南方科技大学
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

Algorithm Design and Analysis (H)

CS216

Prof. Shiqi Yu (于仕琪)

yusq@sustech.edu.cn

http://faculty.sustech.edu.cn/yusq/

南方科技大学
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

Divide and Conquer

PEARSON
Addison Wesley

Divide-and-Conquer

凡治众如治寡 分数是也

孙子兵法

- Divide-and-conquer.
  - Break up problem into several parts.
  - Solve each part recursively.
  - Combine solutions to sub-problems into overall solution.

- Most common usage.
  - Break up problem of size n into two equal parts of size ½n.
  - Solve two parts recursively.
  - Combine two solutions into overall solution in linear time.

- Consequence.
  - Brute force: n^2.
  - Divide-and-conquer: n log n.

1. Mergesort

Sorting

- Sorting. Given n elements, rearrange in ascending order.
- Applications.
  - Sort a list of names.
  - Organize an MP3 library.
  - Display Google PageRank results.
  - List RSS news items in reverse chronological order.

obvious applications

  - Find the median.
  - Find the closest pair.
  - Binary search in a database.
  - Identify statistical outliers.
  - Find duplicates in a mailing list.

problems become easy once items are in sorted order

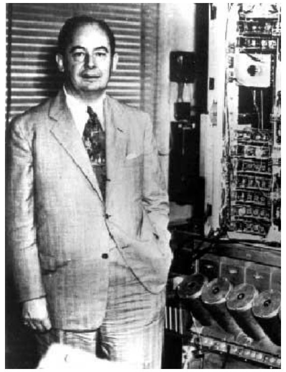
  - Data compression.
  - Computer graphics.
  - Computational biology.
  - Supply chain management.
  - Book recommendations on Amazon.
  - Load balancing on a parallel computer.

  . . .

non-obvious applications

Mergesort

- Mergesort.
  - Divide array into two halves.
  - Recursively sort each half.
  - Merge two halves to make sorted whole.



Jon von Neumann (1945)

A	L	G	O	R	I	T	H	M	S

A	L	G	O	R		I	T	H	M	S	divide	$O(1)$
A	G	L	O	R		H	I	M	S	T	sort	$2T(n/2)$

A	G	H	I	L	M	O	R	S	T	merge	$O(n)$

Merging

- Merging. Combine two pre-sorted lists into a sorted whole.

- How to merge efficiently?
  - ➢ Linear number of comparisons.
  - ➢ Use temporary array.

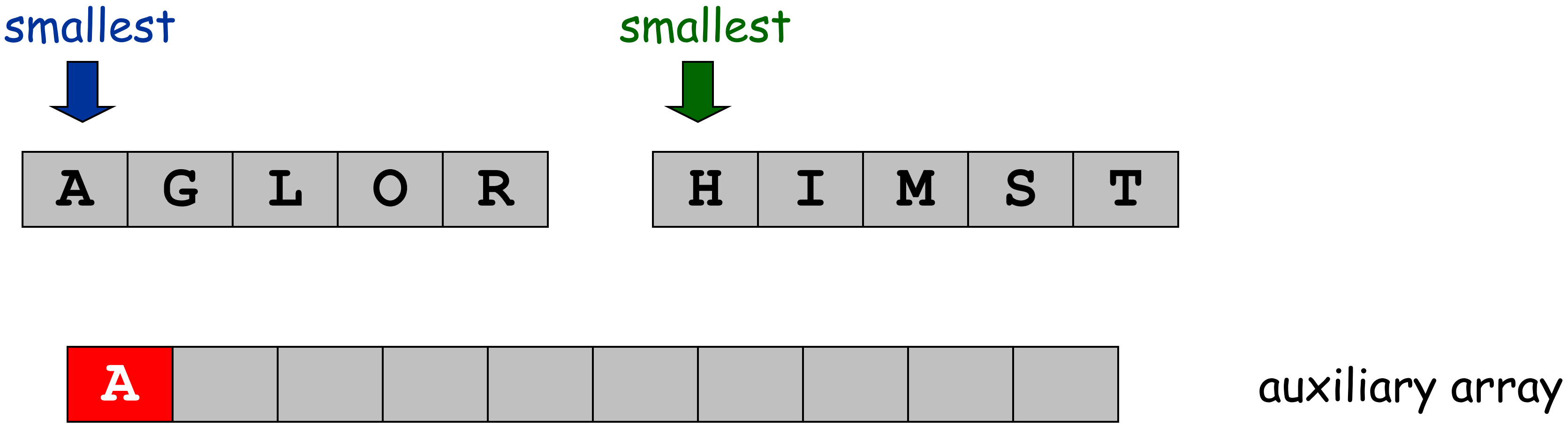
- Challenge for the bored. In-place merge. [Kronrud, 1969]

using only a constant amount of extra storage

Merging

Merge.

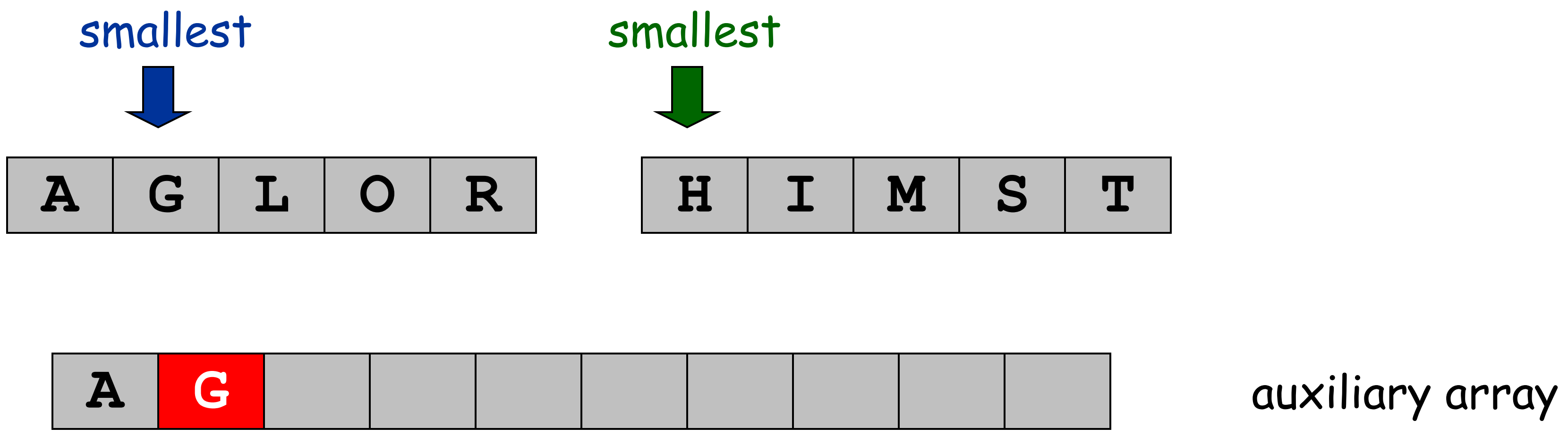
- Keep track of smallest element in each sorted half.
- Insert smallest of two elements into auxiliary array.
- Repeat until done.



Merging

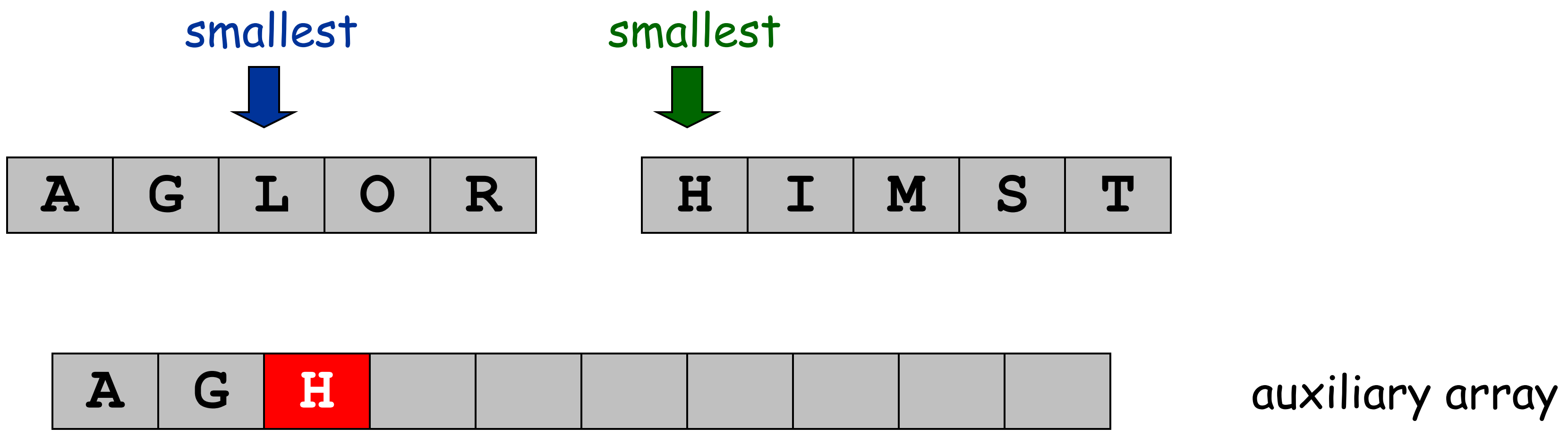
Merge.

- Keep track of smallest element in each sorted half.
- Insert smallest of two elements into auxiliary array.
- Repeat until done.



Merging

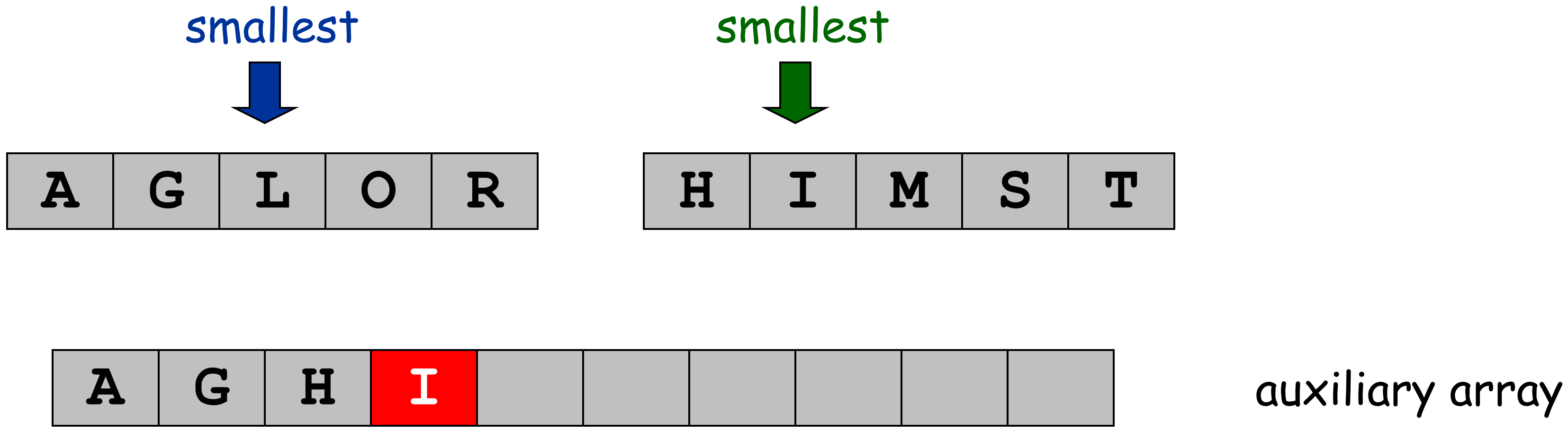
Merge.
- Keep track of smallest element in each sorted half.
- Insert smallest of two elements into auxiliary array.
- Repeat until done.



Merging

Merge.

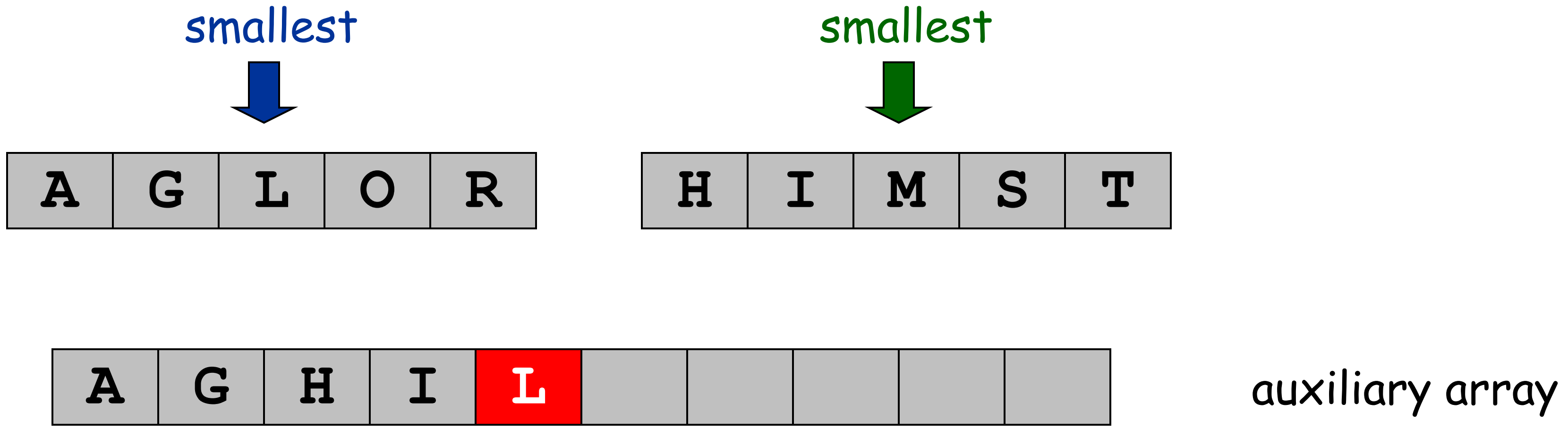
- Keep track of smallest element in each sorted half.
- Insert smallest of two elements into auxiliary array.
- Repeat until done.



Merging

Merge.

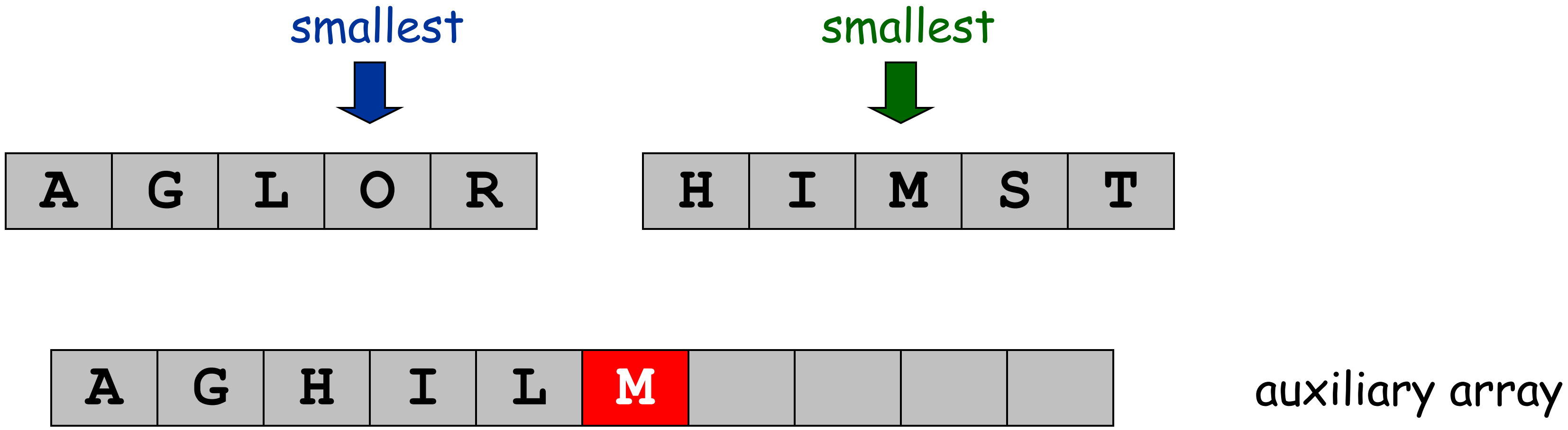
- Keep track of smallest element in each sorted half.
- Insert smallest of two elements into auxiliary array.
- Repeat until done.



Merging

Merge.

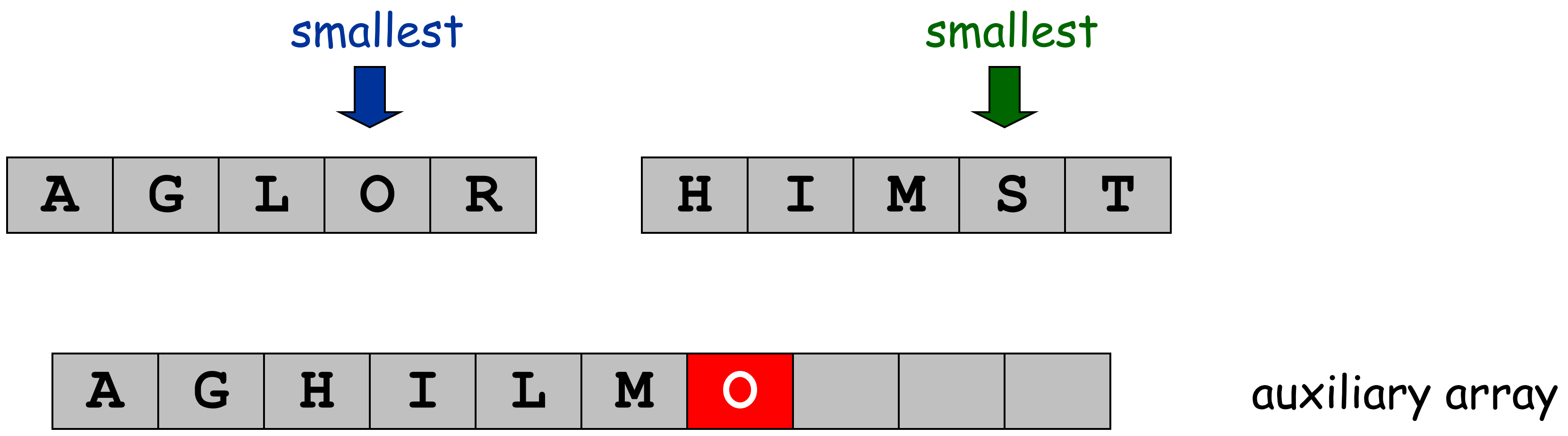
- Keep track of smallest element in each sorted half.
- Insert smallest of two elements into auxiliary array.
- Repeat until done.



Merging

Merge.

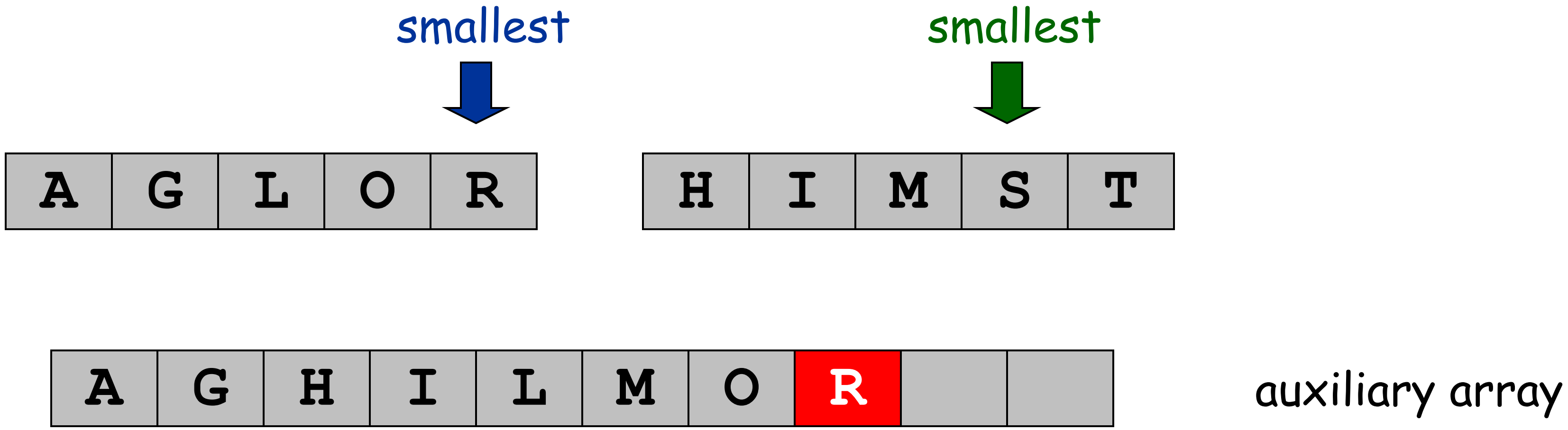
- Keep track of smallest element in each sorted half.
- Insert smallest of two elements into auxiliary array.
- Repeat until done.



Merging

Merge.

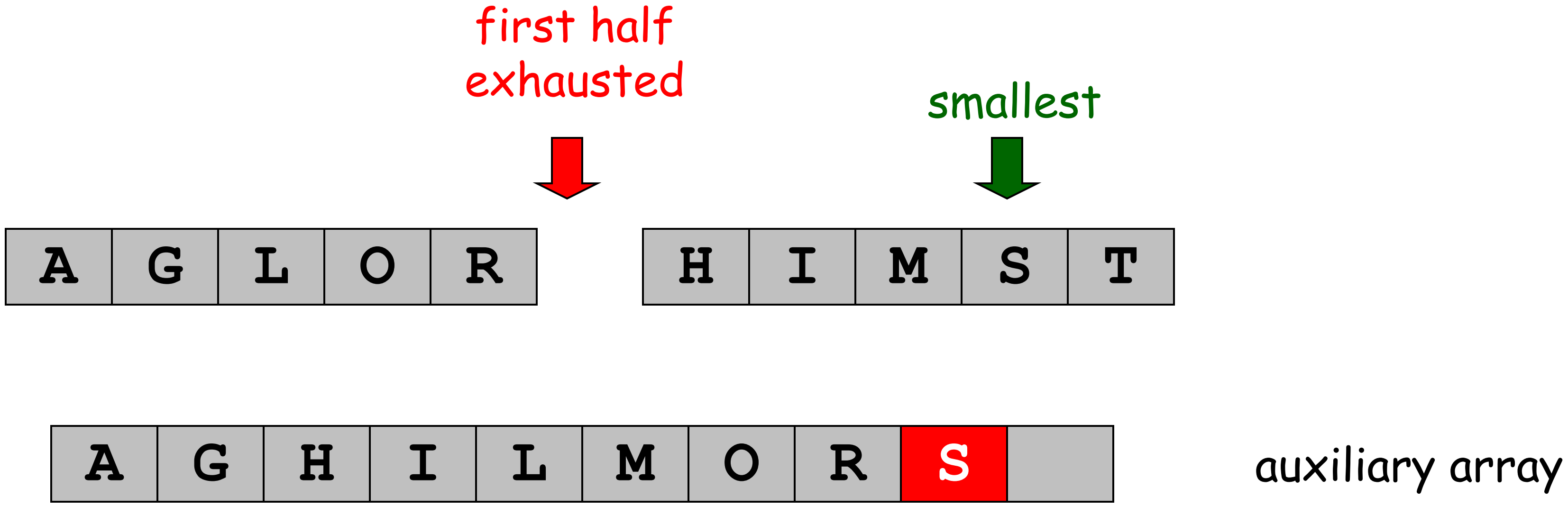
- Keep track of smallest element in each sorted half.
- Insert smallest of two elements into auxiliary array.
- Repeat until done.



Merging

Merge.

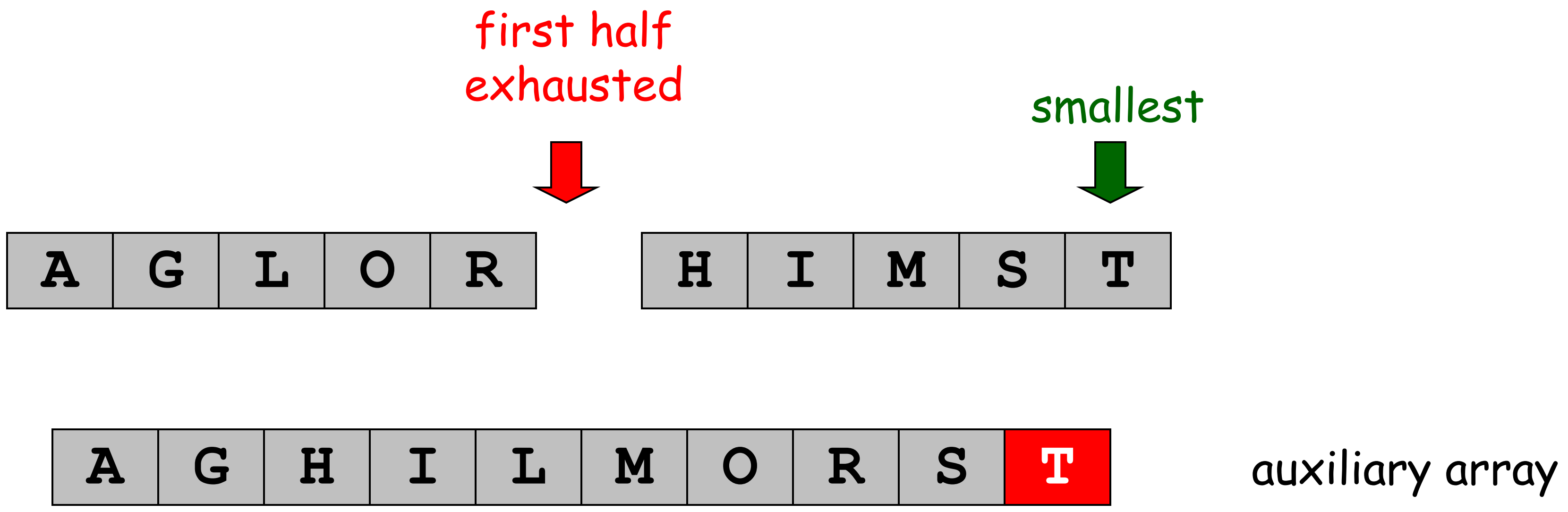
- Keep track of smallest element in each sorted half.
- Insert smallest of two elements into auxiliary array.
- Repeat until done.



Merging

Merge.

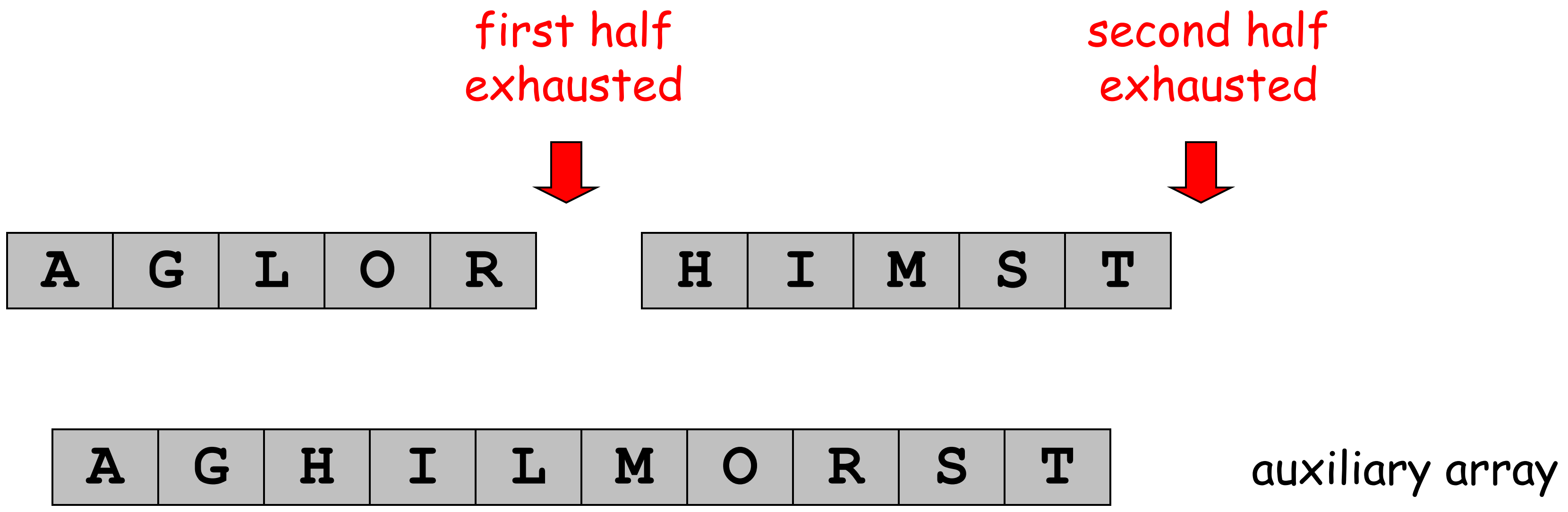
- Keep track of smallest element in each sorted half.
- Insert smallest of two elements into auxiliary array.
- Repeat until done.



Merging

Merge.

- Keep track of smallest element in each sorted half.
- Insert smallest of two elements into auxiliary array.
- Repeat until done.



A Useful Recurrence Relation

Def. T(n) = number of comparisons to mergesort an input of size n.

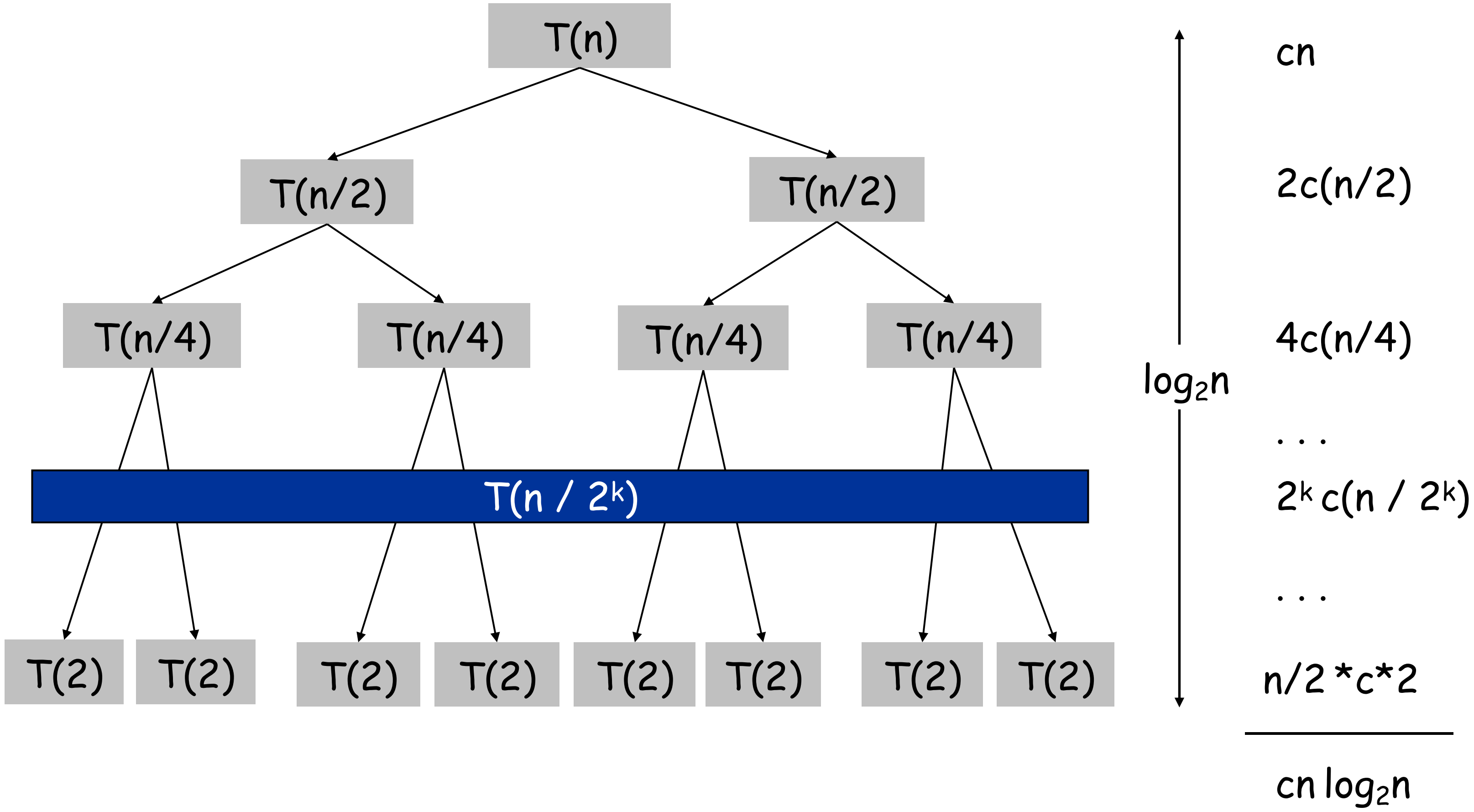
Mergesort recurrence.

- $\mathrm{T}(n) \leq 2T\left(\frac{n}{2}\right) + cn$

Solution. $T(n) = O(n \log_2 n)$.

Assorted proofs. We describe several ways to prove this recurrence. Initially we assume n is a power of 2 and replace $\leq$ with =.

Proof by Recursion Tree



Proof by Telescoping

Claim. If $T(n)$ satisfies this recurrence, then $T(n) \leq cn \log_2 n$.

Pf. For $n > 1$:

$n = 2$: $T(2) \leq 2c$

assume for all $m < n$, $T(m) \leq cm \log_2 m$.

$$
\begin{aligned}
T(n) &\leq 2T(n/2) + cn \\
&\leq 2c(n/2) \log_2(n/2) + cn \\
&= cn \log_2(n/2) + cn \\
&= cn \log_2(n) - cn + cn \\
&= cn \log_2(n)
\end{aligned}
$$

南方科技大学
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

2. Counting Inversions

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, …, n.
  - Your rank: $a_1, a_2, \ldots, a_n$.
  - Songs i and j inverted if $i < j$, but $a_i > a_j$.

Songs

	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.

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

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

8 green-green inversions

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

Conquer: 2T(n / 2)

5 blue-blue inversions

8 green-green inversions

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.

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: $O(n)$

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

two sorted halves

3	7	10	14	18	19		2	11	16	17	23	25
							6					

auxiliary array

2											

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						

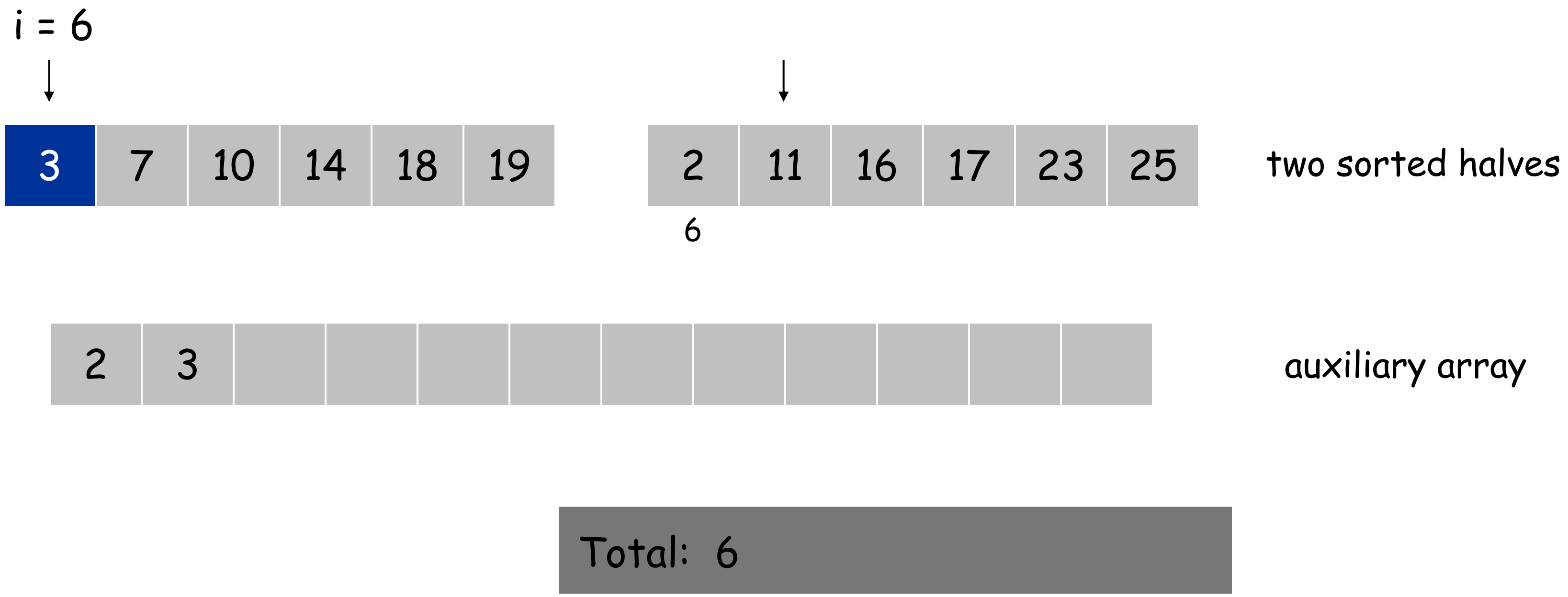
2												auxiliary array

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.



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 = 5

3	7	10	14	18	19		2	11	16	17	23	25
							6					

two sorted halves

2	3										

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 = 5

↓ ↓

3	**7**	10	14	18	19		2	11	16	17	23	25
							6					

two sorted halves

2	3	7									

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 = 4

3	7	10	14	18	19	2	11	16	17	23	25	two sorted halves
						6						

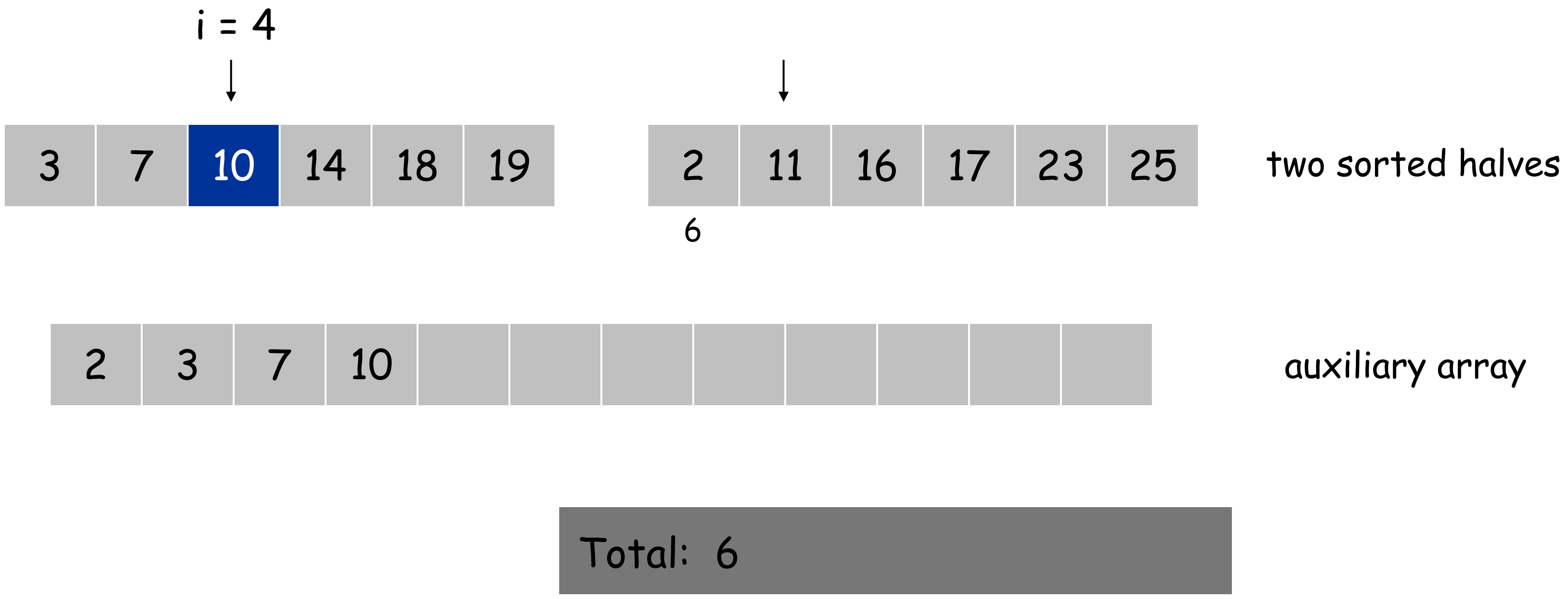
2	3	7										auxiliary array

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.

i = 3

3	7	10	14	18	19		2	11	16	17	23	25	two sorted halves
							6						

2	3	7	10									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 = 3

3	7	10	14	18	19

2	11	16	17	23	25
6	3				

two sorted halves

2	3	7	10	11							

auxiliary array

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.

i = 3

3	7	10	14	18	19	2	11	16	17	23	25	two sorted halves
						6	3					

2	3	7	10	11								auxiliary array

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.

i = 3

two sorted halves

3	7	10	14	18	19		2	11	16	17	23	25
							6	3				

auxiliary array

2	3	7	10	11	14						

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.

i = 2

3	7	10	14	18	19	2	11	16	17	23	25	two sorted halves
						6	3					

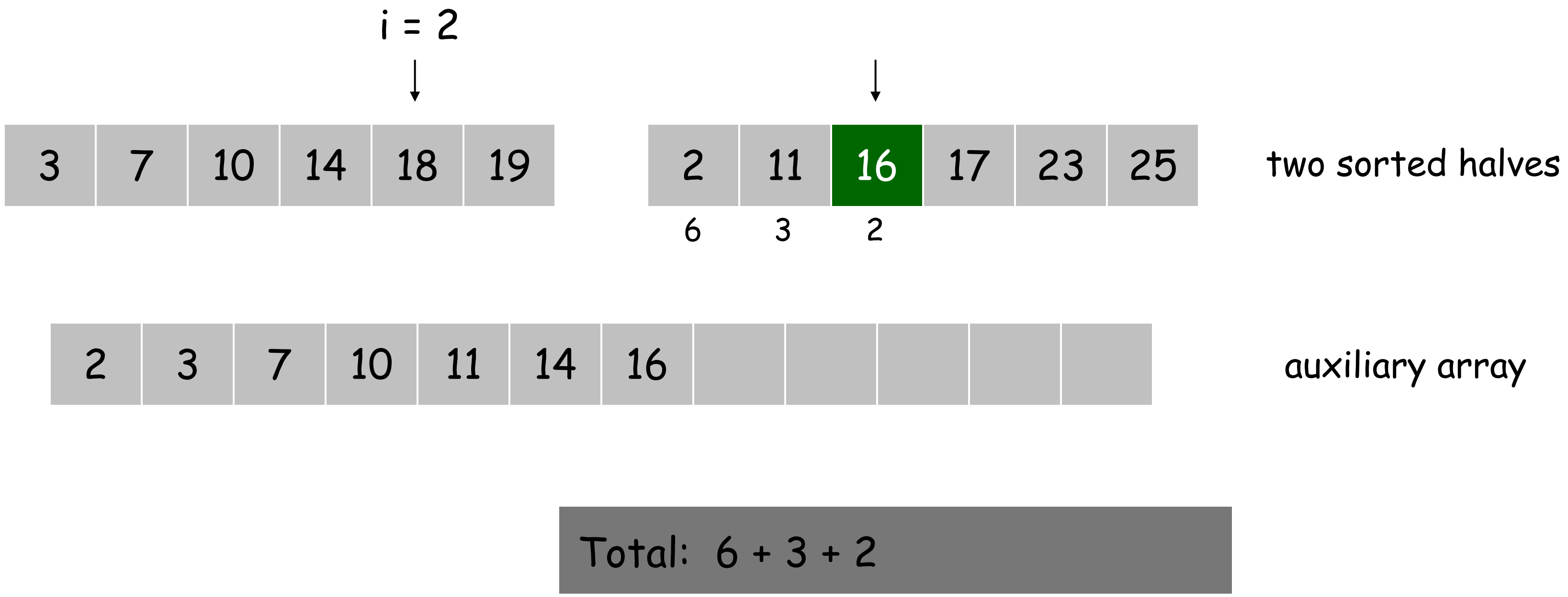
2	3	7	10	11	14							auxiliary array

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.



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 = 2

3	7	10	14	18	19		2	11	16	17	23	25
							6	3	2			

two sorted halves

2	3	7	10	11	14	16					

auxiliary array

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.

i = 2

3	7	10	14	18	19		2	11	16	17	23	25	two sorted halves
							6	3	2	2			

2	3	7	10	11	14	16	17					auxiliary array

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.

i = 2

3	7	10	14	18	19	2	11	16	17	23	25	two sorted halves
						6	3	2	2			

2	3	7	10	11	14	16	17					auxiliary array

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.

i = 2

3	7	10	14	**18**	19	2	11	16	17	23	25	two sorted halves
						6	3	2	2			

2	3	7	10	11	14	16	17	18				auxiliary array

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.

i = 1

3	7	10	14	18	19	2	11	16	17	23	25	two sorted halves
						6	3	2	2			

2	3	7	10	11	14	16	17	18				auxiliary array

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.

i = 1

3	7	10	14	18	19		2	11	16	17	23	25	two sorted halves
							6	3	2	2			

2	3	7	10	11	14	16	17	18	19			auxiliary array

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.

first half exhausted   i = 0

two sorted halves

3	7	10	14	18	19	2	11	16	17	23	25
						6	3	2	2		

auxiliary array

2	3	7	10	11	14	16	17	18	19		

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.

i = 0

3	7	10	14	18	19	2	11	16	17	23	25	two sorted halves
						6	3	2	2	0		

2	3	7	10	11	14	16	17	18	19	23		auxiliary array

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.

i = 0

two sorted halves

3	7	10	14	18	19	2	11	16	17	23	25
						6	3	2	2	0	

auxiliary array

2	3	7	10	11	14	16	17	18	19	23	

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.

i = 0

two sorted halves

3	7	10	14	18	19	2	11	16	17	23	25
						6	3	2	2	0	0

auxiliary array

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

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.

i = 0

3	7	10	14	18	19	2	11	16	17	23	25	two sorted halves
						6	3	2	2	0	0	

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

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
   (rA, A) ← Sort-and-Count(A)
   (rB, B) ← Sort-and-Count(B)
   (rB, L) ← Merge-and-Count(A, B)

   return r = rA + rB + r and the sorted list L
}
```

南方科技大学
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

3. Closest Pair of Points

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.

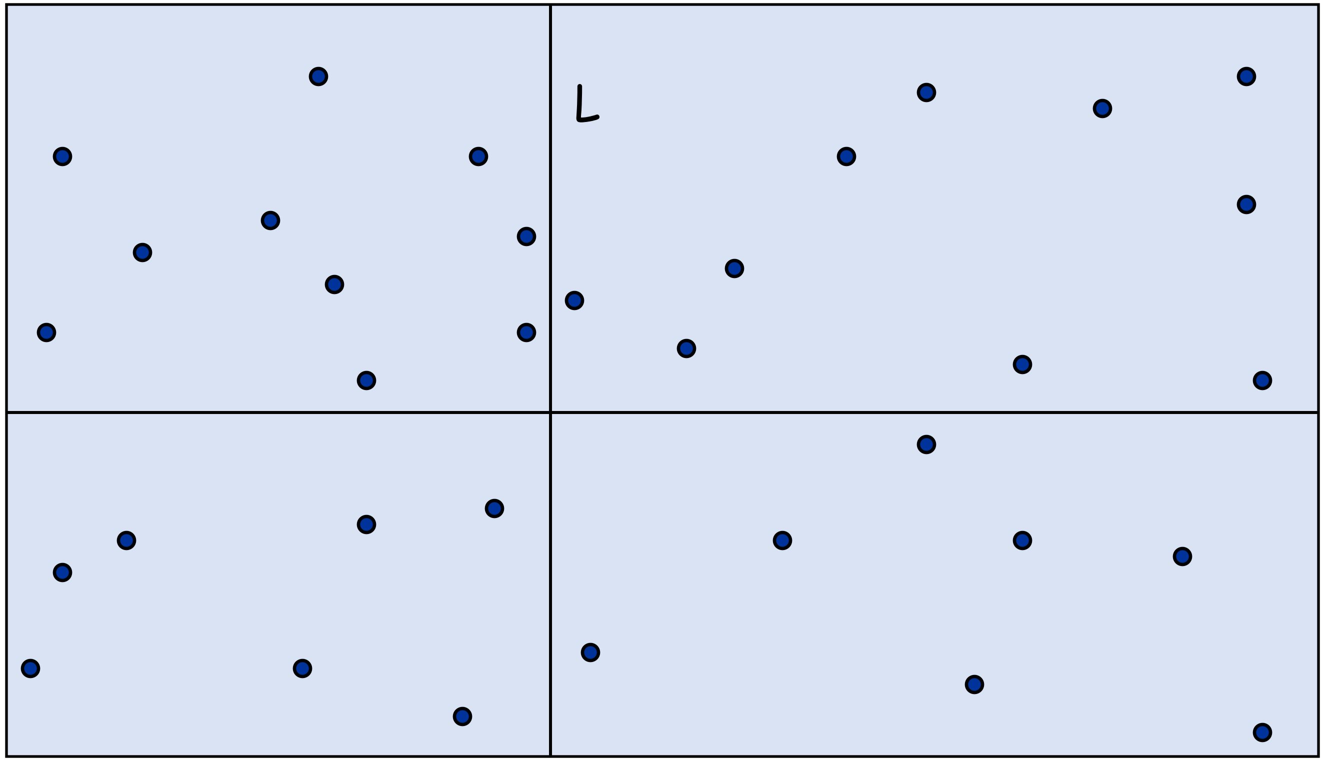
- 1-D version. O(n log n) easy if points are on a line.

- Assumption. No two points have same x coordinate.

to make presentation cleaner

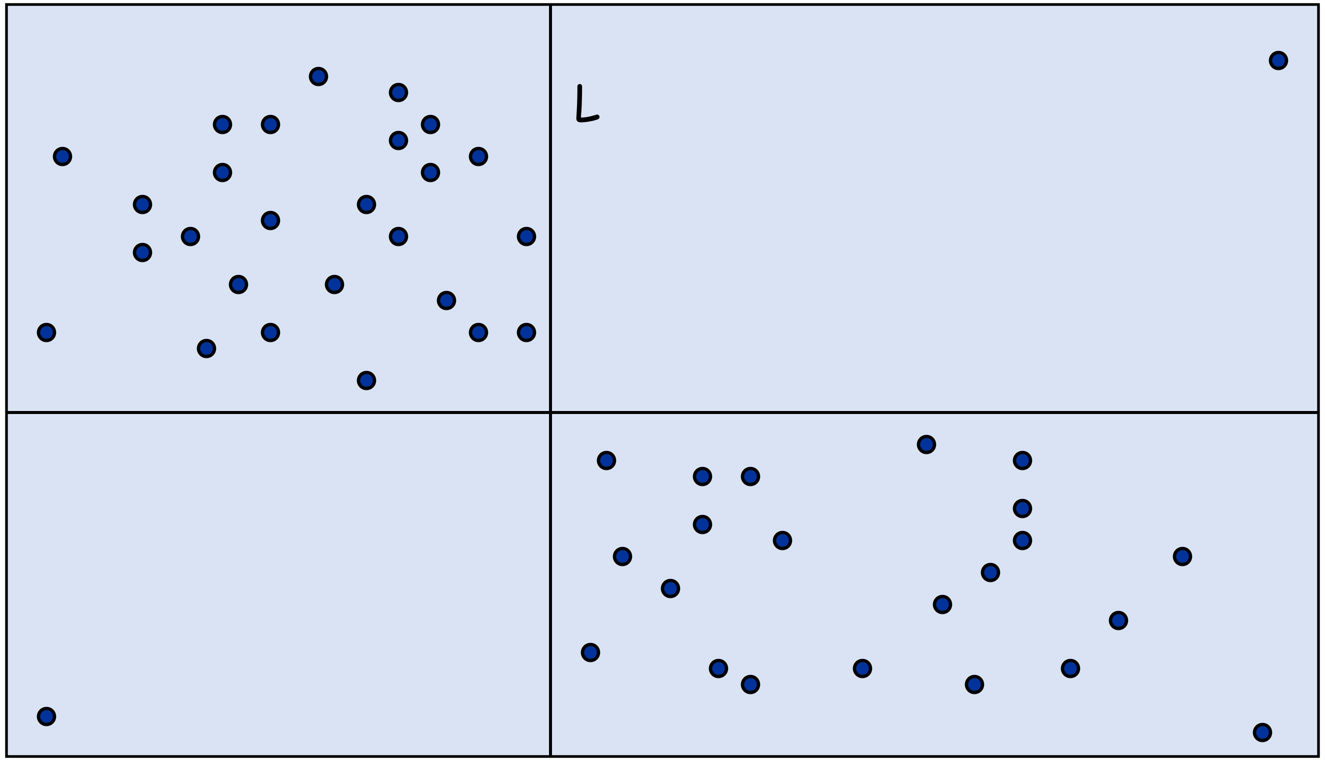
Closest Pair of Points: First Attempt

- Divide. Sub-divide region into 4 quadrants.



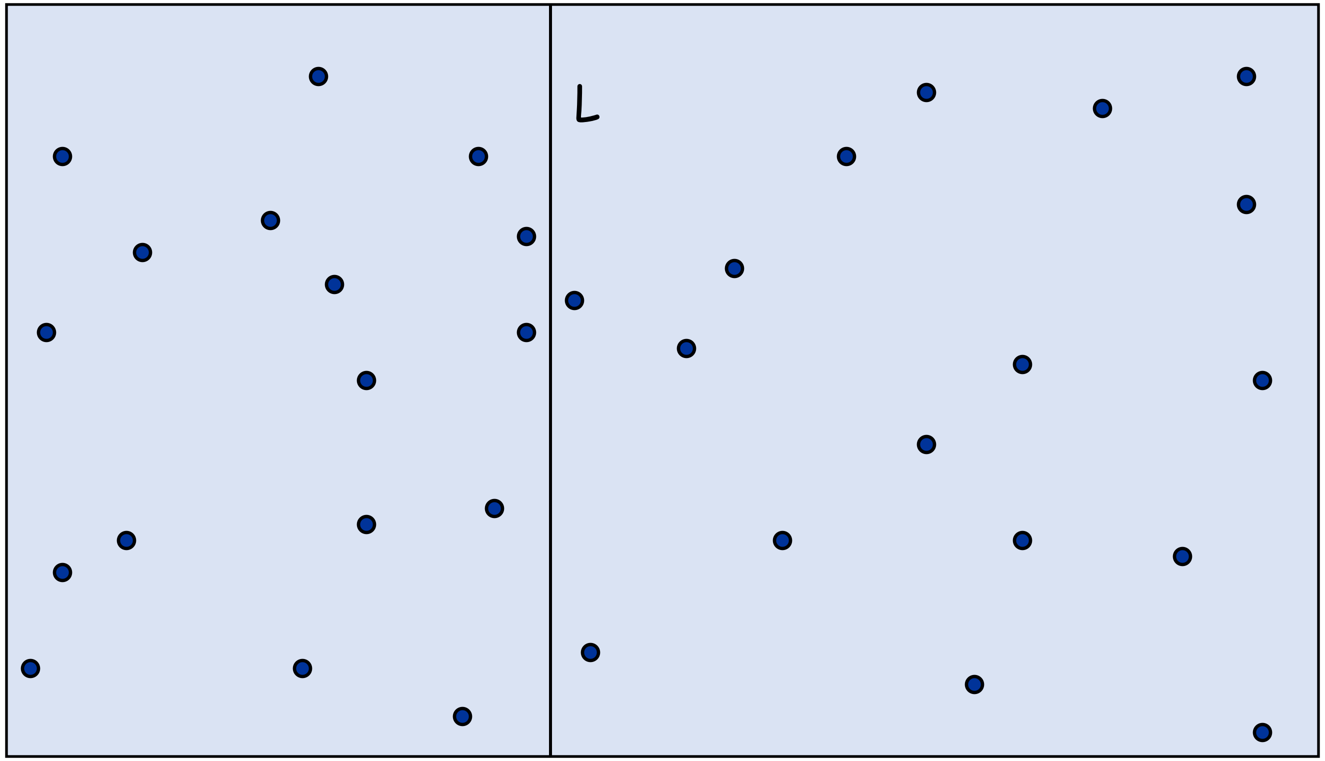
Closest Pair of Points: First Attempt

- Divide. Sub-divide region into 4 quadrants.
- Obstacle. Impossible to ensure n/4 points in each piece.



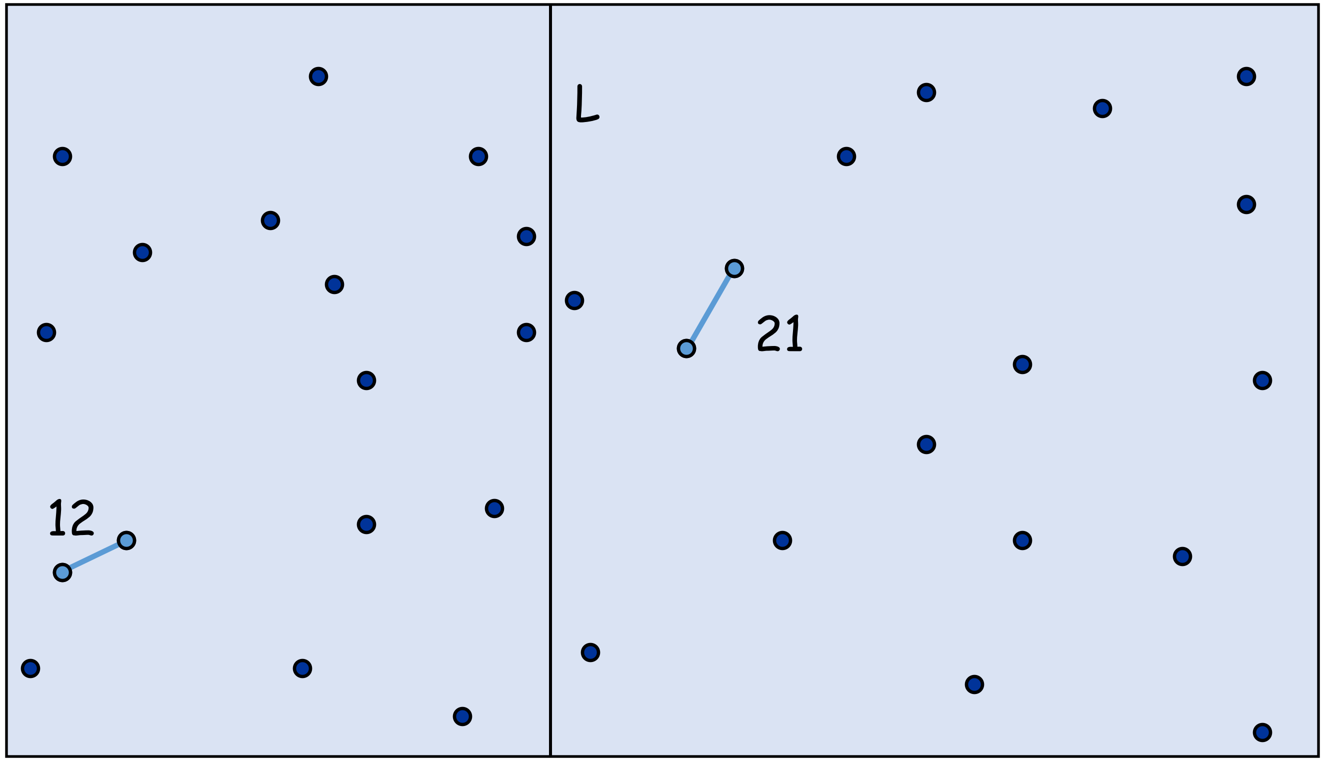
Closest Pair of Points

- Algorithm.
  - Divide: draw vertical line L so that roughly ½n points on each side.



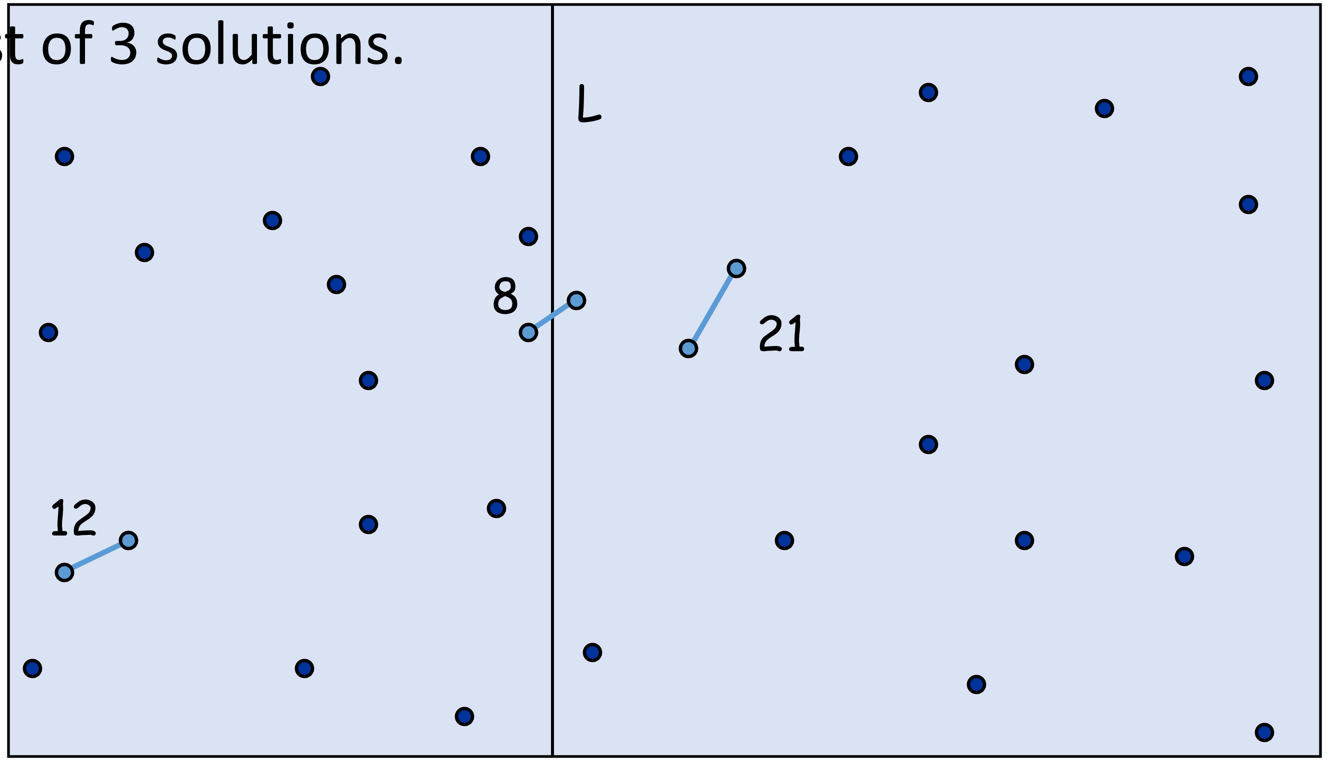
Closest Pair of Points

- Algorithm.
  - Divide: draw vertical line L so that roughly ½n points on each side.
  - Conquer: find closest pair in each side recursively.



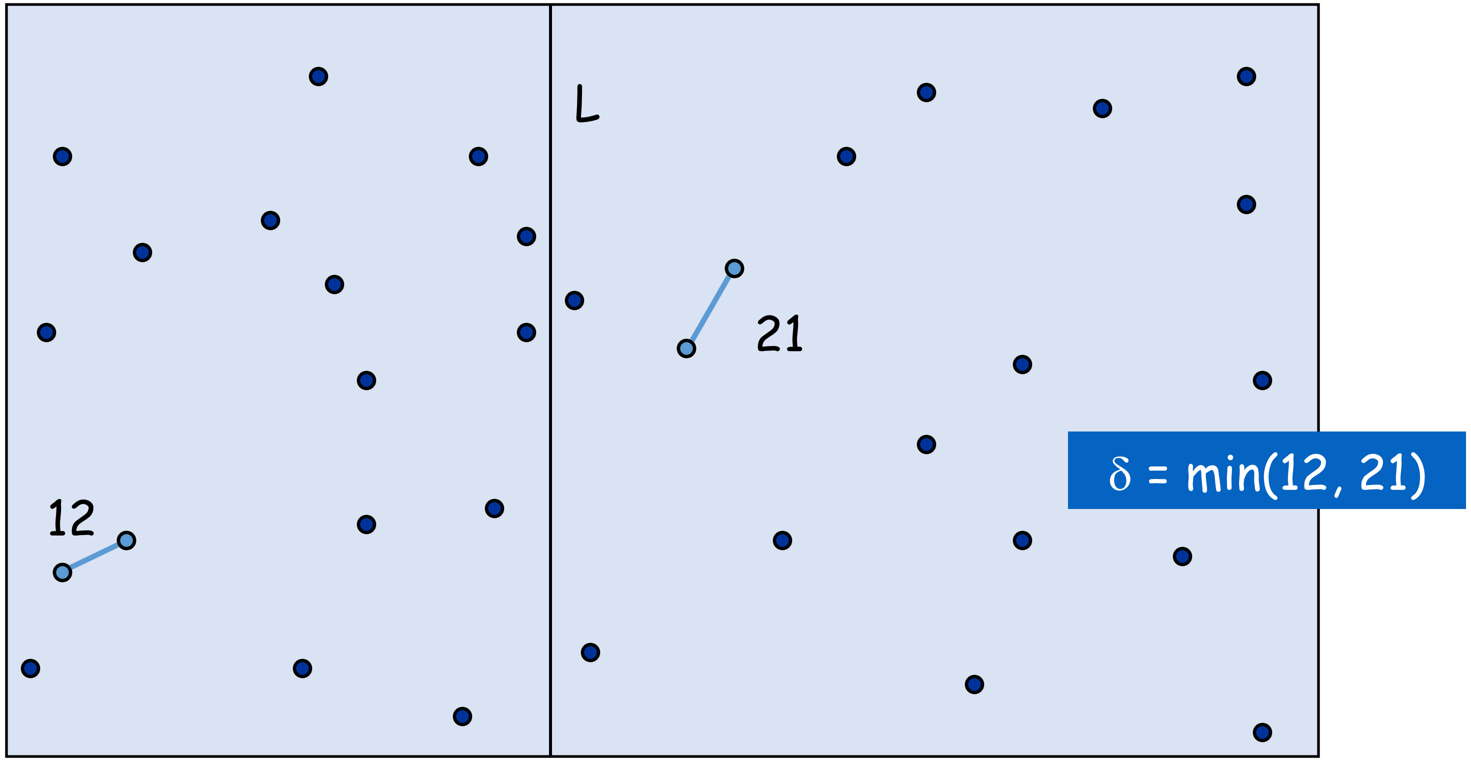
Closest Pair of Points

- Algorithm.
  - Divide: draw vertical line L so that roughly ½n 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

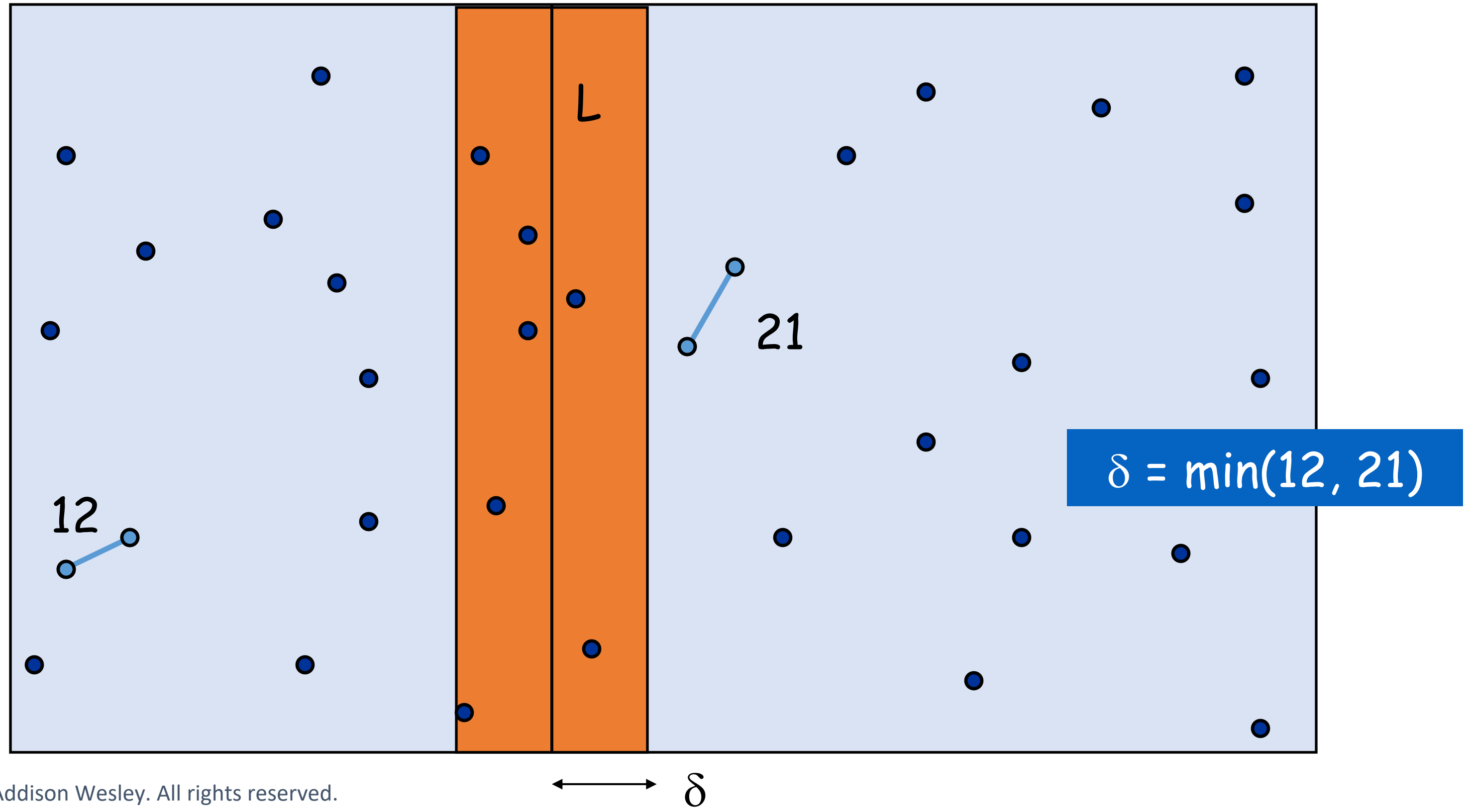
- Find closest pair with one point in each side, assuming that distance < δ.



Closest Pair of Points

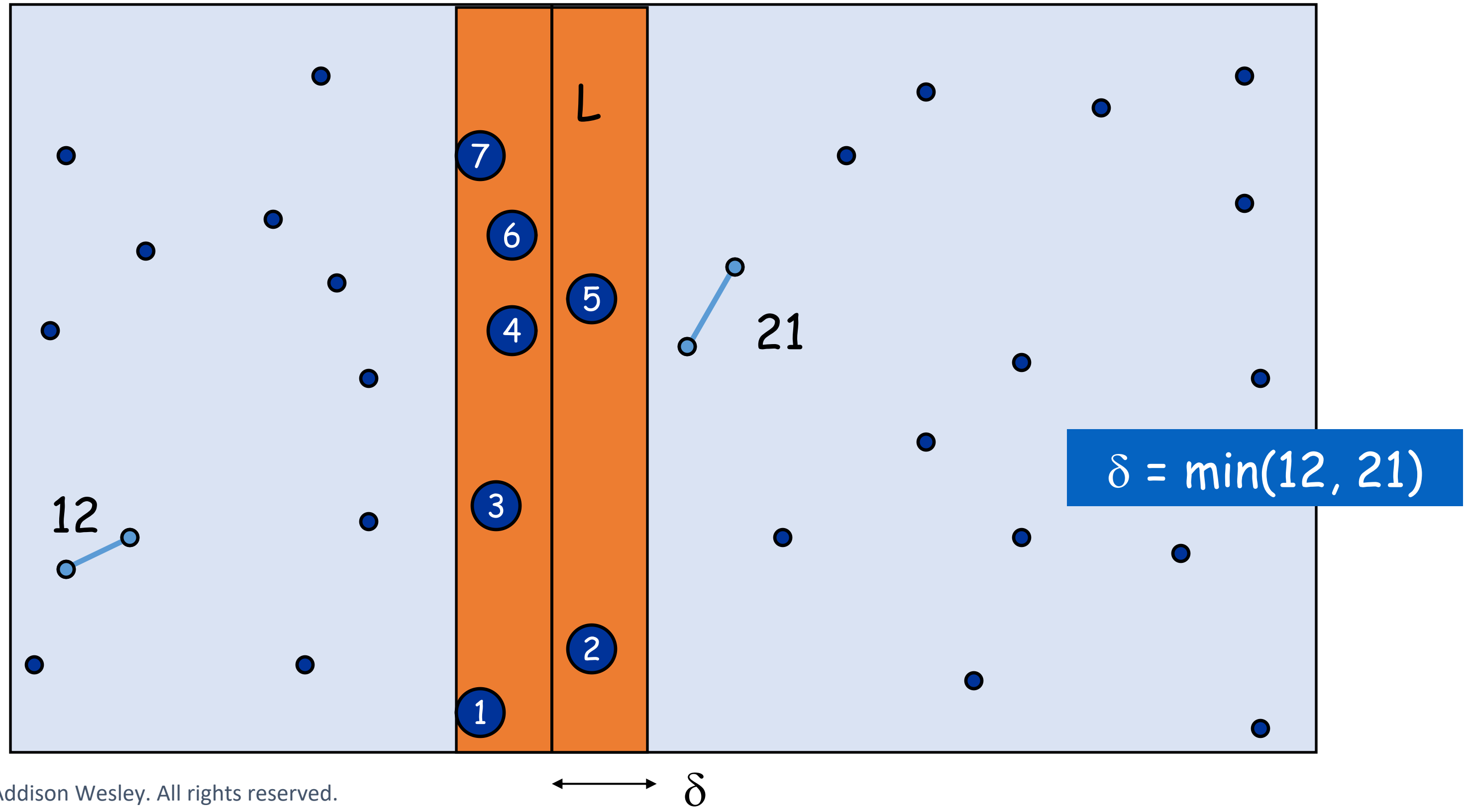
Find closest pair with one point in each side, assuming that distance < δ.

- Observation: only need to consider points within δ of line L.
- If $q \in Q, r \in R, d(q,r) < \delta$, then q, r in stripe $(L_x - \delta, L_x + \delta)$
- Because $L_x - q_x \leq r_x - q_x \leq d(q,r) < \delta$
- $r_x - L_x \leq r_x - q_x \leq d(q,r) < \delta$



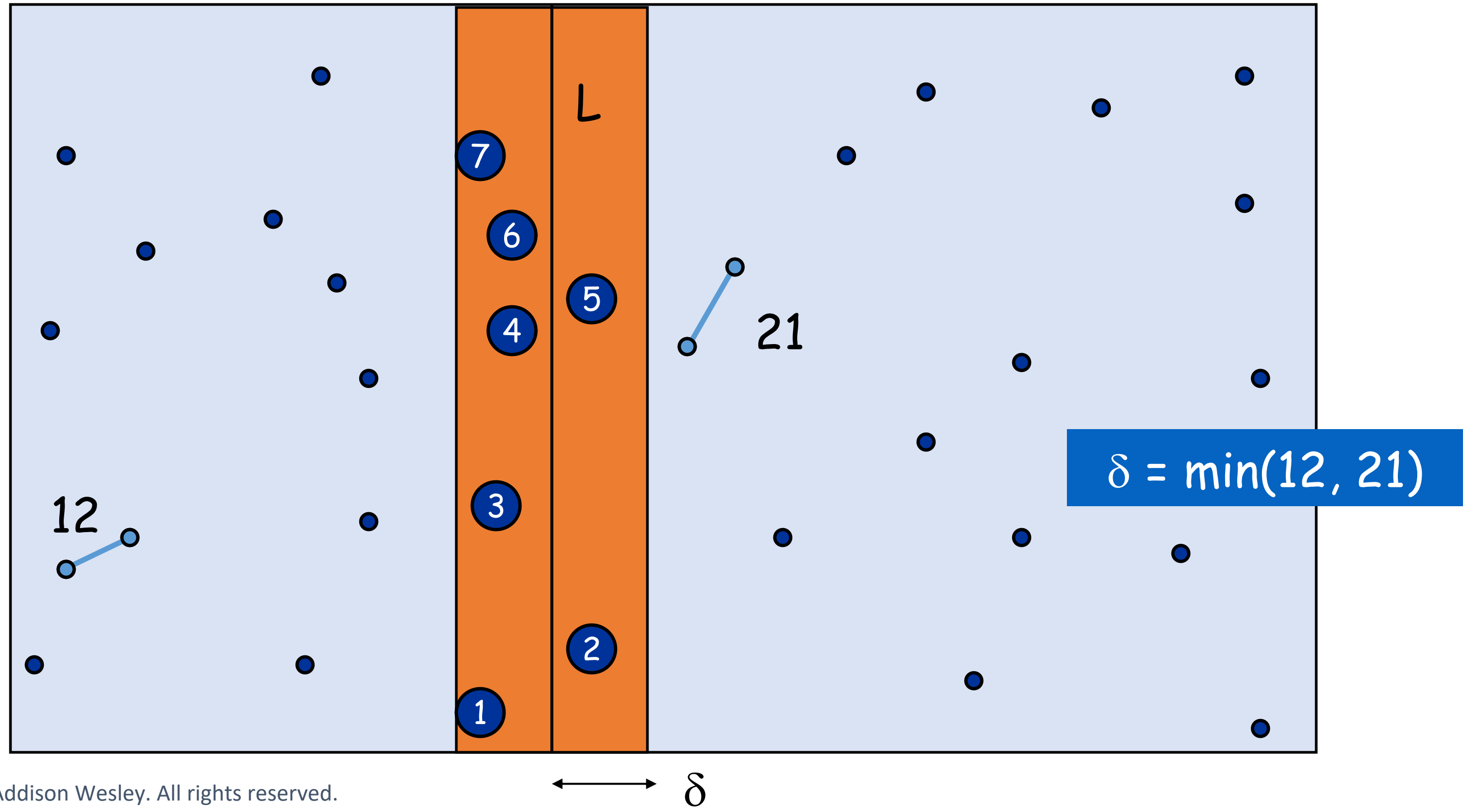
Closest Pair of Points

- Find closest pair with one point in each side, assuming that distance < δ.
  - 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 < δ.
  - 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!

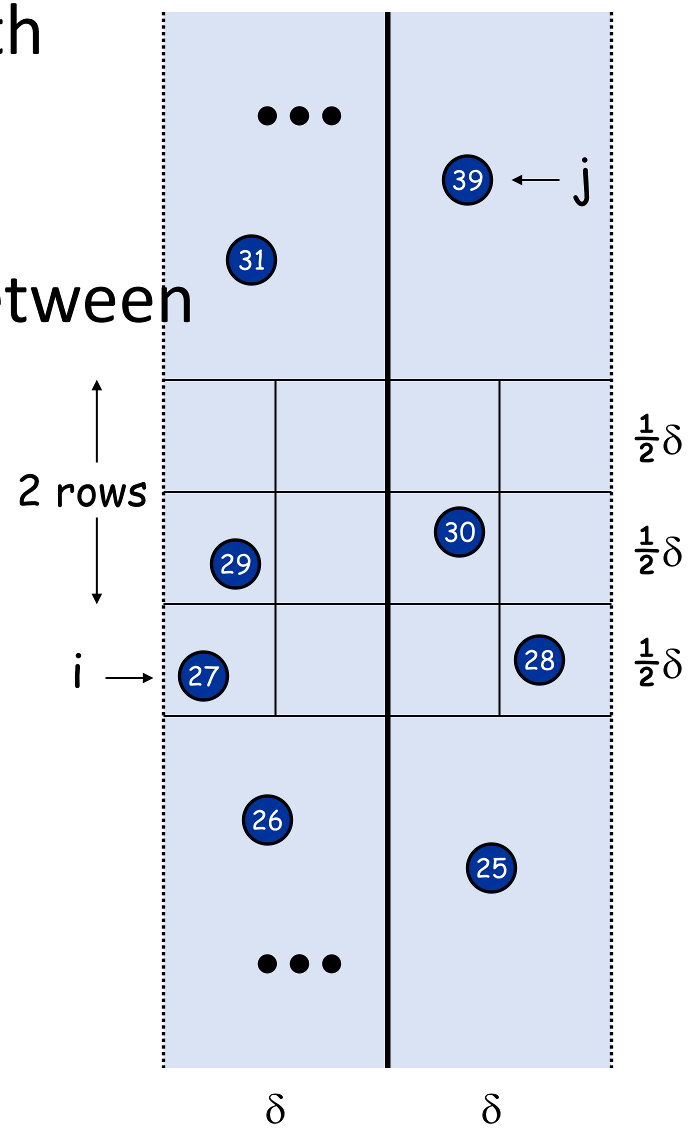


Closest Pair of Points

- 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(p1, …, pn) {
   Compute separation line L such that half the points      O(n log n)
   are on one side and half on the other side.

   δ1 = Closest-Pair(left half)                              2T(n / 2)
   δ2 = Closest-Pair(right half)
   δ  = min(δ1, δ2)

   Delete all points further than δ 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         O(n)
   distances is less than δ, update δ.

   return δ.
}
```

Running time $O(n \log n)$

sort by x axis and y axis $O(n \log n)$