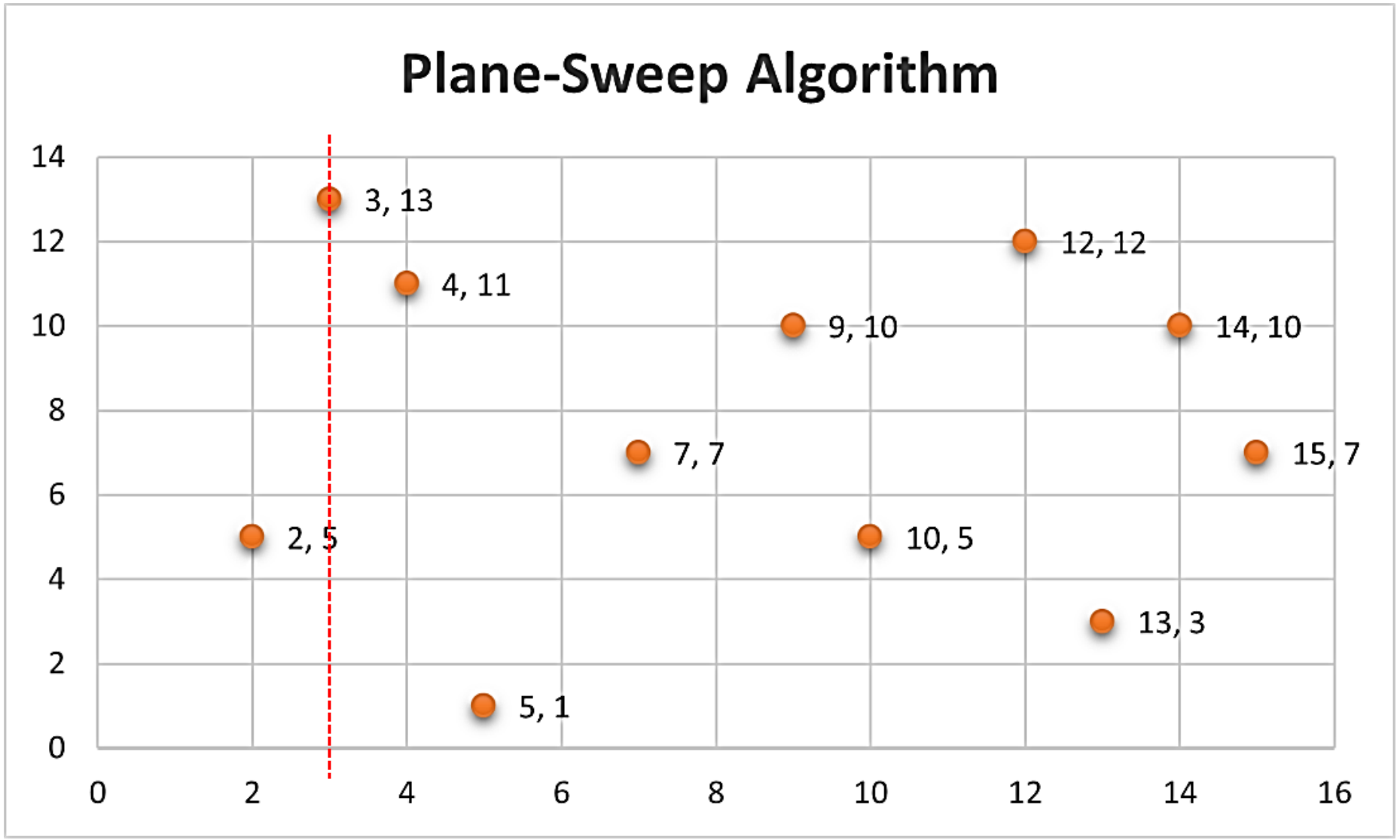
Plane-Sweep Algorithm

- We will sweep a vertical line across the 2D plane from left to right.
- As we sweep, we will build a structure holding the maximal points lying to the left of the sweep line.
- Sweep from minimum x value to maximum x value.
- And update the maximal set accordingly.

- The maximal set will keep updating while we sweep.
- When the sweep line reaches the rightmost point of P, we will have the complete set of maximal points.
- This approach of solving geometric problems is called the *plane sweep*.
- Geometric algorithm is another domain of algorithms.
- First, we will sort all the points in ascending order with respect to x coordinate.

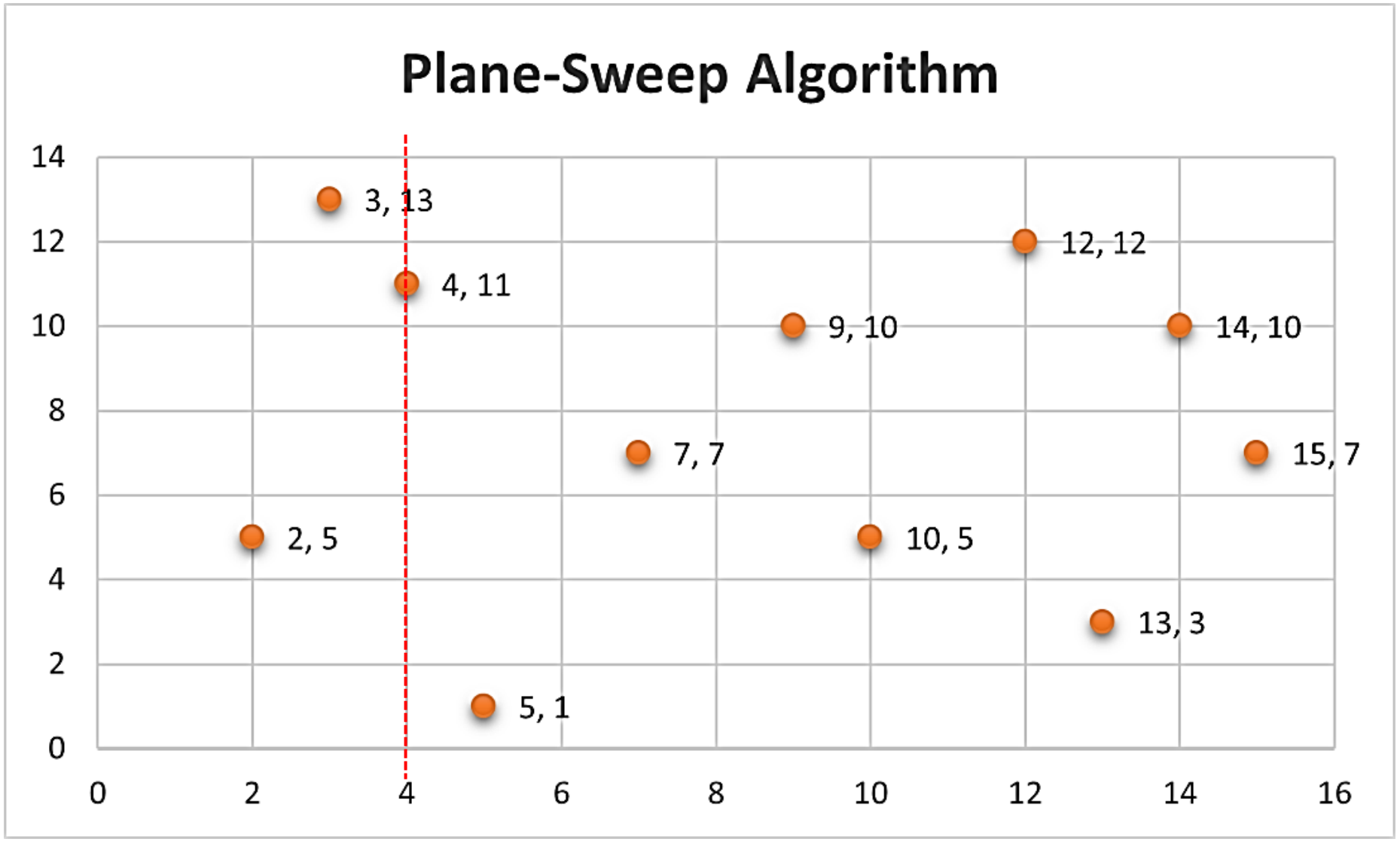

Plane-Sweep Algorithm
14
12
10
8
6
4
2
0
0
2
4
6
8
10
12
14
16
3, 13
4, 11
9, 10
12, 12
14, 10
7, 7
15, 7
2, 5
10, 5
13, 3
5, 1


(2,5)
Stack


Plane-Sweep Algorithm
14
12
10
8
6
4
2
0
0
2
4
6
8
10
12
14
16
3, 13
4, 11
9, 10
12, 12
14, 10
7, 7
15, 7
2, 5
10, 5
13, 3
5, 1

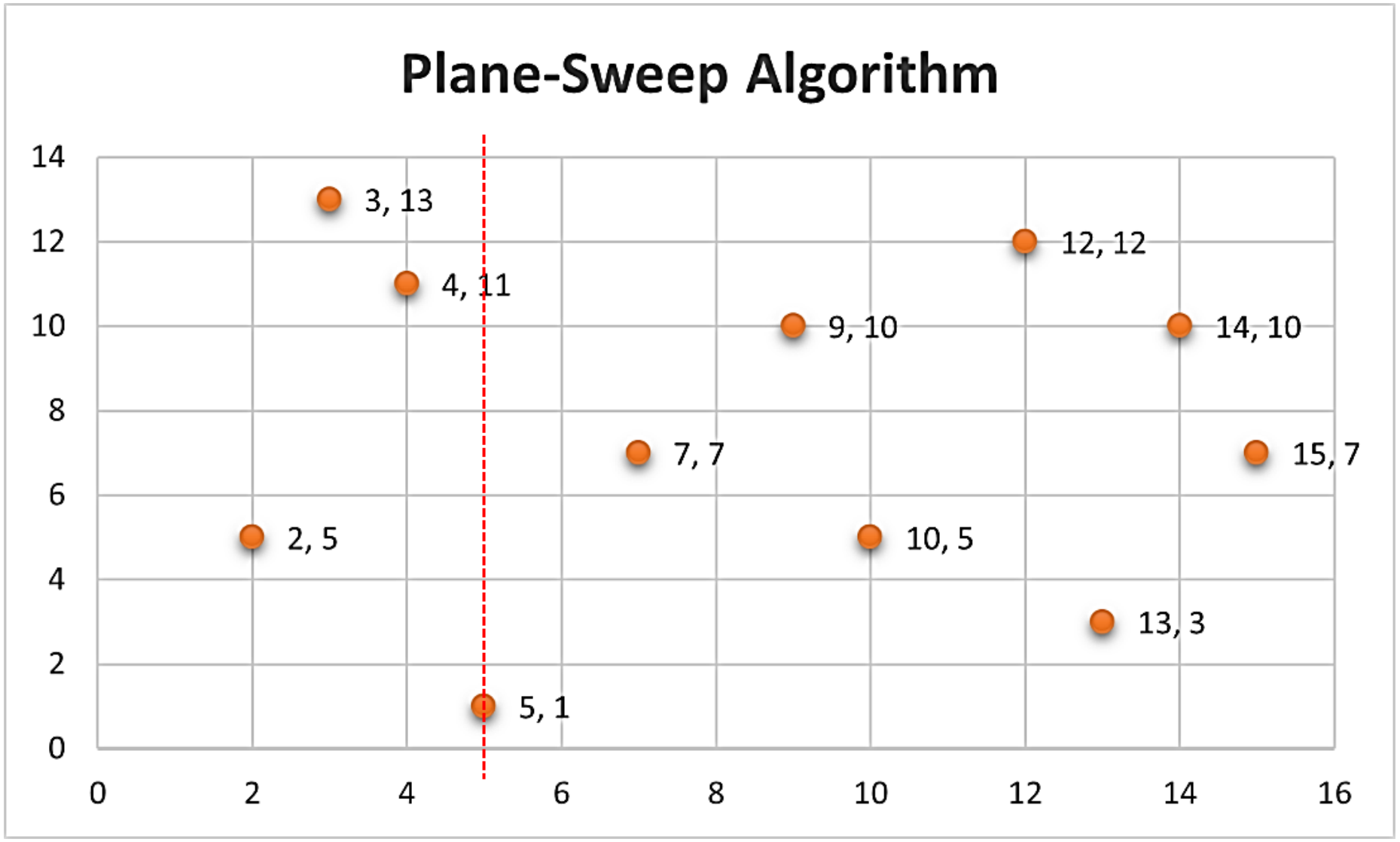
(3,13)

Stack



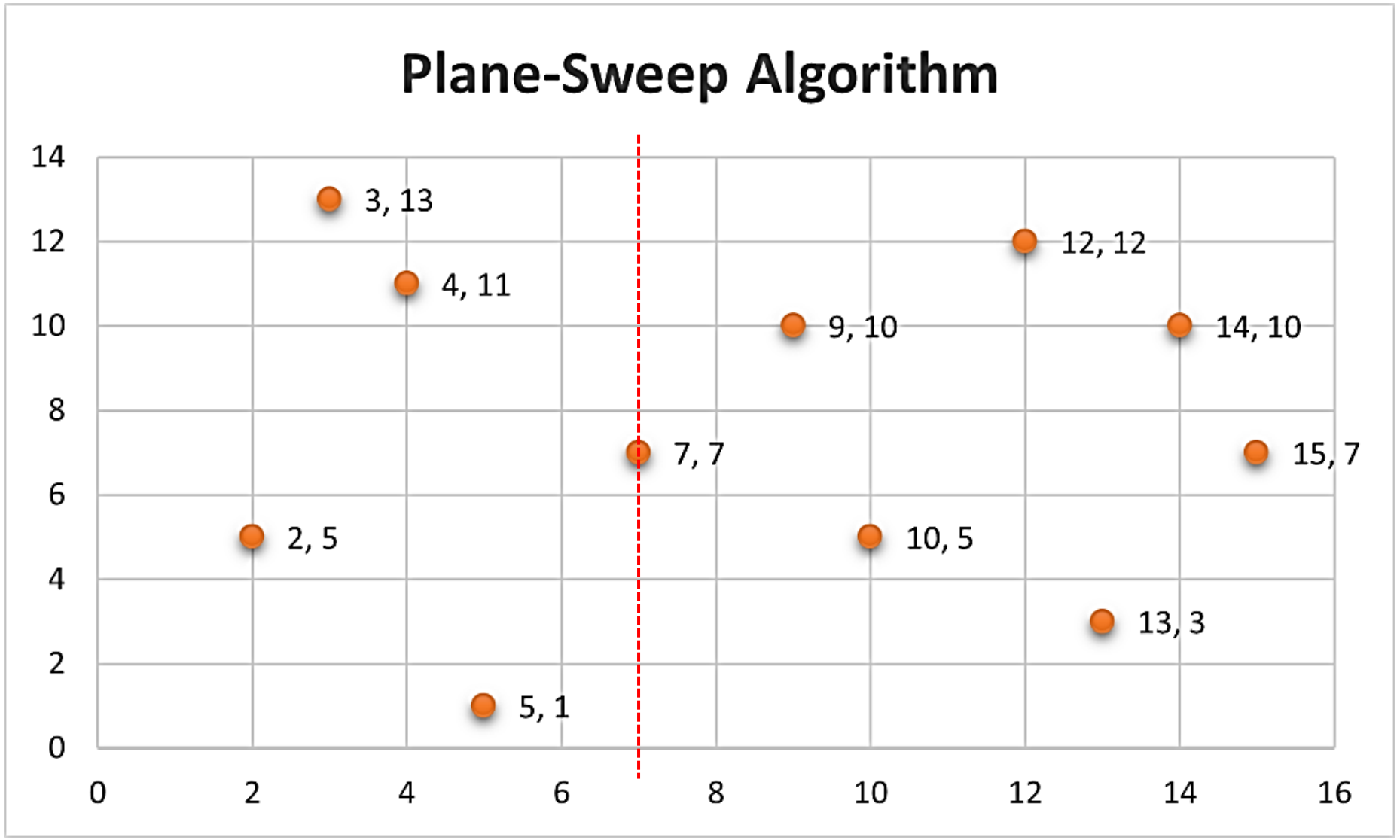
(4,11)
(3,13)

Stack



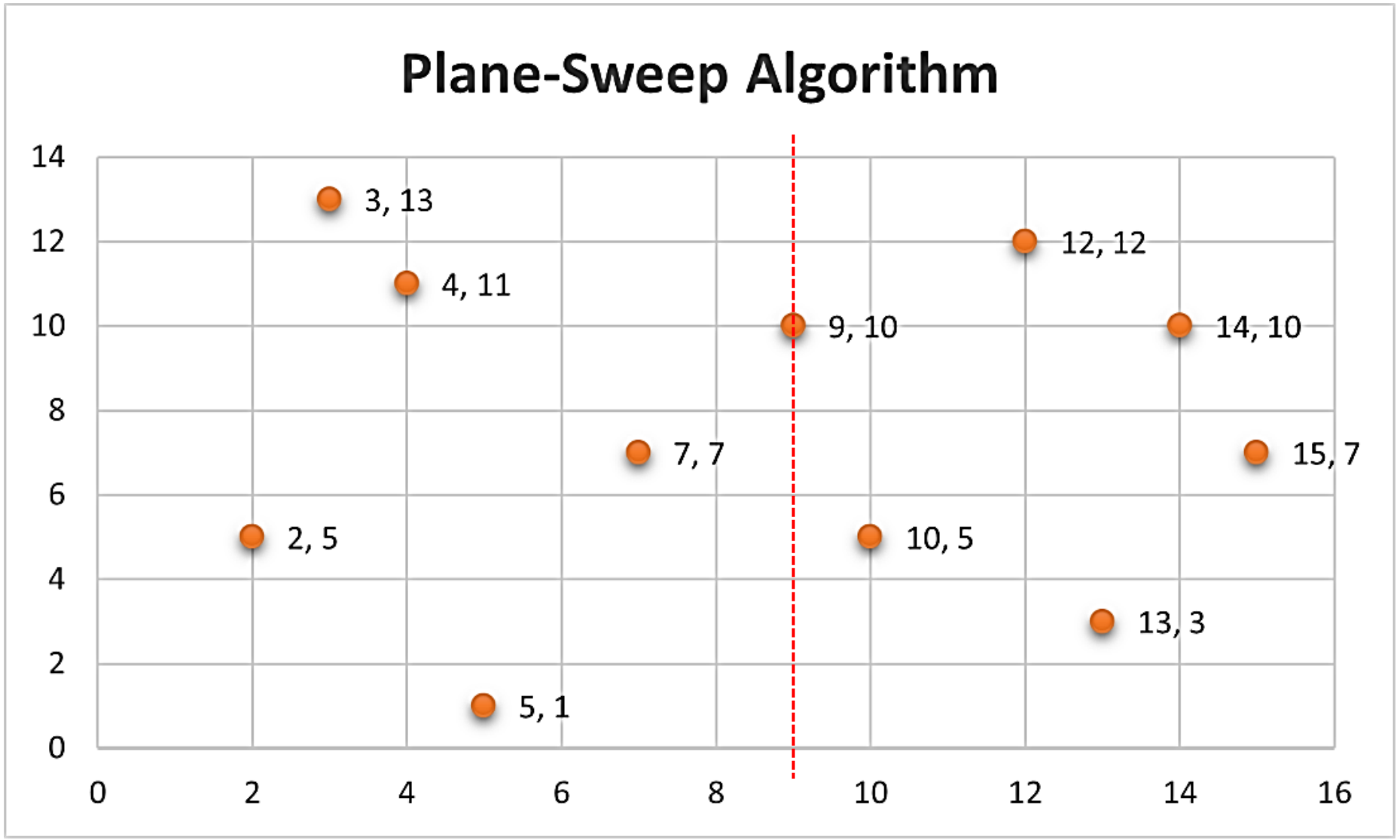
(5,1)
(4,11)
(3,13)

Stack



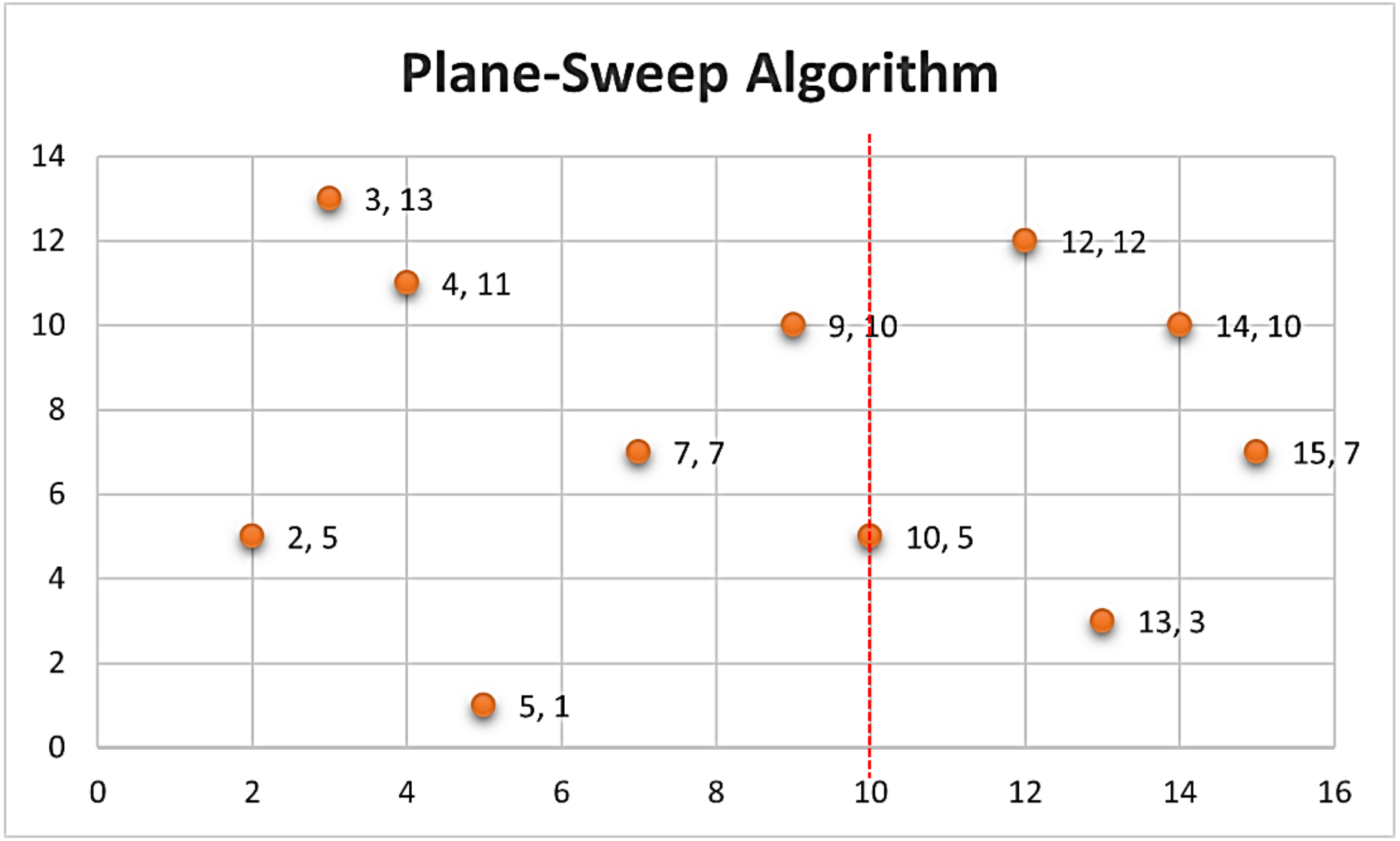
(7,7)
(4,11)
(3,13)

Stack



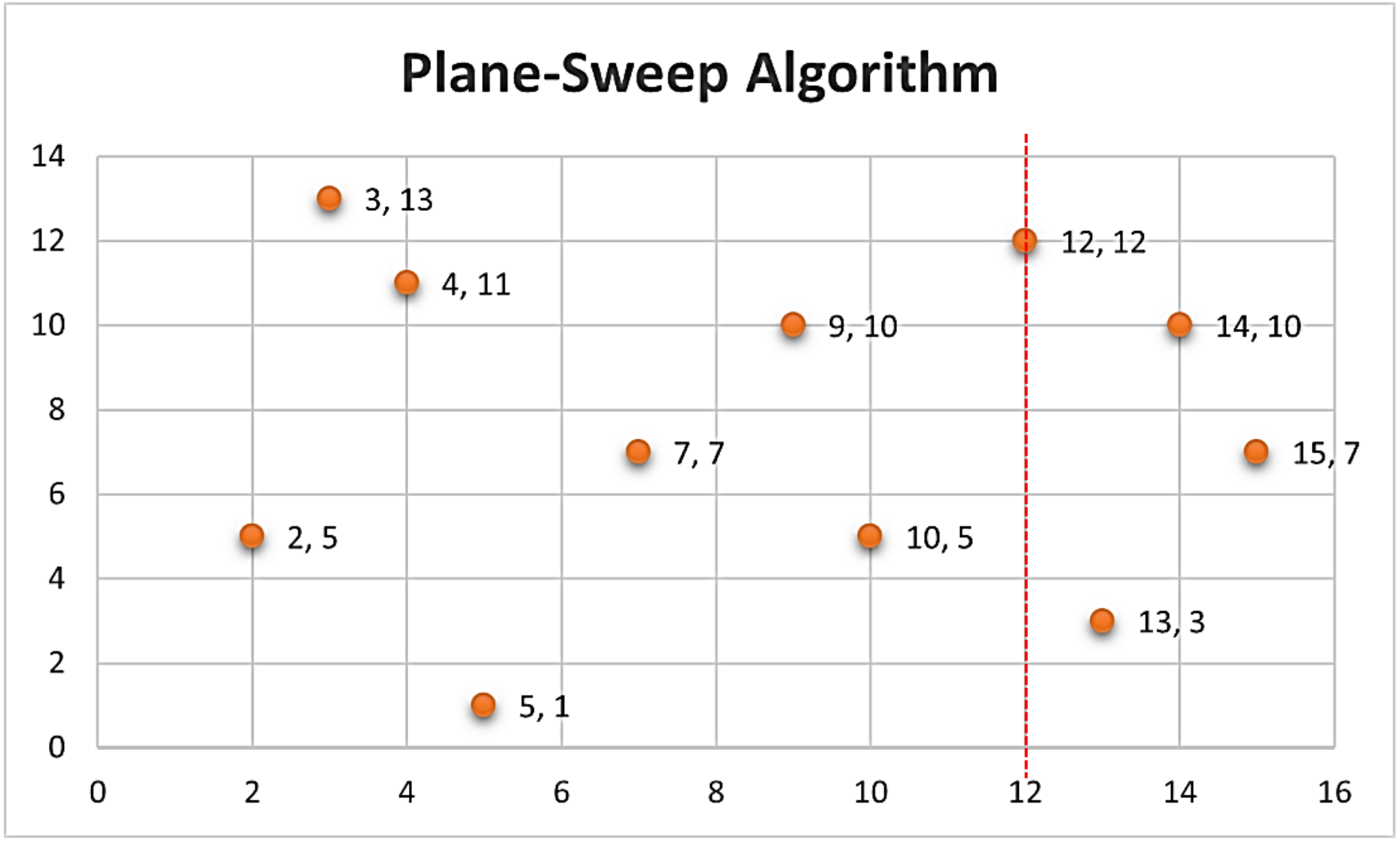
(9,10)
(4,11)
(3,13)

Stack



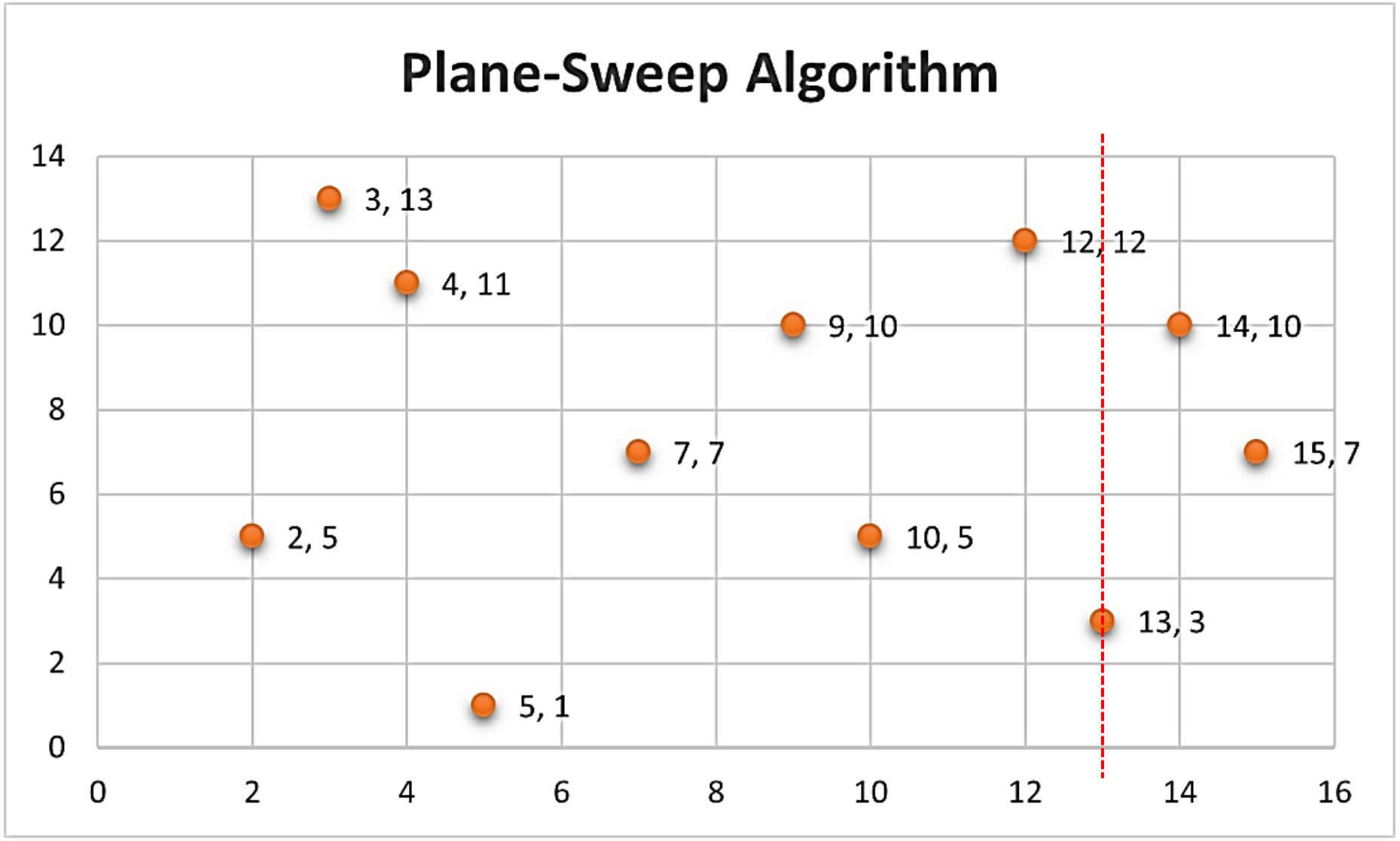
(10,5)
(9,10)
(4,11)
(3,13)

Stack



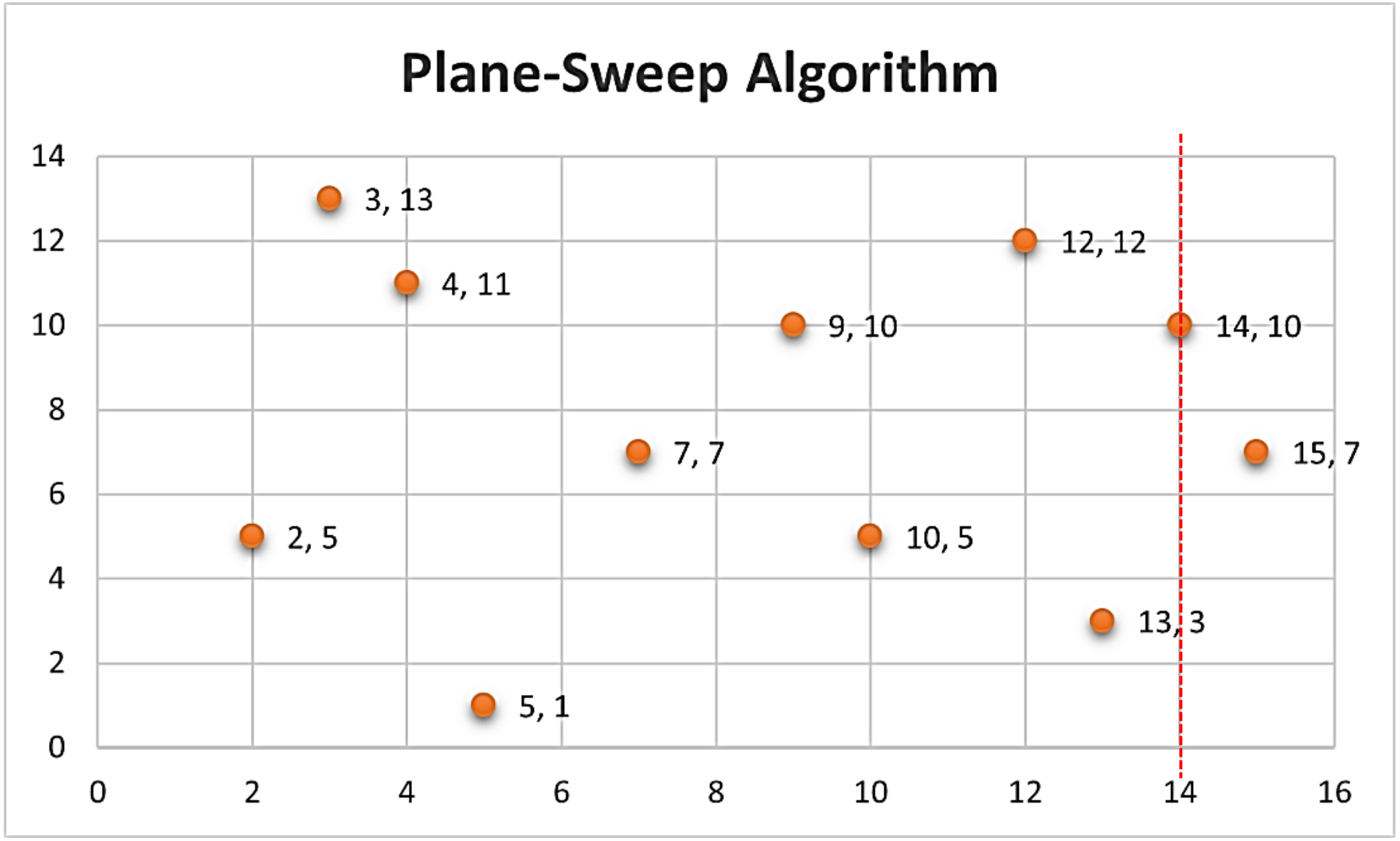
(12,12)
(3,13)

Stack



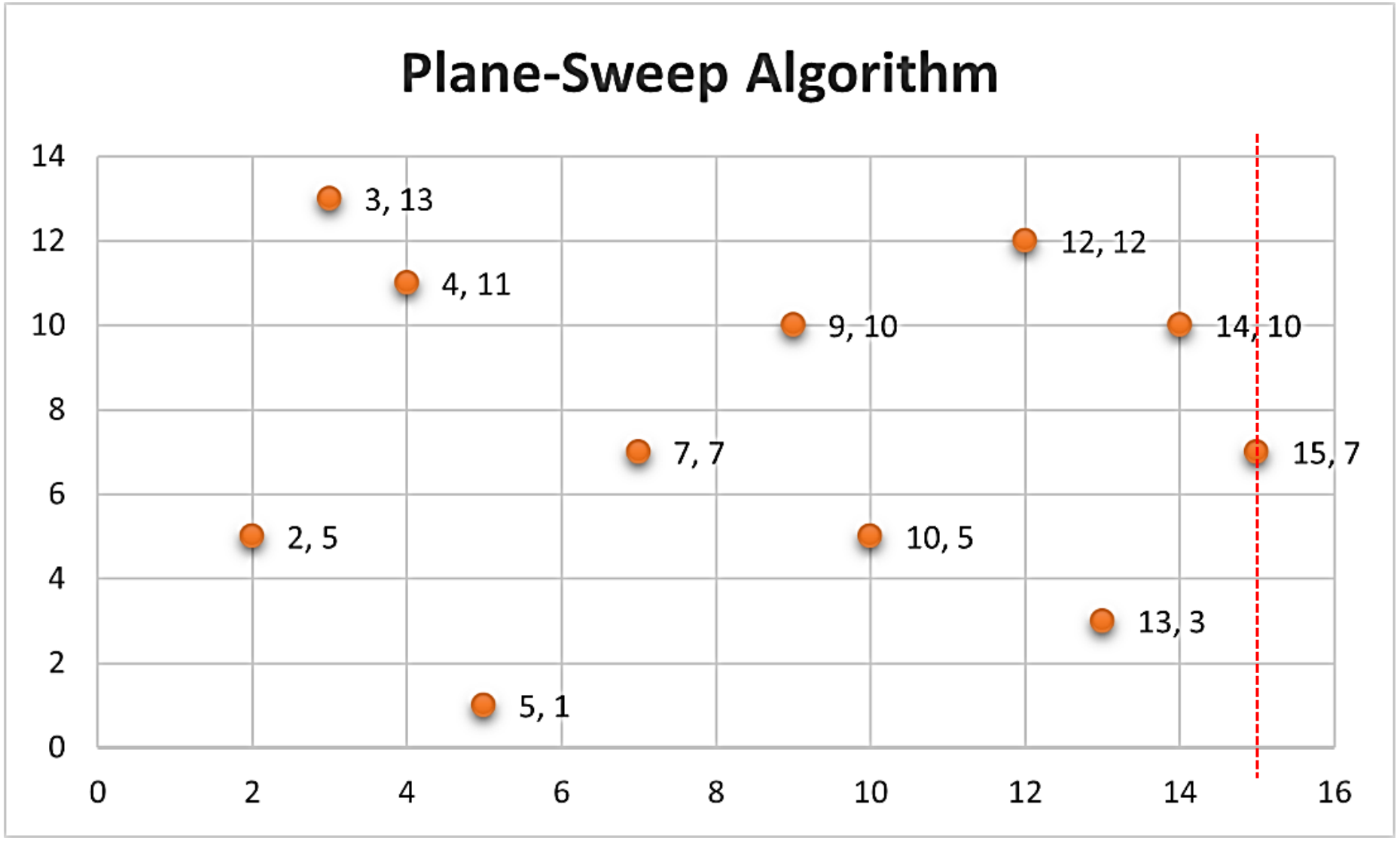
(13,3)
(12,12)
(3,13)

Stack



(14,10)
(12,12)
(3,13)

Stack



(15,7)
(14,10)
(12,12)
(3,13)

Stack

- At this point, the algorithm will stop, and we get the maximal points in the stack.
- We can also perform this whole operation on y axis.
- How we got this idea?

- This is the designing of the algorithm.
- Algorithm designing is an art.
- To master it we have to study existing examples, with experience and by solving different problems in different domains.

Analysis of Plane-Sweep Algorithm

```
PLANE-SWEEP-MAXIMA (n, Point P[1 … n])
1 sort P in increasing order by x coordinate
2 stack s
3 For i ← 1 to n
4   do
5     while (s is not empty AND s.top().y ≤ P[i].y)
6       s.pop()
7   s.push(P[i])
8 output the contents of stack s
```

- Sorting takes $O(n \log_2 n)$; we will show this later when we discuss sorting.
- The *for* loop executes n times.
- The inner loop (seemingly) could be iterated $(n - 1)$ times.
- It seems we still have an $n(n - 1)$ or $O(n^2)$ algorithm.
- Is it same as previous 2D maxima algorithm?
- We got confused by simple minded loop counting.

- We pop an element off the stack each time we go through the inner while-loop.
- It is not possible to pop more elements than are ever pushed on the stack.
- Therefore, the inner while-loop cannot execute more than n times over the entire course of the algorithm.

Worst-Case Scenario of Plane-Sweep Algorithm

- Worst-case scenario is when all the points are same.
- All the points will be pushed to the stack and no point will pop.
- It is not the case that a point will be pushed then pop then pushed again.
- Total n points will be pushed.
- No point will pop.

- The *for* loop iterates n times and the inner *while* loop also iterates n time for a total of $2n \approx O(n)$.
- Combined with the sorting, the runtime of entire plane-sweep algorithm is:

$$(n \log_2 n) + 2n$$

$$= \boldsymbol{O(n \log_2 n)}$$

Comparison

- How much of an improvement is plane-sweep over brute-force?
- Ratio of running times:

$$\frac{n^2}{n \log_2 n}$$

$$= \frac{n}{\log_2 n}$$

n	log n	Ratio (n/log n)
100	7	15
1000	10	100
10000	13	752
100000	17	6021
1000000	20	50171

- For example, if we have n = 1000000,
- The plane-sweep algorithm will take 1 second.
- The brute-force algorithm will take about 14 hours.