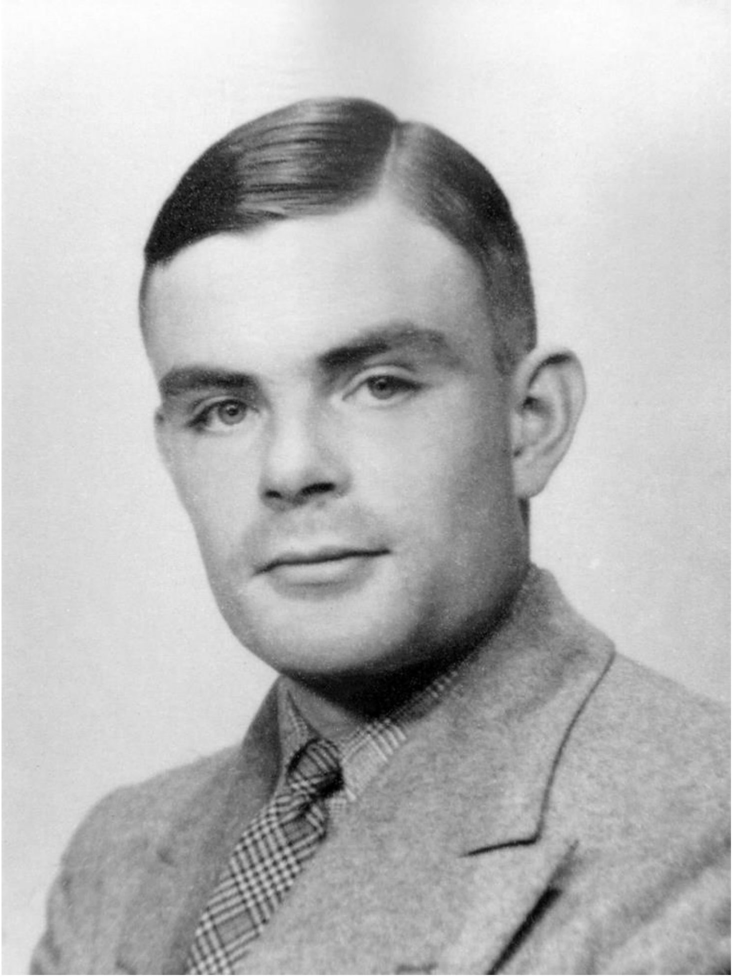
Divide & Conquer



[Alan Turing](#)

David beckham

Divide & Conquer

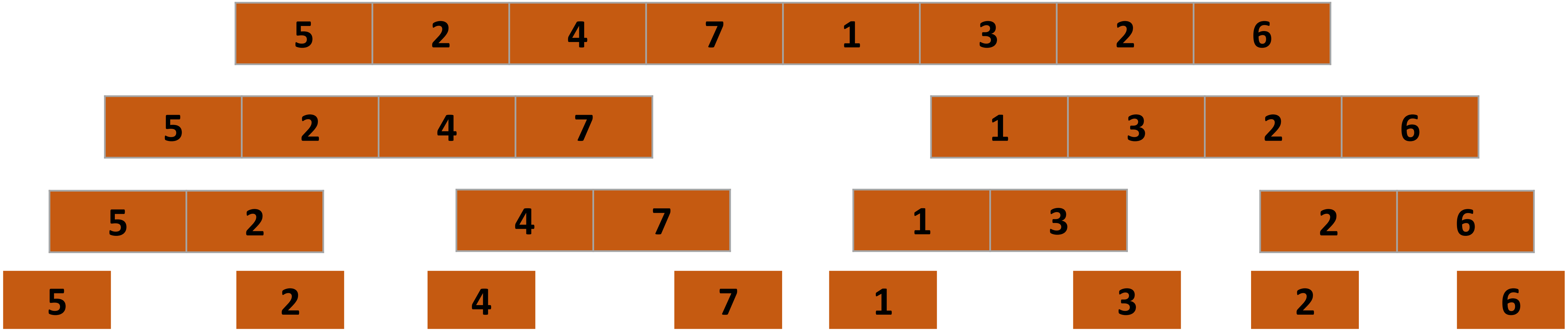
- MergeSort
- Karatsuba
- Closest pair
- Arbitrage
- FFT

Merge-Sort

merge-sort (A, p, r)
    if $p < r$
        $q \leftarrow \lfloor (p+r)/2 \rfloor$
        merge-sort (A, p, q)
        merge-sort $(A, q+1, r)$
        merge(A, p, q, r)

MERGE$(A[1\,..\,n], m)$:
  $i \leftarrow 1$; $j \leftarrow m+1$
  for $k \leftarrow 1$ to n
    if $j > n$
      $B[k] \leftarrow A[i]$; $i \leftarrow i+1$
    else if $i > m$
      $B[k] \leftarrow A[j]$; $j \leftarrow j+1$
    else if $A[i] < A[j]$
      $B[k] \leftarrow A[i]$; $i \leftarrow i+1$
    else
      $B[k] \leftarrow A[j]$; $j \leftarrow j+1$
  for $k \leftarrow 1$ to n
    $A[k] \leftarrow B[k]$

jeff erickson

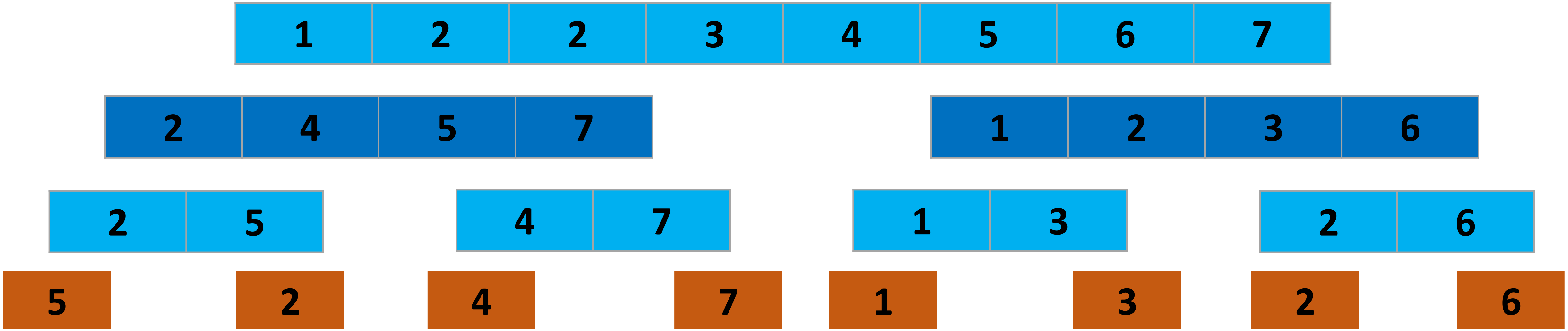


Merge-Sort

merge-sort (A, p, r)
    if $p < r$
        $q \leftarrow \lfloor (p+r)/2 \rfloor$
        merge-sort (A, p, q)
        merge-sort $(A, q+1, r)$
        merge(A, p, q, r)

MERGE$(A[1..n], m)$:
    $i \leftarrow 1$; $j \leftarrow m+1$
    for $k \leftarrow 1$ to n
        if $j > n$
            $B[k] \leftarrow A[i]$; $i \leftarrow i+1$
        else if $i > m$
            $B[k] \leftarrow A[j]$; $j \leftarrow j+1$
        else if $A[i] < A[j]$
            $B[k] \leftarrow A[i]$; $i \leftarrow i+1$
        else
            $B[k] \leftarrow A[j]$; $j \leftarrow j+1$
    for $k \leftarrow 1$ to n
        $A[k] \leftarrow B[k]$

jeff erickson



Merge-Sort

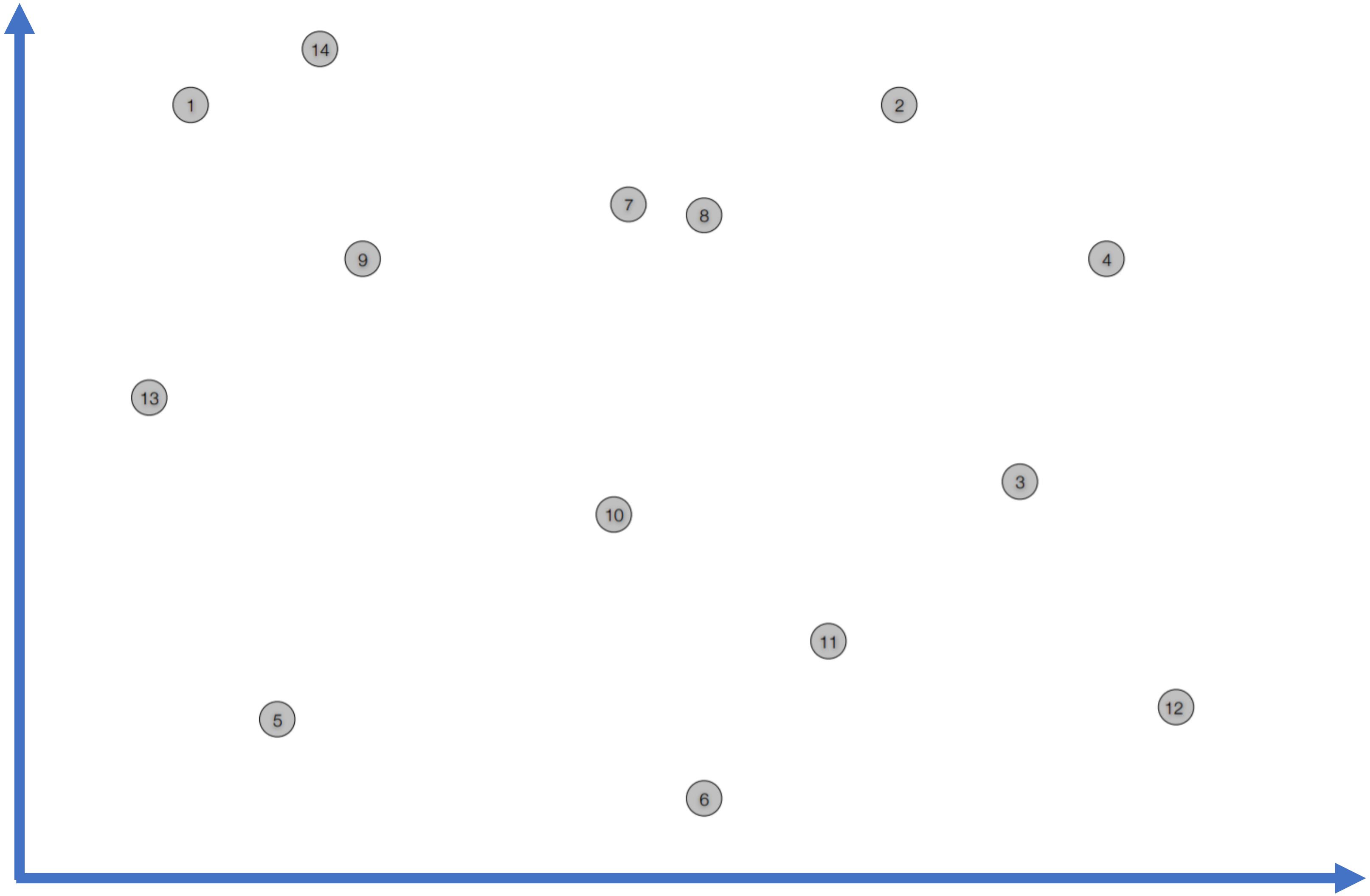
merge-sort (A, p, r)
    if $p < r$
        $q \leftarrow \lfloor (p + r)/2 \rfloor$ O(1)
        merge-sort (A, p, q) T(n/2)
        merge-sort $(A, q + 1, r)$ T(n/2)
        merge(A, p, q, r) Θ(n)

T(n)=2T(n/2)+O(n)

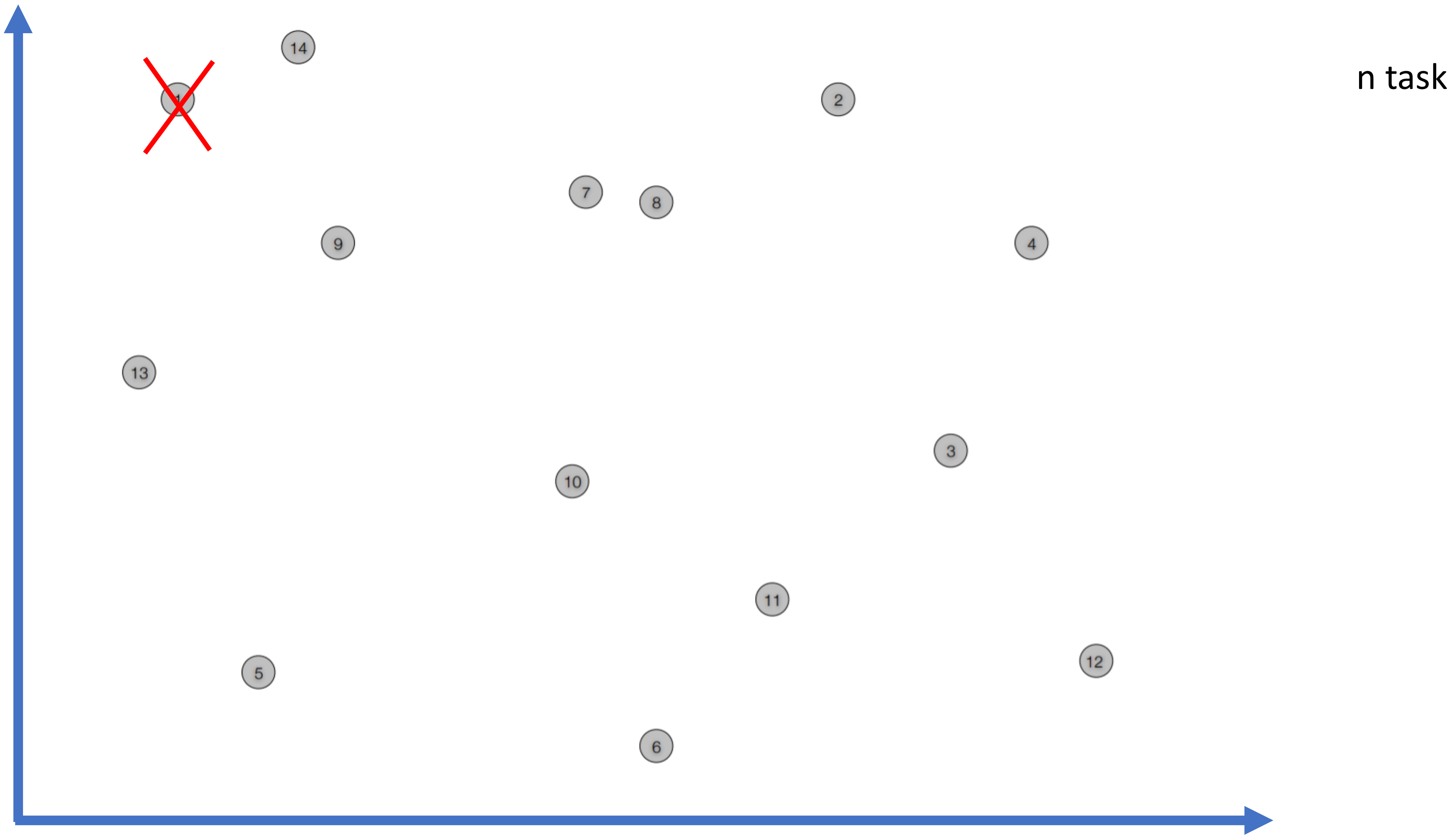
= θ(nlogn)

Closest pair of points

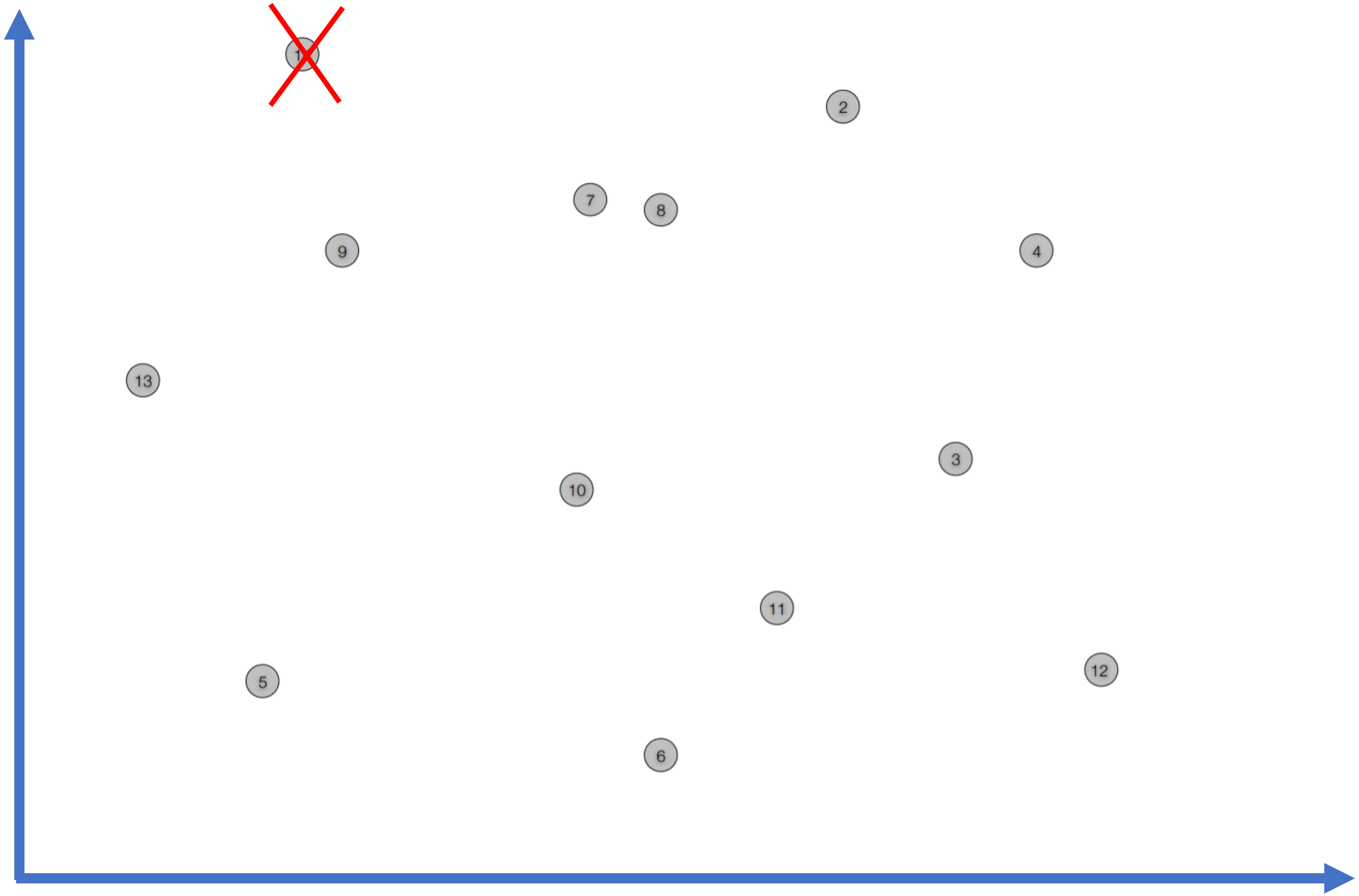
Closest pair problem



Closest pair problem – brute force



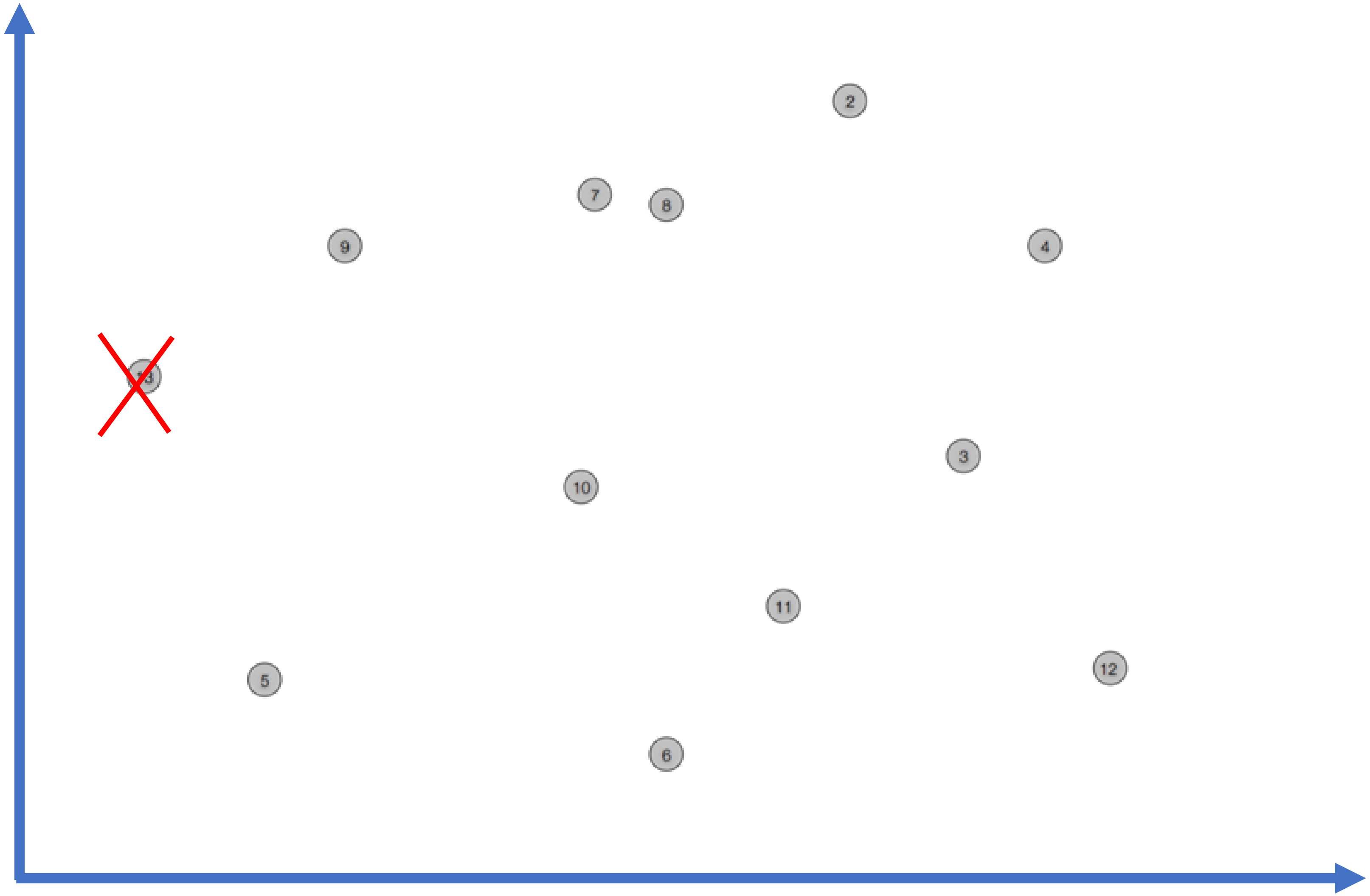
Closest pair problem – brute force



n task

n-1 task

Closest pair problem – brute force



n task

n-1 task

n-2 task

Closest pair problem – brute force

n task

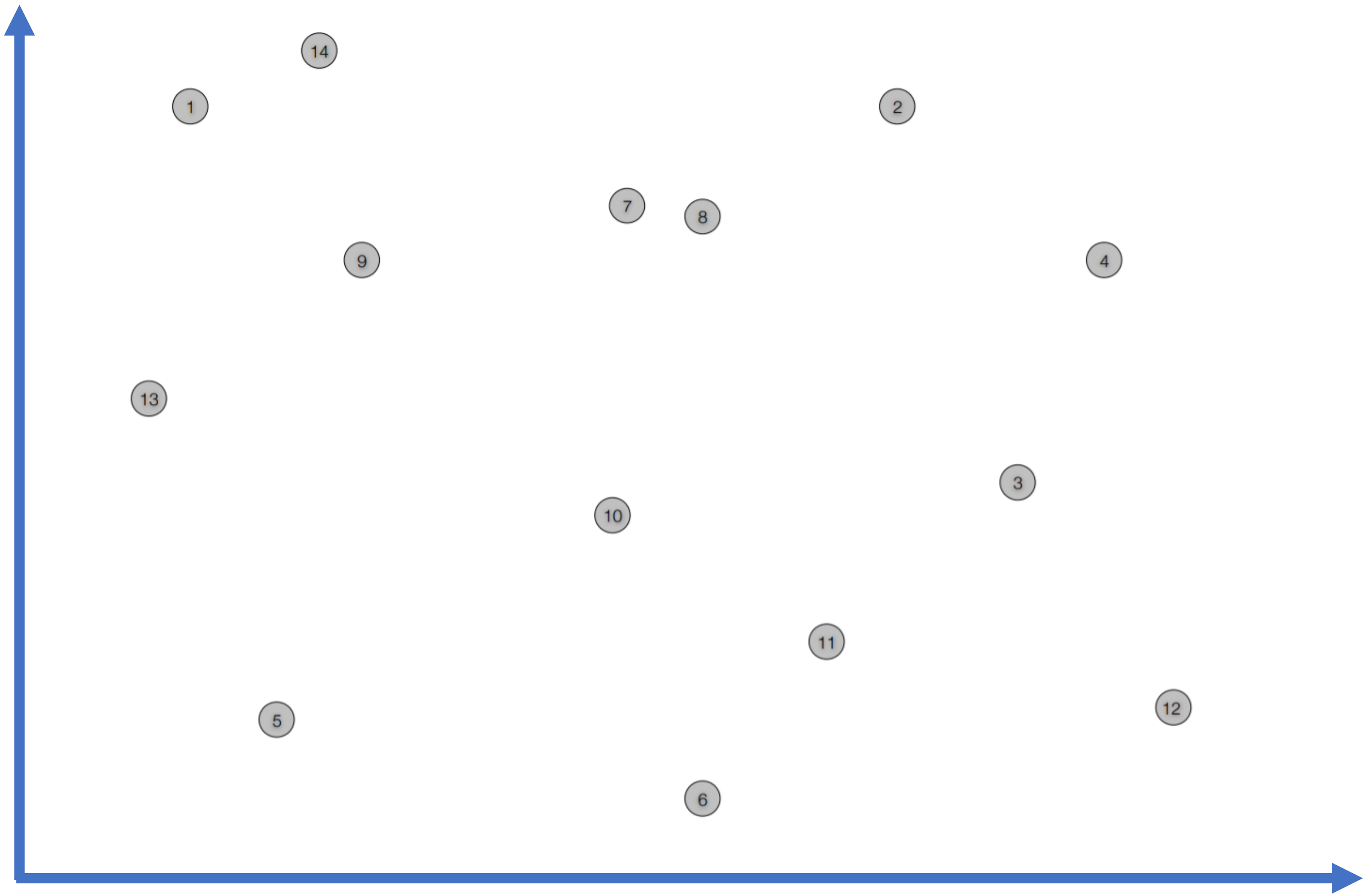
n-1 task

n-2 task

n-3 task

⋮

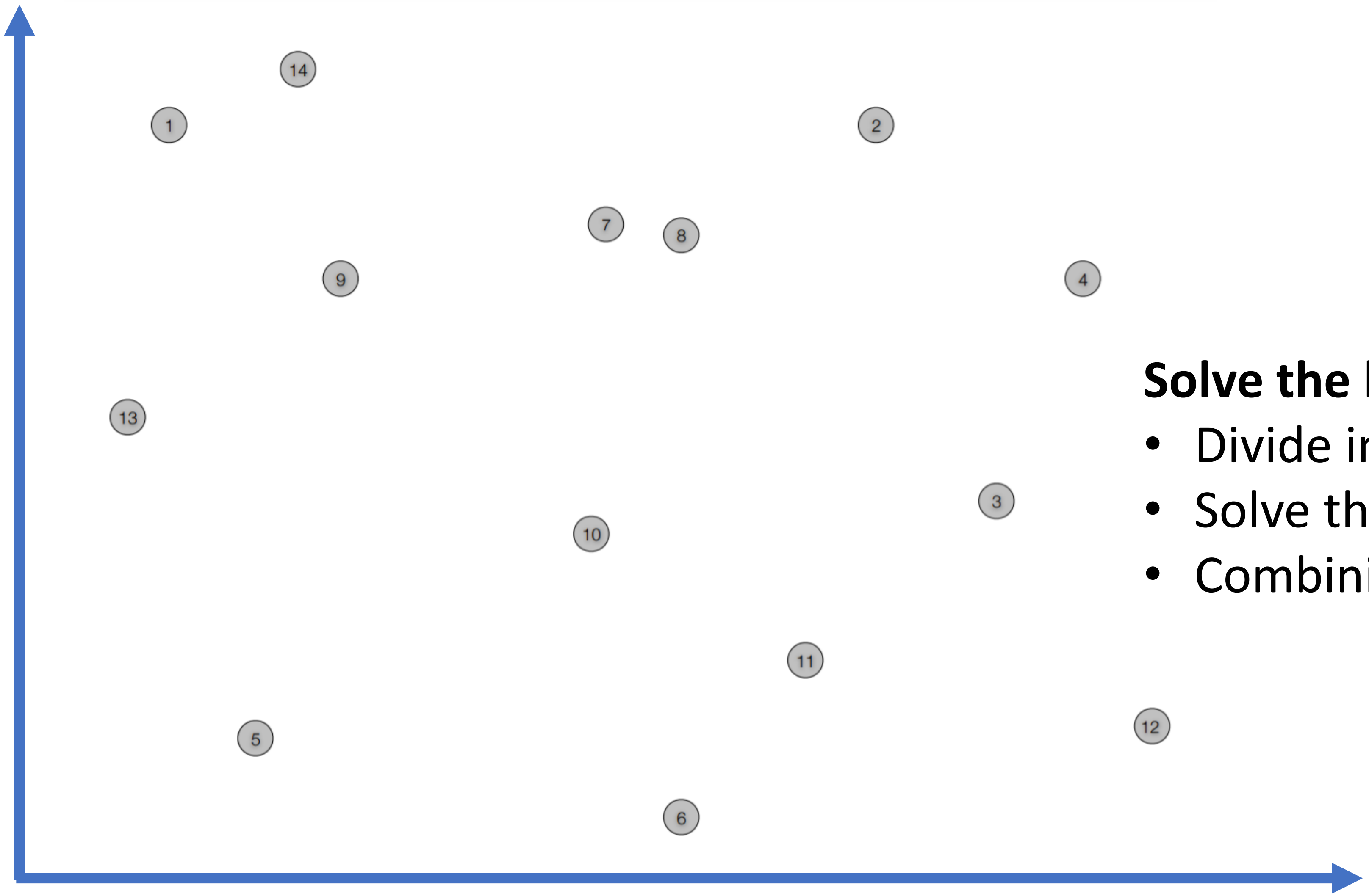
1 task



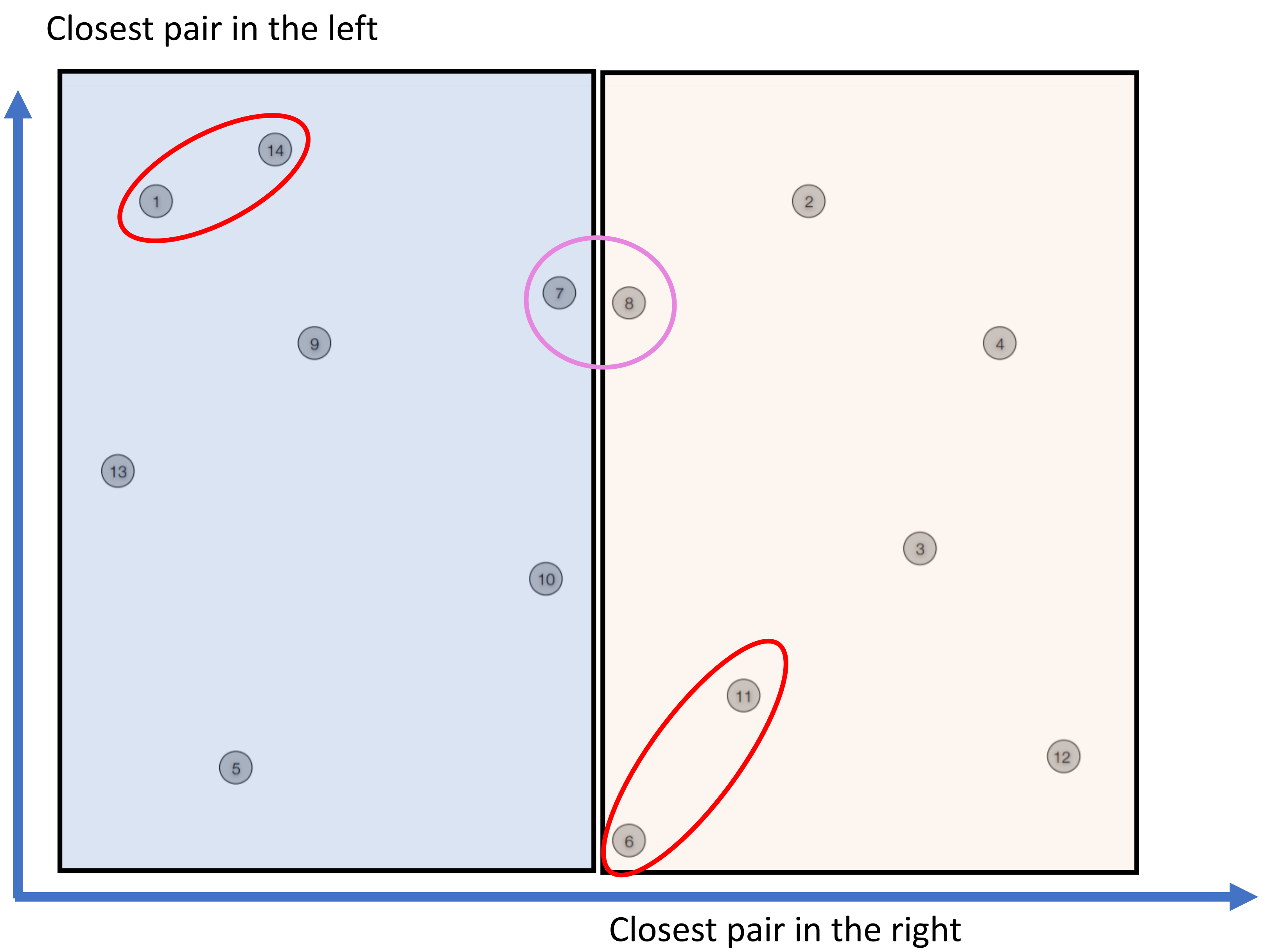
$T(n)=T(n-1)+n$

$T(n)=\theta(n^2)$

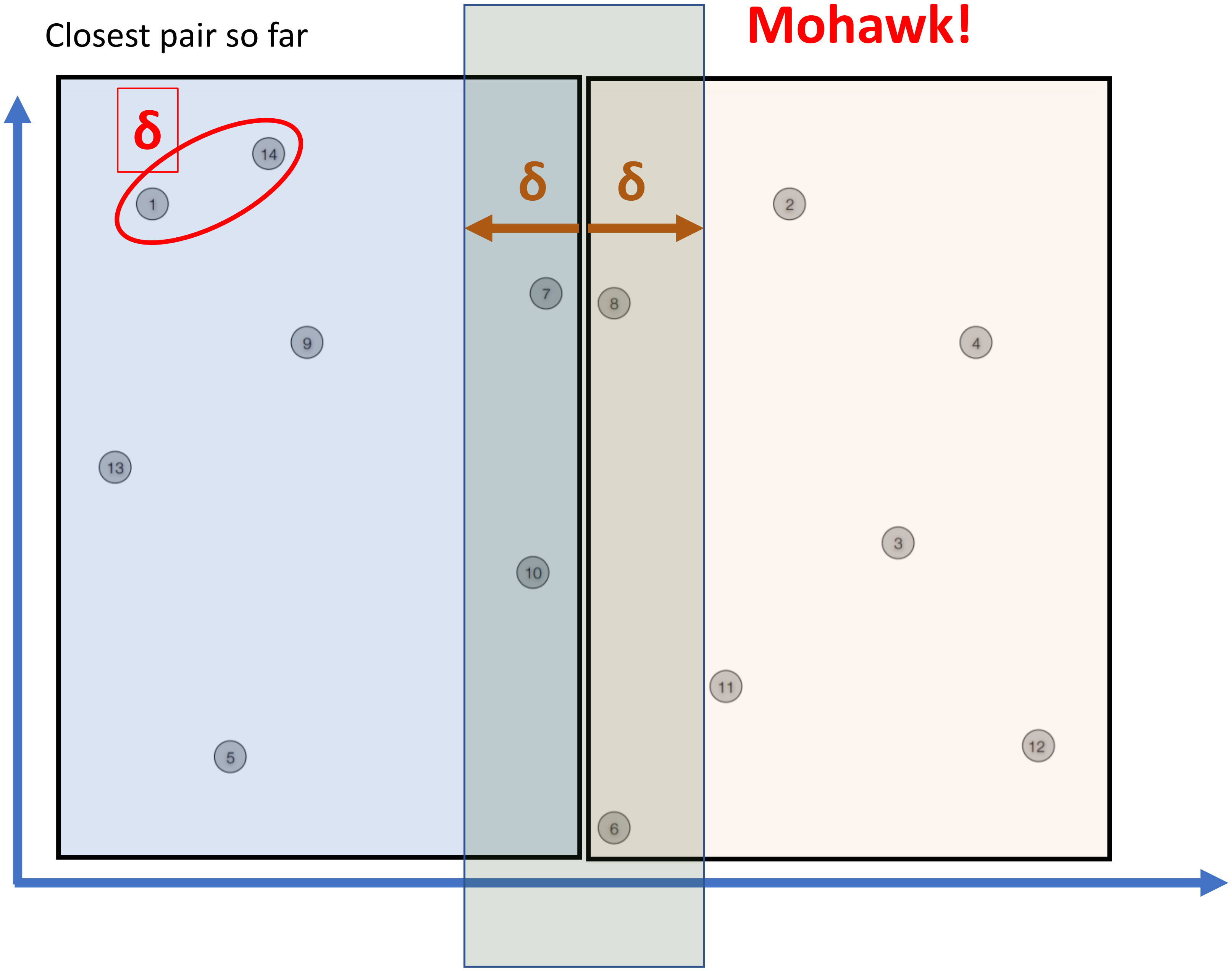
Closest pair problem – Divide & Conquer



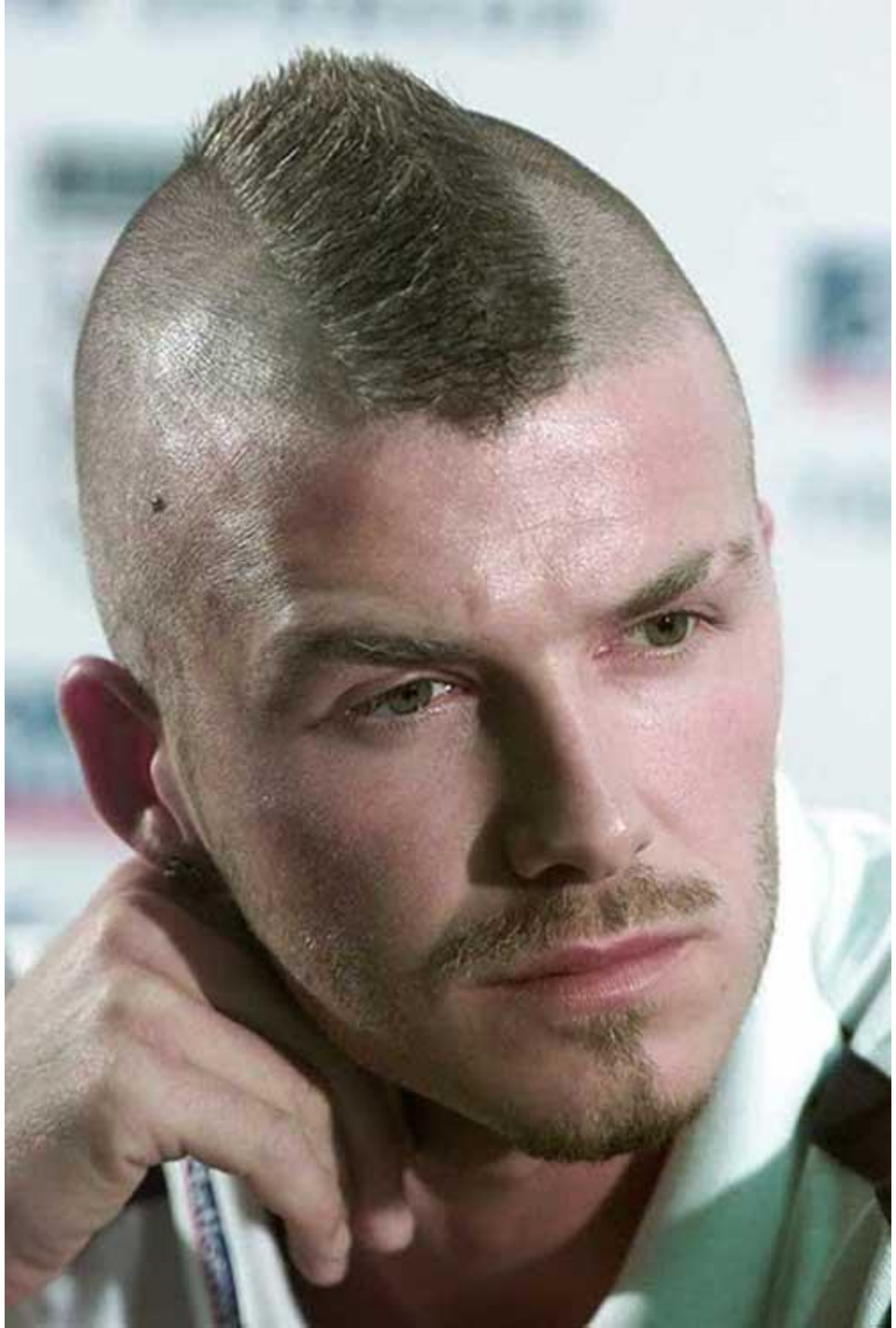
Solve the large problem by
- Divide into smaller problems
- Solve the smaller problems
- Combining smaller solutions

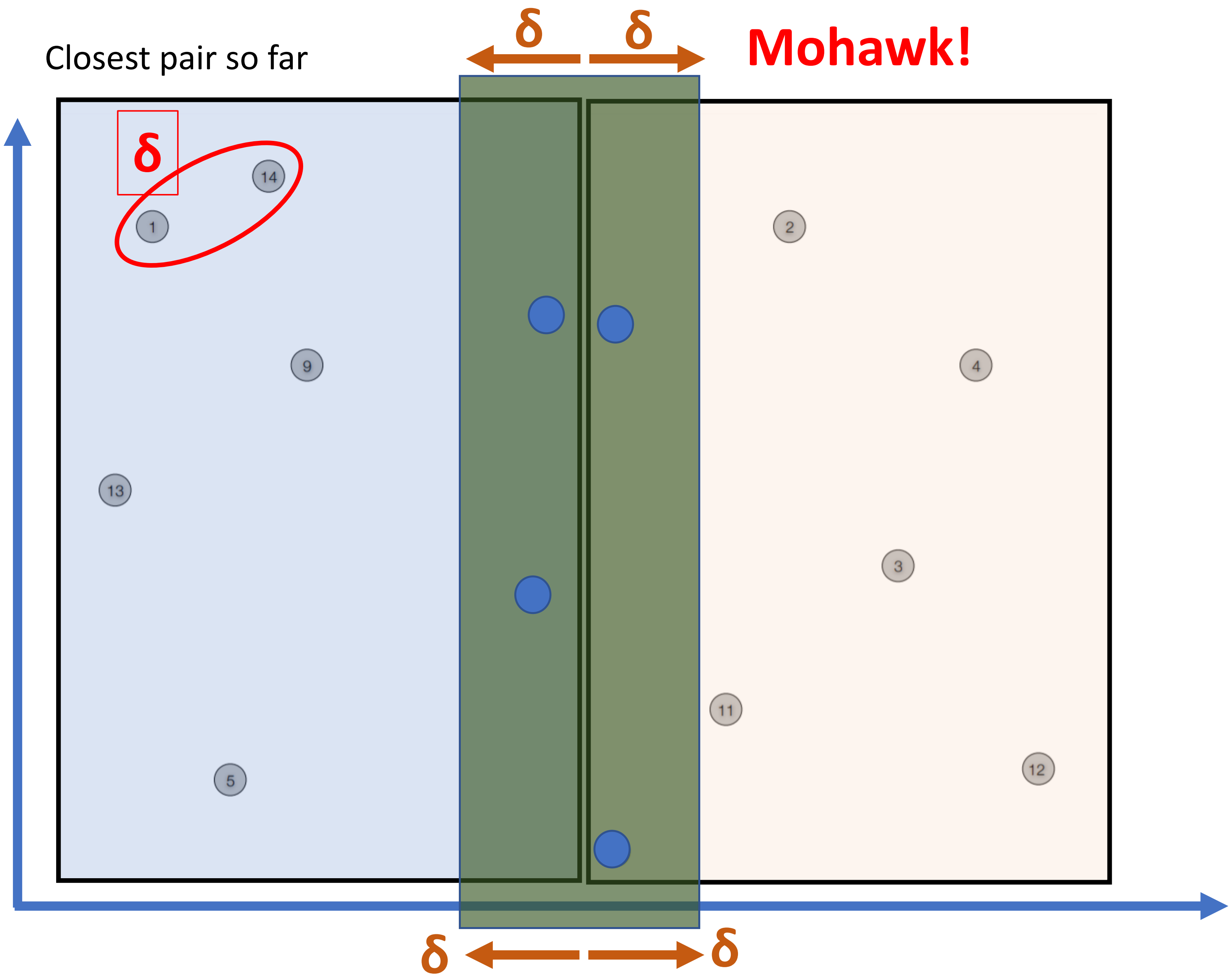


Closest pair in the left
14
1
2
7
8
9
4
13
3
10
11
12
5
6
Closest pair in the right



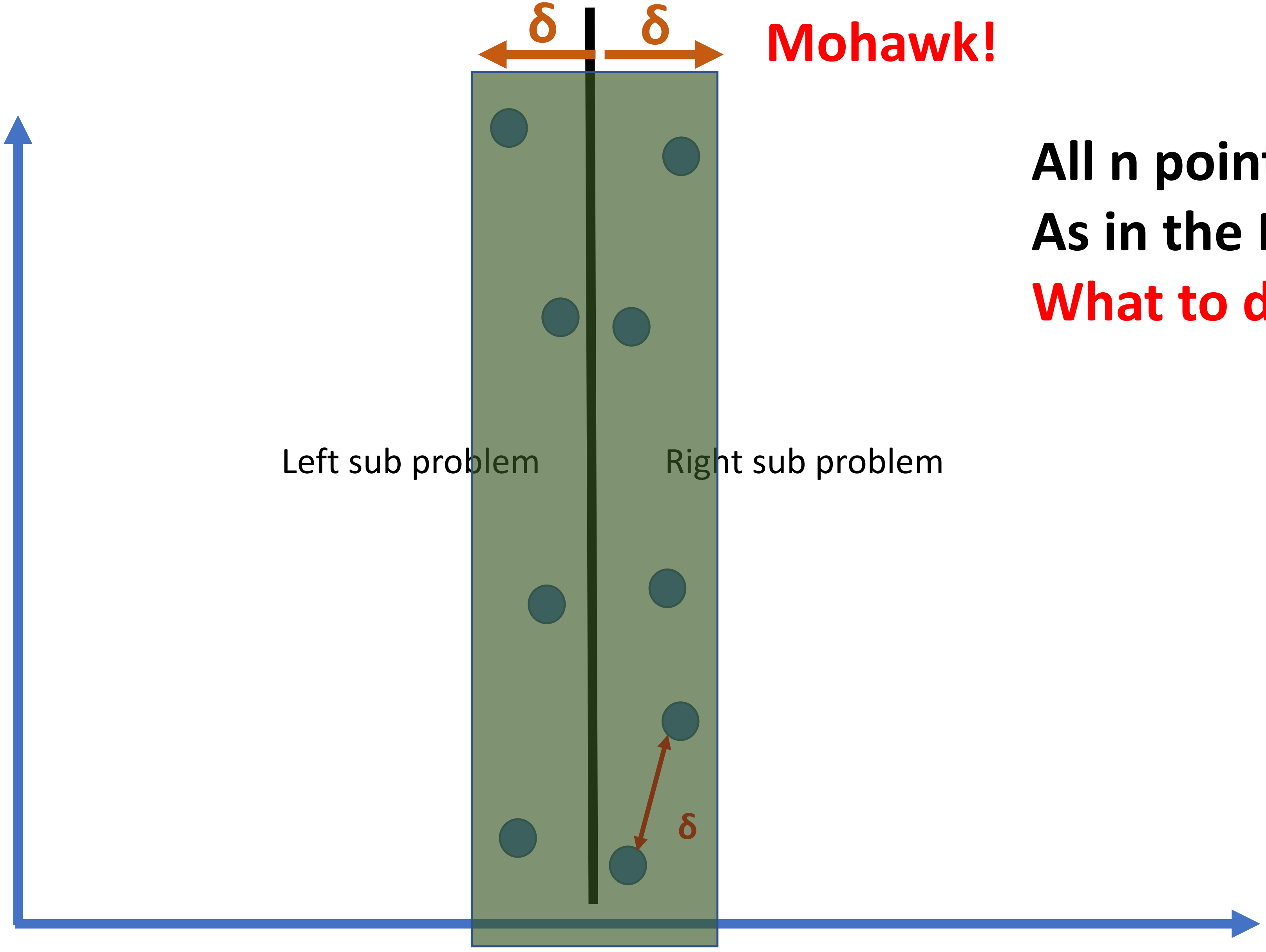
Closest pair so far
Mohawk!
δ
δ
δ
14
1
2
7
8
9
4
13
3
10
11
12
5
6





Mohawk!

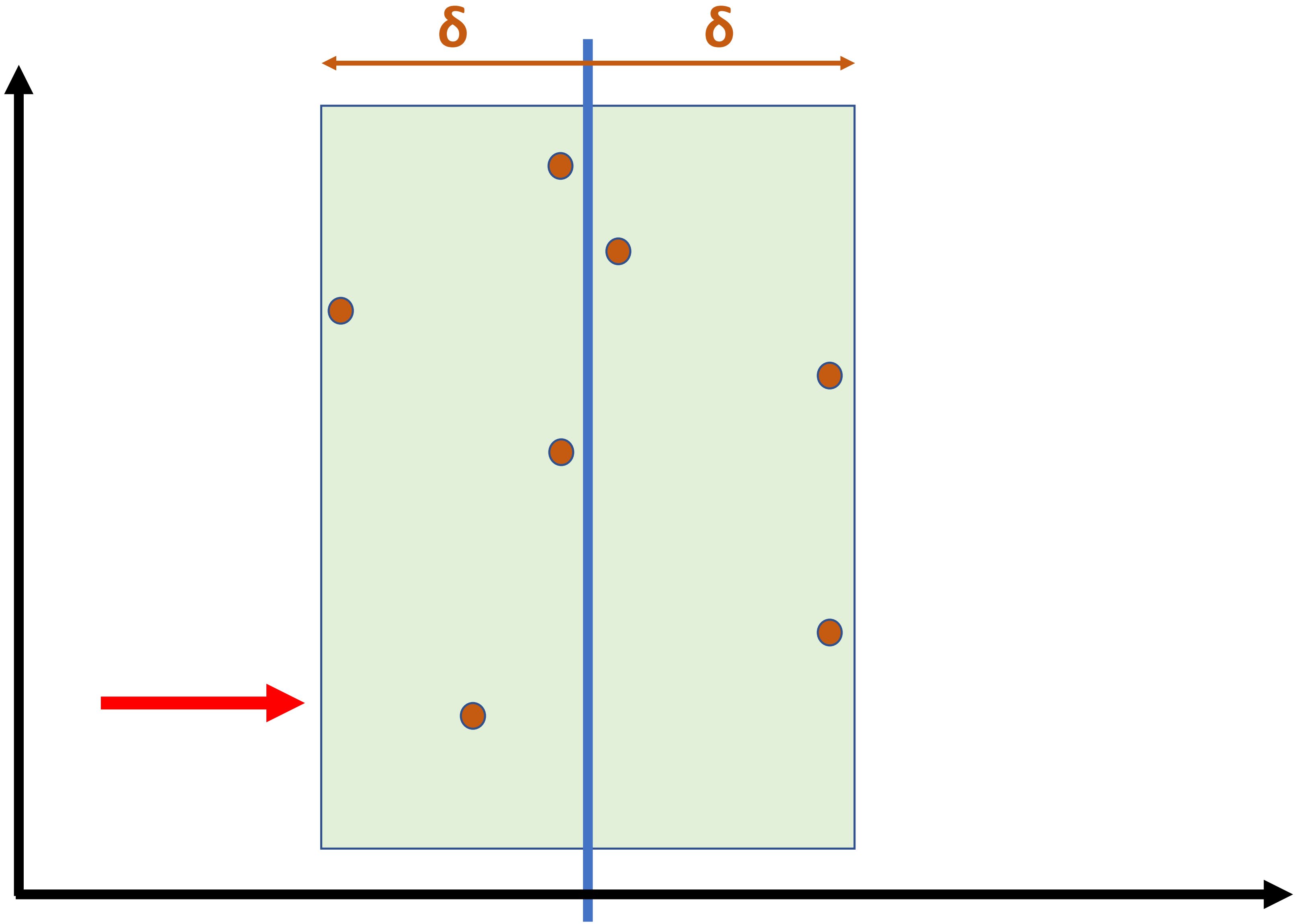
Challenge?



All n points we have as input

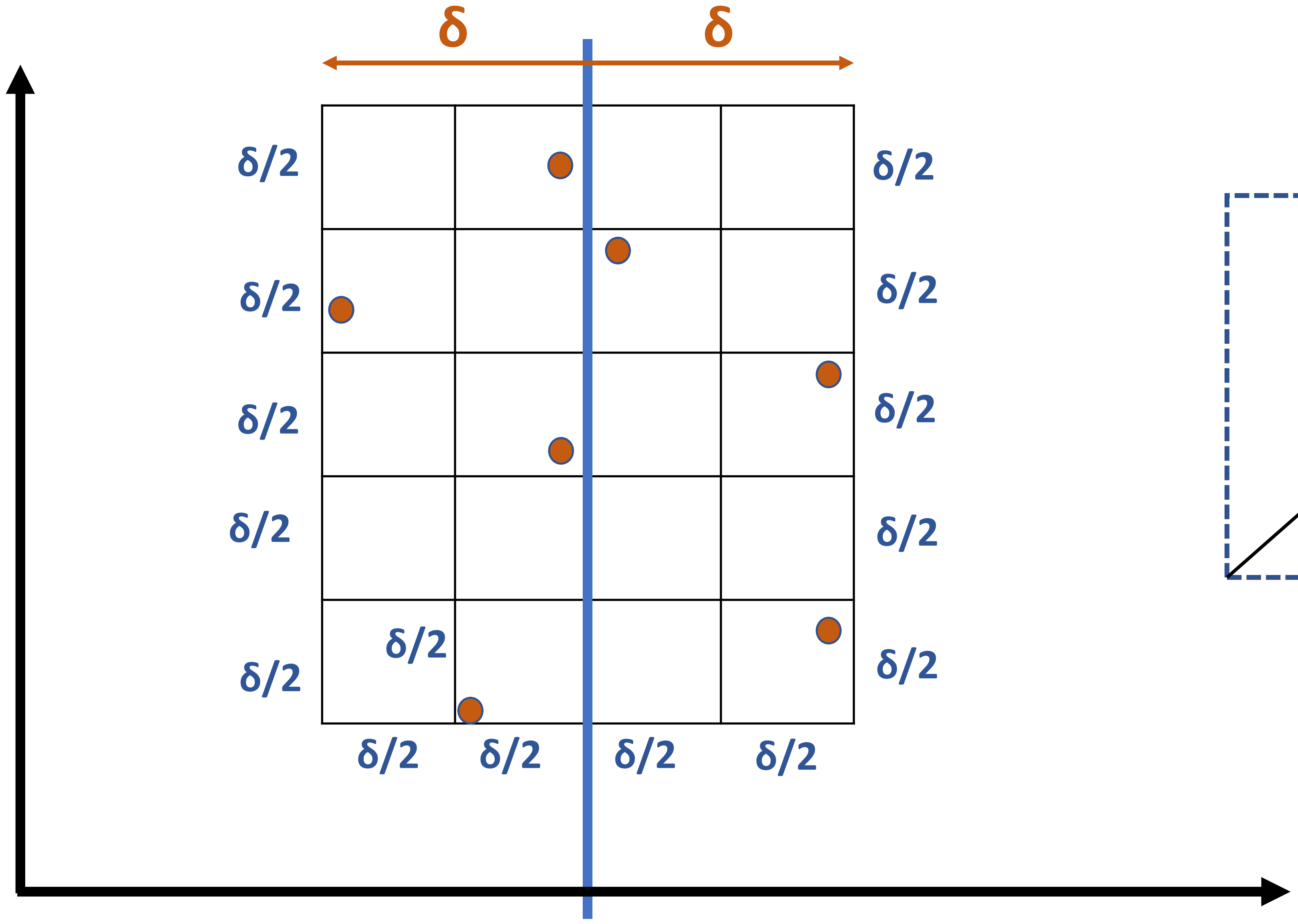
As in the Mohawk!!

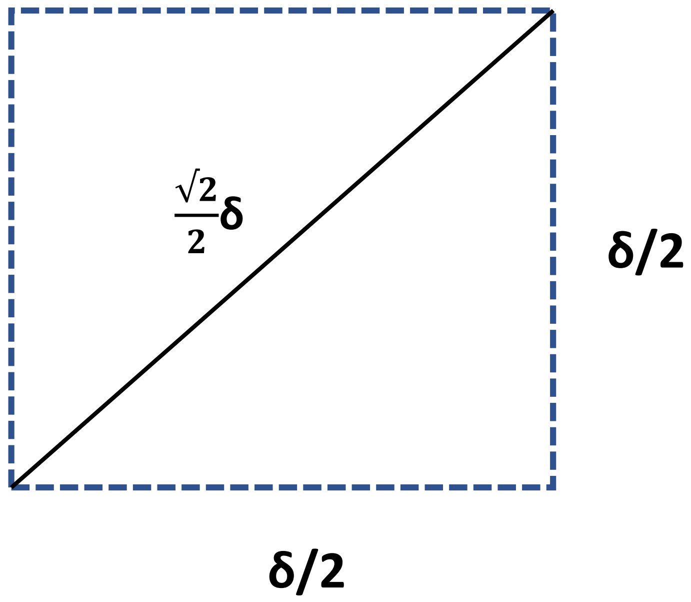
What to do!



Algorithmic insights

Draw **$\delta/2 \times \delta/2$ grids starting at the lowest Y point**

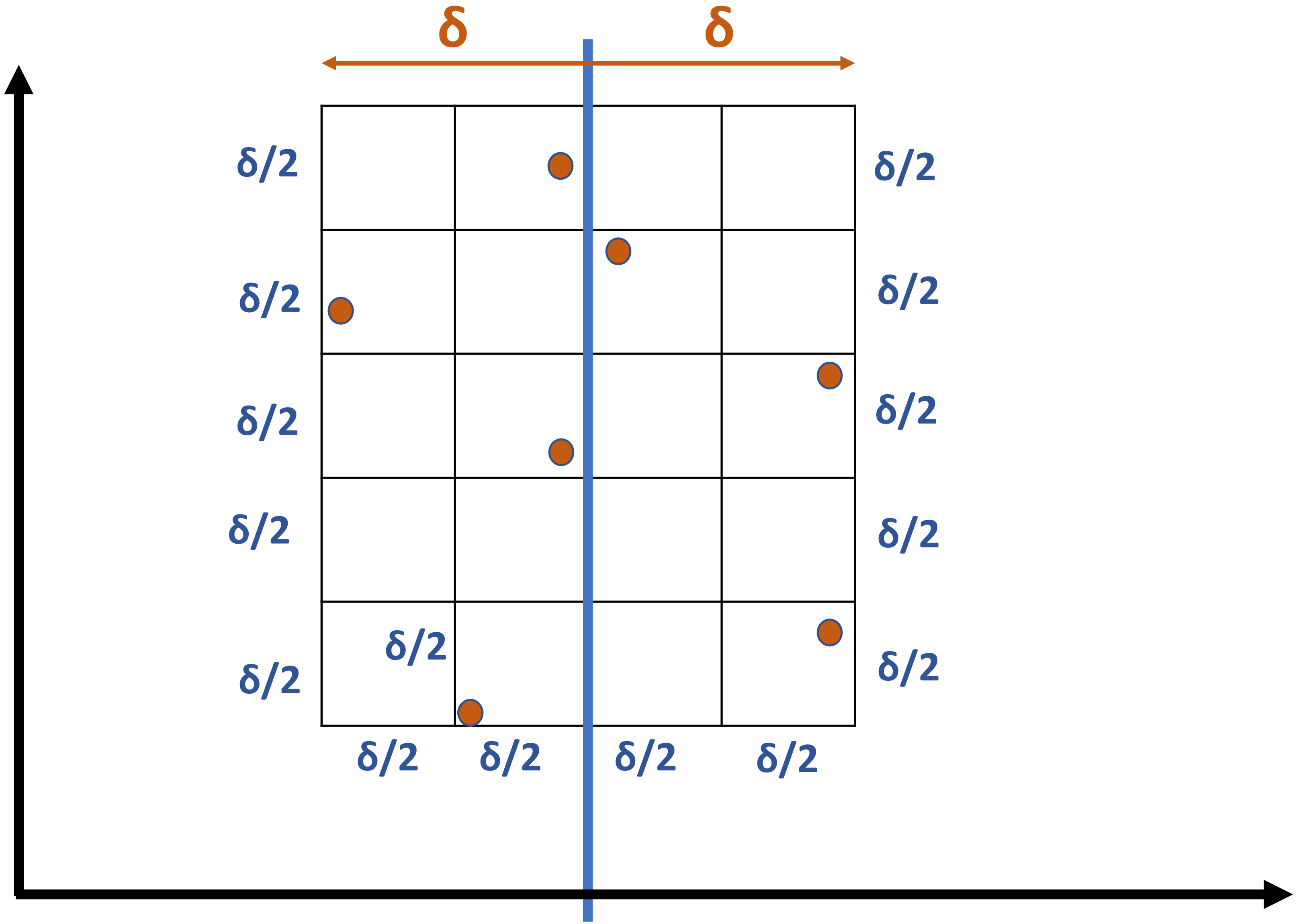




Insight

There can be at most one point in One grid

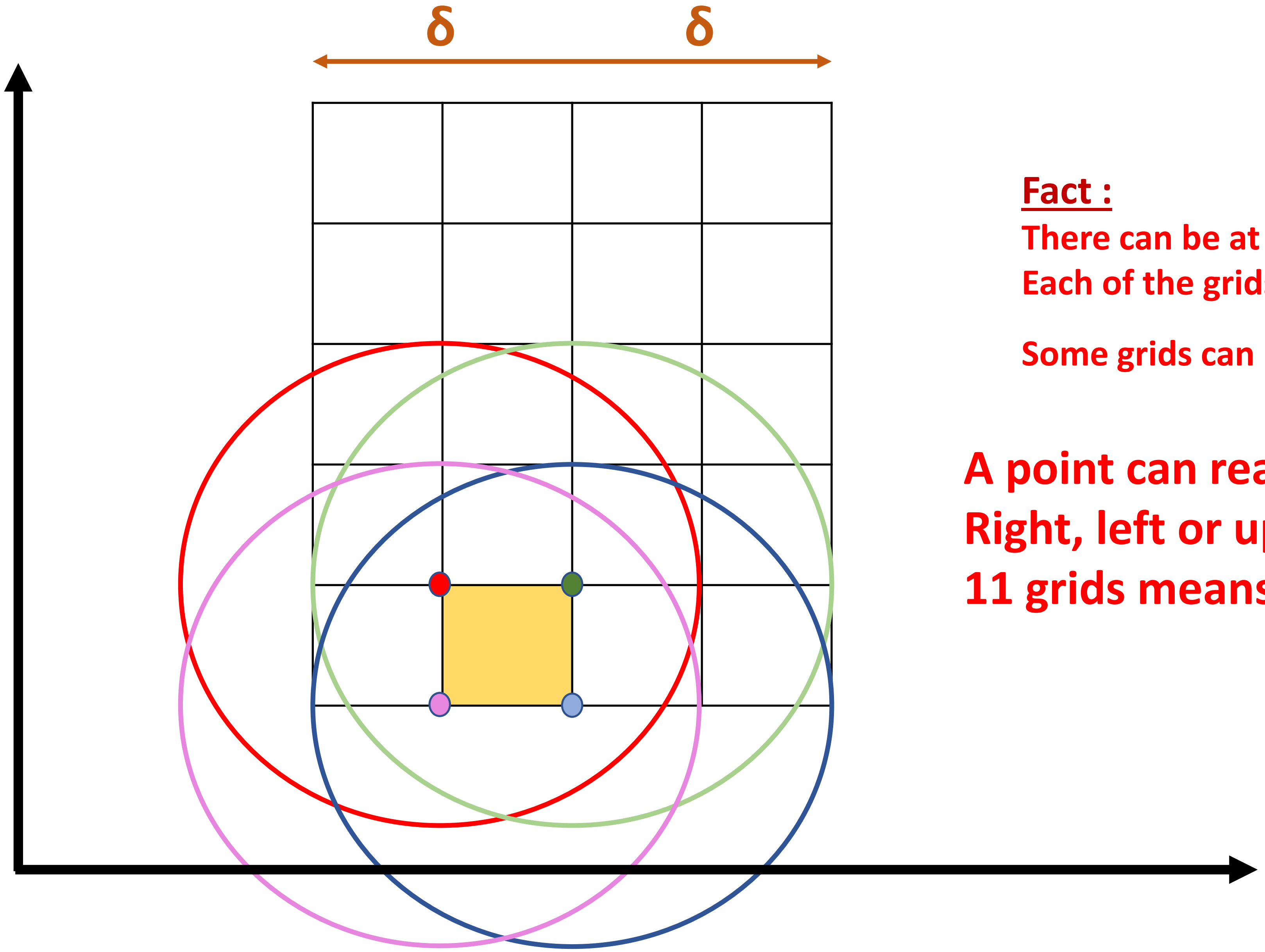
Draw $\delta/2 \times \delta/2$ grids starting at the lowest Y point



Fact :

There can be at most one point in
Each of the grids

Some grids can be empty too.



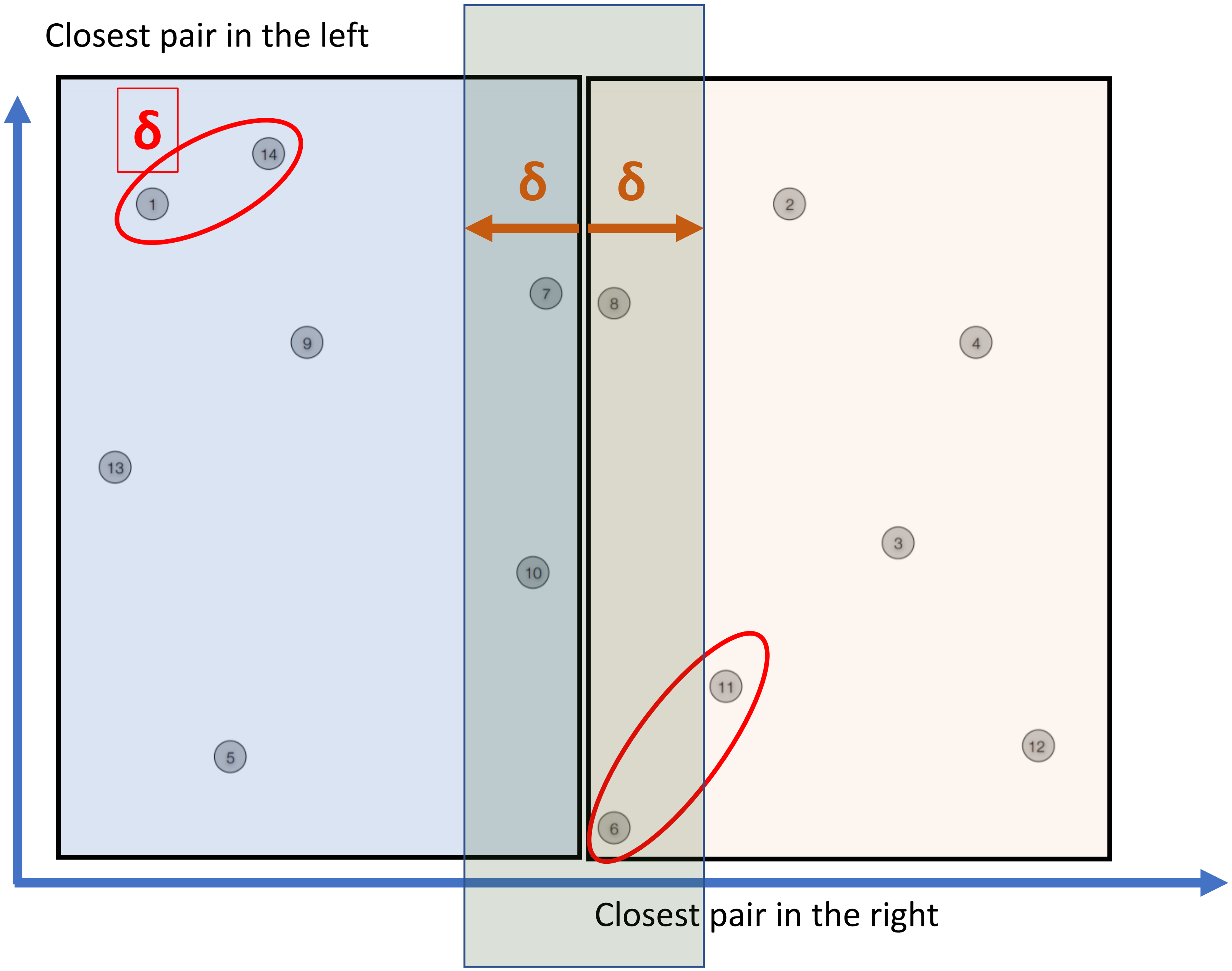
Fact :

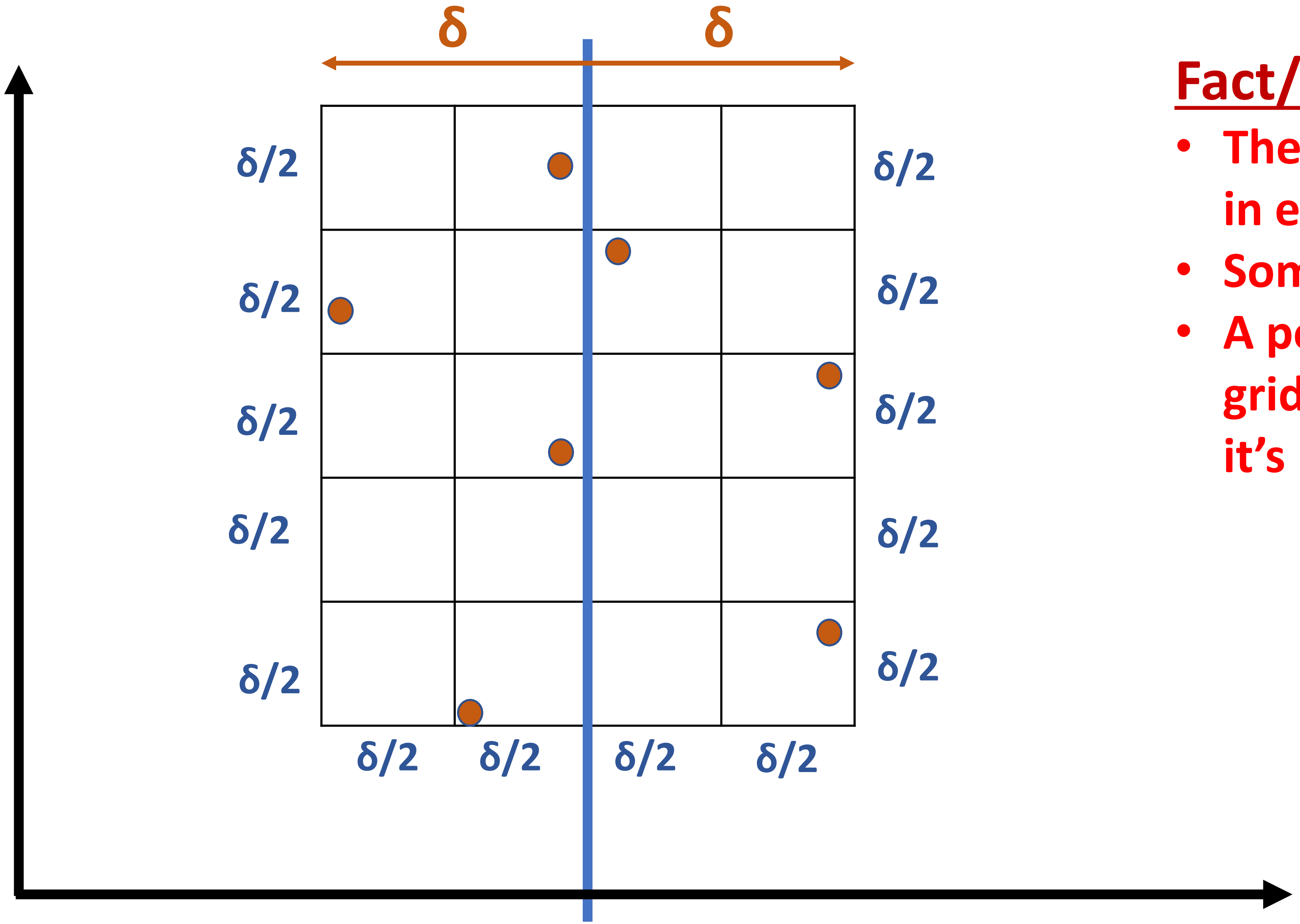
There can be at most one point in
Each of the grids

Some grids can be empty too.

A point can reach at most 11 grids, on it's Right, left or up direction.

11 grids means at-most 11 points! *(upper limit)*


Closest pair in the left
δ
δ
δ
14
1
2
7
8
9
4
13
3
10
11
12
5
6
Closest pair in the right



Fact/insights :

- There can be at most one point in each of the grids
- Some grids can be empty too.
- A point can reach at most 11 grids, meaning, 11 points, on it's right, left or up direction

Closest(P) : input points in 2D

1. Base case: if |P|<=2, brute force. → $O(1)$
2. Let q be the mid-point along X-coordinate. → $\Theta(n)$
3. Divide P into left, right according to q → $\Theta(n)$

4. δ_L=Closest(left) → $T(n/2)$
5. δ_R=Closest(right) → $T(n/2)$
6. $\delta = \mathrm{Min}(\delta_L, \delta_R)$ → $O(1)$

7. Mohawk= {scan P, add points that are in delta (δ) distance from q.x } → $\Theta(n)$

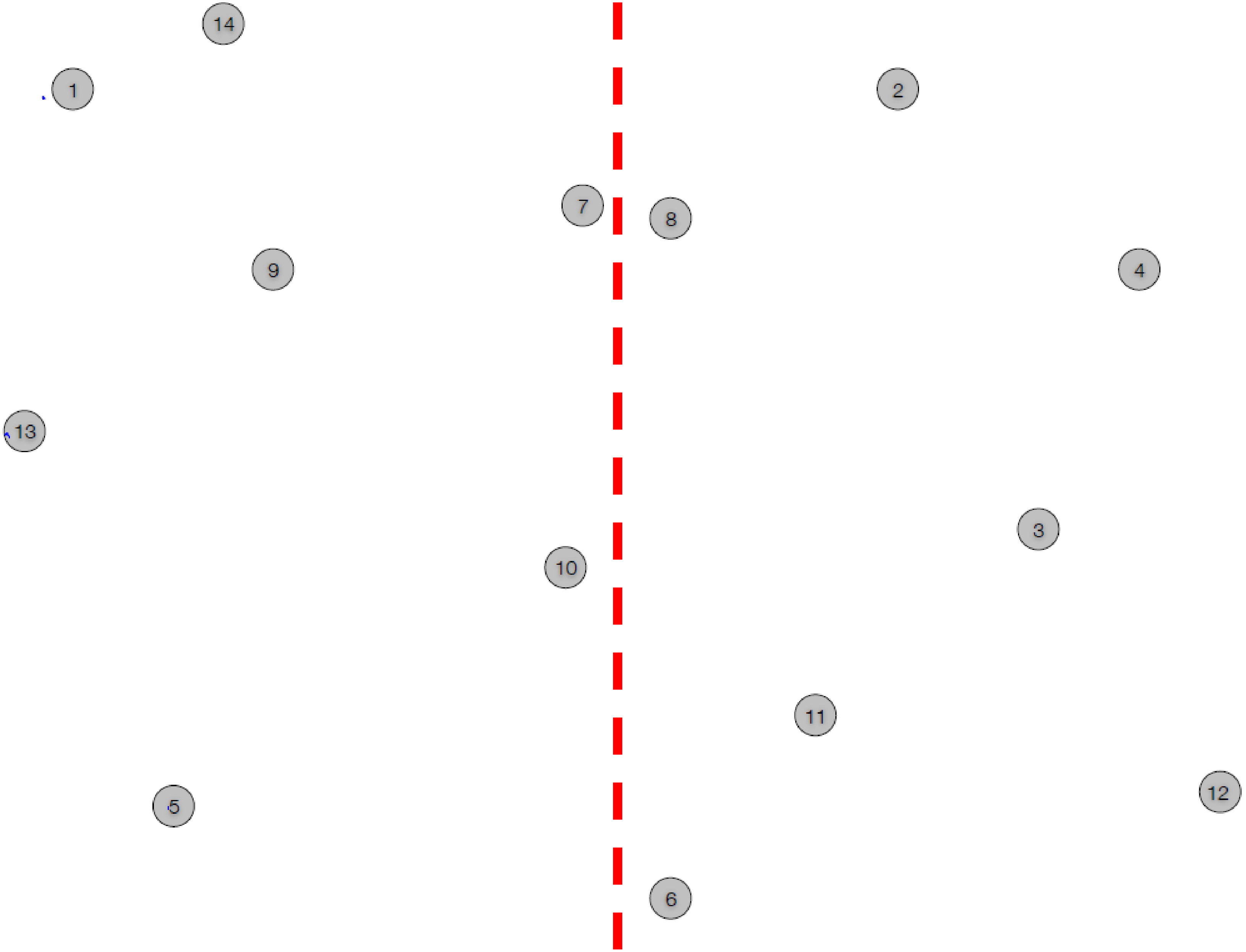
8. For each point x in Mohawk (in y-coordinate):
9. Compute distance to it's next 11 neighbors in higher Y-coordinate
10. Update δ if any pair of points have distance < δ

→ $\Theta(n)$

11. Return δ

$T(n)=2T(n/2)+\Theta(n)$ **$T(n)=\Theta(n\log n)$**

sorted in X: 13 1 5 14 9 10 7 6 8 11 2 3 4 12
sorted in Y: 6 5 12 11 10 3 13 4 9 8 7 2 1 14



ClosestPair(P) :

Compute X coordinate sorted list SX → $\Theta(n\log n)$

Compute X coordinate sorted list SY → $\Theta(n\log n)$

Closest(P,SX,SY) → $\Theta(n\log n)$

Overall solution is still $\Theta(n\log n)$

ClosestPair(P) :

Compute X coordinate sorted list SX → $\Theta(n\log n)$

Compute X coordinate sorted list SY → $\Theta(n\log n)$

Closest(P,SX,SY) → $\Theta(n\log n)$

Closest(P,SX,SY) :

1. Base case: if |P|<=2, brute force.
2. Let q be the mid-element of SX
3. Divide P into left, right according to q
4. δ = Min(Closest(left,LX,LY), Closest(left,RX,RY))
5. Mohawk= {scan SY, add points that are in delta (δ) distance from q.x } → Advantage? Getting them in sorted order of Y coordinate
6. For each point x in Mohawk (in y-coordinate):
7. Compute distance to it's next 11 neighbors in higher Y-coordinate
8. Update δ if any pair of points have distance < δ
9. Return δ

ClosestPair(P) :

Compute X coordinate sorted list SX → Θ(nlogn)

Compute X coordinate sorted list SY → Θ(nlogn)

Closest(P,SX,SY) → Θ(nlogn)

Closest(P,SX,SY) :

1. Base case: if |P|<=2, brute force.
2. Let q be the mid-element of SX
3. Divide P into left, right according to q

4. δ = Min(Closest(left,LX,LY), Closest(left,RX,RY))

5. Mohawk= {scan SY, add points that are in delta (δ) distance from q.x } → Advantage? Getting them in sorted order of Y coordinate

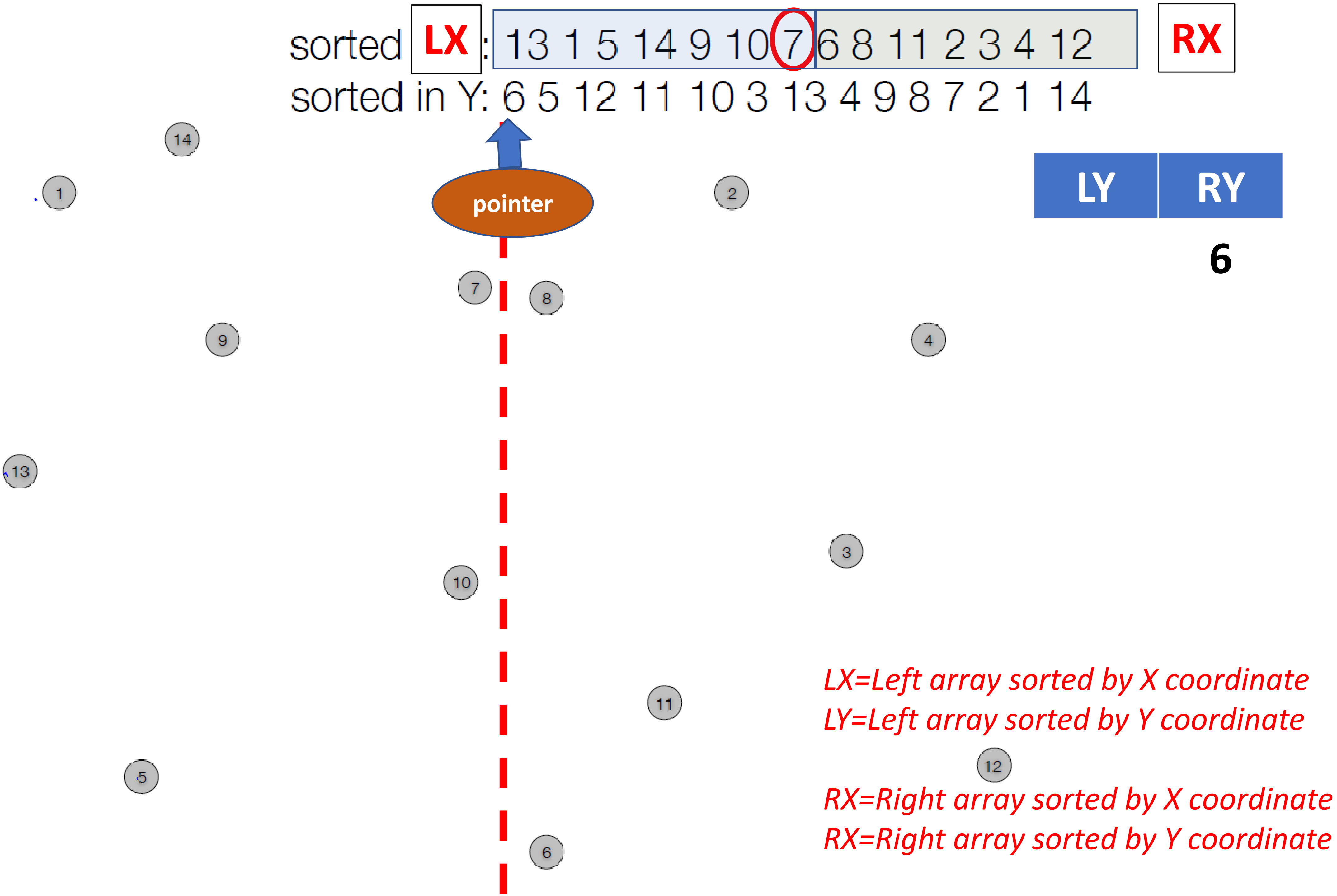
6. For each point x in Mohawk (in y-coordinate):
7. Compute distance to it's next 11 neighbors in higher Y-coordinate
8. Update δ if any pair of points have distance < δ
9. Return δ

LX=Left array sorted by X coordinate

LY=Left array sorted by Y coordinate

RX=Right array sorted by X coordinate

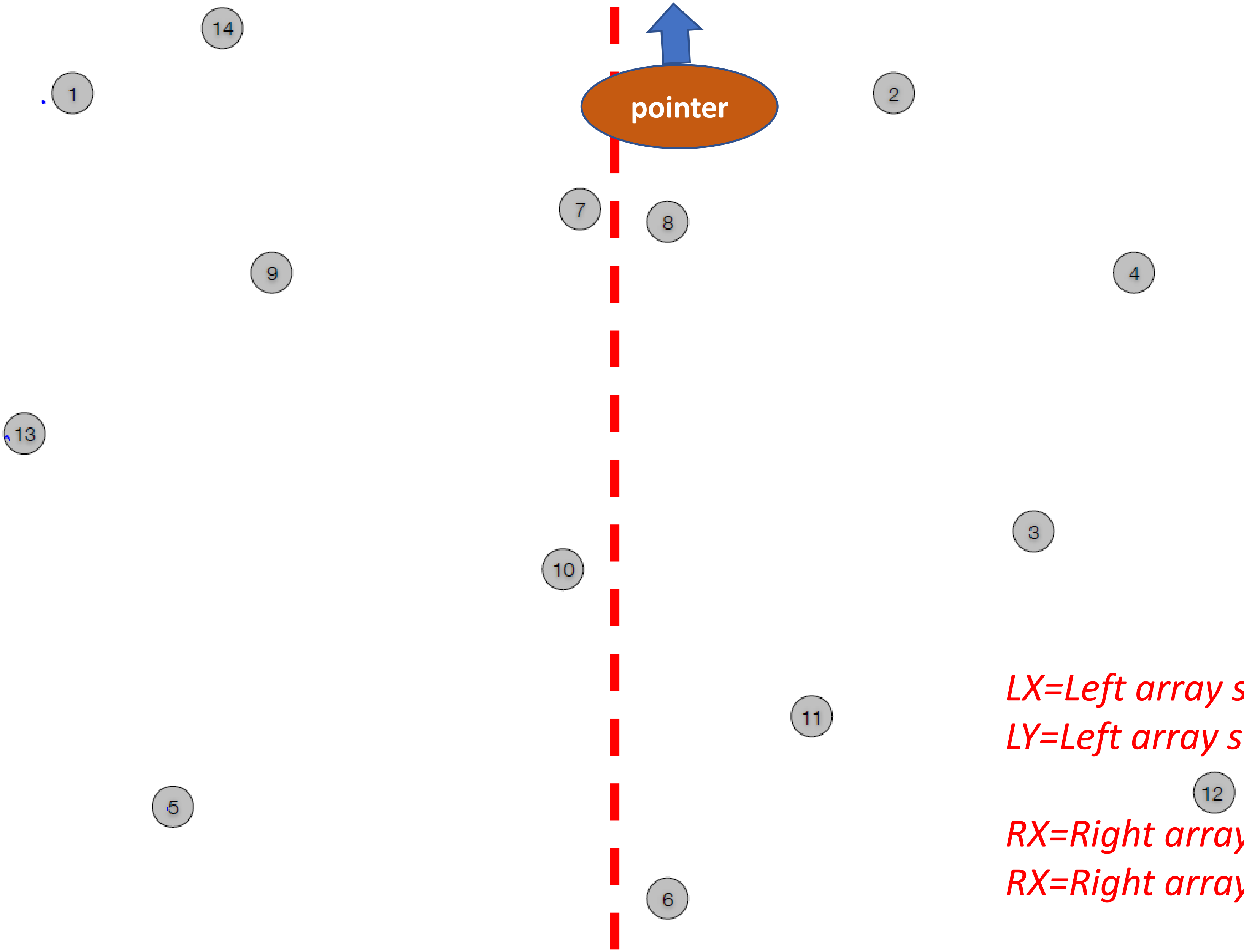
RX=Right array sorted by Y coordinate



sorted LX: 13 1 5 14 9 10 7 6 8 11 2 3 4 12 RX
sorted in Y: 6 5 12 11 10 3 13 4 9 8 7 2 1 14
pointer
LY
RY
6
14
1
2
7
8
9
4
13
3
10
11
12
5
6
LX=Left array sorted by X coordinate
LY=Left array sorted by Y coordinate
RX=Right array sorted by X coordinate
RX=Right array sorted by Y coordinate

sorted **LX** : 13 1 5 14 9 10 7 | 6 8 11 2 3 4 12 **RX**

sorted in Y: 6 5 12 11 10 3 13 4 9 8 7 2 1 14



LY	**RY**
5	**6**

LX=Left array sorted by X coordinate

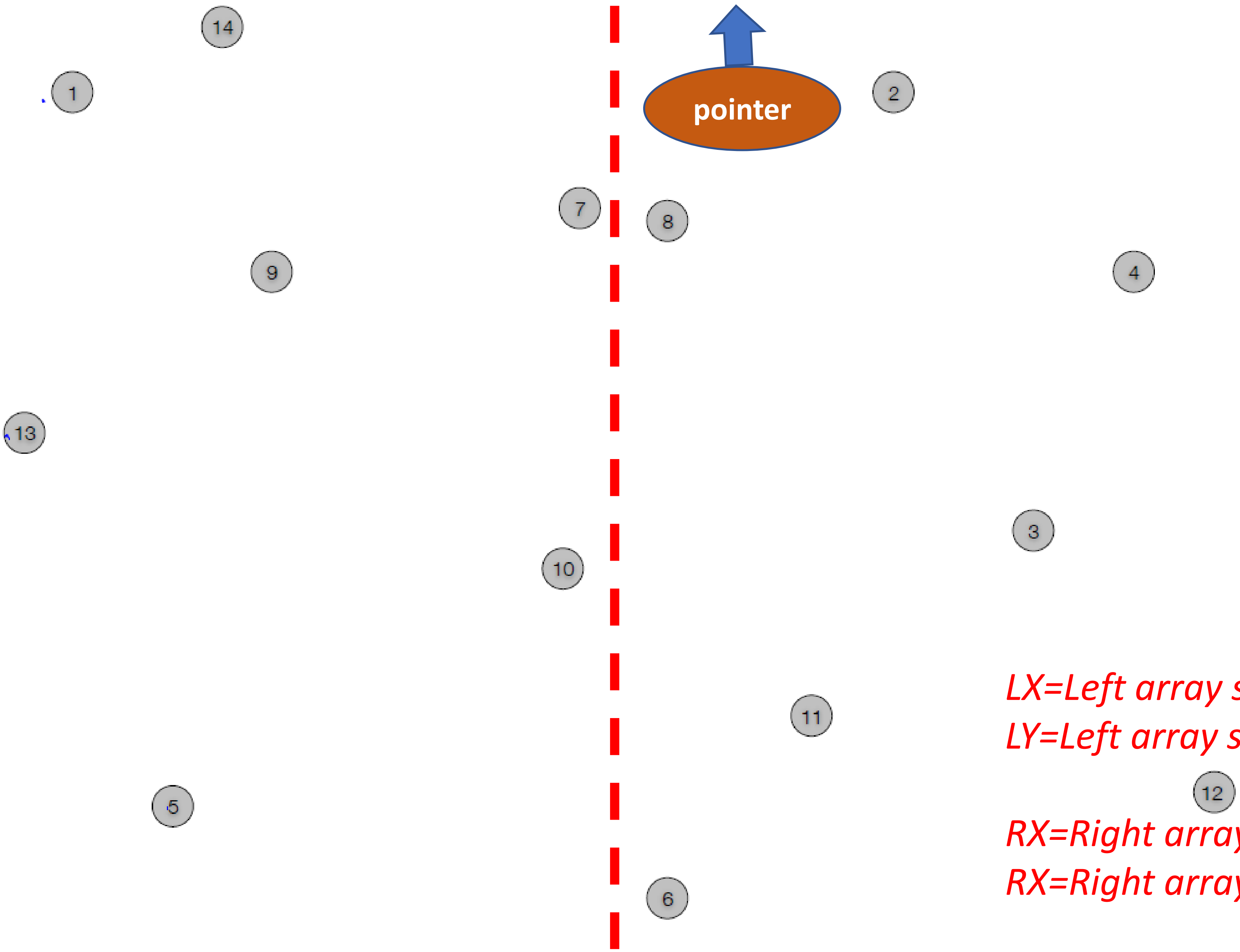
LY=Left array sorted by Y coordinate

RX=Right array sorted by X coordinate

RX=Right array sorted by Y coordinate

sorted **LX**: 13 1 5 14 9 10 7 6 8 11 2 3 4 12 **RX**

sorted in Y: 6 5 12 11 10 3 13 4 9 8 7 2 1 14



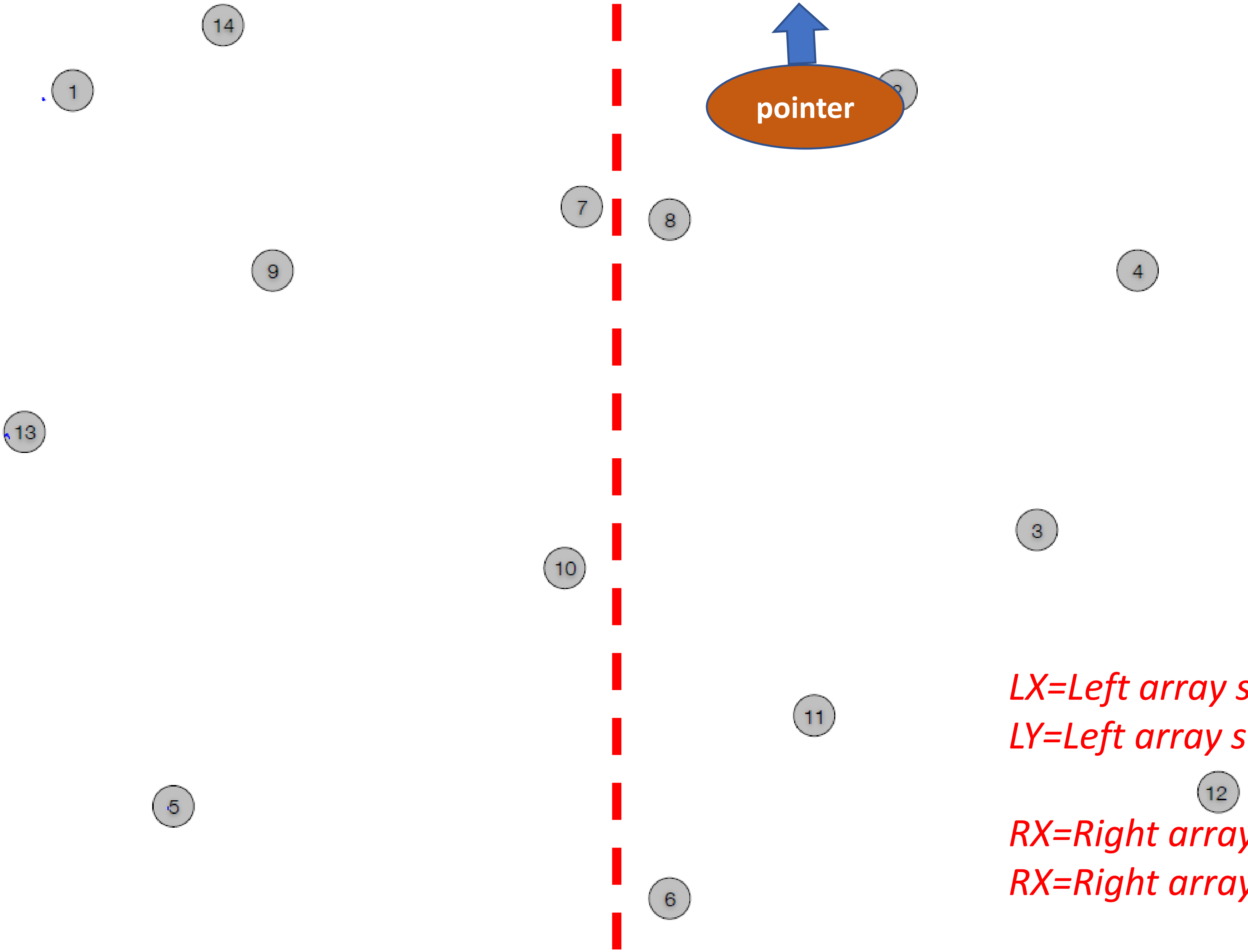
LY	RY
5	6
	12

LX=Left array sorted by X coordinate
LY=Left array sorted by Y coordinate

RX=Right array sorted by X coordinate
RX=Right array sorted by Y coordinate

sorted **LX**: 13 1 5 14 9 10 7 6 8 11 2 3 4 12 **RX**

sorted in Y: 6 5 12 11 10 3 13 4 9 8 7 2 1 14



LY	RY
5	6
	12
	11

LX=Left array sorted by X coordinate

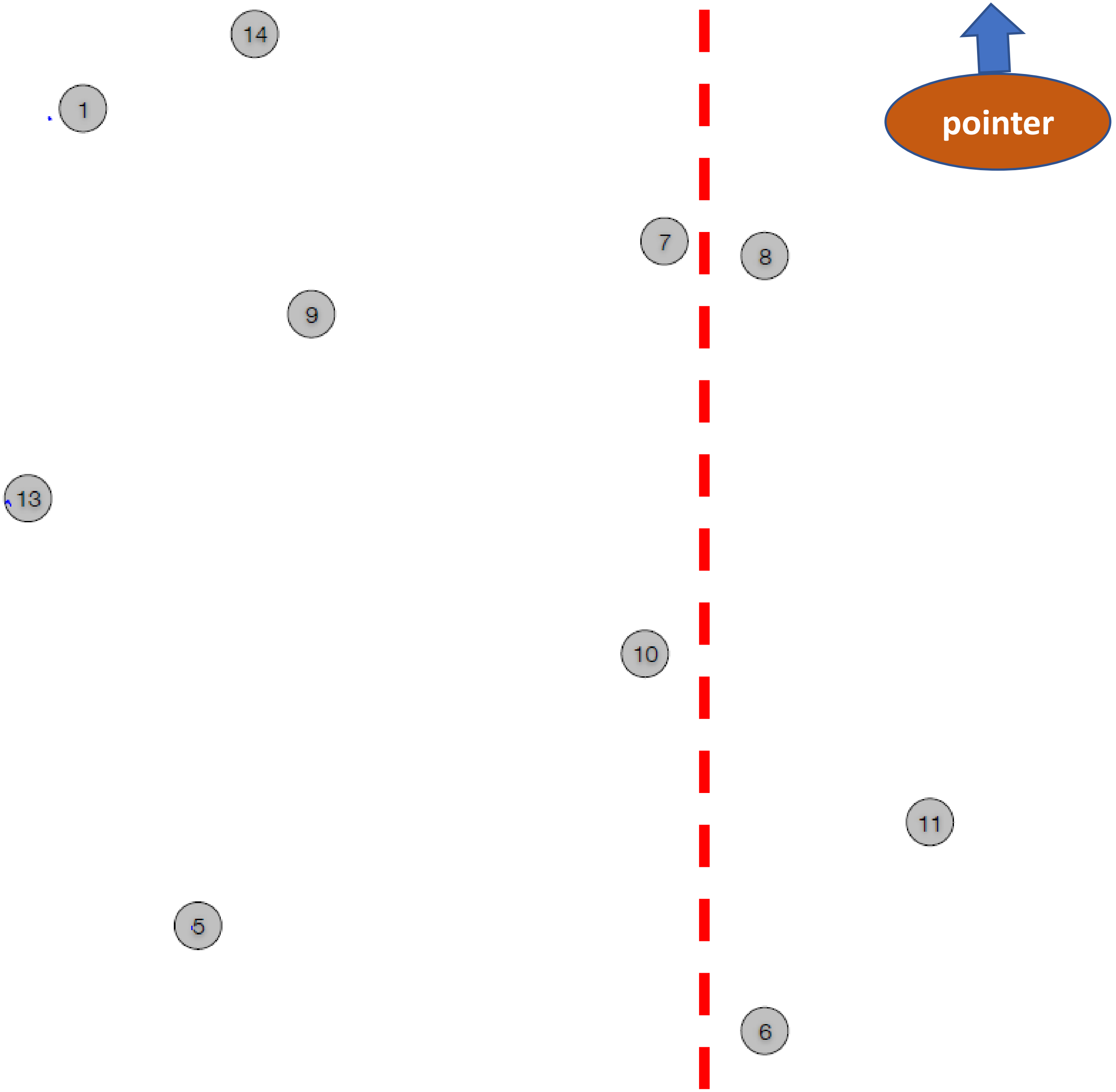
LY=Left array sorted by Y coordinate

RX=Right array sorted by X coordinate

RX=Right array sorted by Y coordinate

sorted **LX**: 13 1 5 14 9 10 7 6 8 11 2 3 4 12 **RX**

sorted in Y: 6 5 12 11 10 3 13 4 9 8 7 2 1 14



LY	**RY**
5	**6**
10	**12**
	11

LX=Left array sorted by X coordinate

LY=Left array sorted by Y coordinate

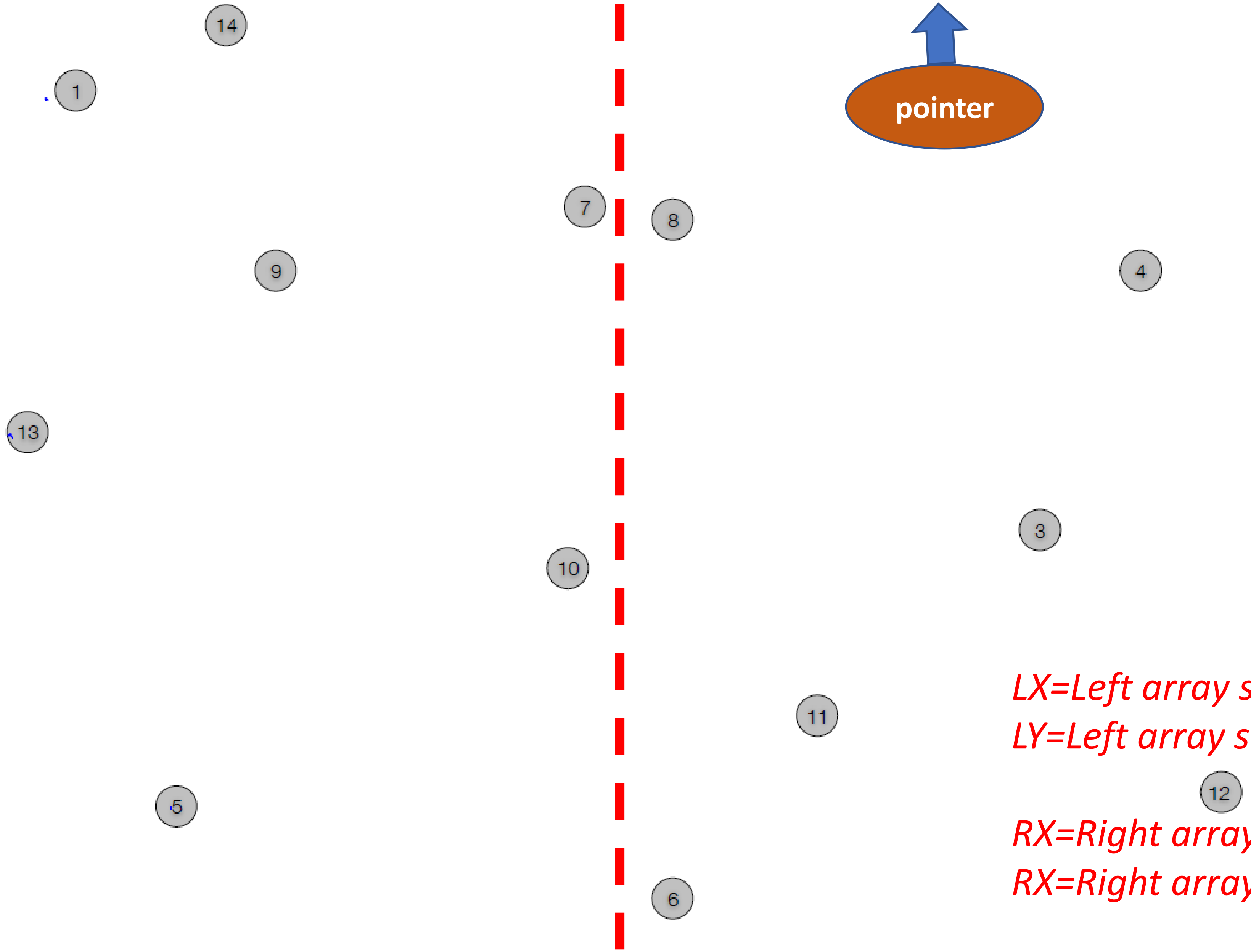
RX=Right array sorted by X coordinate

RX=Right array sorted by Y coordinate

sorted **LX**: 13 1 5 14 9 10 7 6 8 11 2 3 4 12 **RX**

sorted in Y: 6 5 12 11 10 3 13 4 9 8 7 2 1 14

pointer



LY	RY
5	6
10	12
	11
	3

LX=Left array sorted by X coordinate

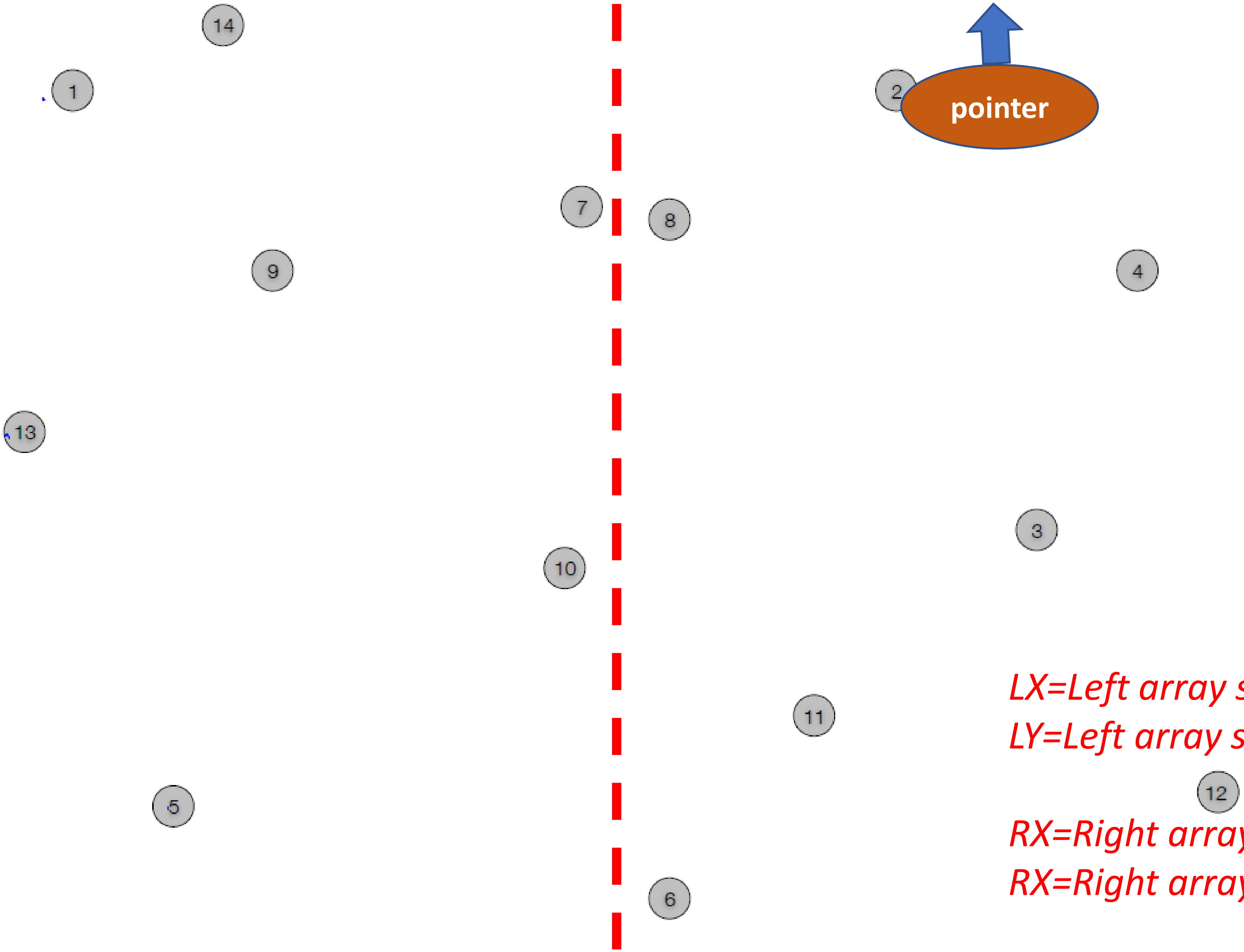
LY=Left array sorted by Y coordinate

RX=Right array sorted by X coordinate

RX=Right array sorted by Y coordinate

sorted **LX**: 13 1 5 14 9 10 7 | 6 8 11 2 3 4 12 **RX**

sorted in Y: 6 5 12 11 10 3 13 4 9 8 7 2 1 14



LY	RY
5	6
10	12
13	11
	3

LX=Left array sorted by X coordinate

LY=Left array sorted by Y coordinate

RX=Right array sorted by X coordinate

RX=Right array sorted by Y coordinate

sorted **LX**: 13 1 5 14 9 10 7 | 6 8 11 2 3 4 12 **RX**

sorted in Y: 6 5 12 11 10 3 13 4 9 8 7 2 1 14

LY	RY
5	6
10	12
13	11
	3
	4

LX=Left array sorted by X coordinate

LY=Left array sorted by Y coordinate

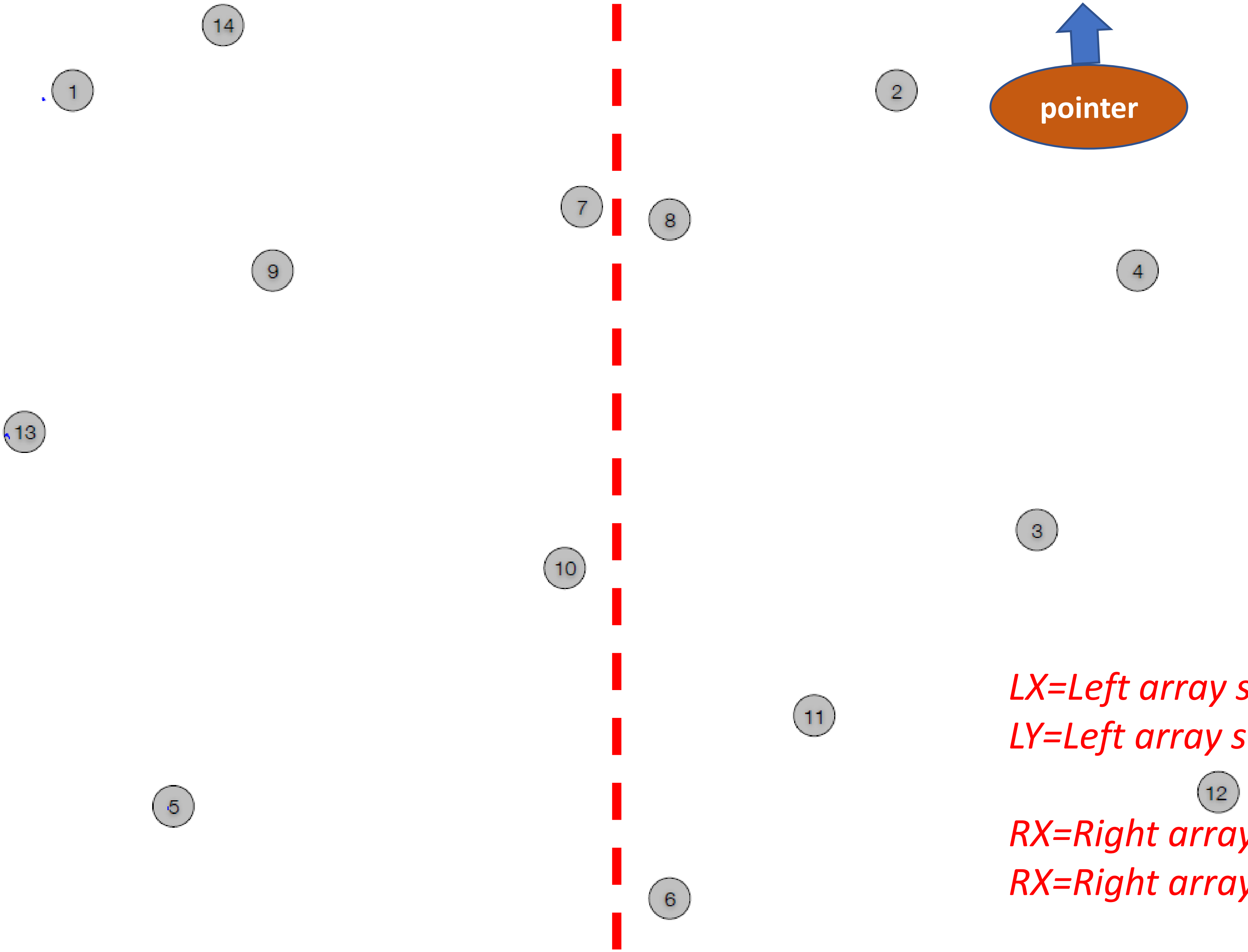
RX=Right array sorted by X coordinate

RX=Right array sorted by Y coordinate

sorted **LX**: 13 1 5 14 9 10 7 6 8 11 2 3 4 12 **RX**

sorted in Y: 6 5 12 11 10 3 13 4 9 8 7 2 1 14

pointer



LY	RY
5	6
10	12
13	11
9	3
	4

LX=Left array sorted by X coordinate

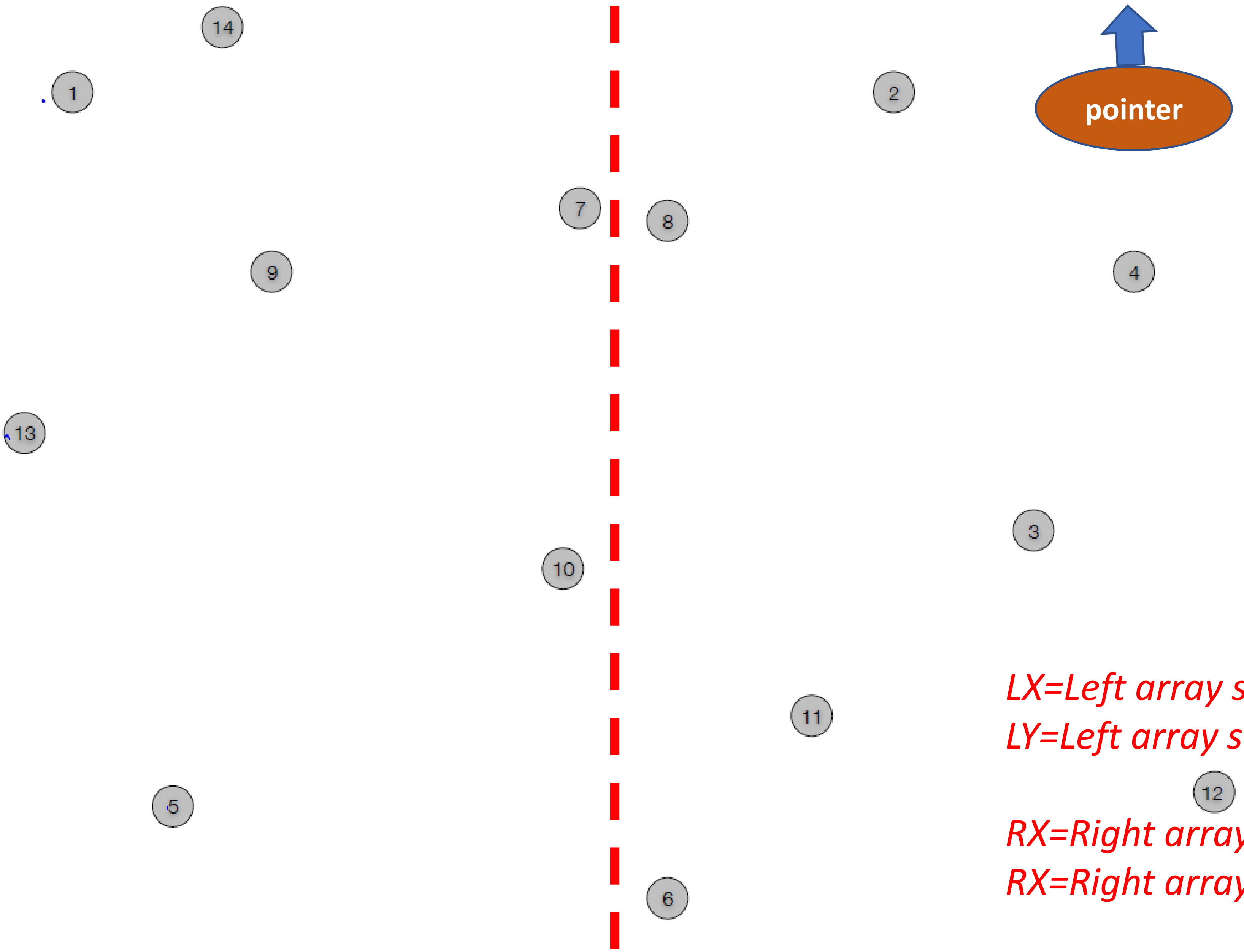
LY=Left array sorted by Y coordinate

RX=Right array sorted by X coordinate

RX=Right array sorted by Y coordinate

sorted **LX**: 13 1 5 14 9 10 7 6 8 11 2 3 4 12 **RX**

sorted in Y: 6 5 12 11 10 3 13 4 9 8 7 2 1 14



LY	RY
5	6
10	12
13	11
9	3
	4
	8

LX=Left array sorted by X coordinate

LY=Left array sorted by Y coordinate

RX=Right array sorted by X coordinate

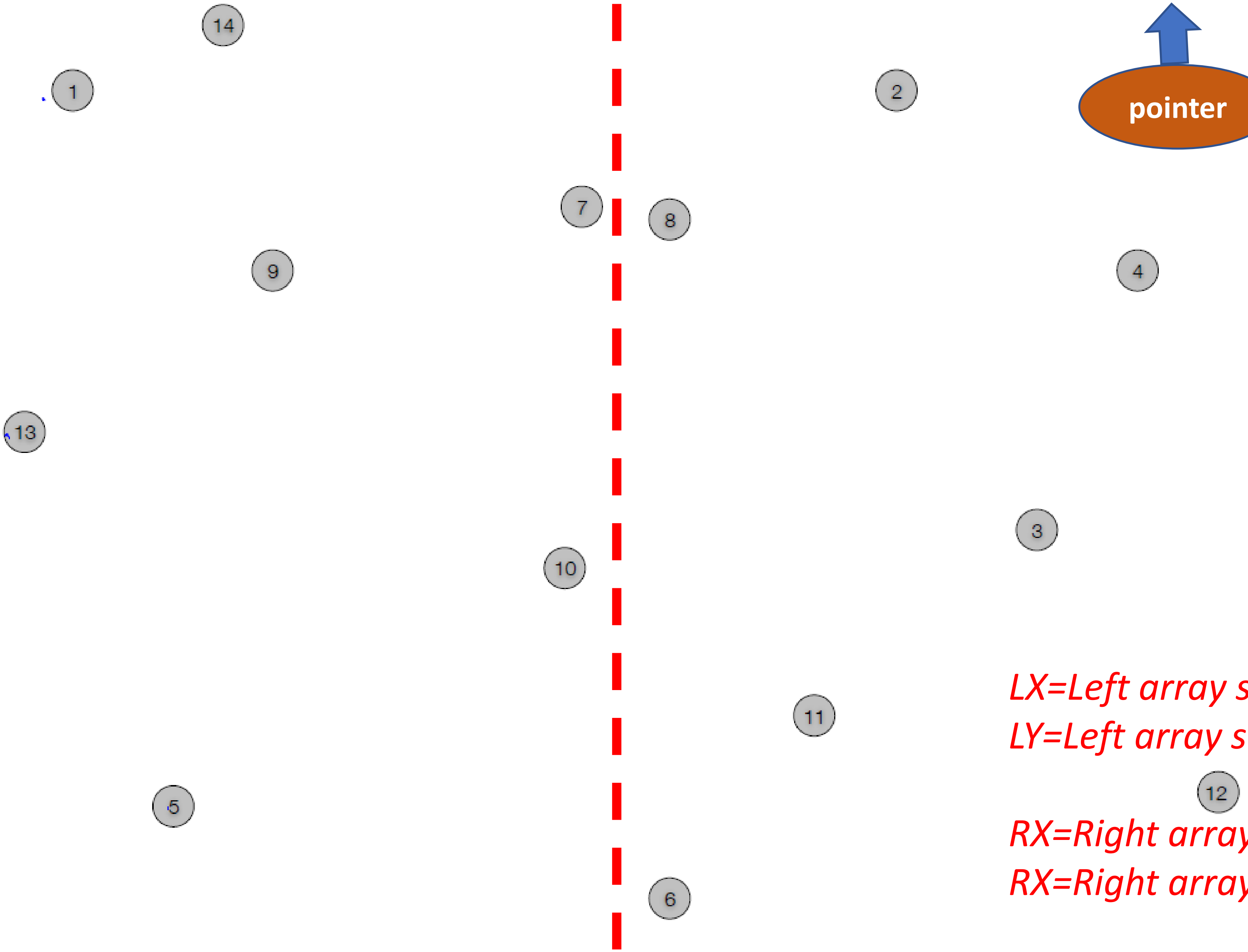
RX=Right array sorted by Y coordinate

sorted **LX**: 13 1 5 14 9 10 7 6 8 11 2 3 4 12 **RX**

sorted in Y: 6 5 12 11 10 3 13 4 9 8 7 2 1 14

pointer

LY	RY
5	6
10	12
13	11
9	3
7	4
	8



LX=Left array sorted by X coordinate

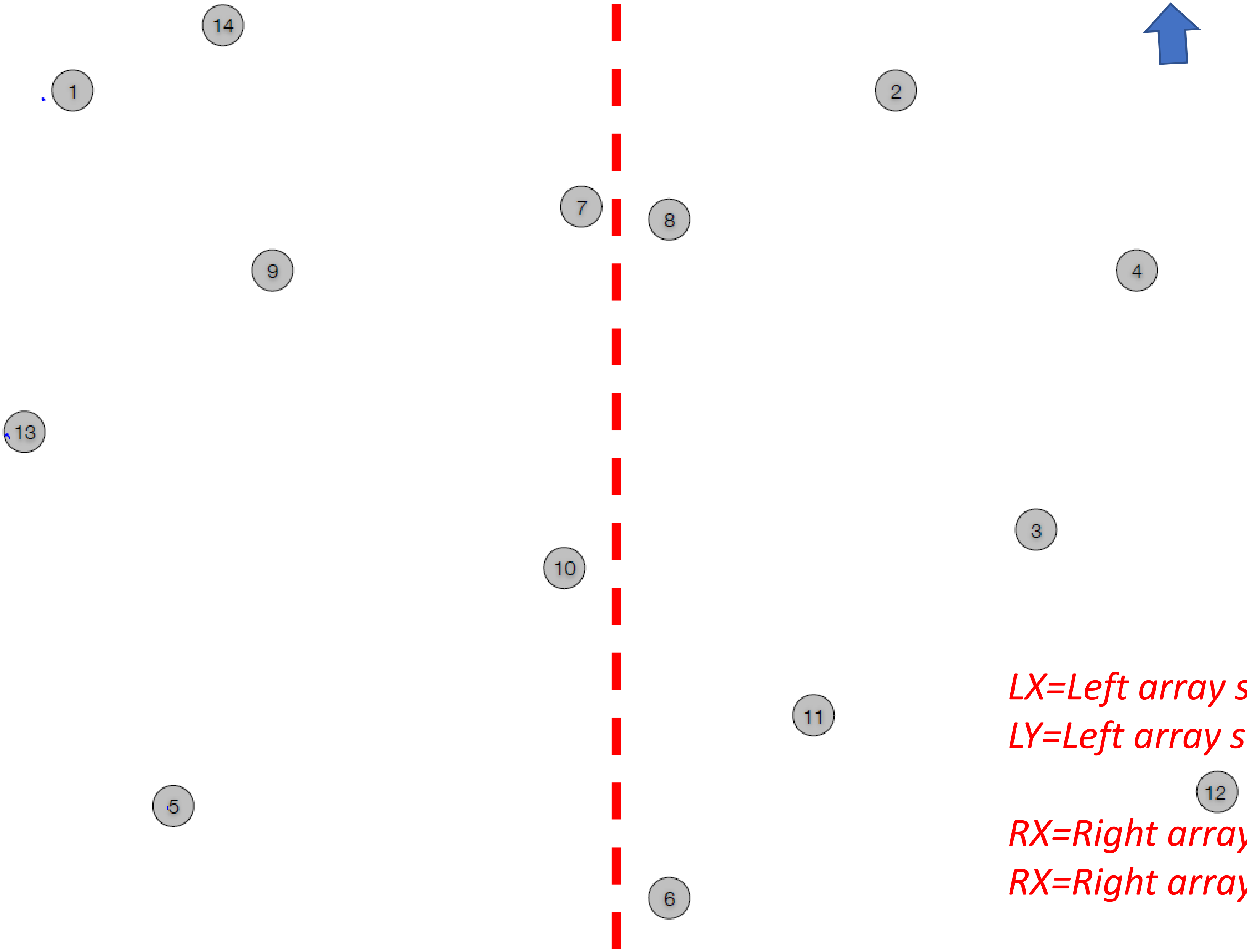
LY=Left array sorted by Y coordinate

RX=Right array sorted by X coordinate

RX=Right array sorted by Y coordinate

sorted **LX**: 13 1 5 14 9 10 7 | 6 8 11 2 3 4 12 **RX**

sorted in Y: 6 5 12 11 10 3 13 4 9 8 7 2 1 14



LY	RY
5	6
10	12
13	11
9	3
7	4
1	8
14	2

LX=Left array sorted by X coordinate
LY=Left array sorted by Y coordinate

RX=Right array sorted by X coordinate
RX=Right array sorted by Y coordinate

ClosestPair(P) :

Compute X coordinate sorted list SX → Θ(nlogn)

Compute X coordinate sorted list SY → Θ(nlogn)

Closest(P,SX,SY) → Θ(nlogn)

Closest(P,SX,SY) :

1. Base case: if |P|<=2, brute force.
2. Let q be the mid-element of SX
3. Divide P into left, right according to q

4. δ = Min(Closest(left,LX,LY), Closest(left,RX,RY))

5. Mohawk= {scan SY, add points that are in delta (δ) distance from q.x } → Advantage? Getting them in sorted order of Y coordinate

6. For each point x in Mohawk (in y-coordinate):
7. Compute distance to it's next 11 neighbors in higher Y-coordinate
8. Update δ if any pair of points have distance < δ
9. Return δ

LX=Left array sorted by X coordinate

LY=Left array sorted by Y coordinate

RX=Right array sorted by X coordinate

RX=Right array sorted by Y coordinate

ClosestPair(P) :

Compute X coordinate sorted list SX → $\Theta(n\log n)$
Compute X coordinate sorted list SY → $\Theta(n\log n)$
Closest(P,SX,SY) → $\Theta(n\log n)$

Closest(P,SX,SY) :

1. Base case: if |P|<=2, brute force.
2. Let q be the mid-element of SX
3. Divide P into left, right according to q
4. δ = Min(Closest(left,LX,LY), Closest(left,RX,RY))
5. Mohawk= {scan SY, add points that are in delta (δ) distance from q.x } → Advantage? Getting them in sorted order of Y coordinate
6. For each point x in Mohawk (in y-coordinate):
7. Compute distance to it's next 11 neighbors in higher Y-coordinate
8. Update δ if any pair of points have distance < δ
9. Return δ

LX=Left array sorted by X coordinate
LY=Left array sorted by Y coordinate

RX=Right array sorted by X coordinate
RX=Right array sorted by Y coordinate

Running time for closest pair algorithm
$T(n)=2T(n/2)+\theta(n) = \theta(n\log n)$