

Counting Inversions


Music site tries to match your song preferences with others.

- You rank n songs.
- Music site consults database to find people with **similar** tastes.

Similarity metric: number of inversions between two rankings.

- My rank: $1, 2, \dots, n$.
- Your rank: a_1, a_2, \dots, a_n .
- Songs i and j **inverted** if $i < j$, but $a_i > a_j$.

<i>Songs</i>					
	A	B	C	D	E
Me	1	2	3	4	5
You	1	3	4	2	5



Inversions

3-2, 4-2

Brute force: check all $\Theta(n^2)$ pairs i and j .

Applications

Applications.

- Voting theory.
- Collaborative filtering.
- Measuring the "sortedness" of an array.
- Sensitivity analysis of Google's ranking function.
- Rank aggregation for meta-searching on the Web.
- Nonparametric statistics (e.g., Kendall's Tau distance).

Counting Inversions: Divide-and-Conquer

Divide-and-conquer.

1	5	4	8	10	2	6	9	12	11	3	7
---	---	---	---	----	---	---	---	----	----	---	---

Counting Inversions: Divide-and-Conquer

Divide-and-conquer.

- **Divide**: separate list into two pieces.

1	5	4	8	10	2	6	9	12	11	3	7
---	---	---	---	----	---	---	---	----	----	---	---

Divide: $O(1)$.

1	5	4	8	10	2	6	9	12	11	3	7
---	---	---	---	----	---	---	---	----	----	---	---

Counting Inversions: Divide-and-Conquer

Divide-and-conquer.

- Divide: separate list into two pieces.
- **Conquer**: recursively count inversions in each half.

1	5	4	8	10	2	6	9	12	11	3	7
---	---	---	---	----	---	---	---	----	----	---	---

Divide: $O(1)$.

1	5	4	8	10	2	6	9	12	11	3	7
---	---	---	---	----	---	---	---	----	----	---	---

Conquer: $2T(n / 2)$

5 blue-blue inversions

8 green-green inversions

5-4, 5-2, 4-2, 8-2, 10-2

6-3, 9-3, 9-7, 12-3, 12-7, 12-11, 11-3, 11-7

Counting Inversions: Divide-and-Conquer

Divide-and-conquer.

- Divide: separate list into two pieces.
- Conquer: recursively count inversions in each half.
- **Combine**: count inversions where a_i and a_j are in different halves, and return sum of three quantities.

1	5	4	8	10	2	6	9	12	11	3	7
---	---	---	---	----	---	---	---	----	----	---	---

Divide: $O(1)$.

1	5	4	8	10	2	6	9	12	11	3	7
---	---	---	---	----	---	---	---	----	----	---	---

5 blue-blue inversions

8 green-green inversions

Conquer: $2T(n / 2)$

9 blue-green inversions

5-3, 4-3, 8-6, 8-3, 8-7, 10-6, 10-9, 10-3, 10-7

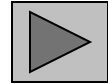
Combine: ???

Total = $5 + 8 + 9 = 22$.

Counting Inversions: Combine

Combine: count blue-green inversions

- Assume each half is **sorted**.
- Count inversions where a_i and a_j are in different halves.
- **Merge** two sorted halves into sorted whole.



to maintain sorted invariant

3	7	10	14	18	19
---	---	----	----	----	----

2	11	16	17	23	25
6	3	2	2	0	0

13 blue-green inversions: $6 + 3 + 2 + 2 + 0 + 0$

Count: $O(n)$

2	3	7	10	11	14	16	17	18	19	23	25
---	---	---	----	----	----	----	----	----	----	----	----

Merge: $O(n)$

$$T(n) \leq T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + O(n) \Rightarrow T(n) = O(n \log n)$$

Merge and Count

Merge and count step.

- Given two sorted halves, count number of inversions where a_i and a_j are in different halves.
- Combine two sorted halves into sorted whole.

$i = 6$



3	7	10	14	18	19
---	---	----	----	----	----



2	11	16	17	23	25
---	----	----	----	----	----

two sorted halves

--	--	--	--	--	--	--	--	--	--	--	--

auxiliary array

Total:

Merge and Count

Merge and count step.

- Given two sorted halves, count number of inversions where a_i and a_j are in different halves.
- Combine two sorted halves into sorted whole.

$i = 6$



3	7	10	14	18	19
---	---	----	----	----	----



2	11	16	17	23	25
---	----	----	----	----	----

two sorted halves

6

2											
---	--	--	--	--	--	--	--	--	--	--	--

auxiliary array

Total: 6

Merge and Count

Merge and count step.

- Given two sorted halves, count number of inversions where a_i and a_j are in different halves.
- Combine two sorted halves into sorted whole.

$i = 6$



3	7	10	14	18	19
---	---	----	----	----	----



2	11	16	17	23	25
---	----	----	----	----	----

two sorted halves

6

2											
---	--	--	--	--	--	--	--	--	--	--	--

auxiliary array

Total: 6

Merge and Count

Merge and count step.

- Given two sorted halves, count number of inversions where a_i and a_j are in different halves.
- Combine two sorted halves into sorted whole.

$i = 6$



two sorted halves

6



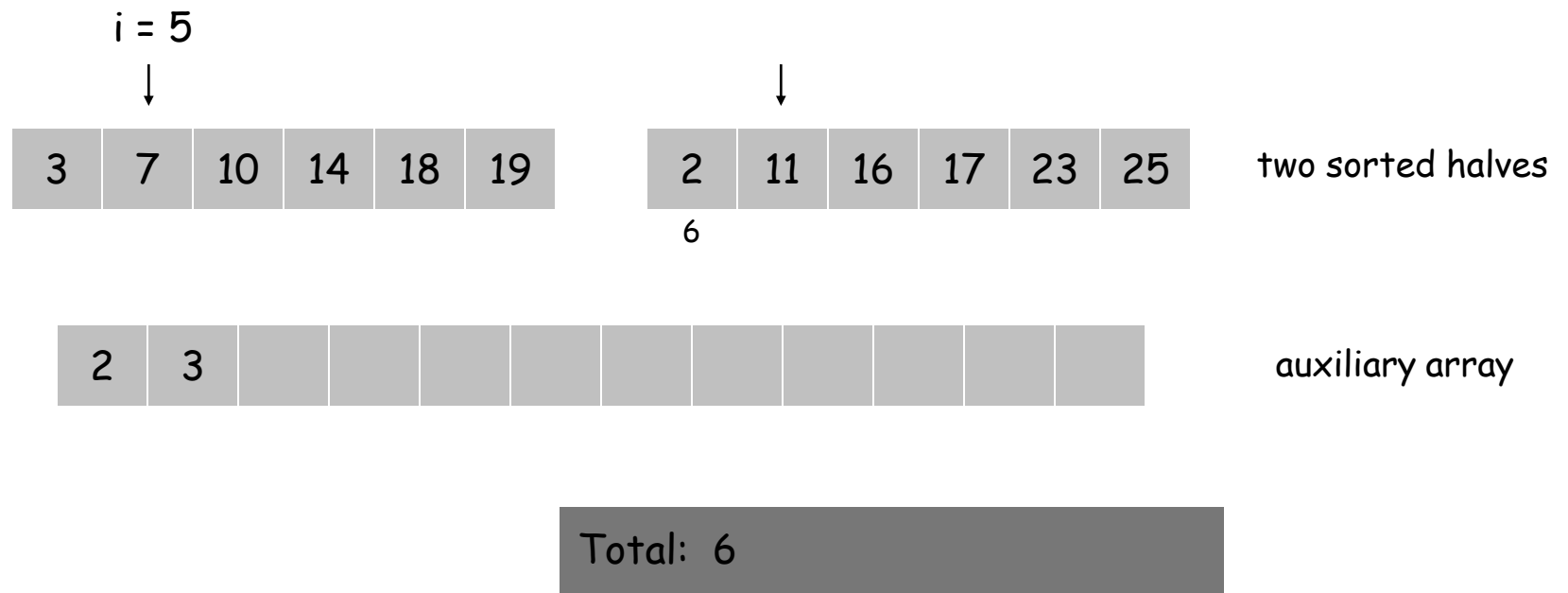
auxiliary array

Total: 6

Merge and Count

Merge and count step.

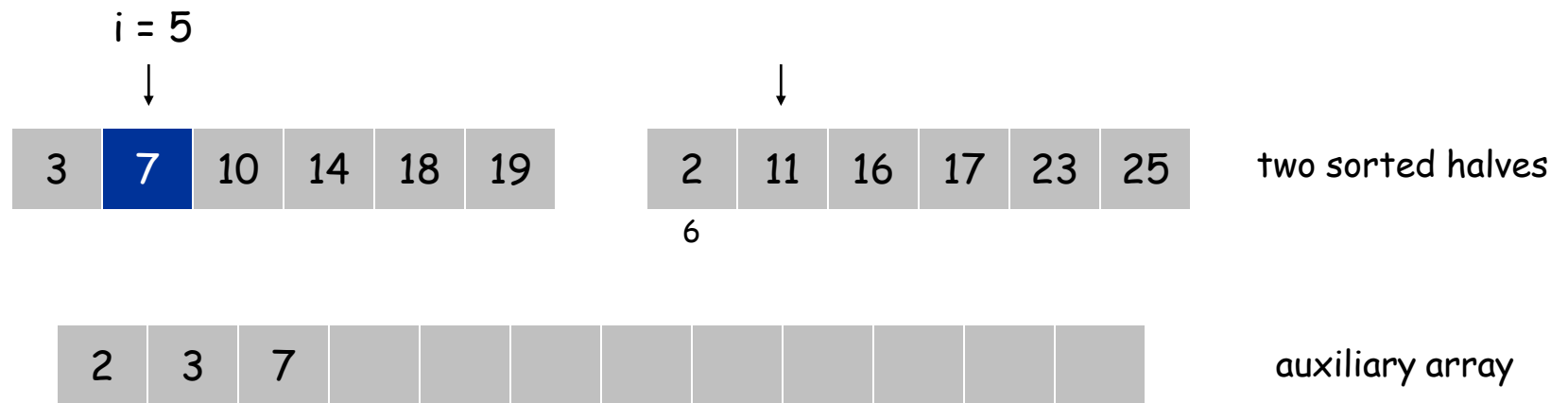
- Given two sorted halves, count number of inversions where a_i and a_j are in different halves.
- Combine two sorted halves into sorted whole.



Merge and Count

Merge and count step.

- Given two sorted halves, count number of inversions where a_i and a_j are in different halves.
- Combine two sorted halves into sorted whole.

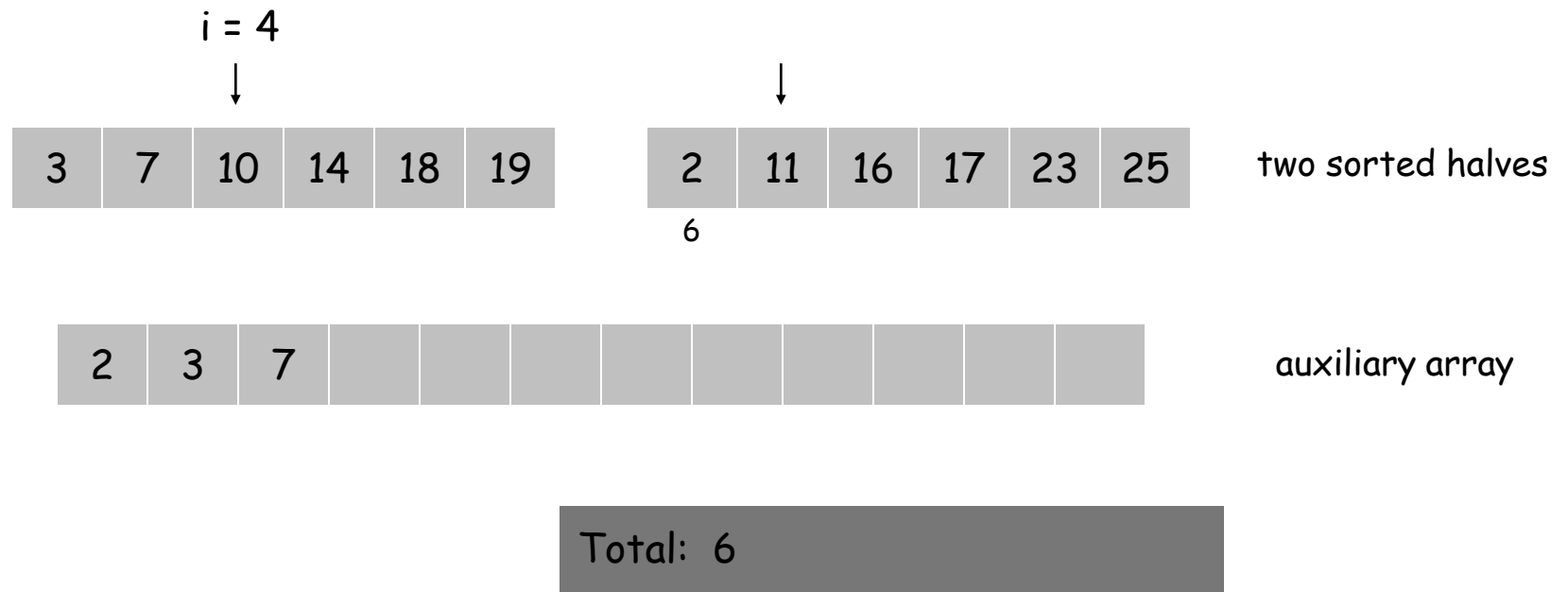


Total: 6

Merge and Count

Merge and count step.

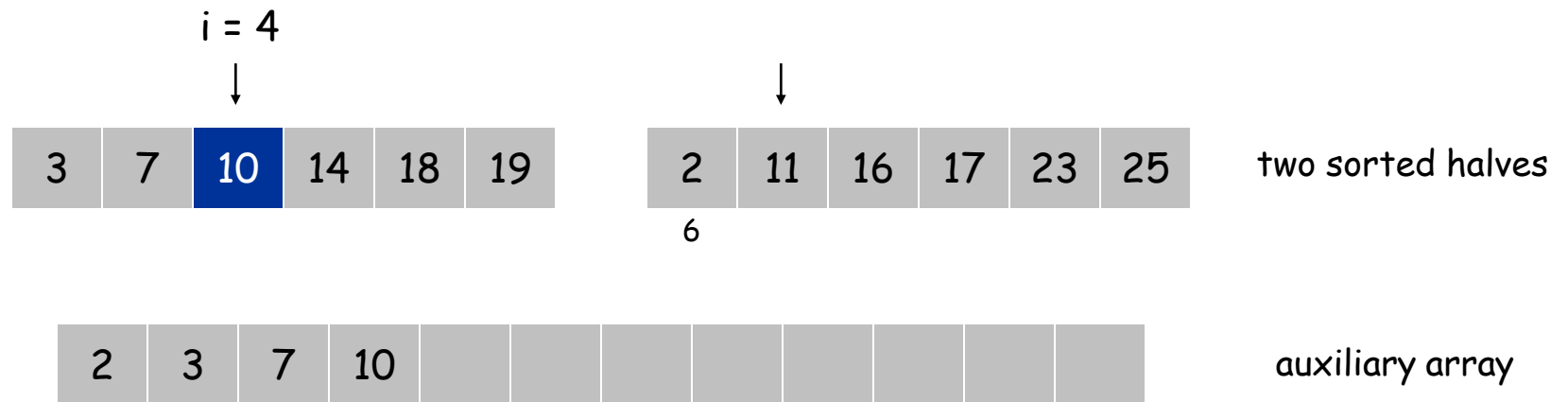
- Given two sorted halves, count number of inversions where a_i and a_j are in different halves.
- Combine two sorted halves into sorted whole.



Merge and Count

Merge and count step.

- Given two sorted halves, count number of inversions where a_i and a_j are in different halves.
- Combine two sorted halves into sorted whole.

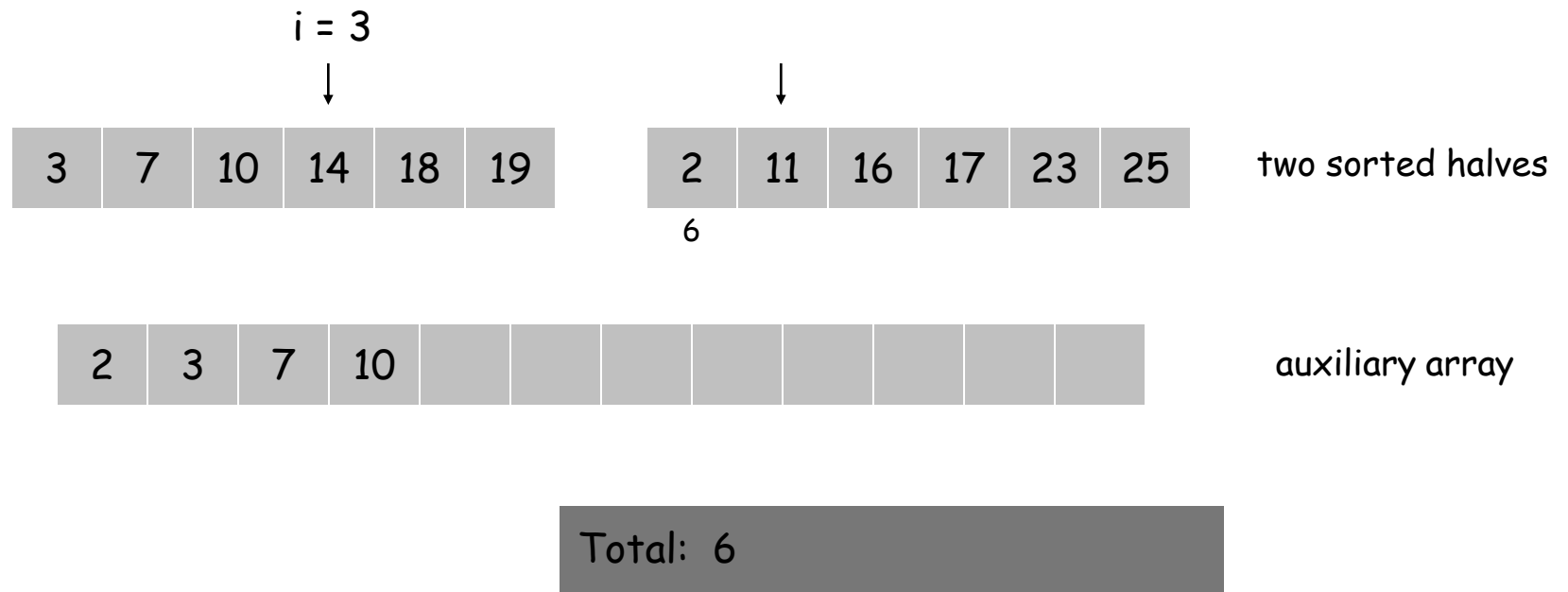


Total: 6

Merge and Count

Merge and count step.

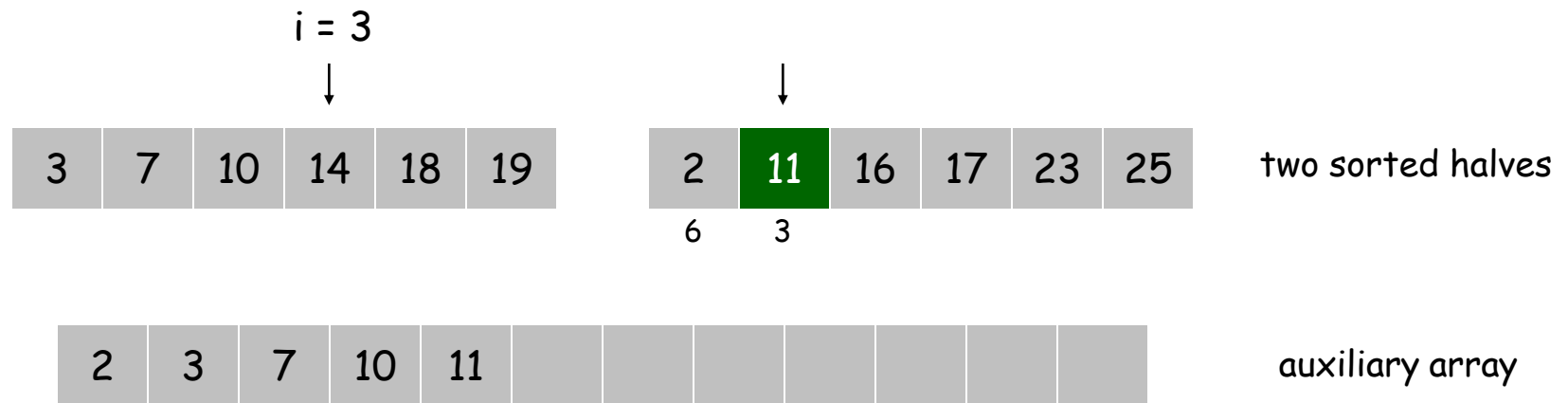
- Given two sorted halves, count number of inversions where a_i and a_j are in different halves.
- Combine two sorted halves into sorted whole.



Merge and Count

Merge and count step.

- Given two sorted halves, count number of inversions where a_i and a_j are in different halves.
- Combine two sorted halves into sorted whole.

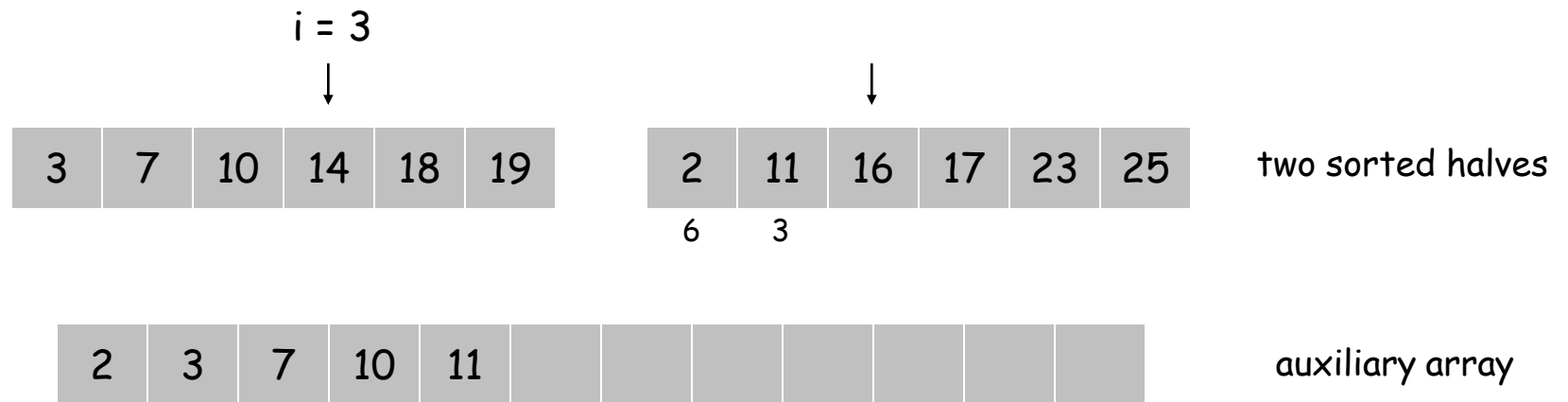


Total: 6 + 3

Merge and Count

Merge and count step.

- Given two sorted halves, count number of inversions where a_i and a_j are in different halves.
- Combine two sorted halves into sorted whole.

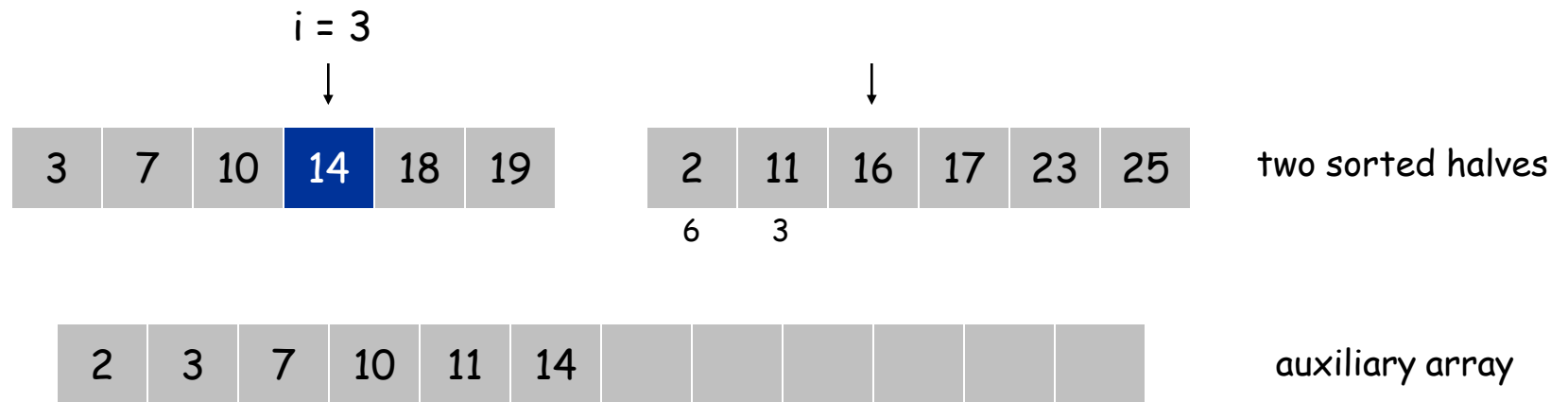


Total: 6 + 3

Merge and Count

Merge and count step.

- Given two sorted halves, count number of inversions where a_i and a_j are in different halves.
- Combine two sorted halves into sorted whole.

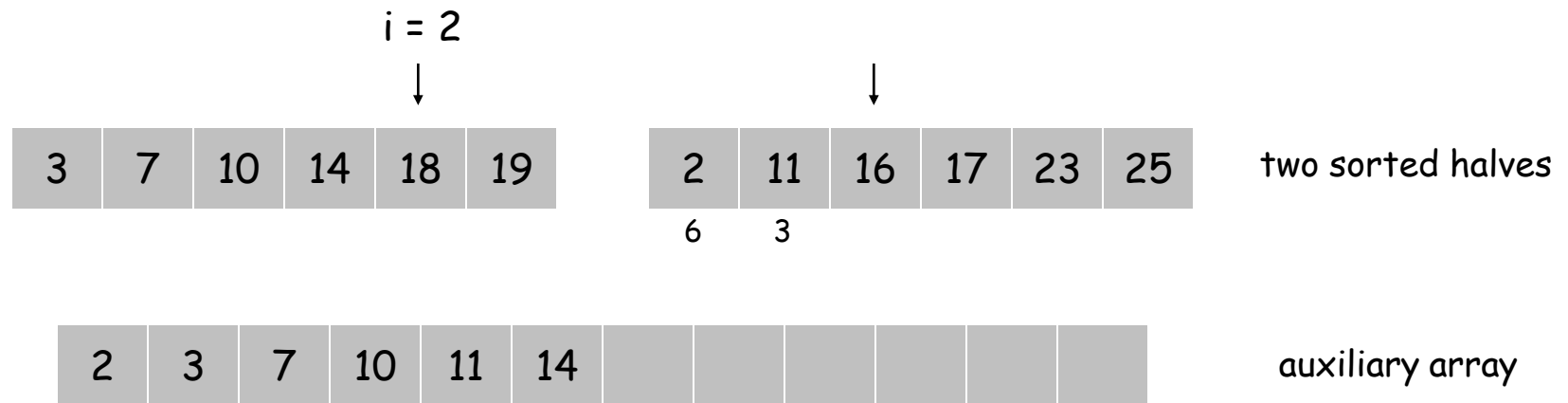


Total: 6 + 3

Merge and Count

Merge and count step.

- Given two sorted halves, count number of inversions where a_i and a_j are in different halves.
- Combine two sorted halves into sorted whole.

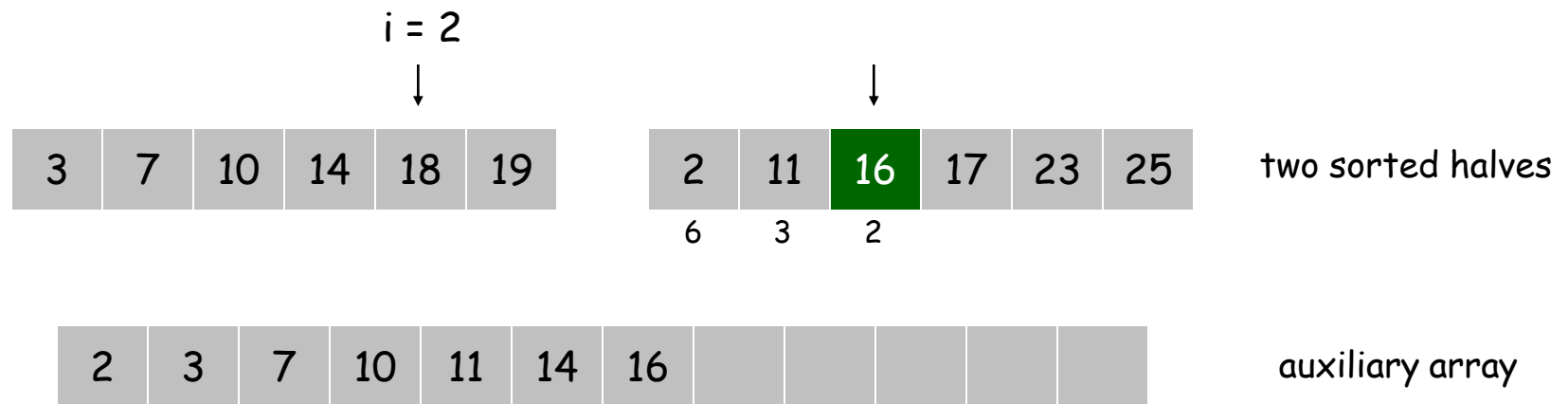


Total: 6 + 3

Merge and Count

Merge and count step.

- Given two sorted halves, count number of inversions where a_i and a_j are in different halves.
- Combine two sorted halves into sorted whole.

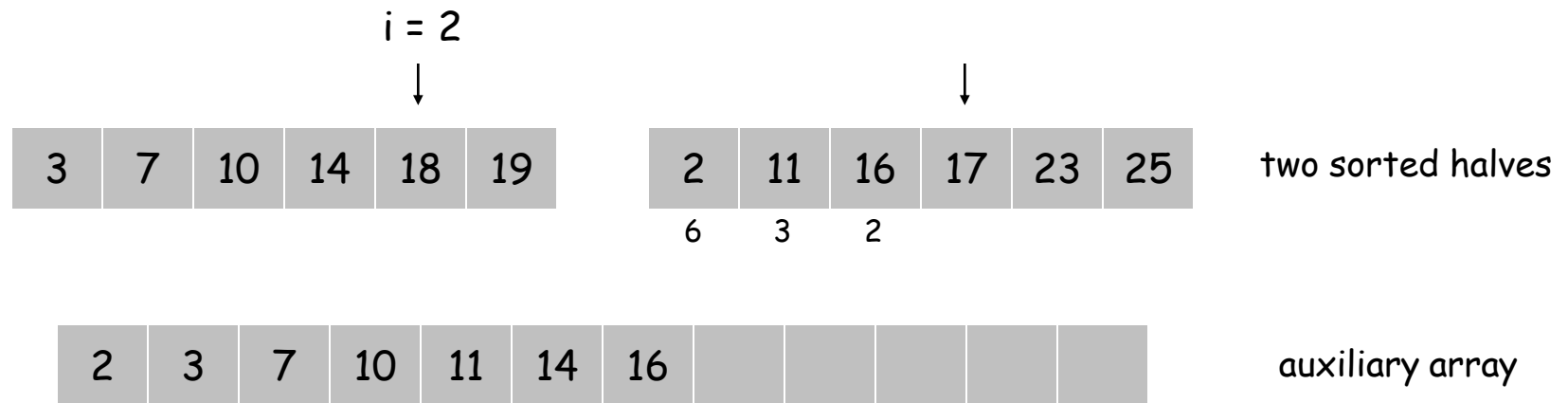


Total: $6 + 3 + 2$

Merge and Count

Merge and count step.

- Given two sorted halves, count number of inversions where a_i and a_j are in different halves.
- Combine two sorted halves into sorted whole.

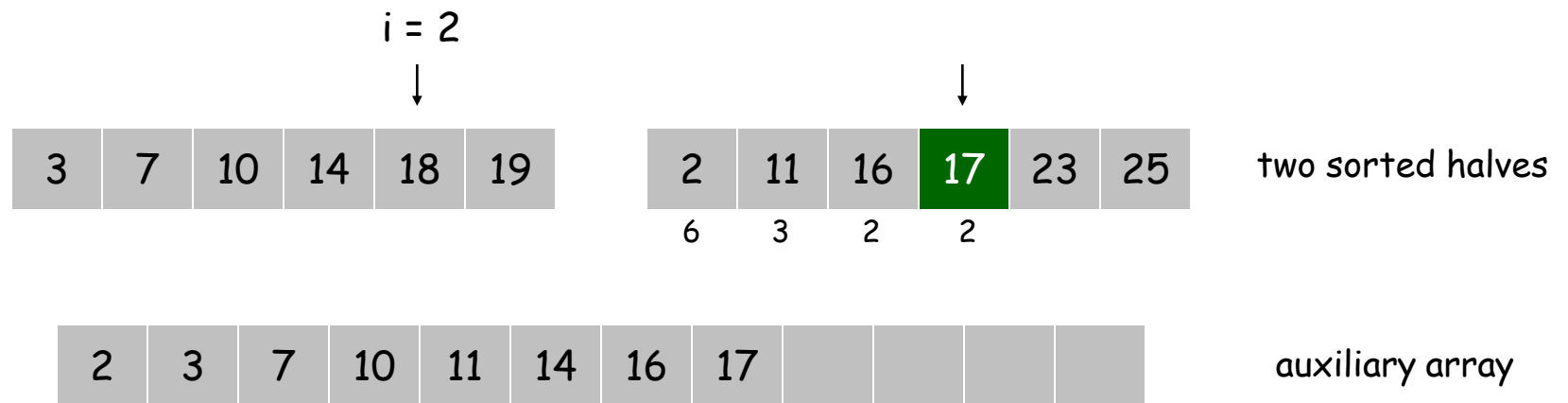


Total: $6 + 3 + 2$

Merge and Count

Merge and count step.

- Given two sorted halves, count number of inversions where a_i and a_j are in different halves.
- Combine two sorted halves into sorted whole.

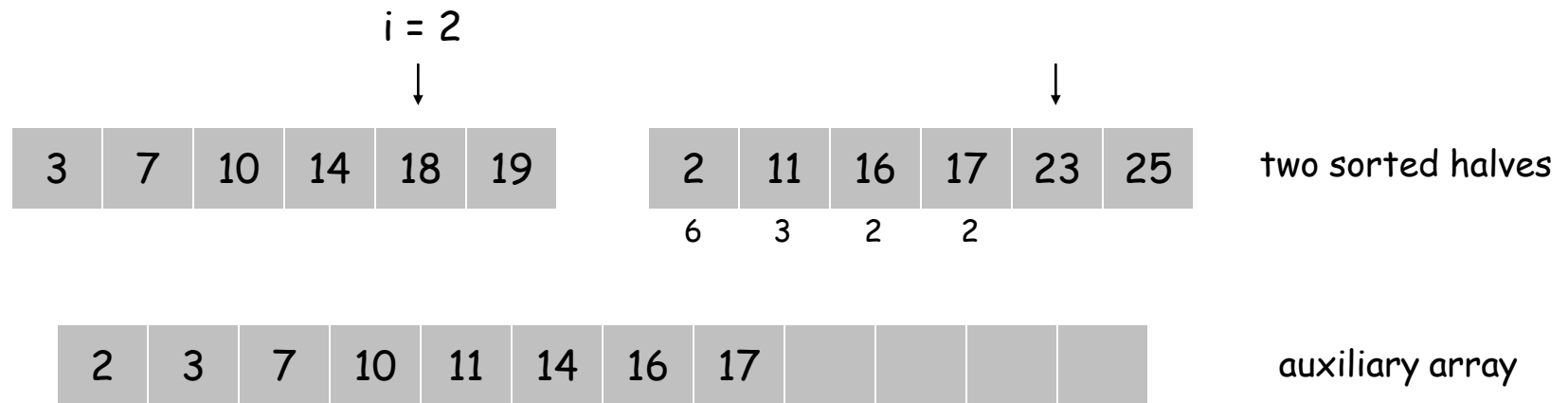


Total: $6 + 3 + 2 + 2$

Merge and Count

Merge and count step.

- Given two sorted halves, count number of inversions where a_i and a_j are in different halves.
- Combine two sorted halves into sorted whole.

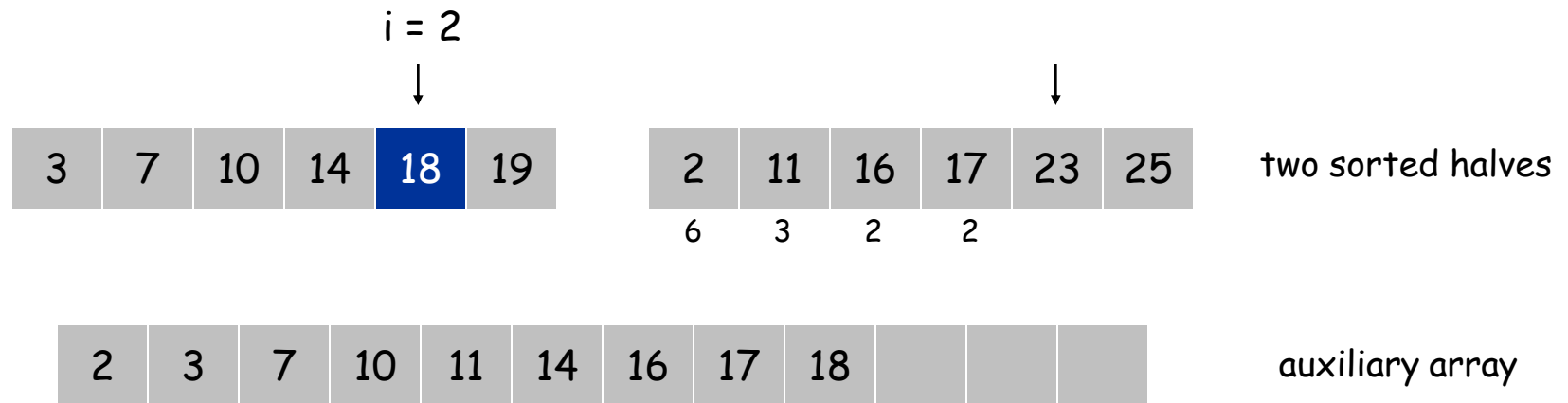


Total: $6 + 3 + 2 + 2$

Merge and Count

Merge and count step.

- Given two sorted halves, count number of inversions where a_i and a_j are in different halves.
- Combine two sorted halves into sorted whole.

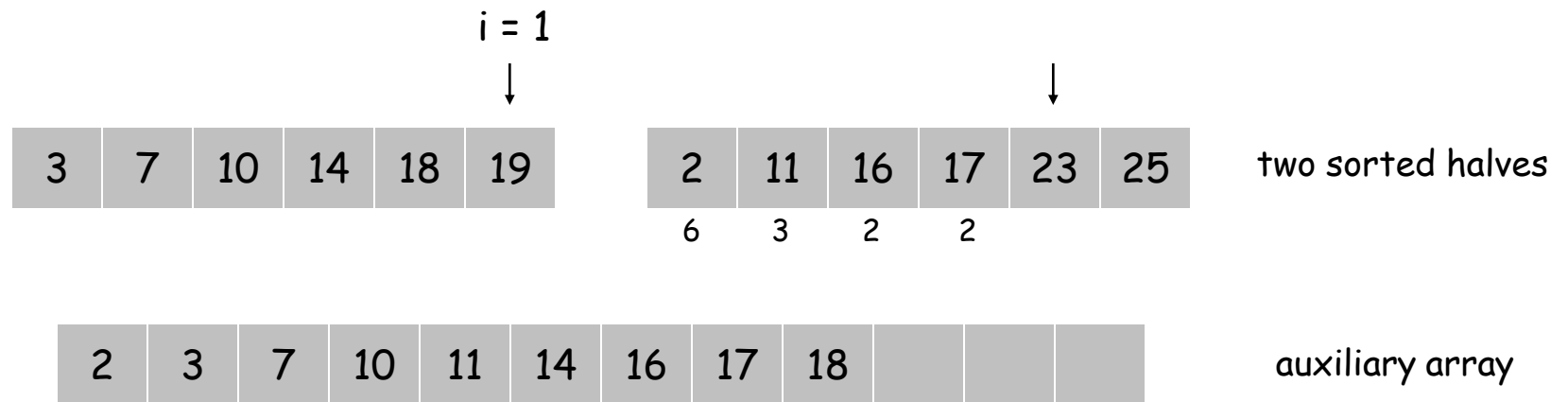


Total: $6 + 3 + 2 + 2$

Merge and Count

Merge and count step.

- Given two sorted halves, count number of inversions where a_i and a_j are in different halves.
- Combine two sorted halves into sorted whole.

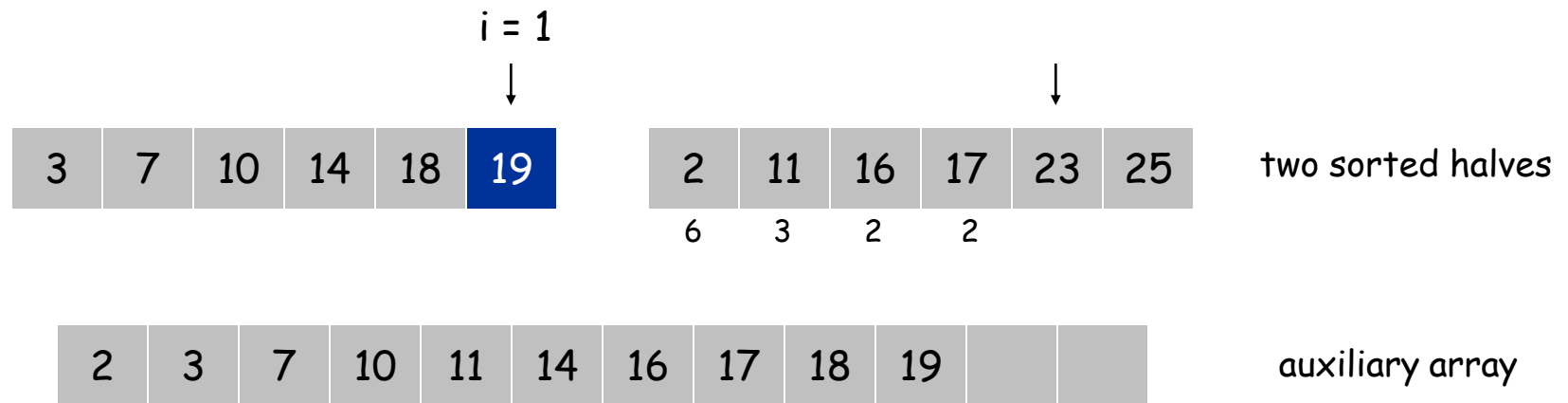


Total: $6 + 3 + 2 + 2$

Merge and Count

Merge and count step.

- Given two sorted halves, count number of inversions where a_i and a_j are in different halves.
- Combine two sorted halves into sorted whole.

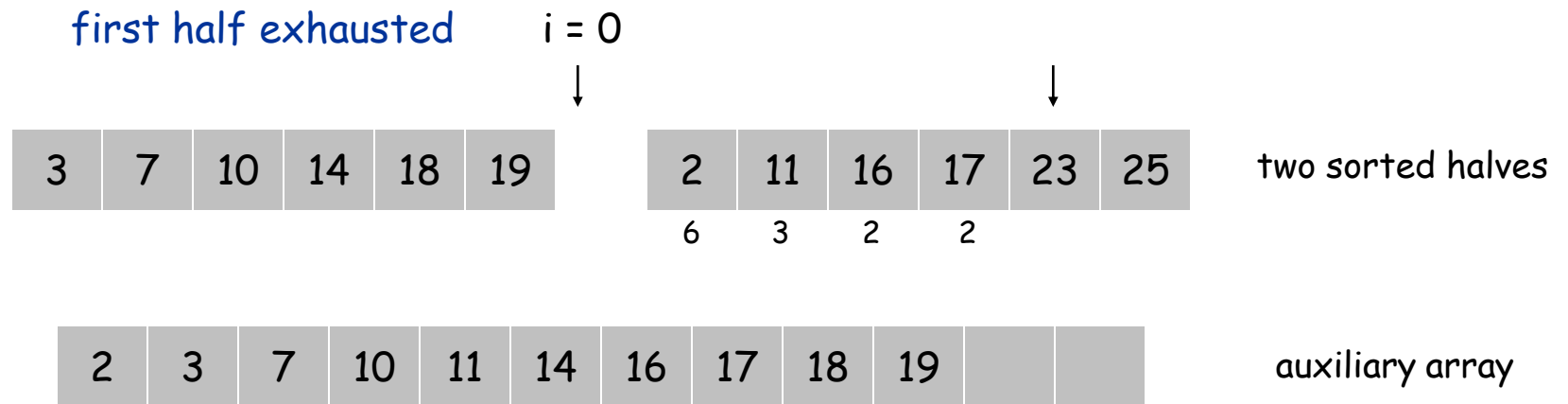


Total: $6 + 3 + 2 + 2$

Merge and Count

Merge and count step.

- Given two sorted halves, count number of inversions where a_i and a_j are in different halves.
- Combine two sorted halves into sorted whole.

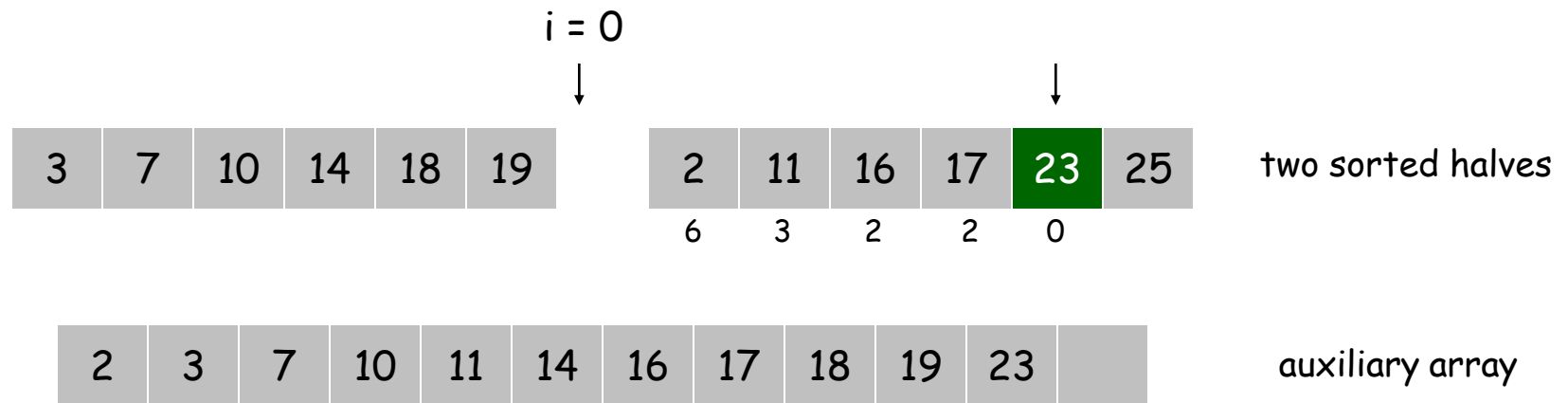


Total: $6 + 3 + 2 + 2$

Merge and Count

Merge and count step.

- Given two sorted halves, count number of inversions where a_i and a_j are in different halves.
- Combine two sorted halves into sorted whole.

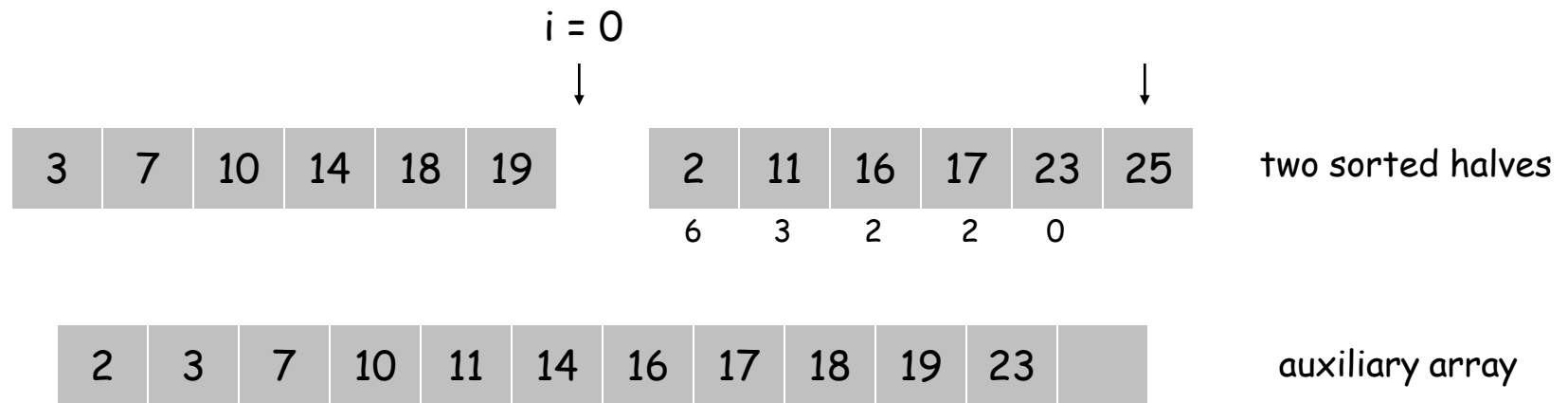


Total: $6 + 3 + 2 + 2 + 0$

Merge and Count

Merge and count step.

- Given two sorted halves, count number of inversions where a_i and a_j are in different halves.
- Combine two sorted halves into sorted whole.

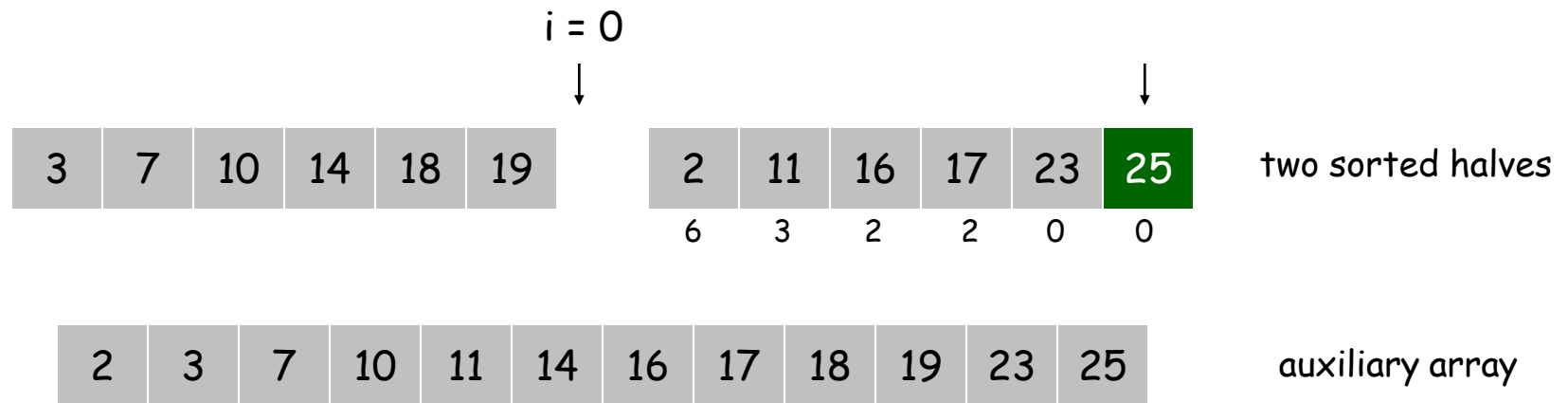


Total: $6 + 3 + 2 + 2 + 0$

Merge and Count

Merge and count step.

- Given two sorted halves, count number of inversions where a_i and a_j are in different halves.
- Combine two sorted halves into sorted whole.

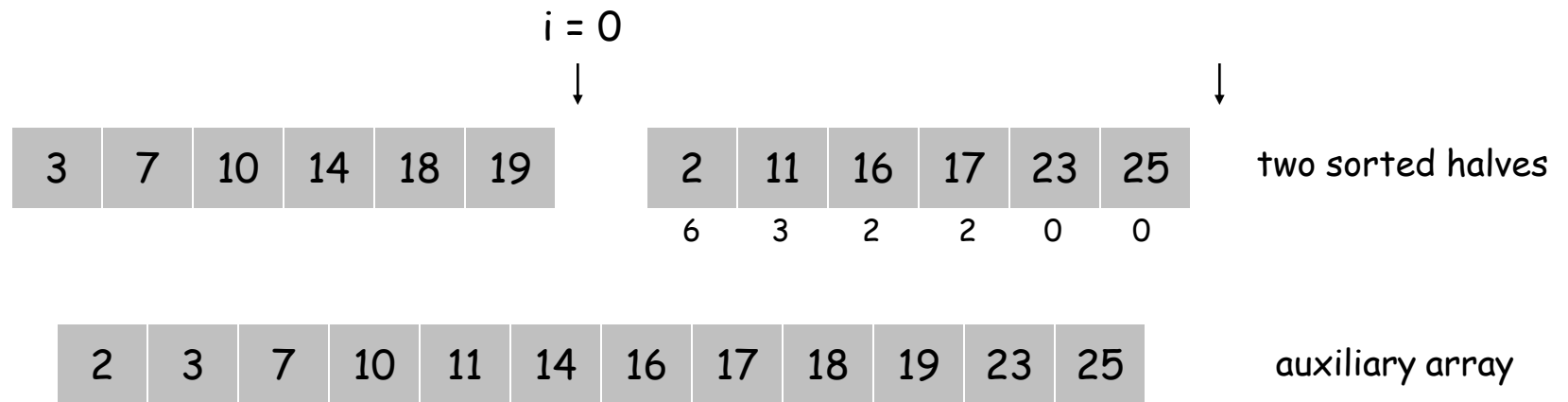


Total: $6 + 3 + 2 + 2 + 0 + 0$

Merge and Count

Merge and count step.

- Given two sorted halves, count number of inversions where a_i and a_j are in different halves.
- Combine two sorted halves into sorted whole.



Total: $6 + 3 + 2 + 2 + 0 + 0 = 13$

Counting Inversions: Implementation

Pre-condition. [Merge-and-Count] A and B are sorted.

Post-condition. [Sort-and-Count] L is sorted.

```
Sort-and-Count(L) {  
    if list L has one element  
        return 0 and the list L  
  
    Divide the list into two halves A and B  
    ( $r_A$ , A)  $\leftarrow$  Sort-and-Count(A)  
    ( $r_B$ , B)  $\leftarrow$  Sort-and-Count(B)  
    ( $r$ , L)  $\leftarrow$  Merge-and-Count(A, B)  
  
    return  $r = r_A + r_B + r$  and the sorted list L  
}
```

Closest Pair of Points

Closest pair. Given n points in the plane, find a pair with smallest Euclidean distance between them.

Fundamental geometric primitive.

- Graphics, computer vision, geographic information systems, molecular modeling, air traffic control.
- Special case of nearest neighbor, Euclidean MST, Voronoi.

↑
fast closest pair inspired fast algorithms for these problems

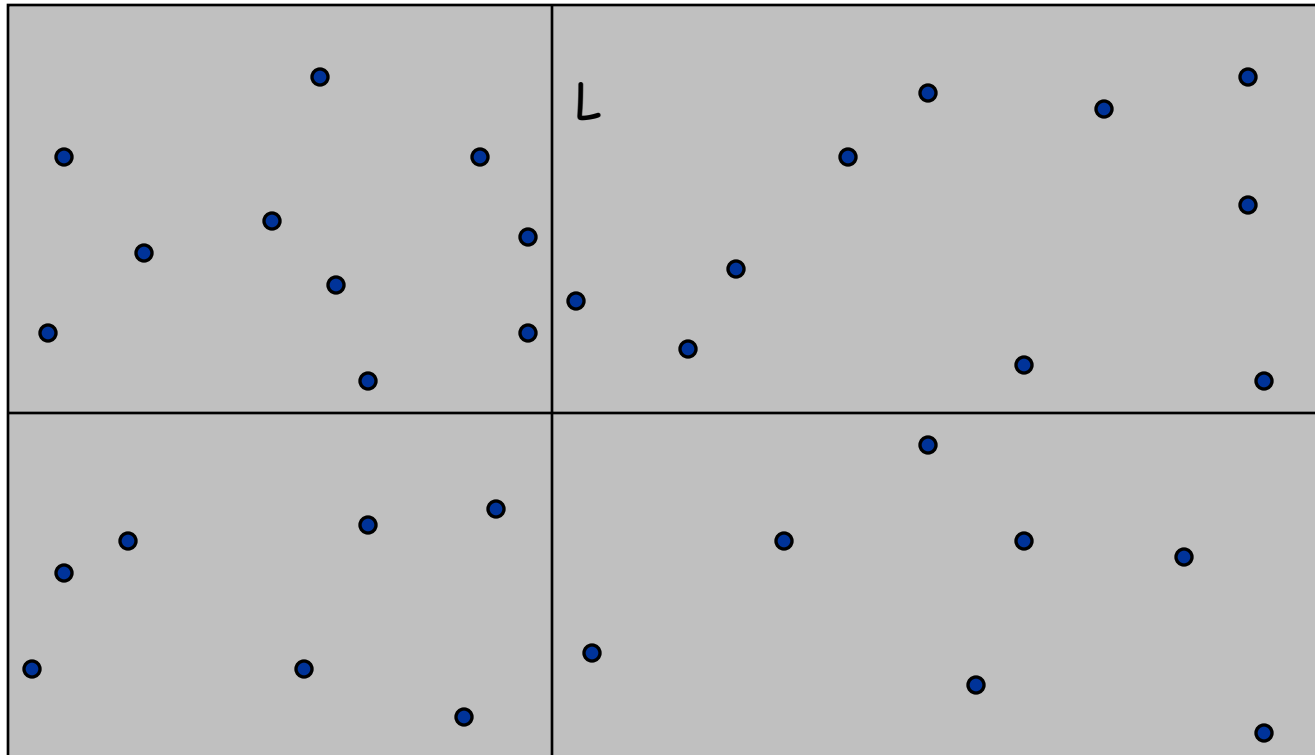
Brute force. Check all pairs of points p and q with $\Theta(n^2)$ comparisons.

1-D version. $O(n \log n)$ easy if points are on a line.

Assumption. No two points have same x coordinate.

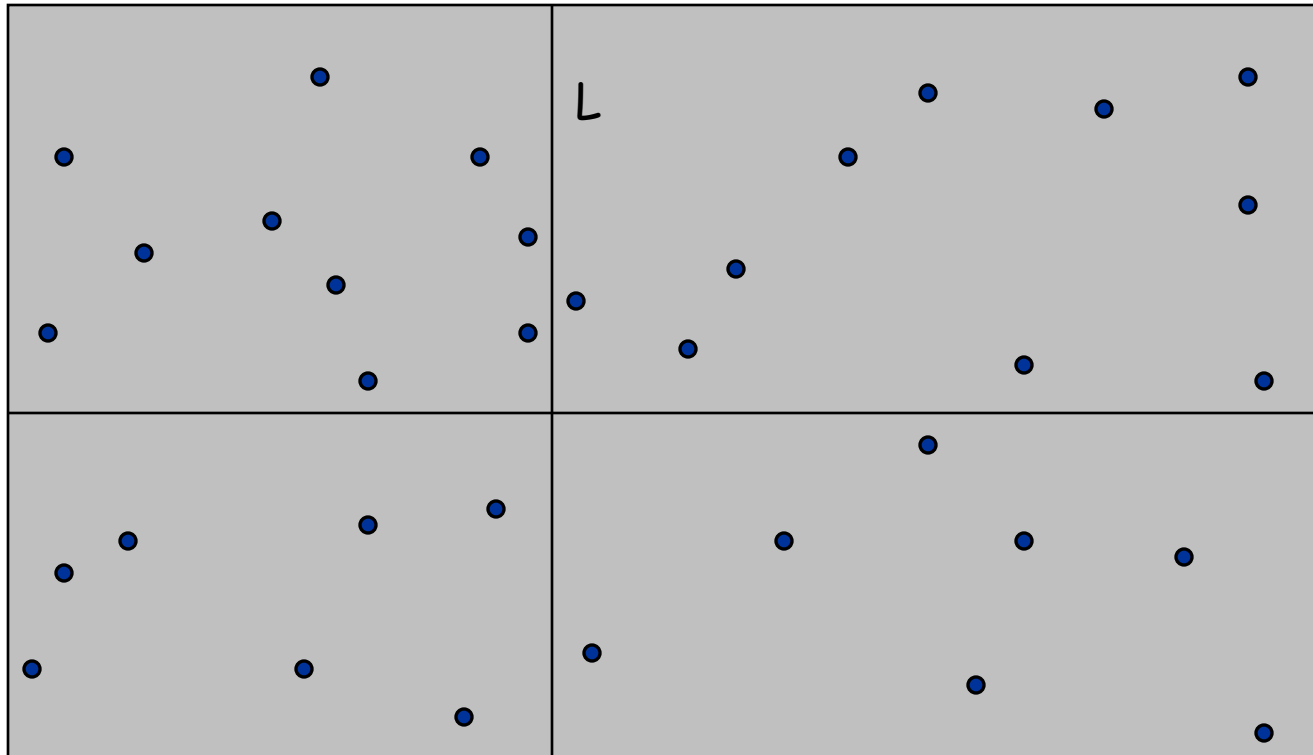
↑
to make presentation cleaner

Closest Pair of Points: First Attempt



Closest Pair of Points: First Attempt

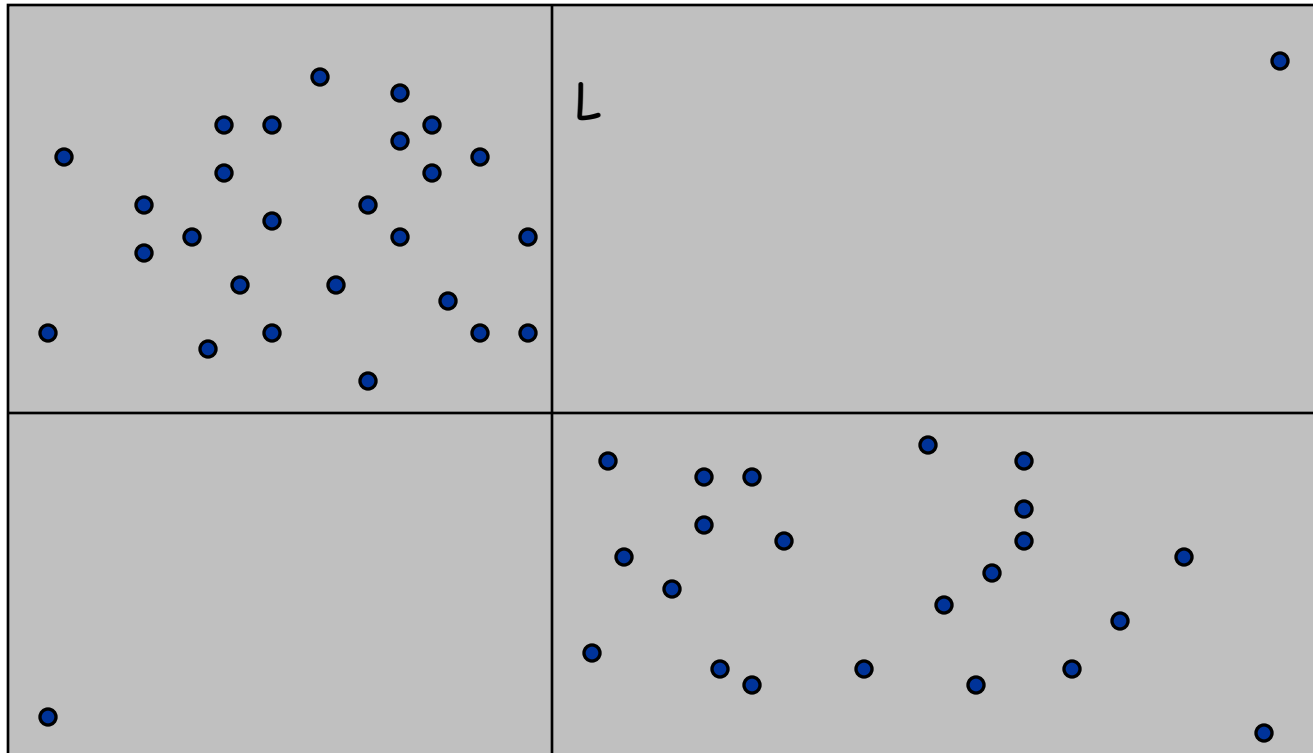
Divide. Sub-divide region into 4 quadrants.



Closest Pair of Points: First Attempt

Divide. Sub-divide region into 4 quadrants.

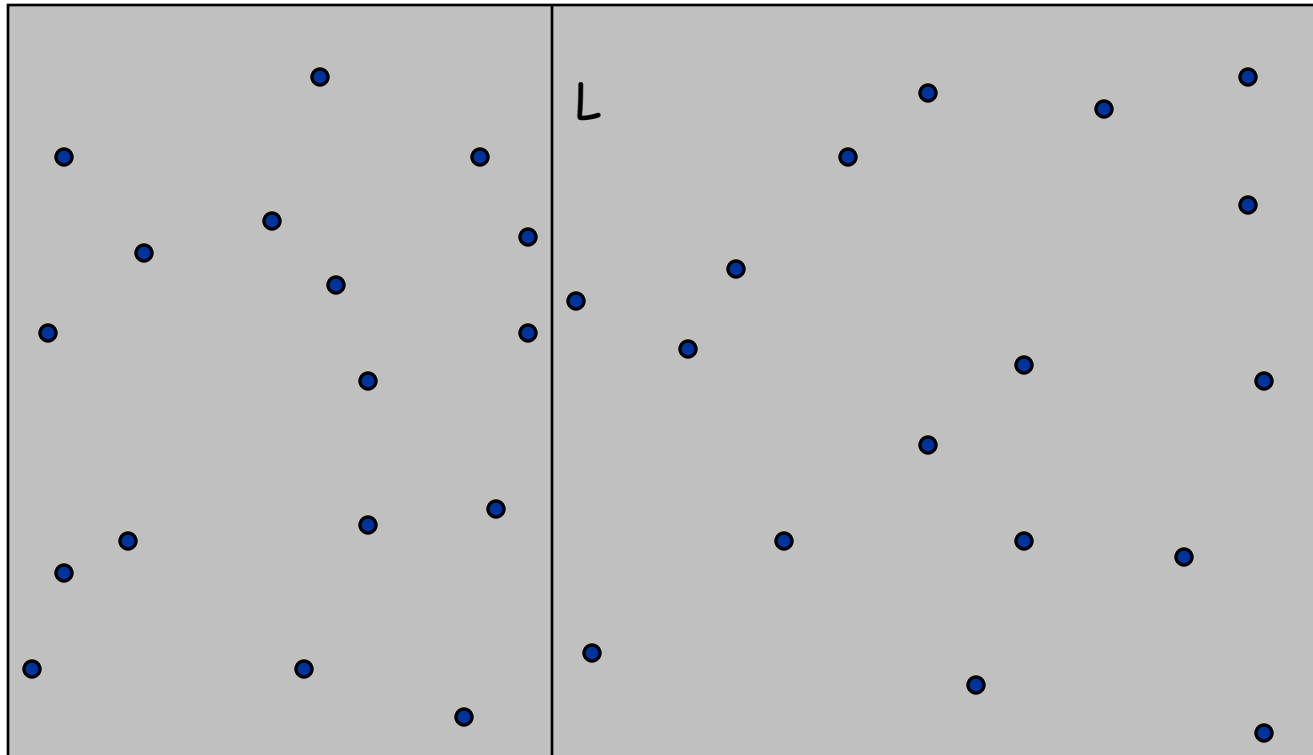
Obstacle. Impossible to ensure $n/4$ points in each piece.



Closest Pair of Points

Algorithm.

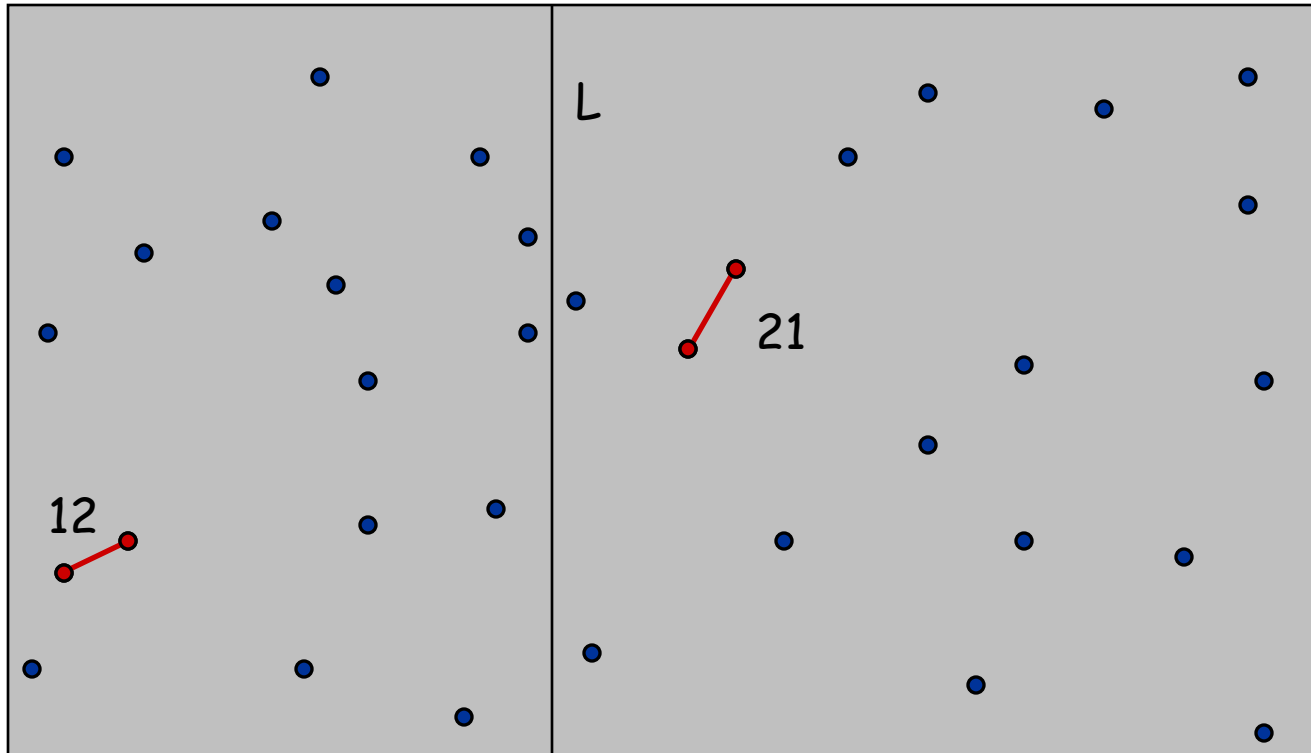
- **Divide:** draw vertical line L so that roughly $\frac{1}{2}n$ points on each side.



Closest Pair of Points

Algorithm.

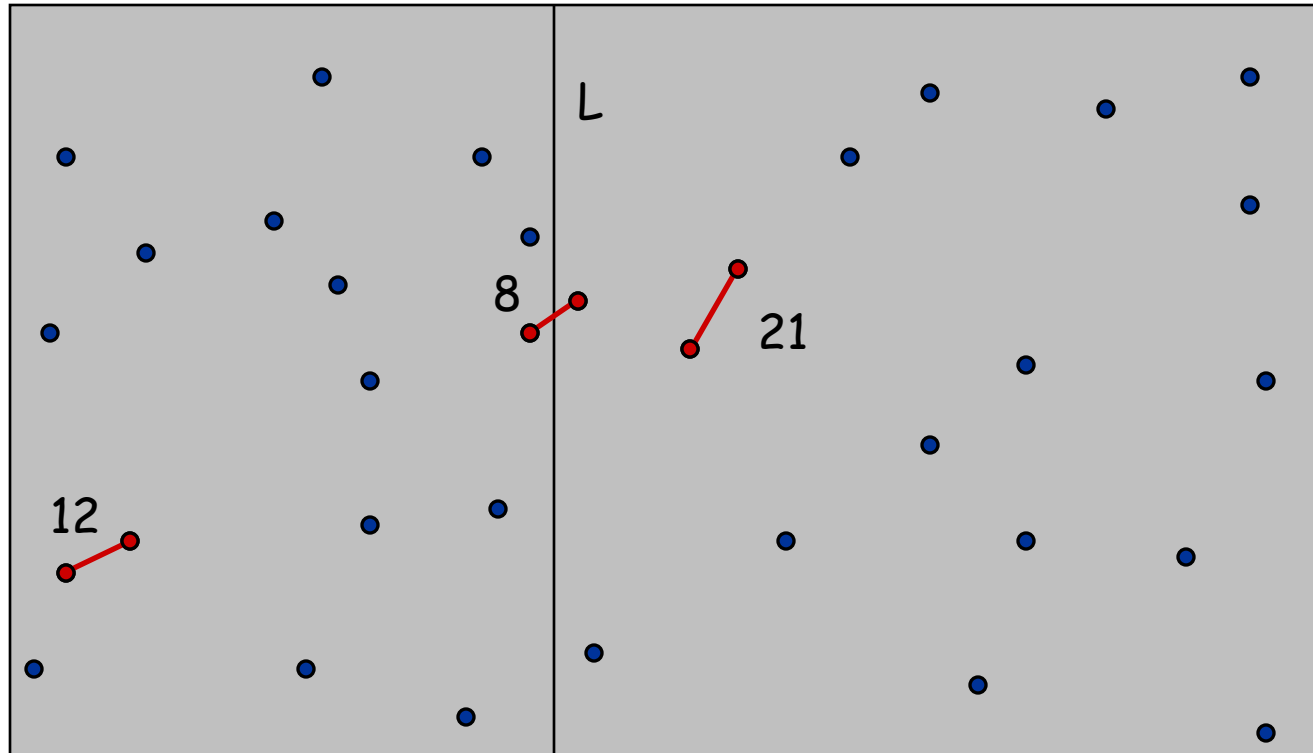
- Divide: draw vertical line L so that roughly $\frac{1}{2}n$ points on each side.
- **Conquer**: find closest pair in each side recursively.



Closest Pair of Points

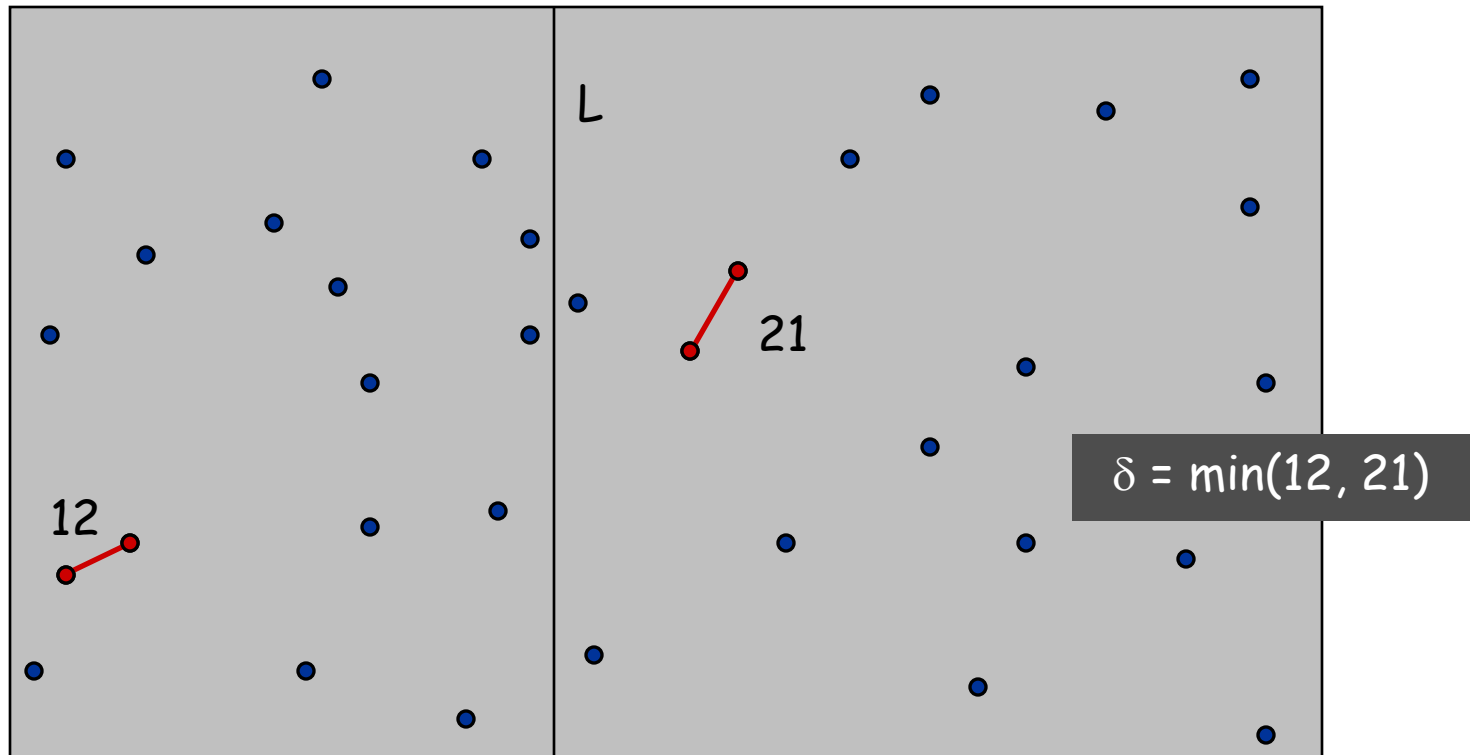
Algorithm.

- Divide: draw vertical line L so that roughly $\frac{1}{2}n$ points on each side.
- Conquer: find closest pair in each side recursively.
- **Combine**: find closest pair with one point in each side. ← seems like $\Theta(n^2)$
- Return best of 3 solutions.



Closest Pair of Points

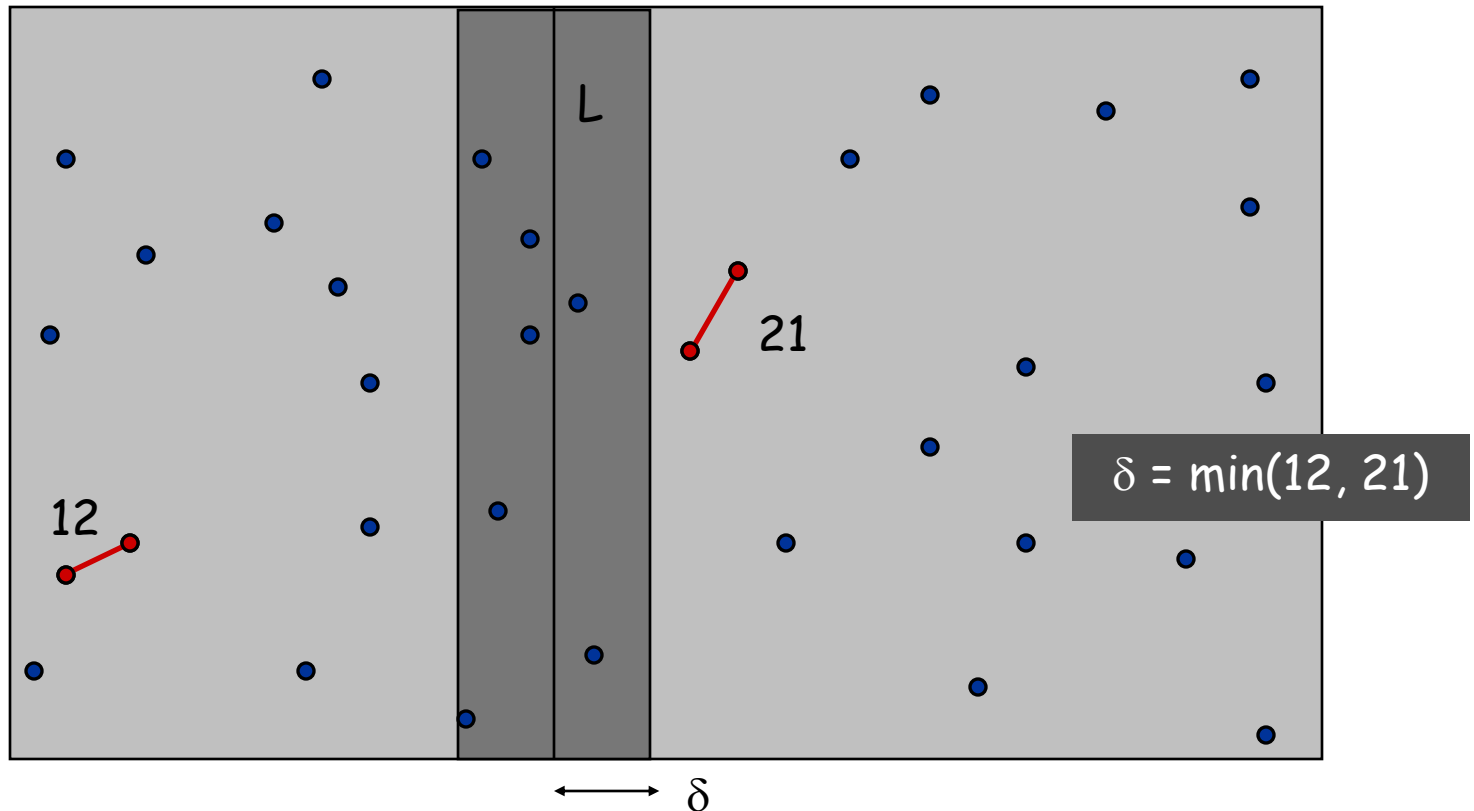
Find closest pair with one point in each side, **assuming that distance $< \delta$** .



Closest Pair of Points

Find closest pair with one point in each side, **assuming that distance $< \delta$** .

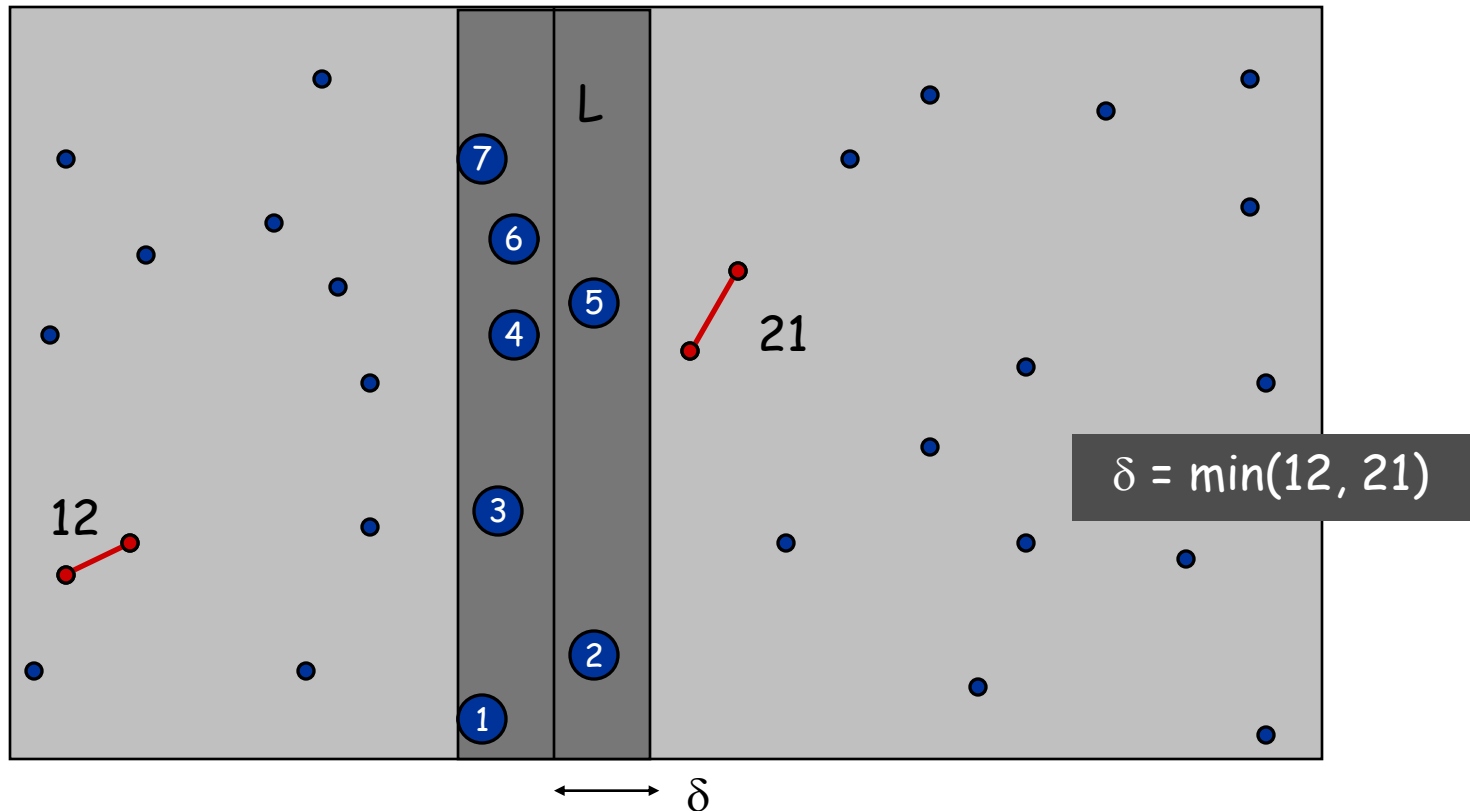
- Observation: only need to consider points within δ of line L .



Closest Pair of Points

Find closest pair with one point in each side, **assuming that distance $< \delta$** .

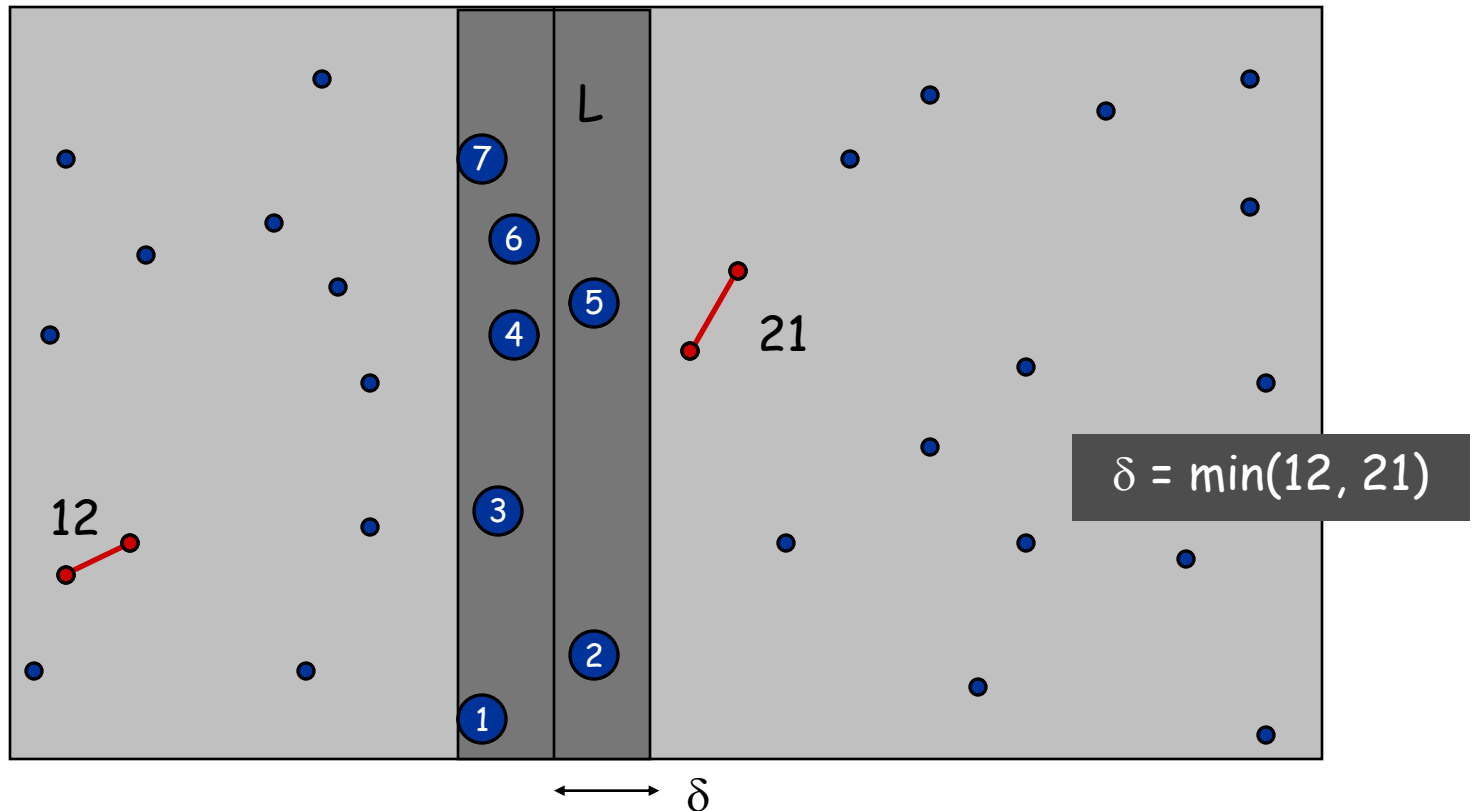
- Observation: only need to consider points within δ of line L .
- Sort points in 2δ -strip by their y coordinate.



Closest Pair of Points

Find closest pair with one point in each side, **assuming that distance $< \delta$** .

- Observation: only need to consider points within δ of line L .
- Sort points in 2δ -strip by their y coordinate.
- Only check distances of those within 11 positions in sorted list!



Closest Pair of Points

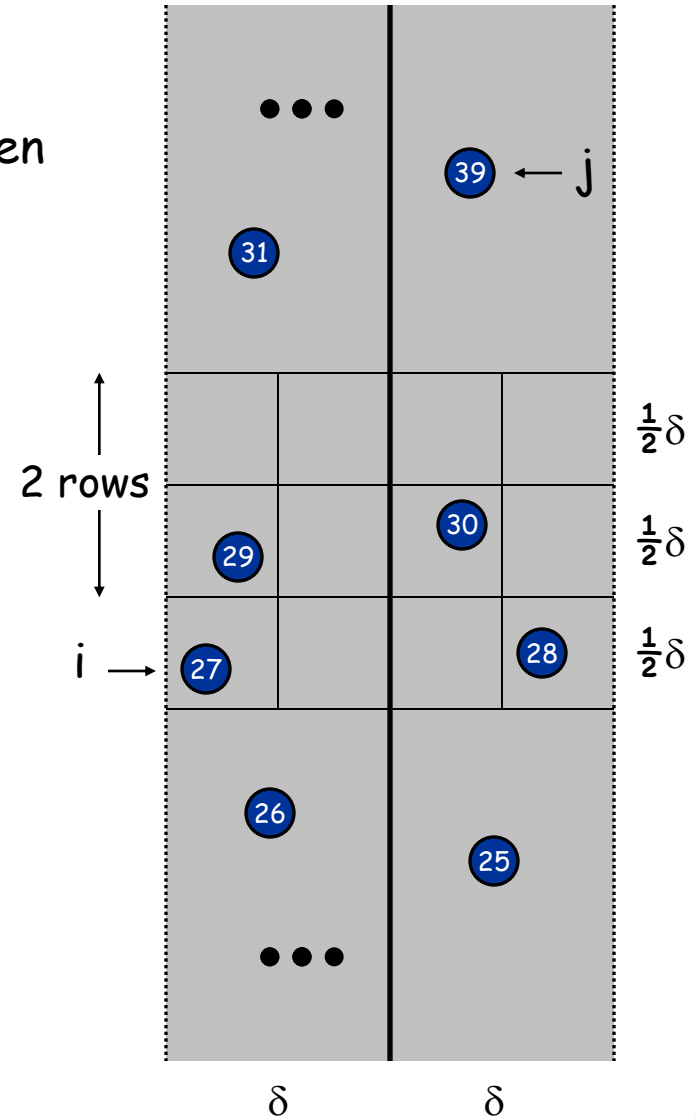
Def. Let s_i be the point in the 2δ -strip, with the i^{th} smallest y -coordinate.

Claim. If $|i - j| \geq 12$, then the distance between s_i and s_j is at least δ .

Pf.

- No two points lie in same $\frac{1}{2}\delta$ -by- $\frac{1}{2}\delta$ box.
- Two points at least 2 rows apart have distance $\geq 2(\frac{1}{2}\delta)$. ▪

Fact. Still true if we replace 12 with 7.



Closest Pair Algorithm

```
Closest-Pair( $p_1, \dots, p_n$ ) {  
    Compute separation line  $L$  such that half the points  
    are on one side and half on the other side.  $O(n \log n)$   
  
     $\delta_1 = \text{Closest-Pair}(\text{left half})$   
     $\delta_2 = \text{Closest-Pair}(\text{right half})$   $2T(n / 2)$   
     $\delta = \min(\delta_1, \delta_2)$   
  
    Delete all points further than  $\delta$  from separation line  $L$   $O(n)$   
  
    Sort remaining points by  $y$ -coordinate.  $O(n \log n)$   
  
    Scan points in  $y$ -order and compare distance between  
    each point and next 11 neighbors. If any of these  
    distances is less than  $\delta$ , update  $\delta$ .  $O(n)$   
  
    return  $\delta$ .  
}
```

Closest Pair of Points: Analysis

Running time.

$$T(n) \leq 2T(n/2) + O(n \log n) \Rightarrow T(n) = O(n \log^2 n)$$

Q. Can we achieve $O(n \log n)$?

A. Yes. Don't sort points in strip from scratch each time.

- Each recursive returns two lists: all points sorted by y coordinate, and all points sorted by x coordinate.
- Sort by **merging** two pre-sorted lists.

$$T(n) \leq 2T(n/2) + O(n) \Rightarrow T(n) = O(n \log n)$$