UVa Online Judge

Bey Battle

Time Limit: 4 Second

Dark Blader has returned with new tactics. They are now able to create an energy field around its blade and if any blade enters inside any of these energy fields the energy level of the Bit-beast drastically decreases. So Tyson had to avoid these energy fields and finally he has won!

I was lucky enough to be around Kenny who was analyzing the game with his PC and helping Tyson to avoid the energy field. I saw that the energy field was Square in shape and the blade was at its centre. At that instant a problem came to my mind and let me see how efficiently you can solve that problem.

There will be N points in a 2D plane. Find out the maximum size such that if you draw such size squares around each point (that point will be at the center of the square) no two squares will intersect each other (can touch but not intersect). To make the problem simple the sides of the squares will be parallel to X and Y axis.

[http://uva.onlinejudge.org/external/113/11378.pdf]

Divide and conquer

CPSC490 2014/15WT2

Nasa Rouf

UVa Online Judge

Bey Battle

Time Limit: 4 Second

Dark Blader has returned with new tactics. They are now able to create an energy field around its blade and if any blade enters inside any of these energy fields the energy level of the Bit-beast drastically decreases. So Tyson had to avoid these energy fields and finally he has won!

I was lucky enough to be around Kenny who was analyzing the game with his PC and helping Tyson to avoid the energy field. I saw that the energy field was Square in shape and the blade was at its centre. At that instant a problem came to my mind and let me see how efficiently you can solve that problem.

There will be N points in a 2D plane. Find out the maximum size such that if you draw such size squares around each point (that point will be at the center of the square) no two squares will intersect each other (can touch but not intersect). To make the problem simple the sides of the squares will be parallel to X and Y axis.

[http://uva.onlinejudge.org/external/113/11378.pdf]

Divide and conquer

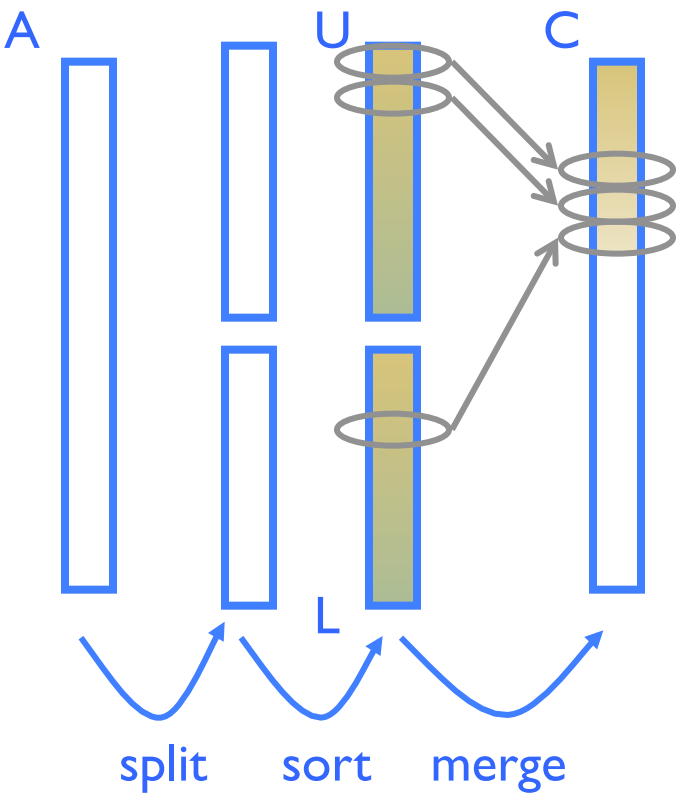
CPSC490 2014/15WT2

Nasa Rouf

merge sort

```
MS(A: array[1..n]) returns array[1..n] {
    If(n=1) return A;
    New U:array[1:n/2] = MS(A[1..n/2]);
    New L:array[1:n/2] = MS(A[n/2+1..n]);
    Return(Merge(U,L));
    }

Merge(U,L: array[1..n]) {
    New C: array[1..2n];
    a=1; b=1;
    For i = 1 to 2n
        C[i] = "smaller of U[a], L[b] and correspondingly a++ or b++";
    Return C;
    }
```



Alternative "divide & conquer" algorithm:

Sort n-1

Sort last 1

Merge them

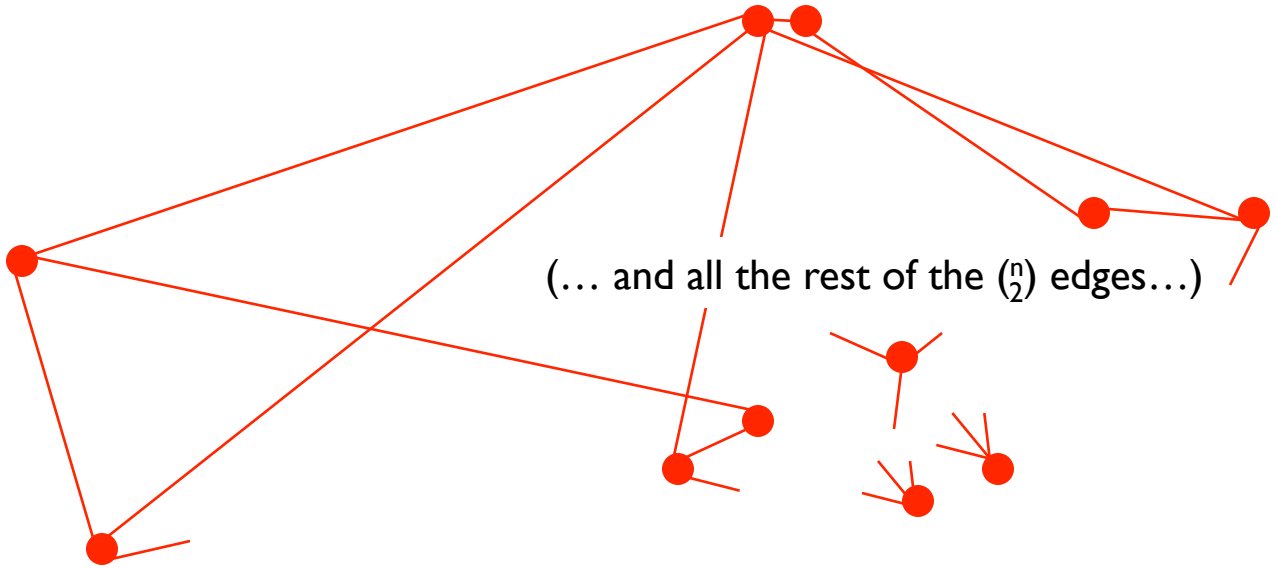
$T(n) = T(n-1)+T(1)+3n \quad \text{for } n \geq 2$

$T(1) = 0$

Solution: $3n + 3(n-1) + 3(n-2) \ldots = \Theta(n^2)$

closest pair of points: non-geometric version

Given n points and *arbitrary* distances between them, find the closest pair. (E.g., think of distance as airfare – definitely not Euclidean distance!)

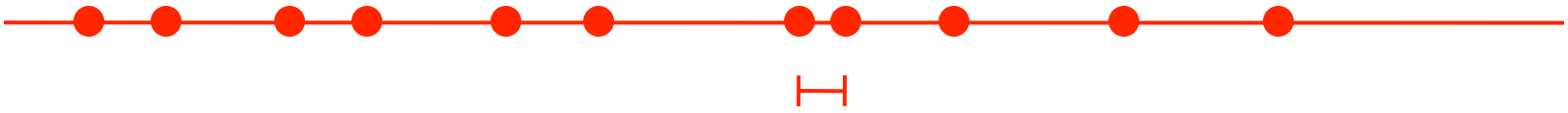


Must look at all n choose 2 pairwise distances, else any one you didn't check might be the shortest.

Also true for Euclidean distance in 1-2 dimensions?

closest pair of points: 1 dimensional version

Given n points on the real line, find the closest pair



Closest pair is *adjacent* in ordered list

Time O(n log n) to sort, if needed

Plus O(n) to scan adjacent pairs

Key point: do *not* need to calc distances between all pairs: exploit geometry + ordering

closest pair of points: 2 dimensional version

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

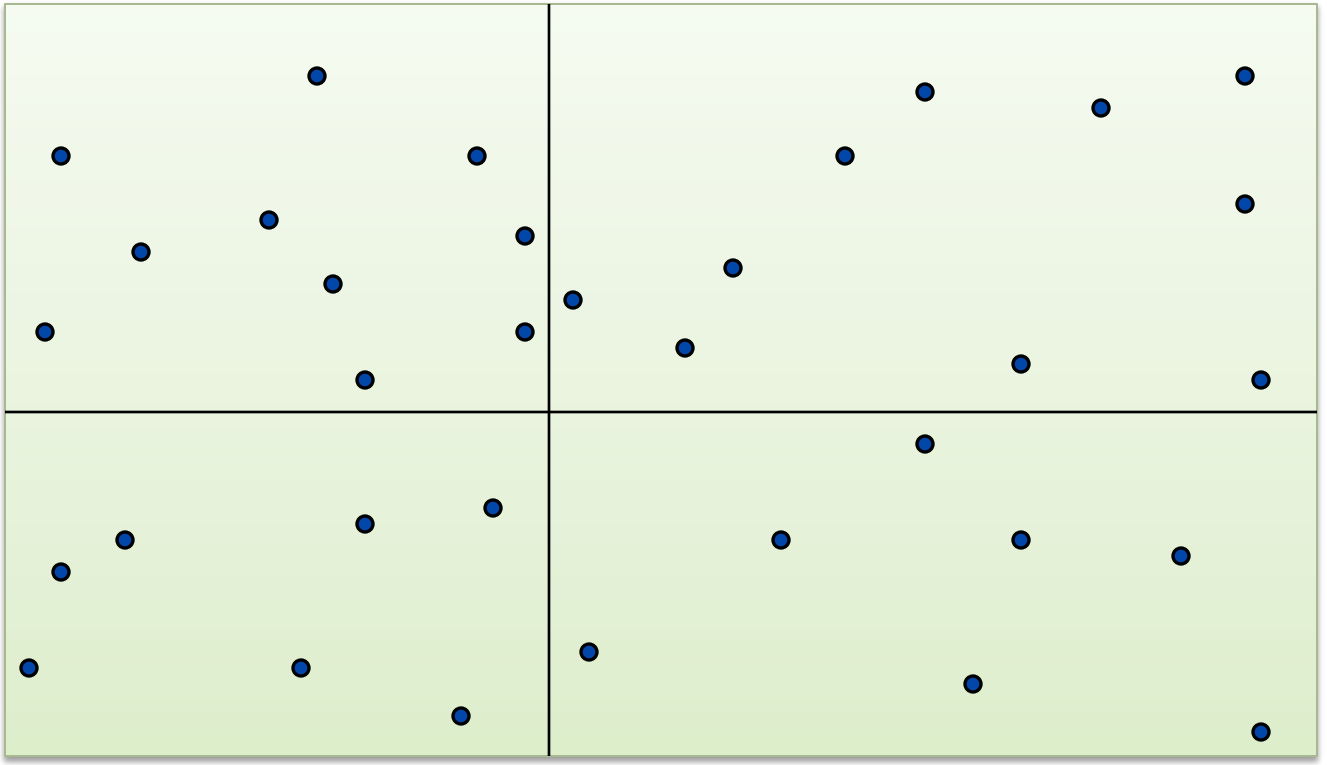
Brute force. Check all pairs of points p and q with $\Theta(n^2)$ comparisons.

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

Assumption. No two points have same x coordinate.

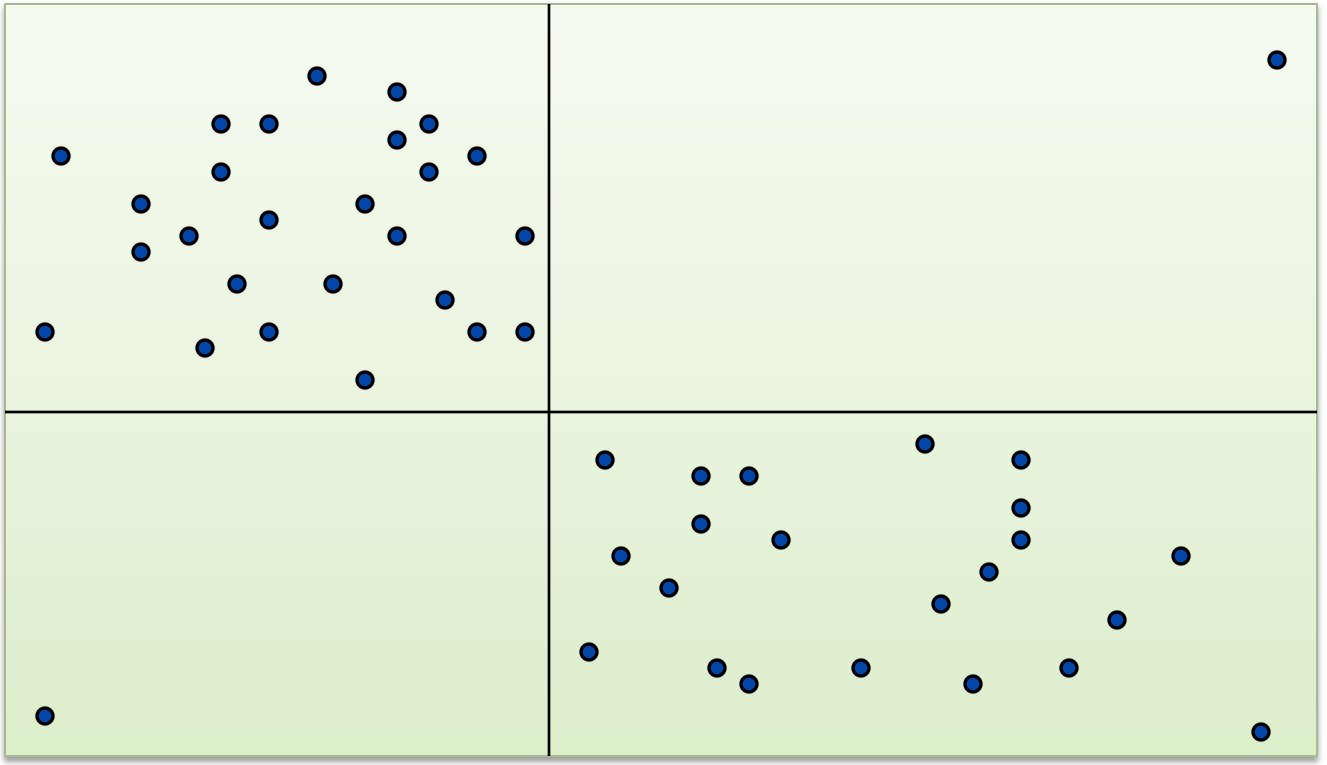
Just to simplify presentation

Divide. Sub-divide region into 4 quadrants.



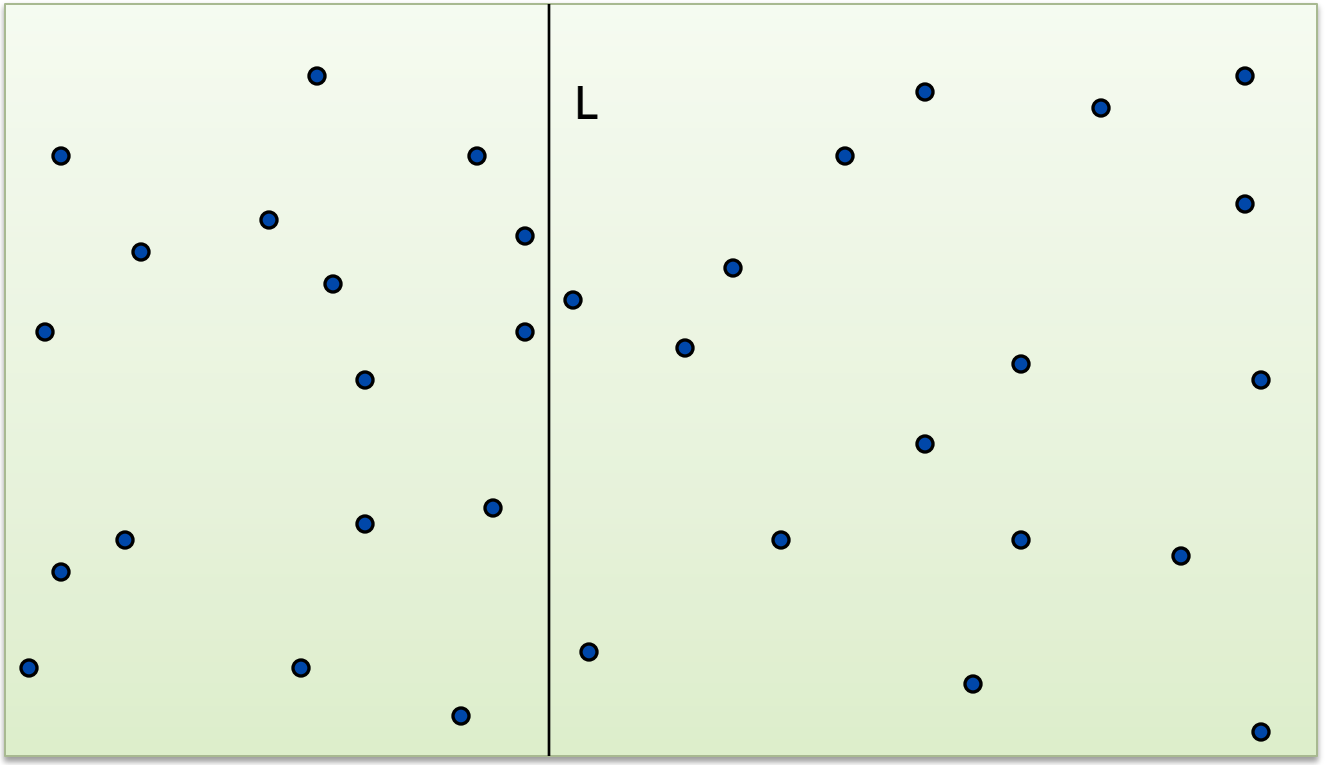
Divide. Sub-divide region into 4 quadrants.

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



Algorithm.

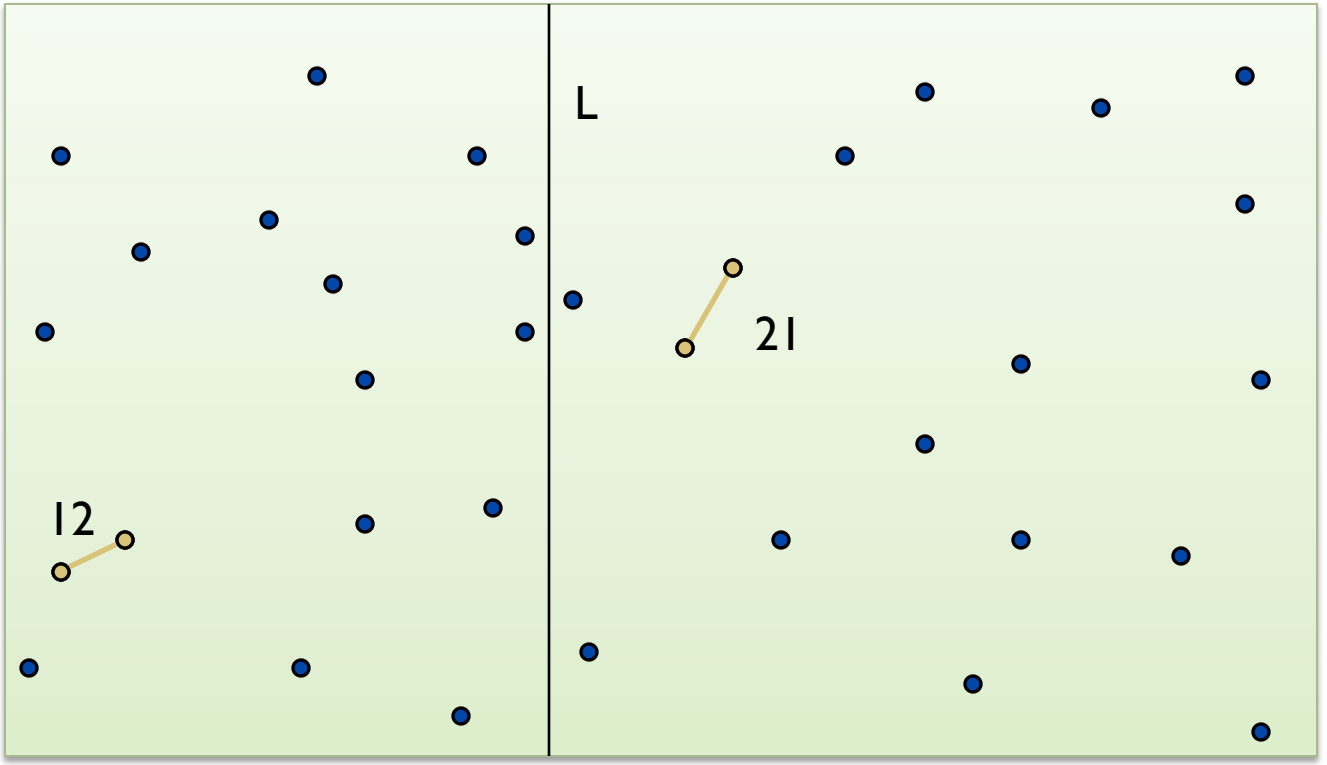
Divide: draw vertical line L with ≈ n/2 points on each side.



Algorithm.

Divide: draw vertical line L with ≈ n/2 points on each side.

Conquer: find closest pair on each side, recursively.



Algorithm.

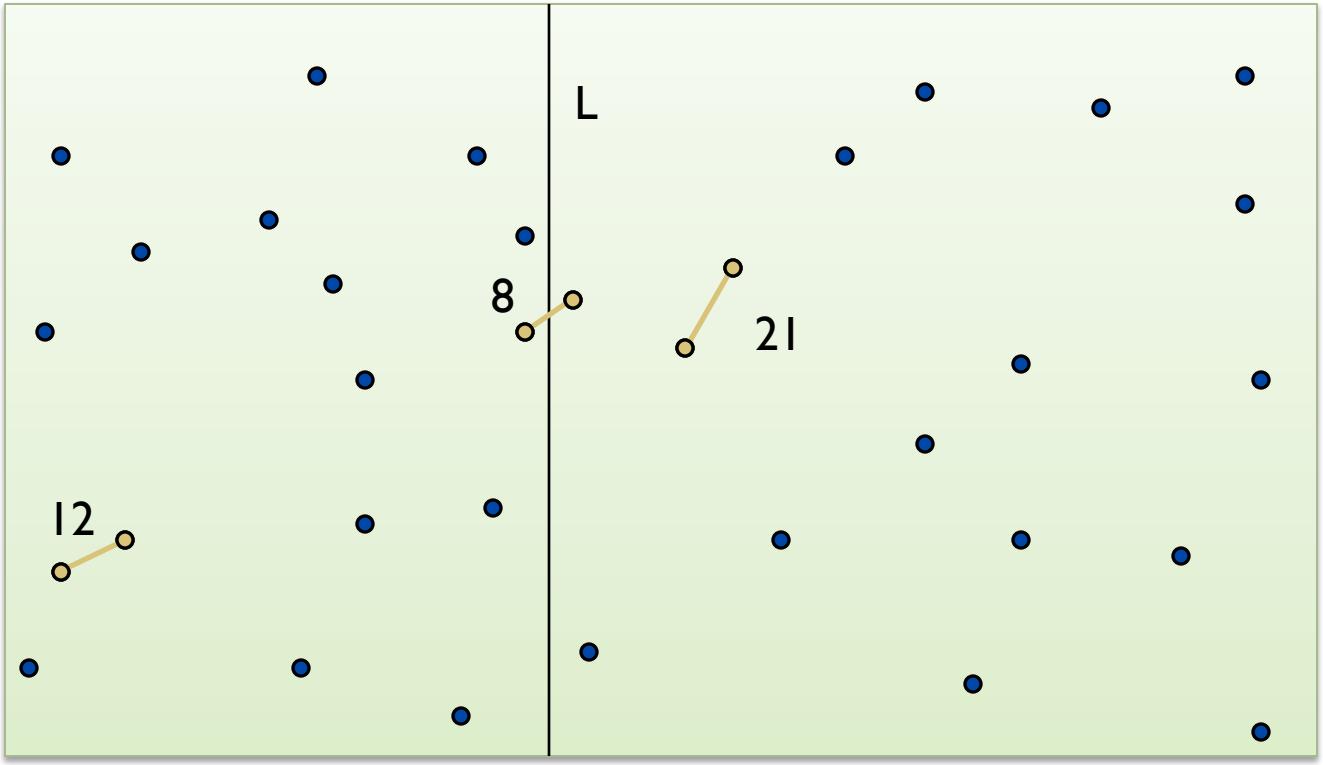
Divide: draw vertical line L with ≈ n/2 points on each side.

Conquer: find closest pair on each side, recursively.

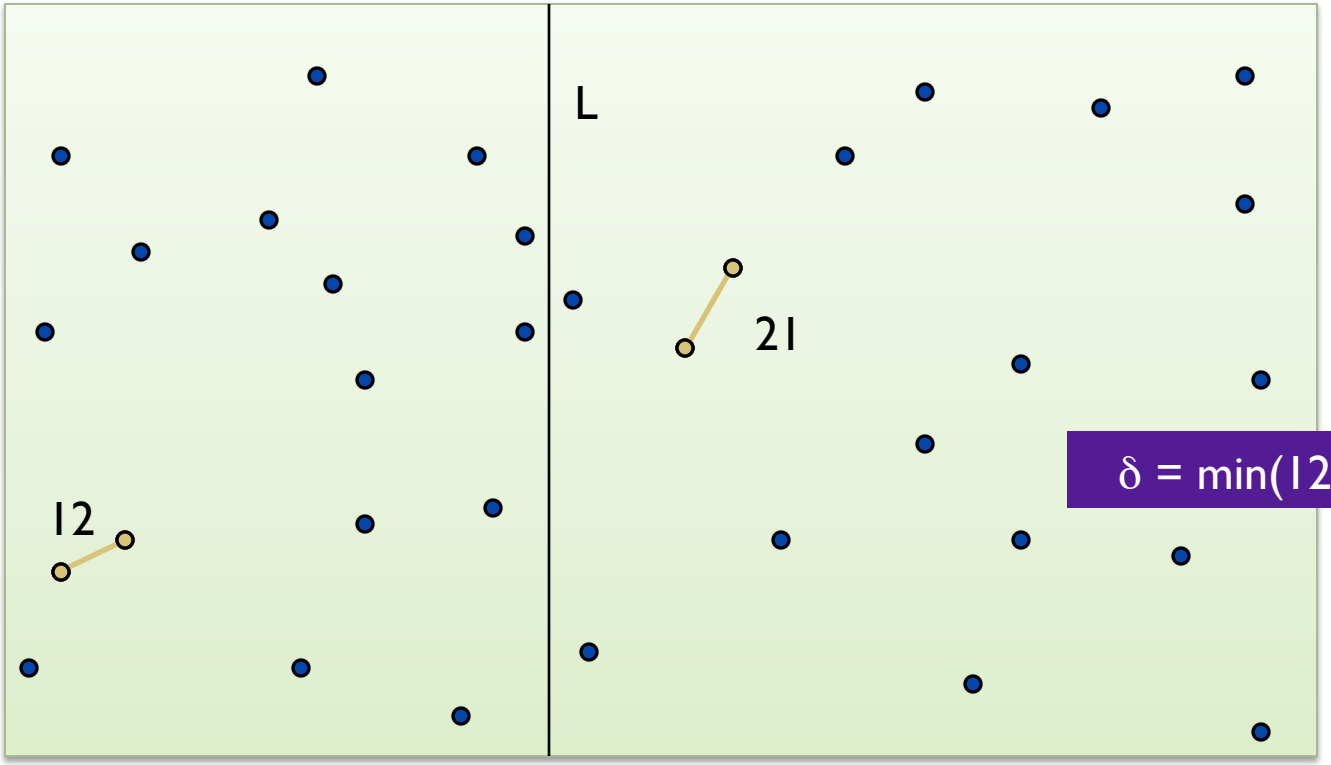
Combine: find closest pair with one point in each side.

Return best of 3 solutions.

seems like $\Theta(n^2)$?

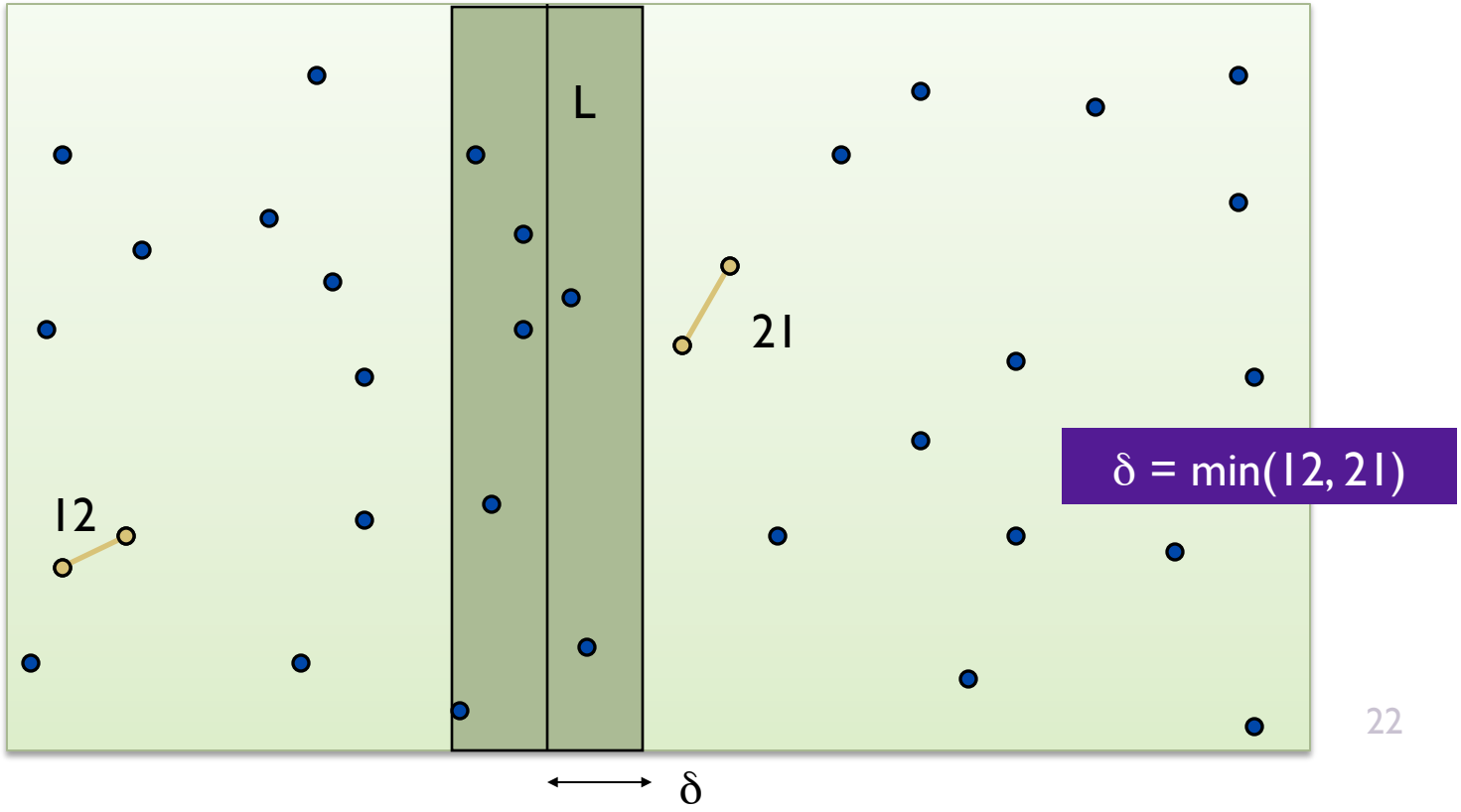


Find closest pair with one point in each side, *assuming* distance < δ.



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

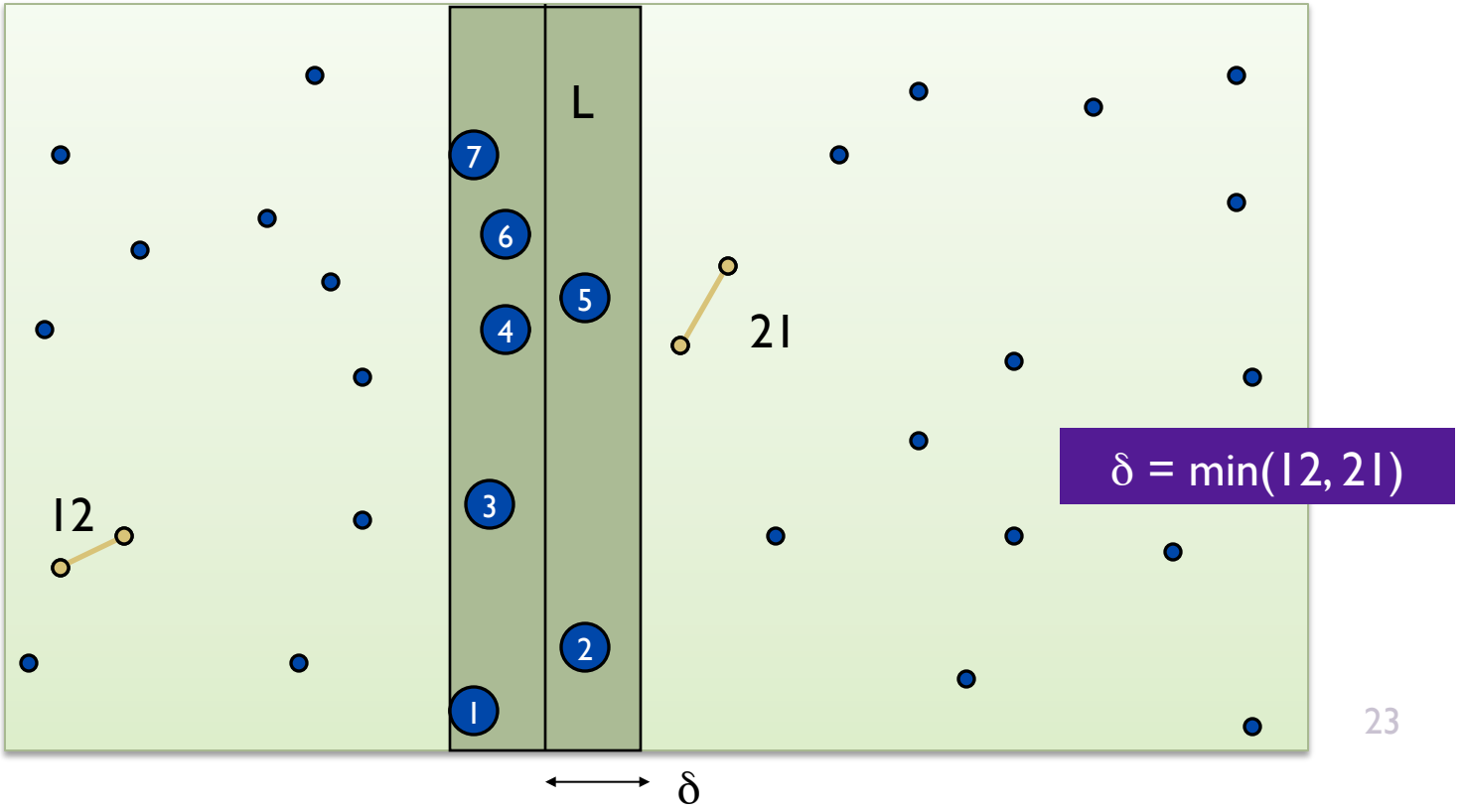
Observation: suffices to consider points within δ of line L.



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

Observation: suffices to consider points within δ of line L.

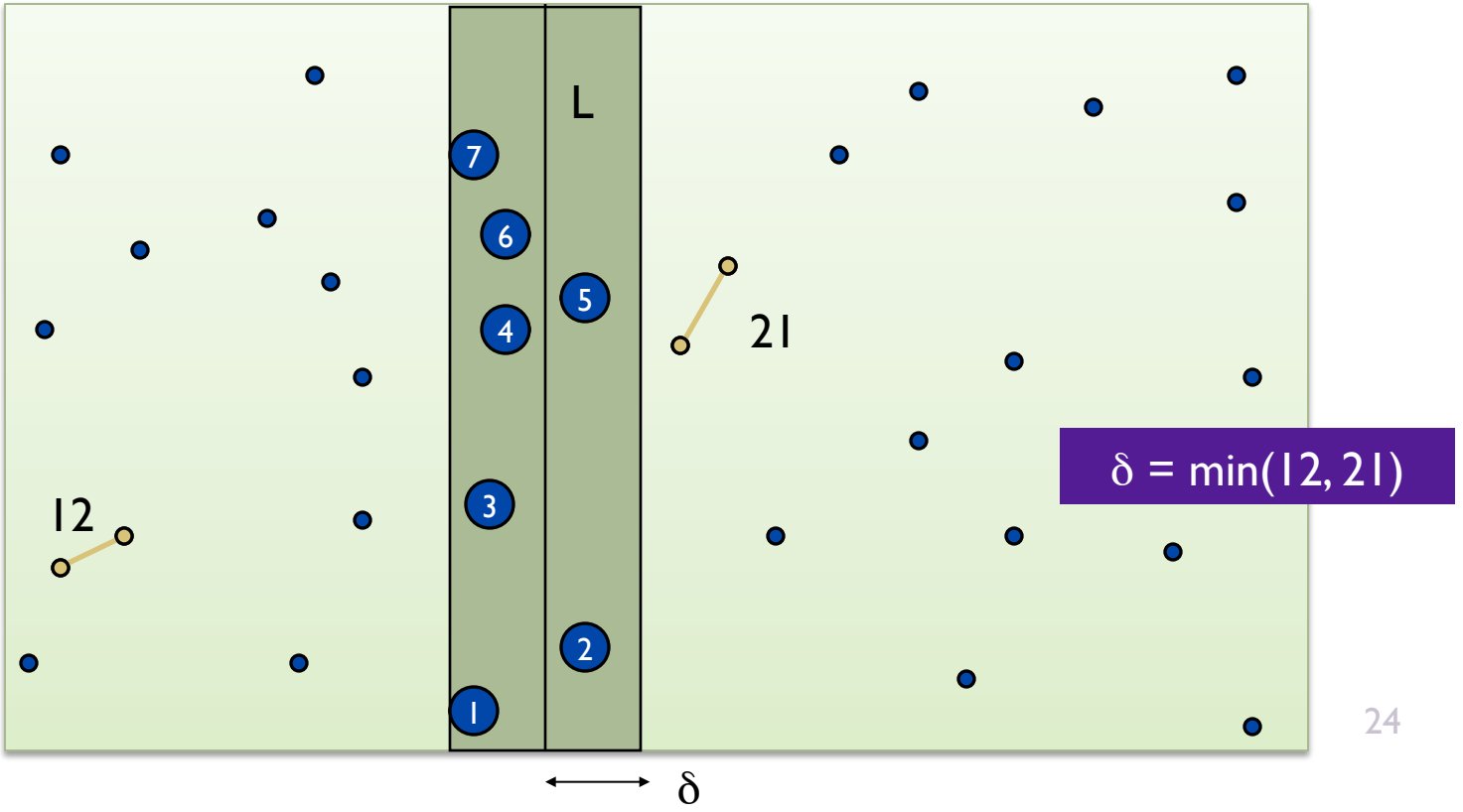
Almost the one-D problem again: Sort points in 2δ-strip by their y coordinate.



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

Observation: suffices to consider points within δ of line L.

Almost the one-D problem again: Sort points in 2δ-strip by their y coordinate. Only check pts within 8 in sorted list!



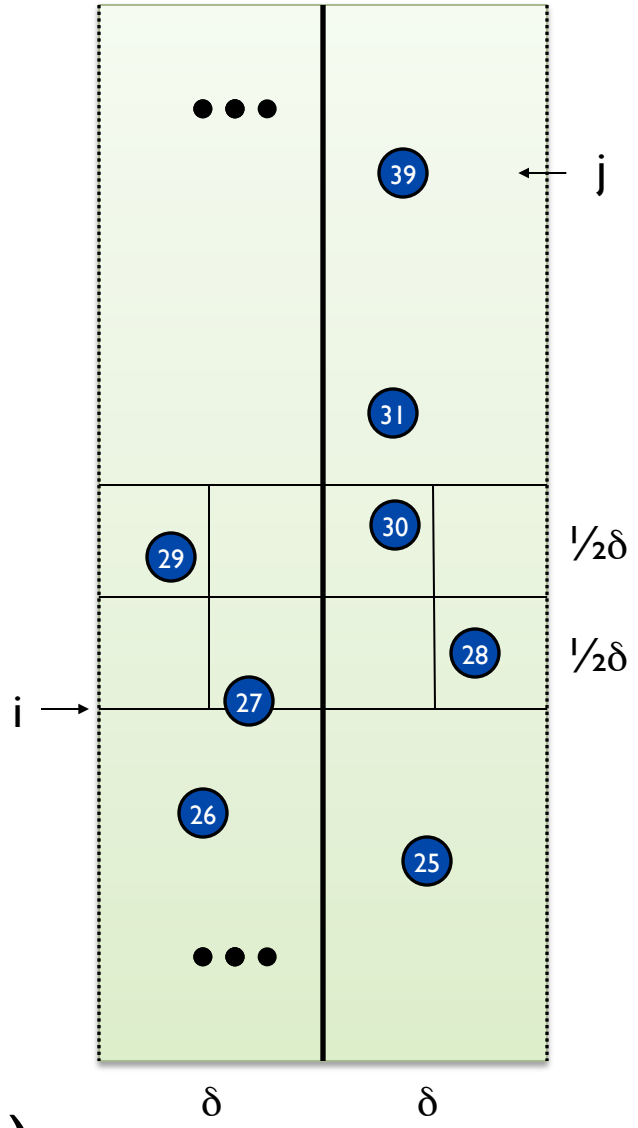
Def. Let s_i have the i^{th} smallest y-coordinate among points in the 2δ-width-strip.

Claim. If $|i - j| > 8$, then the distance between s_i and s_j is > δ.

Pf: No two points lie in the same ½δ-by-½δ box:

$$\sqrt{\left(\frac{1}{2}\right)^2 + \left(\frac{1}{2}\right)^2} = \sqrt{\frac{1}{2}} = \frac{\sqrt{2}}{2} \approx 0.7 < 1$$

only 8 boxes within +δ of $y(s_i)$.



closest pair algorithm

```
Closest-Pair(p1, …, pn) {
   if(n <= ??) return ??

   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 p[1]…p[m] by y-coordinate.

   for i = 1..m
      k = 1
      while i+k <= m && p[i+k].y < p[i].y + δ
        δ = min(δ, distance between p[i] and p[i+k]);
        k++;

   return δ.
}
```

Analysis, I: Let D(n) be the number of pairwise distance calculations in the Closest-Pair Algorithm when run on n ≥ I points

$$D(n) \le \begin{cases} 0 & n = 1 \\ 2D(n/2) + 7n & n > 1 \end{cases} \Rightarrow D(n) = O(n \log n)$$

BUT – that's only the number of *distance calculations*

What if we counted comparisons?

Analysis, II: Let C(n) be the number of comparisons between coordinates/distances in the Closest-Pair Algorithm when run on n ≥ 1 points

$$C(n) \le \begin{cases} 0 & n = 1 \\ 2C(n/2) + O(n \log n) & n > 1 \end{cases} \Rightarrow C(n) = O(n \log^2 n)$$

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

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

Sort by x at top level only.

Each recursive call returns δ *and* list of all points sorted by y

Sort by merging two pre-sorted lists.

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

Going From Code to Recurrence

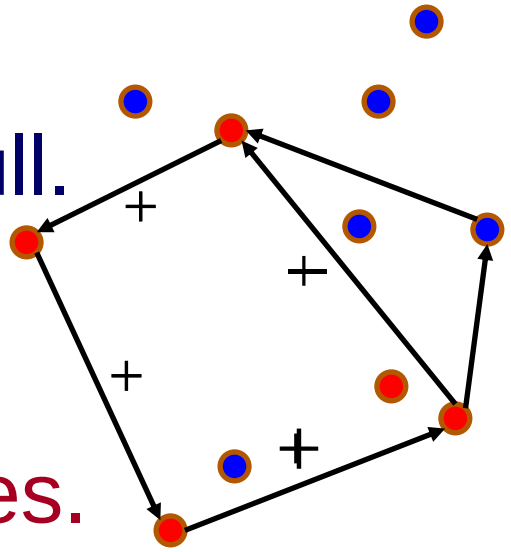
Incremental Algorithm

- IncrementalAlgorithm($S = \{p_1, \ldots, p_n\}$):
  - $H_3 \leftarrow$ ConvexHull($\{p_1, p_2, p_3\}$)
  - For $k \in [4, n]$
    - $H_k \leftarrow$ AddtoHull(H_{k-1} , p_k)

Complexity: $O(n^2)$

To add to a point to the hull, mark each edge, indicating if the points is to the left or right:

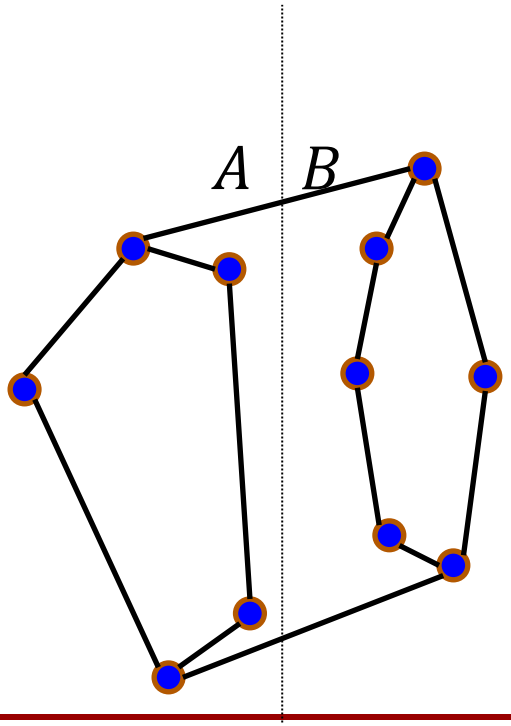
- If it's left of all edges, it's interior and isn't on the hull.
- Otherwise, there are two transition vertices.
  - »Connect to those vertices.



Divide And Conquer

Recursively:

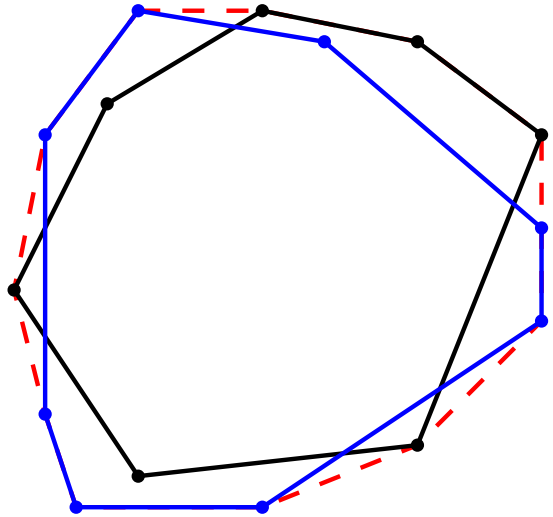
- Split the point-set in two.
- Compute the hull of both halves
- Merge the hulls



Other approaches: divide-and-conquer

Divide-and-conquer: split the point set in two halves, compute the convex hulls recursively, and merge

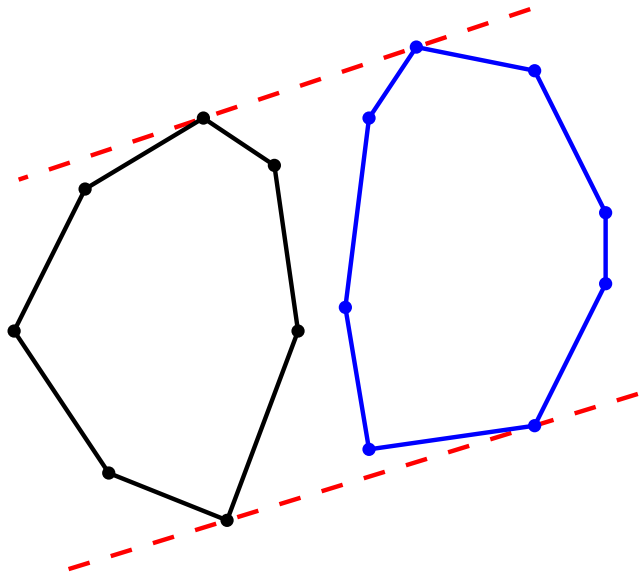
A merge involves finding "extreme vertices" in every direction



Other approaches: divide-and-conquer

Alternatively: split the point set in two halves on x-coordinate, compute the convex hulls recursively, and merge

A merge now comes down to finding two common tangent lines



Union of convex hulls – I