南方科技大学
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

Algorithm Design and Analysis (H)

CS216

Instructor: Shan CHEN (陈杉)

chens3@sustech.edu.cn

(slides edited from Prof. Shiqi Yu)

Divide and Conquer

Divide-and-Conquer Paradigm

- **Divide-and-conquer.**
  - Divide up problem into several subproblems (of the same kind).
  - Solve (conquer) each subproblem recursively.
  - Combine solutions to subproblems into overall solution.
- **Most common usage.**
  - Divide problem of size n into two subproblems of size $n/2$.
  - Solve (conquer) two subproblems recursively.
  - Combine two solutions into overall solution.

O(n) time

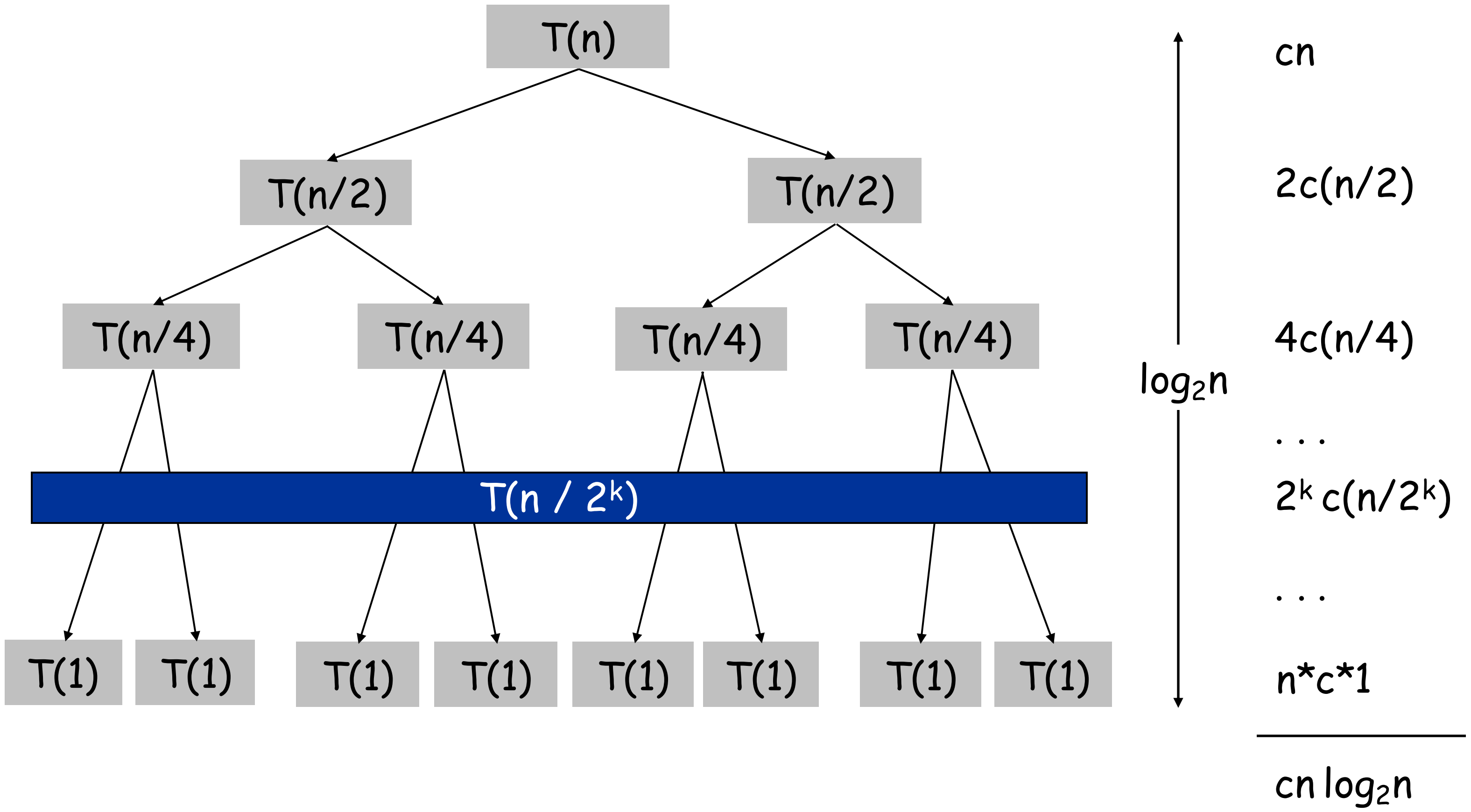
- **Time complexity.**
  - Brute force: $\Theta(n^2)$.
  - Divide-and-conquer: $O(n \log n)$. ← $T(n) = 2T(n/2) + O(n)$

凡治众如治寡 分数是也

孙子兵法

Divide-and-Conquer Recurrences

- **Example:** $T(n) = 2T(n/2) + O(n)$ with $T(1) = \Theta(1)$



Divide-and-Conquer Recurrences

- **Divide-and-conquer recurrences:** $T(n) = aT(n/b) + f(n)$ with $T(1) = \Theta(1)$

- **Master theorem.** Let $a \geq 1$, $b \geq 2$, and $c \geq 0$ and suppose that $T(n)$ is a function on the non-negative integers that satisfies the recurrence

$$T(n) = aT(n/b) + \Theta(n^c)$$

with $T(1) = \Theta(1)$, where n/b means either $\lfloor n/b \rfloor$ or $\lceil n/b \rceil$. Then,

  - **Case 1.** If $c > \log_b a$, then $T(n) = \Theta(n^c)$.
  - **Case 2.** If $c = \log_b a$, then $T(n) = \Theta(n^c \log n)$.
  - **Case 3.** If $c < \log_b a$, then $T(n) = \Theta(n^{\log_b a})$.

1. Counting Inversions

Counting Inversions

- **Match song preferences.**
  - Every person ranks *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* are 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

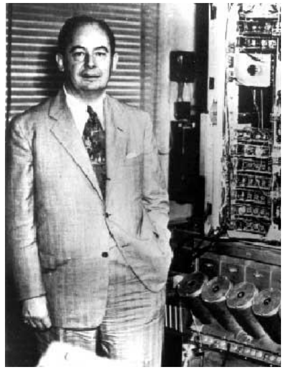
Inversions

3-2, 4-2

- **Brute force:** check all $\Theta(n^2)$ pairs.
- **Q.** Can we count inversions faster?

Recall: 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 $O(n)$
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 $O(n)$

total $O(n \log n)$

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.

easy to count if both sorted

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

Divide: O(n)

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

9 blue-green inversions

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

Combine: O(n) ?

how to count blue-green inversions ?

Total = 5 + 8 + 9 = 22.

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.

easy to count if both sorted

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

Divide: O(n)

1	2	4	5	8	10	3	6	7	9	11	12
						4	2	2	1	0	0

Conquer: 2T(n/2)

9 blue-green inversions: 4 + 2 + 2 + 1 + 0 + 0

Count: O(n)

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

Merge: O(n)

Counting Inversions: O(*n log n*) Algorithm

- **Divide-and-conquer algorithm:**

```
Sort-and-Count(L) {
   if list L has one element
      return (0, L)
   Divide the list into two halves A and B
      (rA, A) = Sort-and-Count(A)
      (rB, B) = Sort-and-Count(B)
      (rAB, L) = Merge-and-Count(A, B)
   return (rA + rB + rAB, L)
}
```

- **Pre-condition.** [Sort-and-Count] A and B are sorted.
- **Post-condition.** [Merge-and-Count] L is sorted.

2. Median and Selection

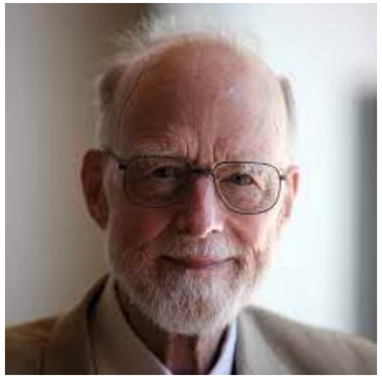
Median and Selection

- **Median and selection.** Given n elements from a totally ordered universe, find the median element or in general the k-th smallest.
  - Minimum or maximum: $k = 1$ or $k = n$.
  - Median: $k = \lfloor (n + 1) / 2 \rfloor$.
  - $O(n)$ compares for min or max.
  - $O(n \log n)$ compares by sorting.
  - $O(n \log k)$ compares with a binary heap.
- **Applications.** Order statistics, find the "top k", bottleneck paths, etc.

- **Q.** Can we do it with $O(n)$ compares?
- A. Yes! Selection is easier than sorting.

Recall: Randomized Quicksort

- **Randomized quicksort.**
  - Pick a random pivot element *p*.
  - 3-way partition the array into *L*, *M*, and *R*.
    - ✓L elements < p, M elements = p, R elements > p.
  - Recursively sort both *L* and *R*.



Tony Hoare (1959)

A L G O R I T H M S	pick random pivot $O(1)$
A L G I H M O R T S	3-way partition $O(n)$
A G H I L M O R S T	sort $T(\|L\|) + T(\|R\|)$
A G H I L M O R S T	total $O(n \log n)$ on average

Median and Selection: Divide-and-Conquer

- **Divide and conquer.**
  - Pick a random pivot element *p*.
  - 3-way partition the array into *L*, *M*, and *R*.
    - ✓L elements < p, M elements = p, R elements > p.
  - Recursively select in one subarray: the one containing the *k*-th smallest element.

A	L	G	O	R	I	T	H	M	S

pick random pivot $O(1)$

A	L	G	I	H	M		O		R	T	S

3-way partition $O(n)$

select $T(|L|)$ or $O(1)$ or $T(|R|)$

Randomized Quickselect

- **Divide and conquer.**
  - Pick a random pivot element *p*.
  - 3-way partition the array into *L*, *M*, and *R*.
    - ✓L elements < p, M elements = p, R elements > p.
  - Recursively select in one subarray: the one containing the *k*-th smallest element.

```
Quick-Select(A, k) { // 1 ≤ k ≤ |A|
   Pick pivot p uniformly at random from A
   Partition the list into two three parts L, M and R
   if k ≤ |L|
      return Quick-Select(L, k)
   else if k > |L| + |M|
      return Quick-Select(R, k - |L| - |M|)
   else
      return p
}
```

Randomized Quickselect: Time Complexity

- **Randomized quickselect.**
  - ➢ Pick a random pivot element p.
  - ➢ 3-way partition the array into L, M, and R.
    - ✓ L elements < p, M elements = p, R elements > p.
  - ➢ Recursively select in one subarray: the one containing the k-th smallest element.
- **Intuition.** Split length-n array uniformly ⇒ expected larger size ~ $3n/4$.
  - ➢ $T(n) \le T(3n/4) + n \Rightarrow T(n) \le 4n$
  - ➢ However, not rigorous because we cannot assume $\mathbf{E}(T(k)) \le T(\mathbf{E}(k))$.
- **Q.** What is the expected time complexity of randomized quickselect?
  - ➢ Time complexity is measured by the number of compares.

Randomized Quickselect: Time Complexity

- **Def.** $T(n, k)$ = expected number of compares to select the k-th smallest element in an array of length $\leq n$.
- **Def.** $T(n) = \max_k T(n, k)$.

- **Claim.** $T(n) \leq 4n$.
- Pf. (by strong induction on n)

can assume always recur in larger subarrays since $T(n)$ is monotone non-decreasing

$$
\begin{aligned}
T(n) &\leq n + 1/n\ [\ 2T(n/2) + \ldots + 2T(n-3) + 2T(n-2) + 2T(n-1)\ ] \\
&\leq n + 1/n\ [\ 8(n/2) + \ldots + 8(n-3) + 8(n-2) + 8(n-1)\ ] \\
&\leq n + 1/n\ (3n^2) \\
&= 4n.
\end{aligned}
$$

inductive hypothesis

More on Median and Selection

- We learned that randomized quickselection runs in $O(n)$ on average.

- [Blum–Floyd–Pratt–Rivest–Tarjan 1973] There exists a compare-based deterministic selection algorithm whose worst-case running time is $O(n)$.
  - This algorithm is also known as the median of medians.
  - Optimized version requires $\leq 5.4305n$ compares.

- In practice, we use randomized selection algorithms since deterministic algorithms have too large constants.
  - However, deterministic algorithms can be used as a fallback for pivot selection.

3. Closest Pair of Points

Closest Pair of Points

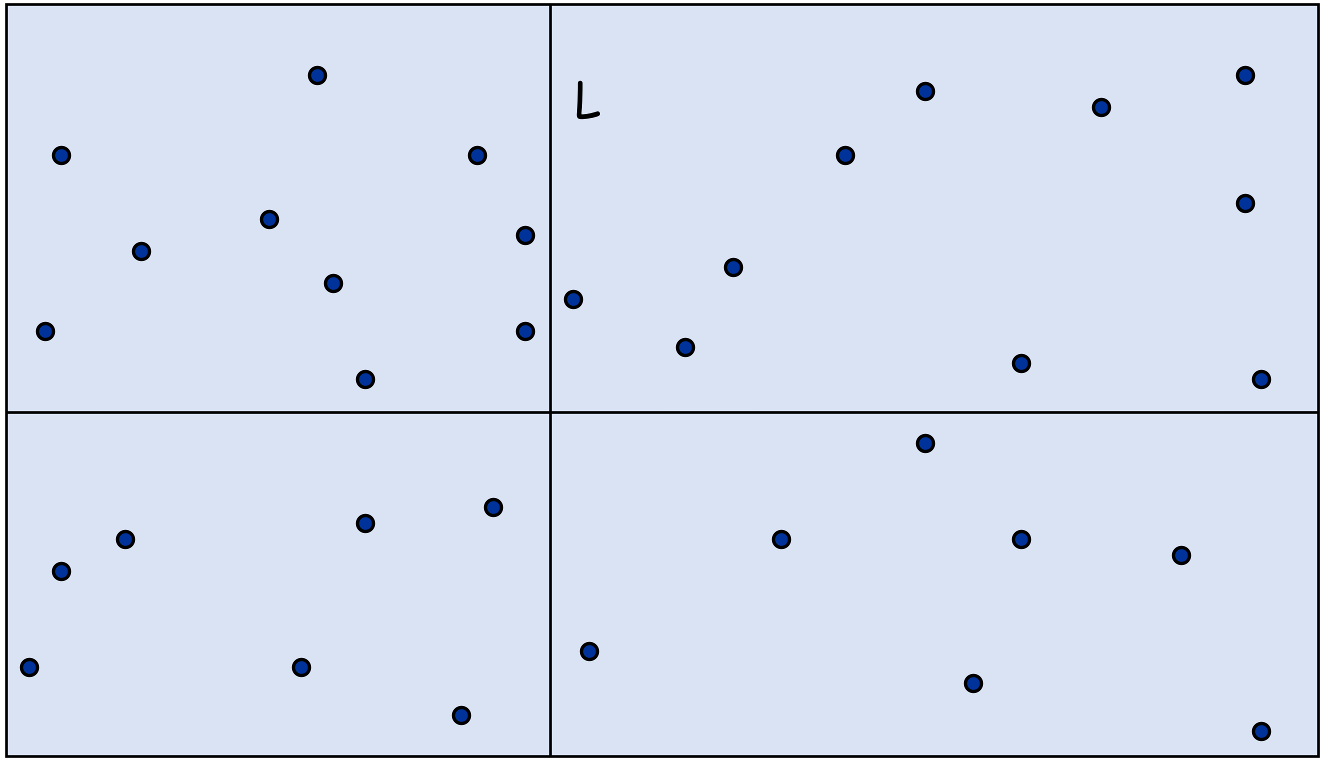
- **Closest pair problem.** Given n points in the plane, find a pair with smallest Euclidean distance between them.
- **Fundamental geometric primitive.**
  - Widely used in graphics, computer vision, geographic information systems, molecular modeling, air traffic control, etc.
  - Special case of nearest neighbor, Euclidean MST, Voronoi diagram, etc.

fast closest pair inspired fast algorithms for these problems

- **Brute force.** Check all pairs with $\Theta(n^2)$ distance calculations.
- **1-D version.** Easy $O(n \log n)$ algorithm if points are on a line.

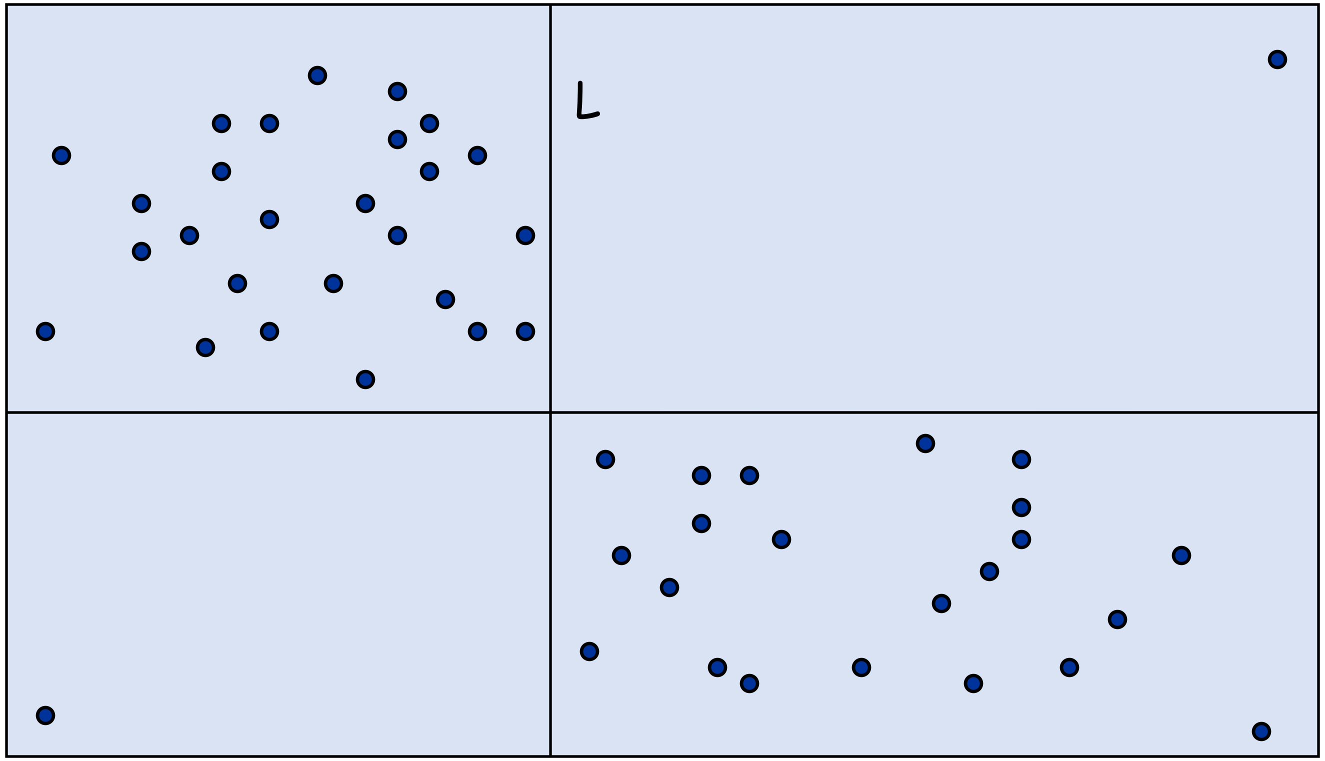
Closest Pair of Points: First Attempt

- **Divide.** Sub-divide region into 4 quadrants.



Closest Pair of Points: First Attempt

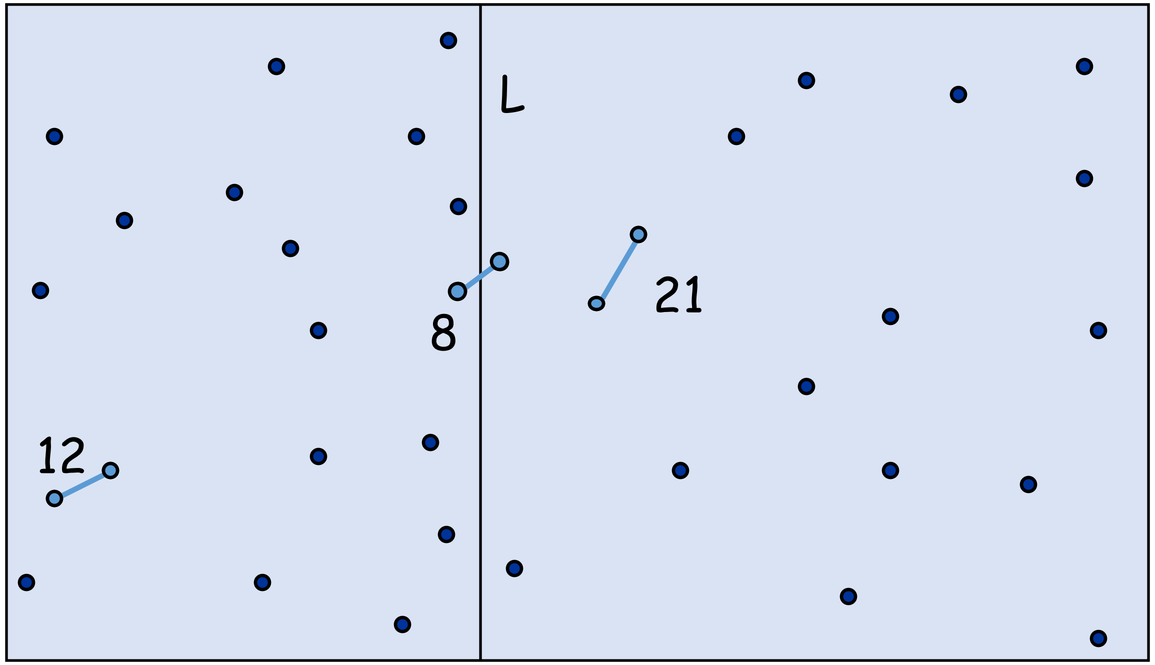
- **Divide.** Sub-divide region into 4 quadrants.
- **Obstacle.** Impossible to ensure n/4 points in each piece.



Closest Pair of Points: Divide and Conquer

- **Divide and conquer.**
  - Divide: draw vertical line L so that roughly n/2 points lie in each side.
  - Conquer: find closest pair in each side recursively.
  - Combine: find closest pair with one point in each side.
  - Return best of 3 solutions.

can we beat $\Theta(n^2)$?

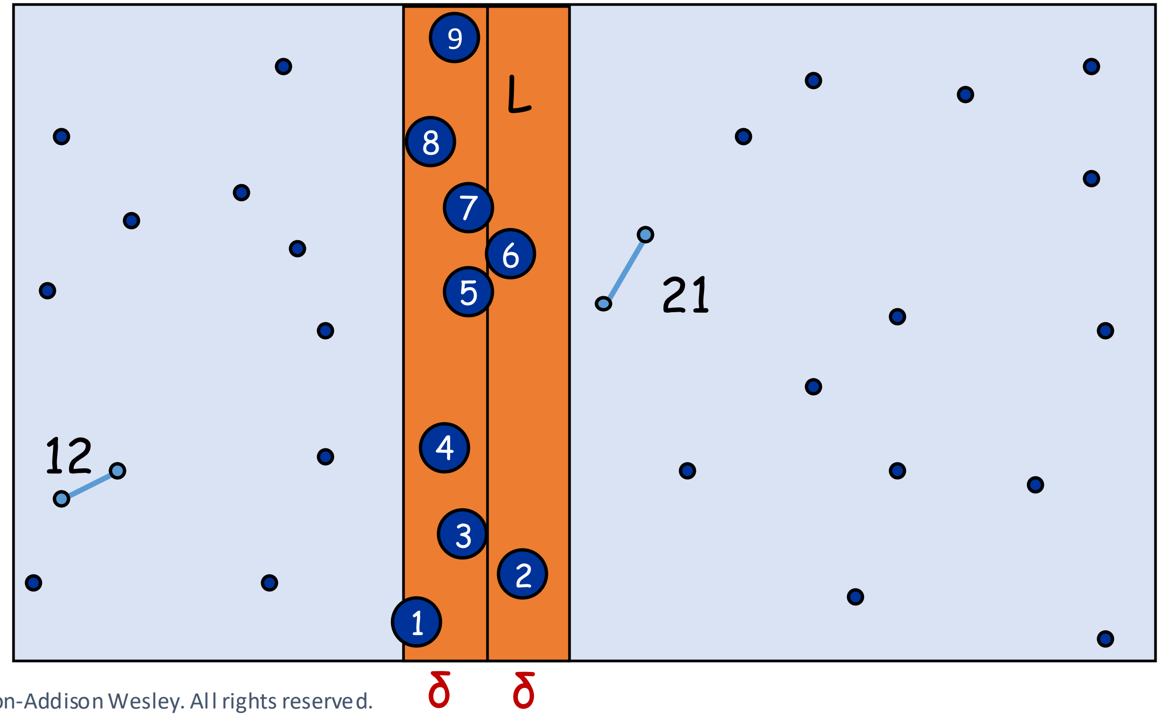


Closest Pair of Points: Divide and Conquer

- **Combine:** Find closest pair with one point in each side.
  - Observation: suffices to consider only points within δ of line *L*, where δ is the distance of closest pair with both points in one side.
  - Sort points in 2δ-strip by their *y*-coordinate.
  - Check distances of only those points within 7 positions in sorted list!

known from recursion

why?

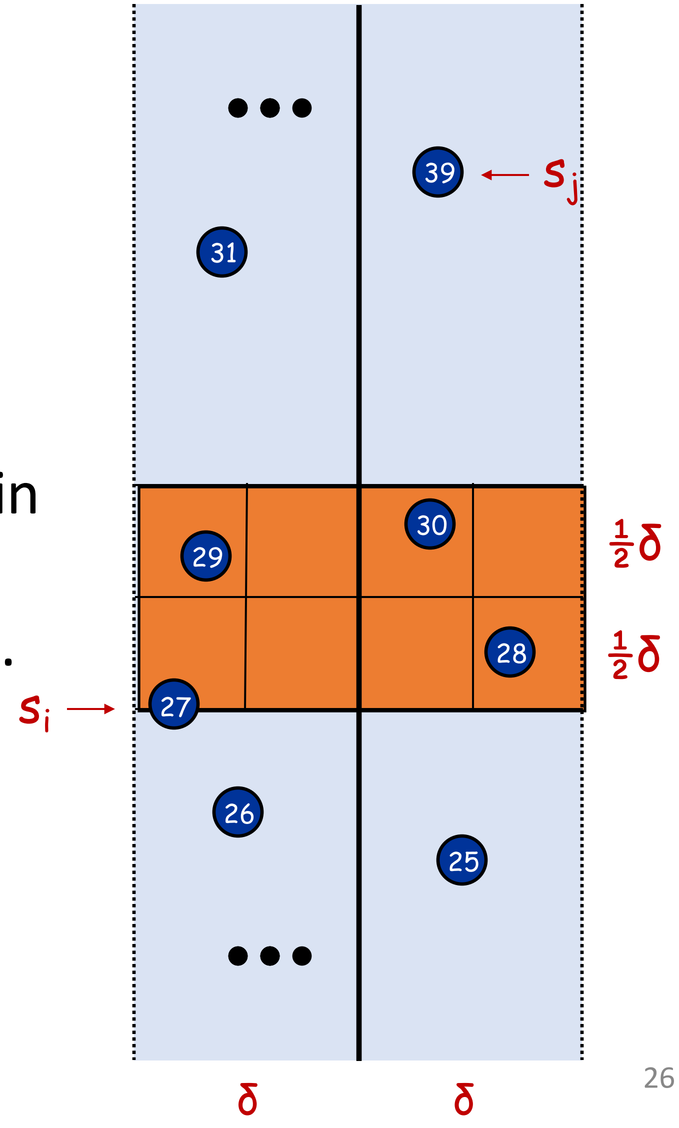


δ = min(12, 21)

Closest Pair of Points: Divide and Conquer

- **Def.** Let s_i be the point in the 2δ-strip, with the i-th smallest y-coordinate.
- **Claim.** If $|i - j| > 7$, then the distance between s_i and s_j is at least δ.
- Pf.
  - Consider the 2δ-by-δ rectangle R in strip whose min y-coordinate is y-coordinate of s_i.
  - Distance between s_i and any point s_j above R is ≥ δ.
  - Subdivide R into 8 squares.
  - At most 1 point per square. square diameter < δ
  - At most 7 other points can be in R. ▪

constant can be improved
with more refined argument



Closest Pair of Points: $O(n \log n)$ Algorithm

- **Divide-and-conquer algorithm:** [sort by x-axis and y-axis beforehand]

```
Closest-Pair(p1, …, pn) {
   Compute vertical line L such that half the points
   are on one side and half on the other side.

   δ1 = Closest-Pair(left half)
   δ2 = Closest-Pair(right half)
   δ  = min(δ1, δ2)

   Delete all points further than δ from line L

   Sort remaining points by y-coordinate.

   Scan points in y-order and compare distance between
   each point and next 7 neighbors. If any of these
   distances is less than δ, update δ.

   return δ.
}
```

use x-sorted list

$O(n)$

$2T(n / 2)$

$O(n)$

$O(n)$

use y-sorted list

$O(n)$

More on Closest Pair of Points

- [Rabin 1976] There exists an algorithm to find the closest pair of points in the plane whose expected running time is $O(n)$.

- There are divide-and-conquer algorithms that solve the following core 2D geometric problems in $O(n \log n)$ time:
  - Farthest pair
  - Convex hull
  - Delaunay/Voronoi
  - Euclidean MST