5.4 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

Q. How much comparisons do we need in a brute-force method?

Q. How much comparisons do we need if points are on a line?

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

Q. How much comparisons do we need in a brute-force method?

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

Q. How much comparisons do we need if points are on a line?

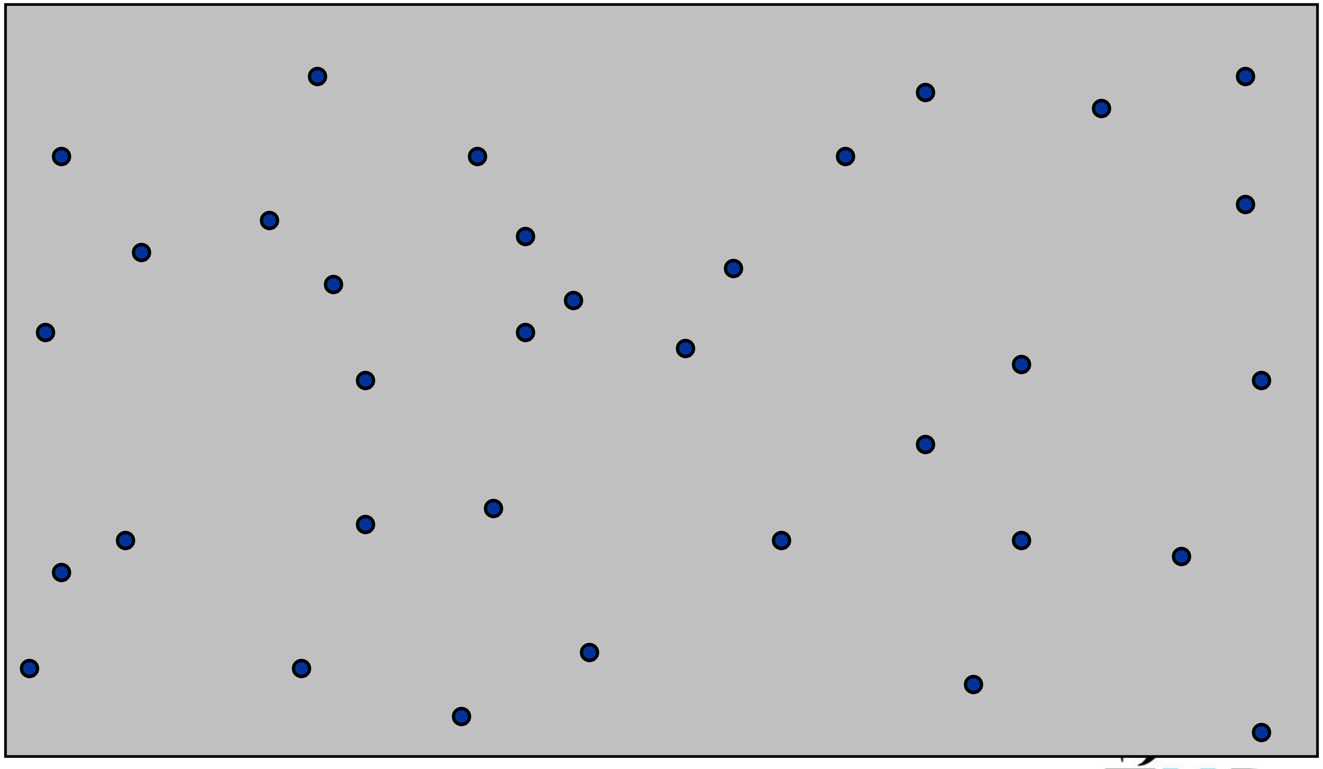
A. O(n log n): easy if points are on a line (or O(n) if already sorted).

to make presentation cleaner

Assumption. No two points have same x coordinate.

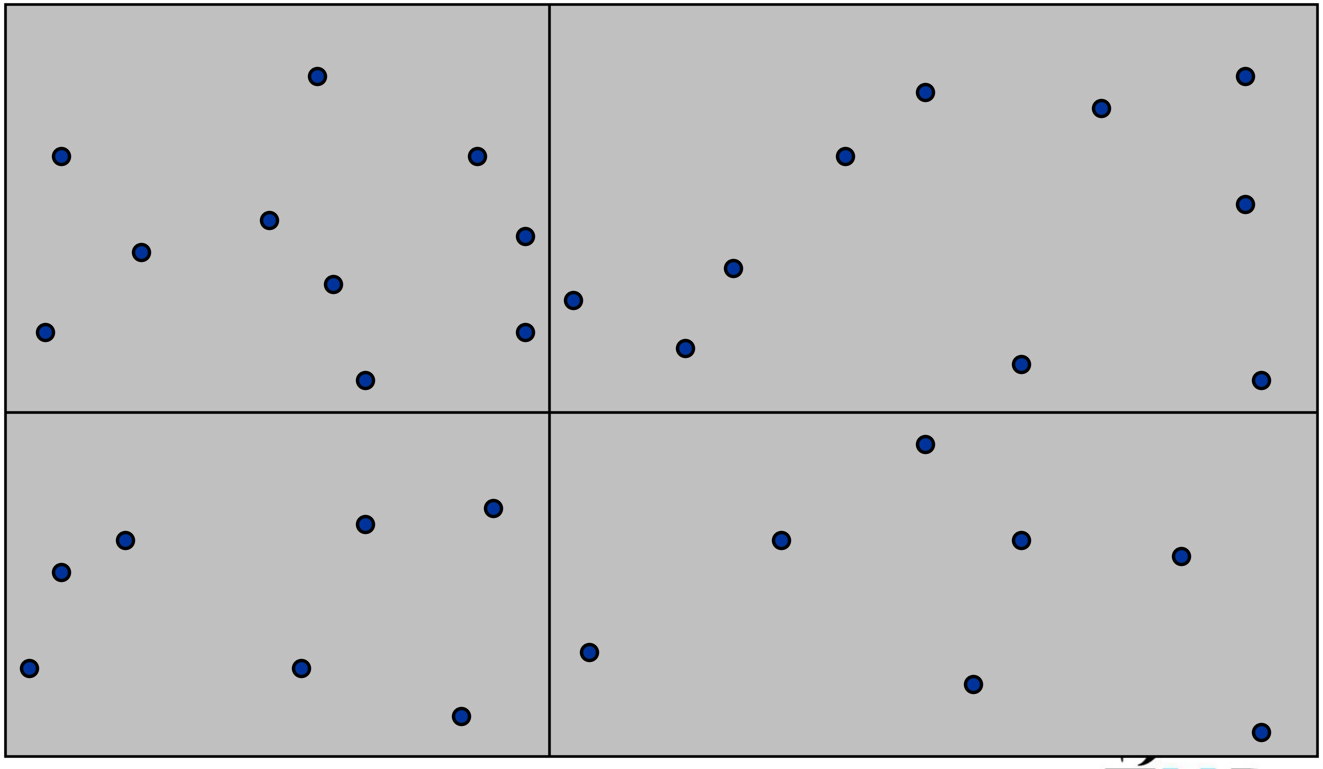
Closest Pair of Points: First Attempt

Q. How would a divide & conquer approach look like?



Closest Pair of Points: First Attempt

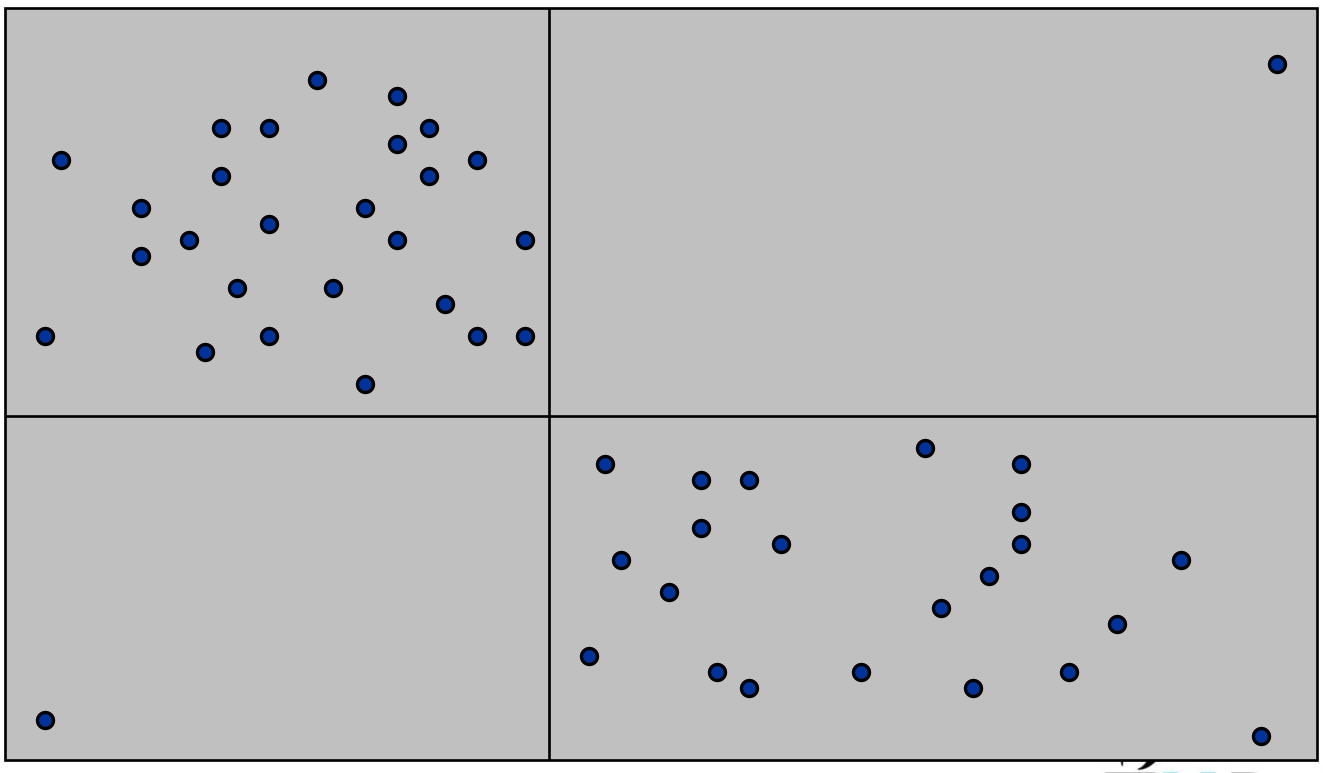
Divide. Sub-divide region into 4 quadrants.



Closest Pair of Points: First Attempt

Divide. Sub-divide region into 4 quadrants.

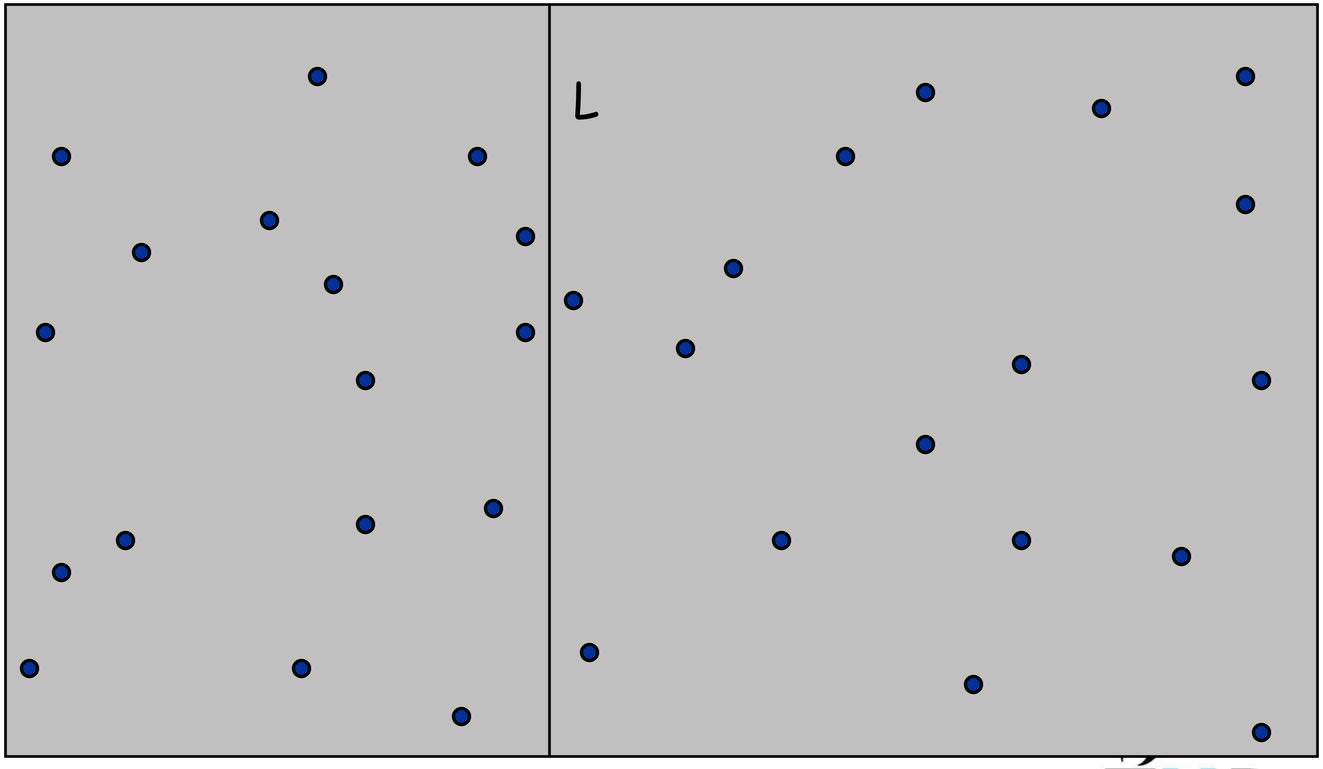
Obstacle. Impossible to ensure n/4 points in each piece.



Closest Pair of Points

Algorithm.

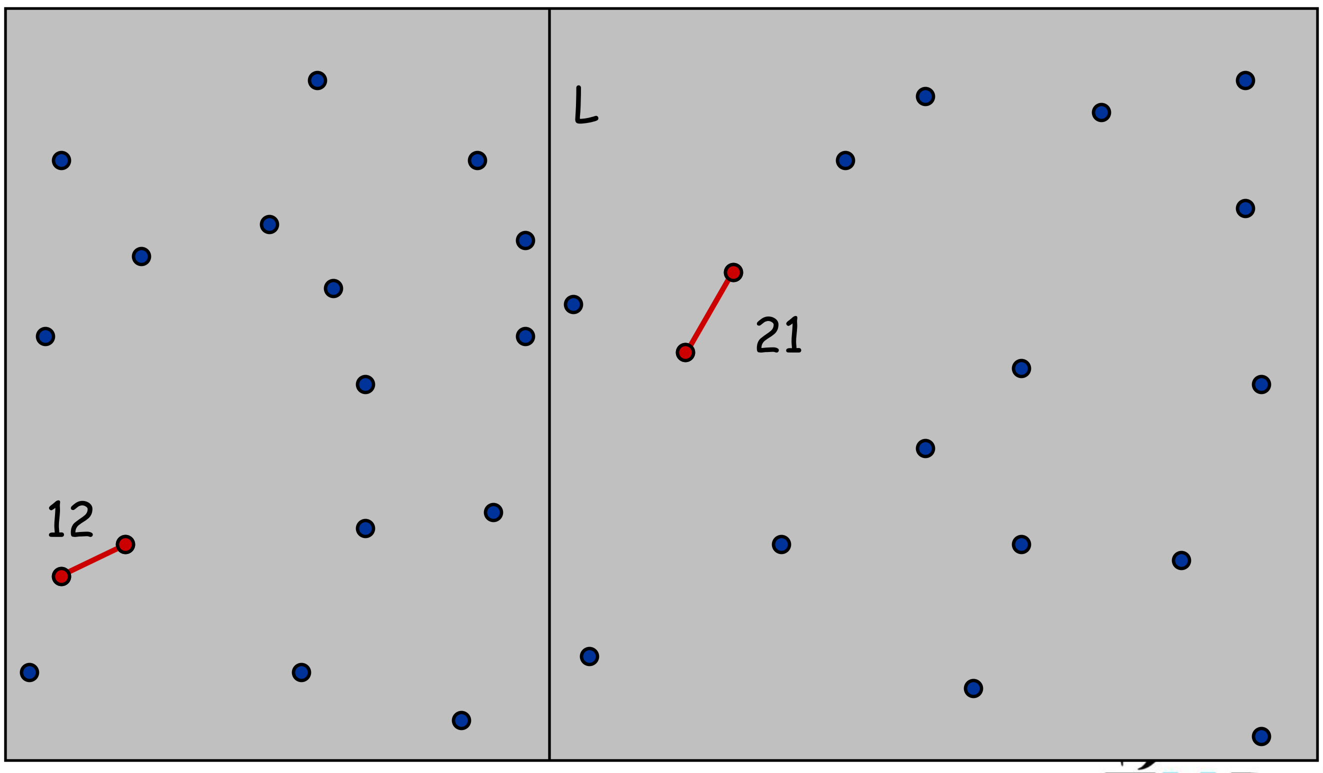
- Divide: draw vertical line L so that roughly ½n points on each side.



Closest Pair of Points

Algorithm.

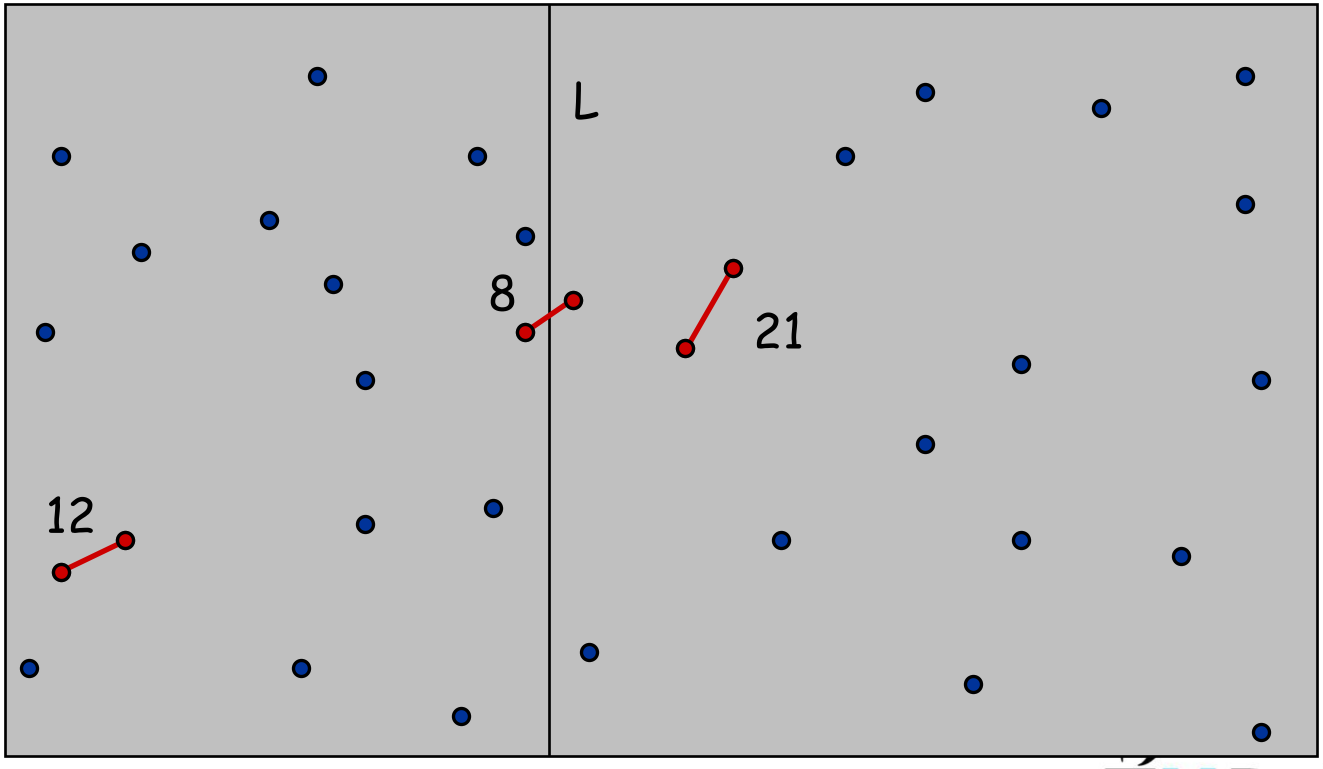
- Divide: draw vertical line L so that roughly ½n points on each side.
- Conquer: find closest pair in each side recursively.



Closest Pair of Points

Algorithm.

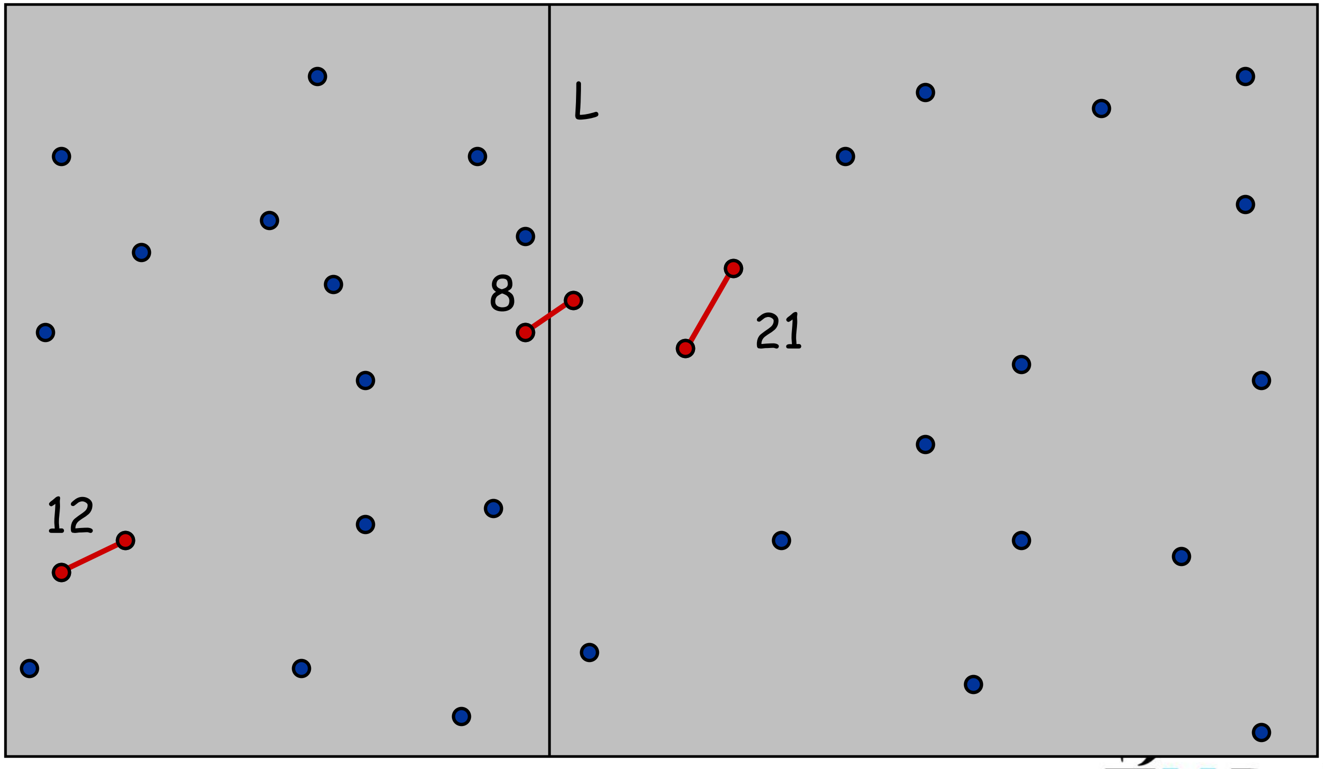
- Divide: draw vertical line L so that roughly ½n points on each side.
- Conquer: find closest pair in each side recursively.
- Combine?: find closest pair with one point on each side.
- Return best of 3 solutions.



Closest Pair of Points

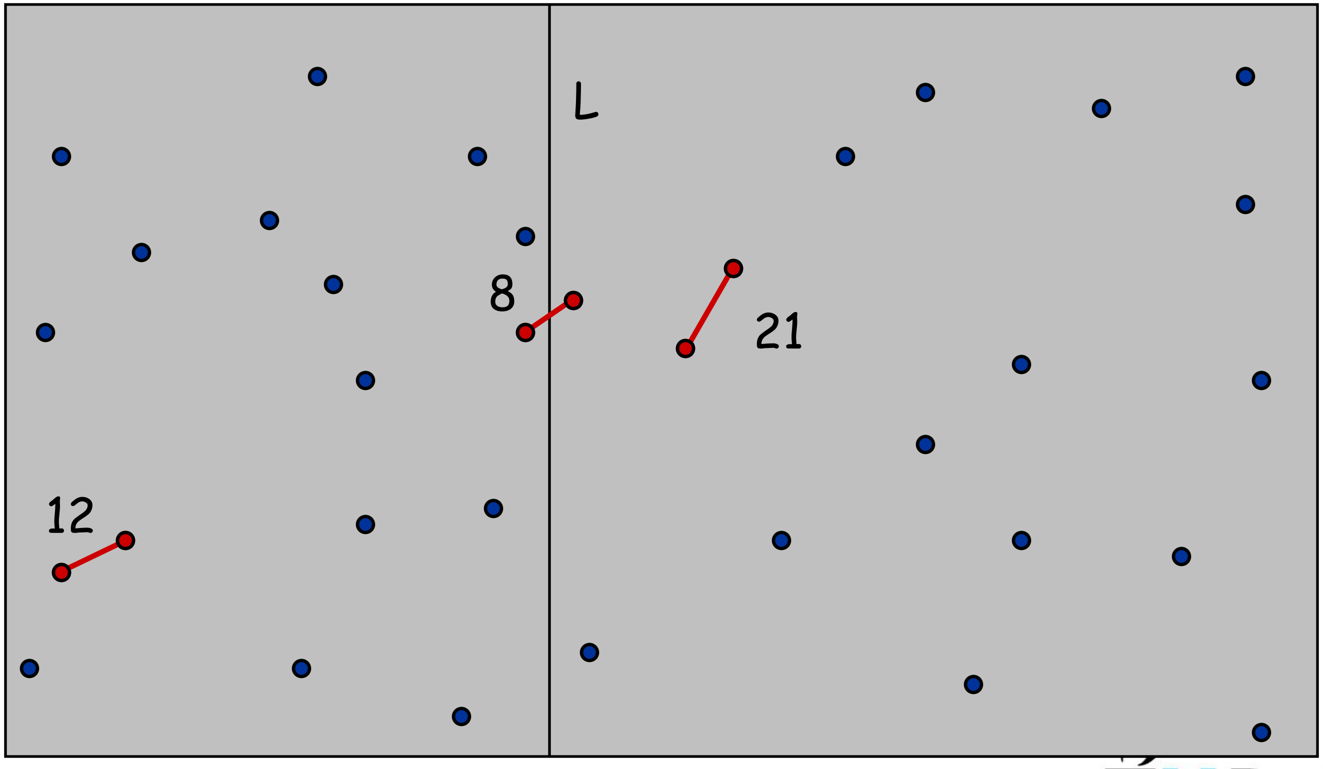
Algorithm.

- 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 on each side. ← seems like $\Theta(n^2)$
- Return best of 3 solutions.



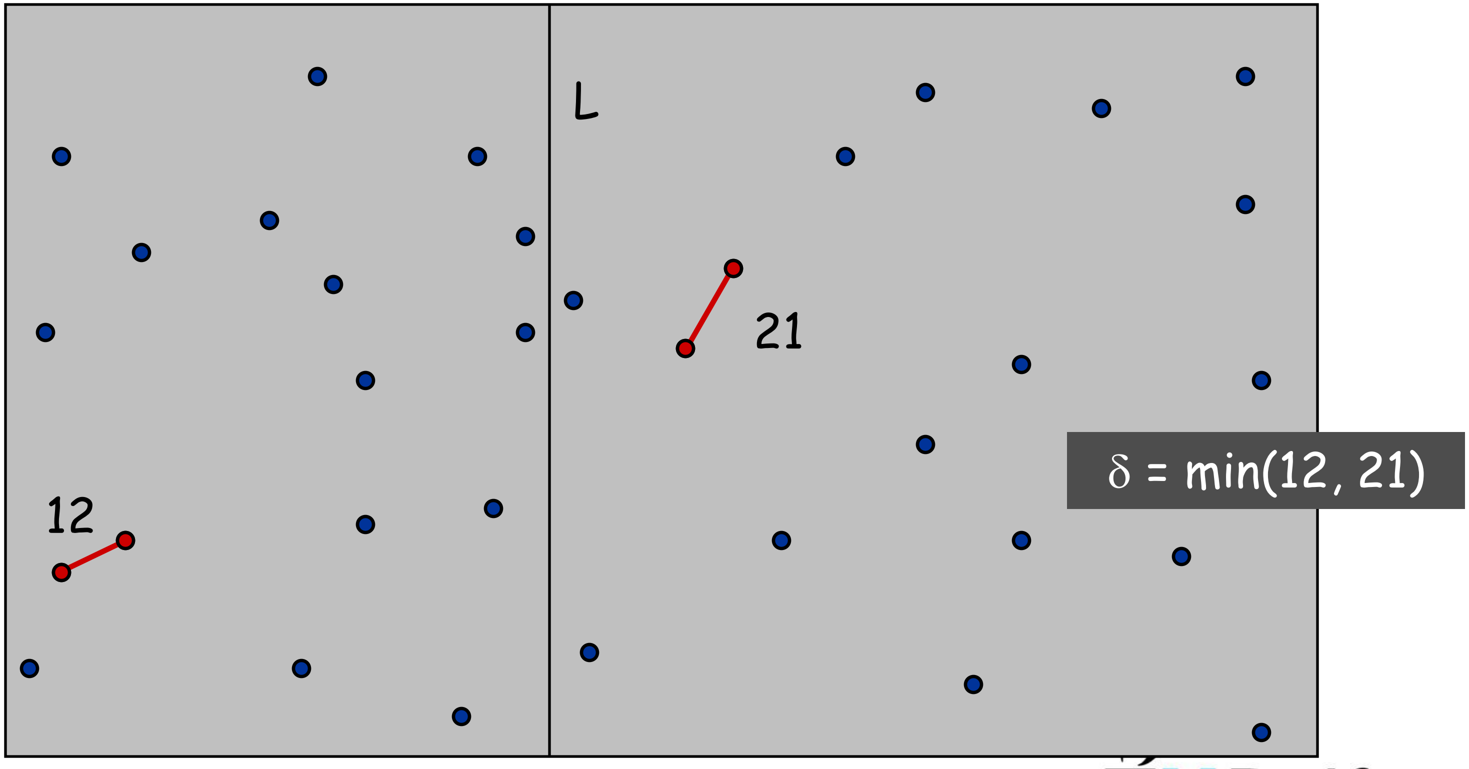
Closest Pair of Points

Q. To combine, do we need to consider all points on the left and all points on the right of L?



Closest Pair of Points

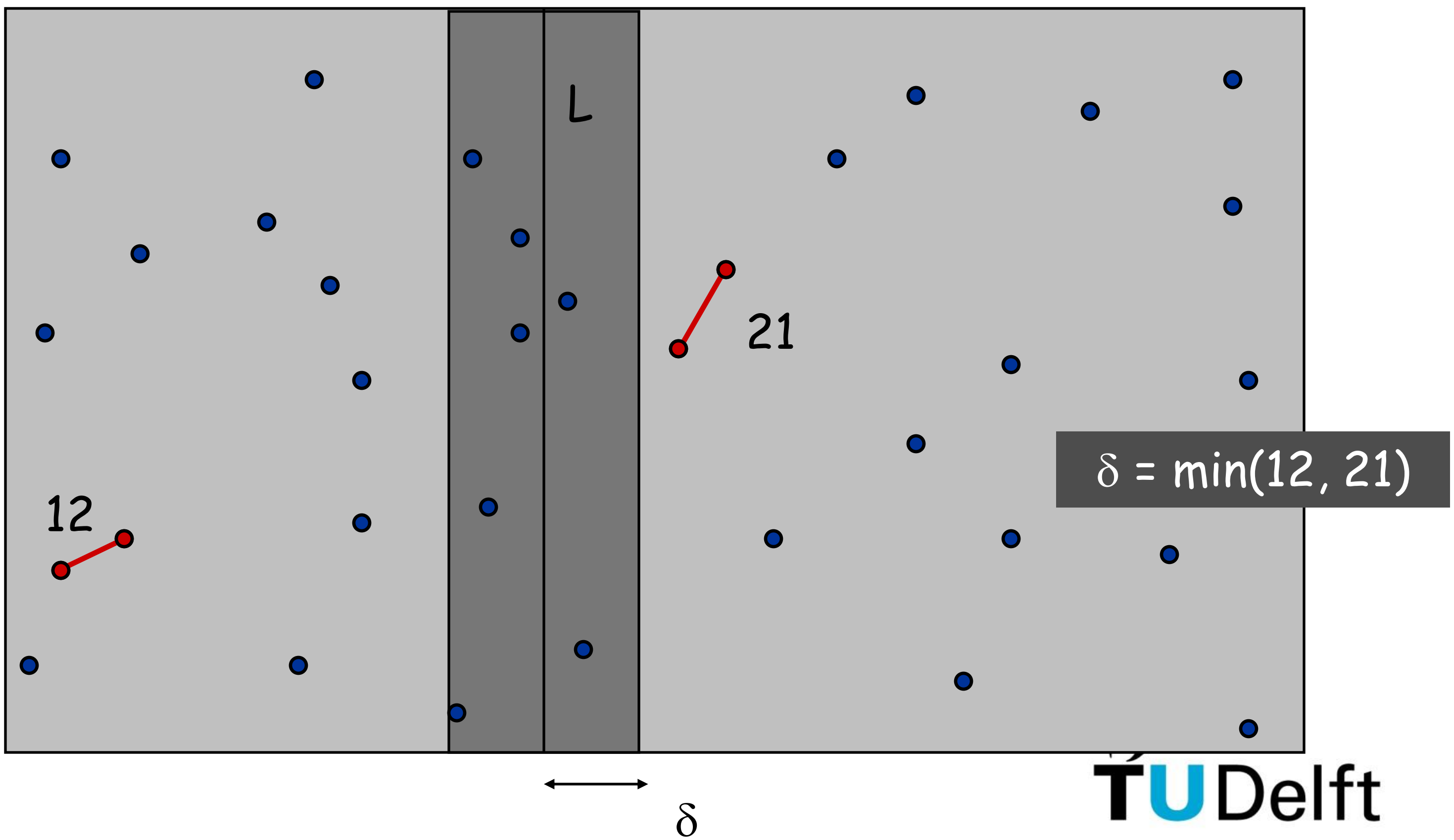
Find closest pair with one point on each side, assuming that distance < δ.



Closest Pair of Points

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

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

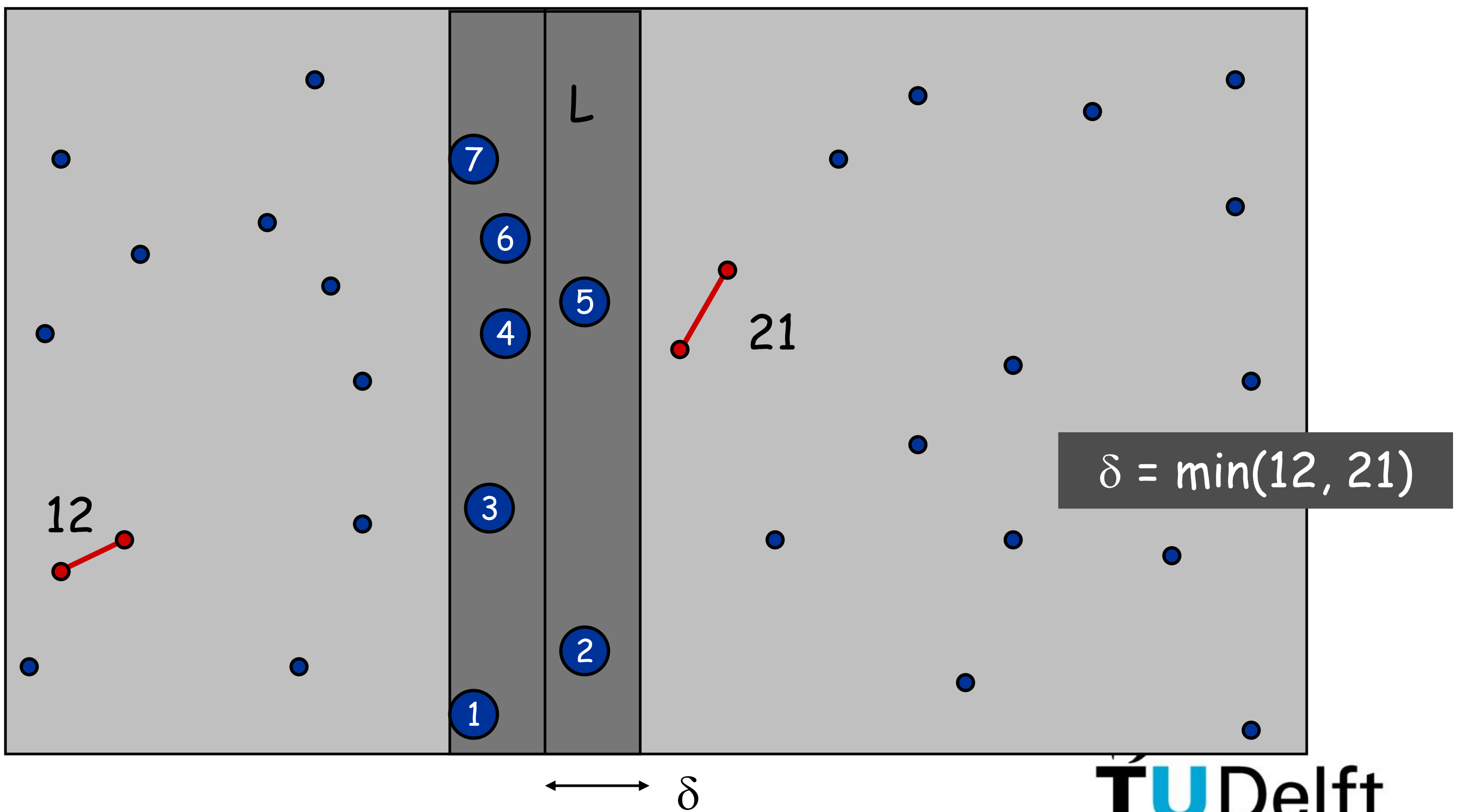


Closest Pair of Points

Find closest pair with one point on 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.

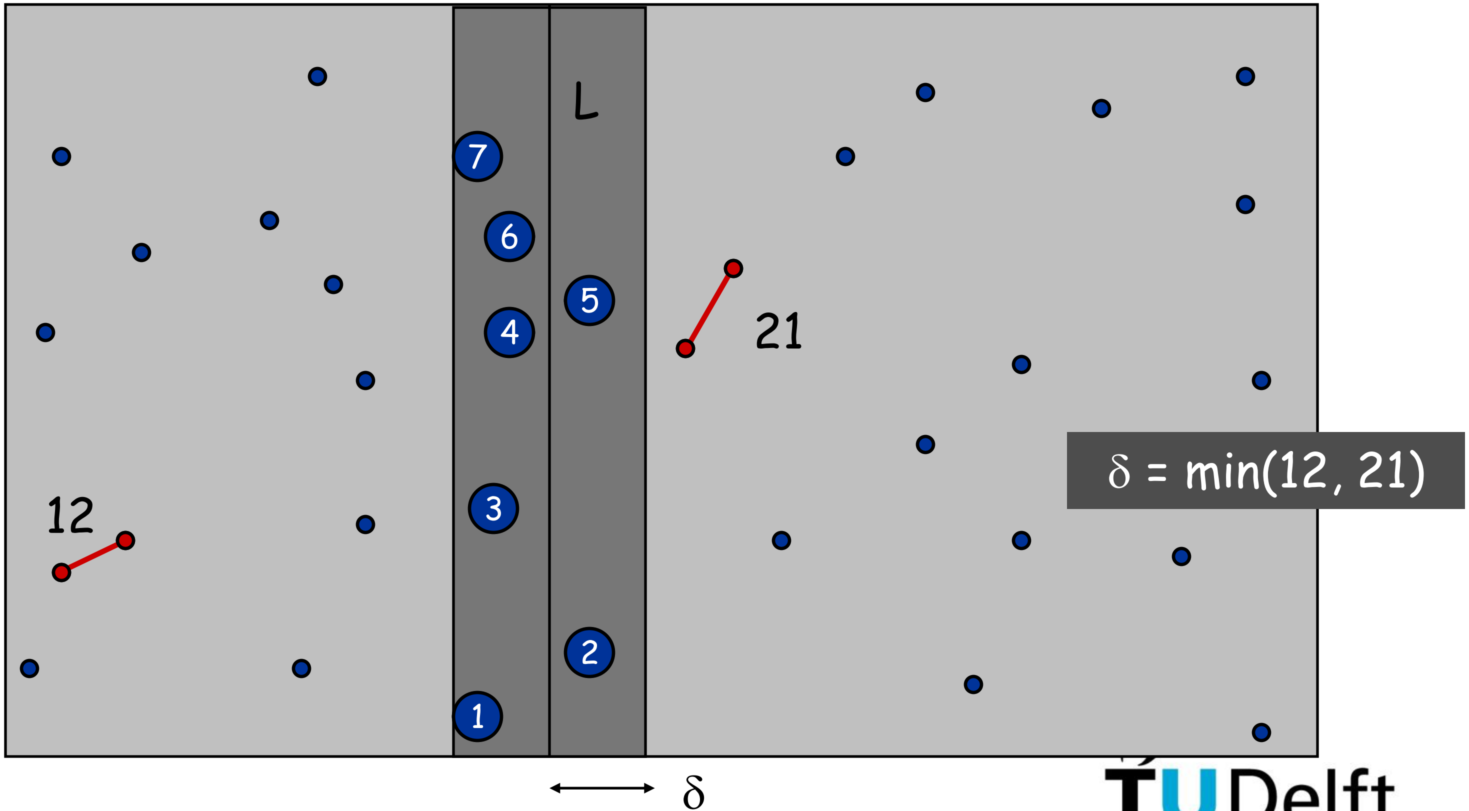
Q. Do we need to consider all points in this strip?



Closest Pair of Points

Find closest pair with one point on 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!



TU Delft

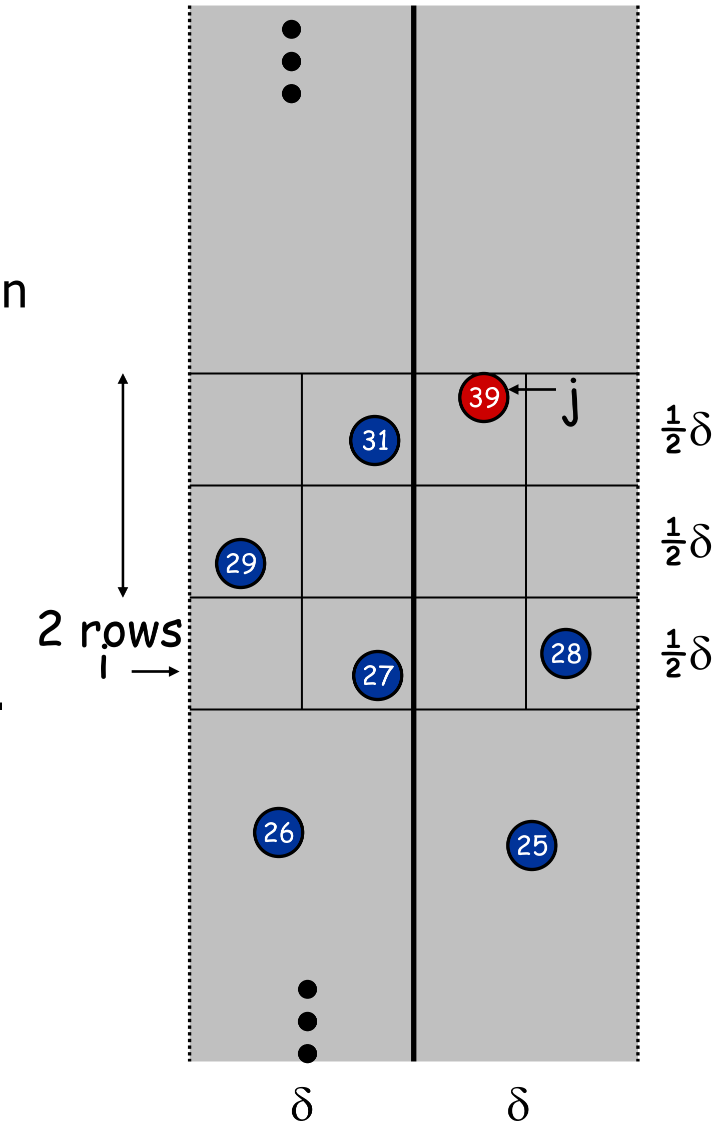
Closest Pair of Points

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

Claim. If $|i - j| > 11$, 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 $\geq 2(\frac{1}{2}\delta)$.
- Only consider points within 0, 1, or 2 rows. ▪



Fact. Still true if we replace 11 with 7.

Fact. Or even less if we consider left and right columns separately (e.g. 6).

TU Delft

Closest Pair Algorithm

```
Closest-Pair(p1, …, pn) {
   if n = 1
       return infinity (MAXINT)

   Compute separation 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 separation
line L
   Sort remaining points by y-coordinate.

   Scan points in y-order and compare distance
between
   each point and next 11 neighbors. If any of
these
   distances is less than δ, update δ.
```

Q. What is the run-time of this algorithm? (1 min)

TU Delft

Closest Pair Algorithm

```
Closest-Pair(p1, …, pn) {
   if n = 1
       return infinity (MAXINT)

   Compute separation 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 separation
line L
   Sort remaining points by y-coordinate.

   Scan points in y-order and compare distance
between
   each point and next 11 neighbors. If any of
these
   distances is less than δ, update δ.
```

2T(n / 2)

O(n)

O(n log n)

O(n)

Q. What is the run-time of this algorithm? (1 min)

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. Improve this algorithm to obtain a runtime of O(n log n).

A.

Q. What then should be the run-time of one call to **`Closest-Pair`**?

TU Delft

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. Improve this algorithm to obtain a runtime of O(n log n).

A. 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 (with mutual links).
- (Or sort up front, and make selection in O(n) time.)

Similarly, solving the problem of finding the convex hull.

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