CSE 421

Divide and Conquer

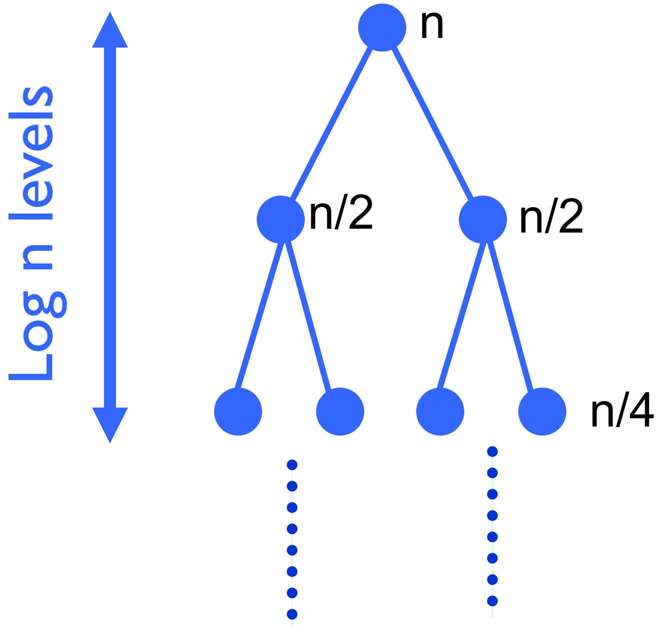
Shayan Oveis Gharan

Divide and Conquer Approach

Divide and Conquer

Similar to algorithm design by induction, we reduce a problem to several subproblems.

Typically, each sub-problem is at most a constant fraction of the size of the original problem



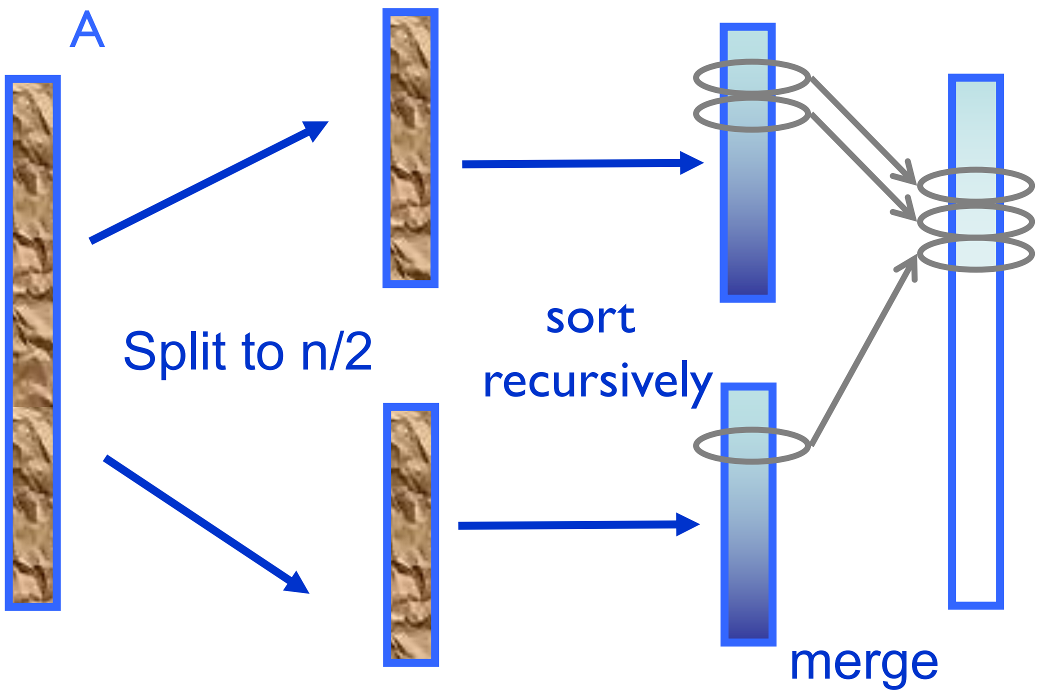
Recursively solve each subproblem

Merge the solutions

Examples:

- Mergesort, Binary Search, Strassen's Algorithm,

A Classical Example: Merge Sort



Why Balanced Partitioning?

An alternative "divide & conquer" algorithm:

- Split into n-1 and 1
- Sort each sub problem
- Merge them

Runtime

$$T(n) = T(n-1) + T(1) + n$$

Solution:

$$T(n) = n + T(n-1) + T(1)$$

$$= n + n - 1 + \ T(n-2)$$

$$= n + n - 1 + n - 2 + T(n-3)$$

$$= n + n - 1 + n - 2 + \cdots + 1 = O(n^2)$$

D&C: The Key Idea

Suppose we've already invented Bubble-Sort, and we know it takes n^2

Try just one level of divide & conquer:

Bubble-Sort(first n/2 elements)

Bubble-Sort(last n/2 elements)

Merge results

Time: $2\,T(n/2) + n = n^2/2 + n \ll n^2$

Almost twice as fast!

D&C in a nutshell

D&C approach

- “the more dividing and conquering, the better”
  - Two levels of D&C would be almost 4 times faster, 3 levels almost 8, etc., even though overhead is growing.
  - Best is usually full recursion down to a small constant size (balancing "work" vs "overhead").

  In the limit: you’ve just rediscovered mergesort!
- Even unbalanced partitioning is good, but less good
  - Bubble-sort improved with a 0.1/0.9 split:

    $$(.1n)^2 + (.9n)^2 + n = .82n^2 + n$$

    The 18% savings compounds significantly if you carry recursion to more levels, actually giving $O(n \log n)$, but with a bigger constant.
- This is why Quicksort with random splitter is good – badly unbalanced splits are rare, and not instantly fatal.

Finding the Root of a Function

Finding the Root of a Function

Given a continuous function f and two points a < b such that

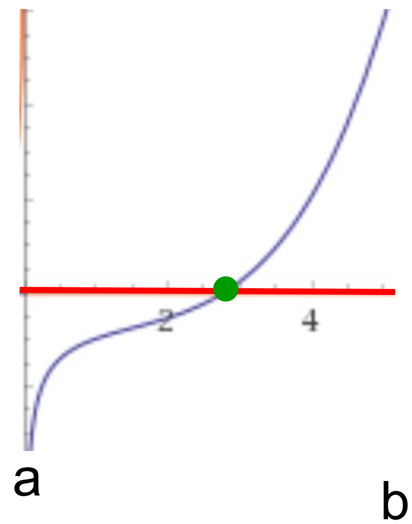
$$f(a) \leq 0$$
$$f(b) \geq 0$$

Find an approximate root of f (a point c where there is r s.t., $|r - c| \leq \epsilon$ and $f(r) = 0$).

$$\mathrm{f}(x) = \sin(x) - \frac{100}{\sqrt{x}} + x^4$$

Note f has a root in $[a, b]$ by

intermediate value theorem



Note that roots of f may be irrational,

So, we want to approximate

the root with an arbitrary precision!

A Naiive Approch

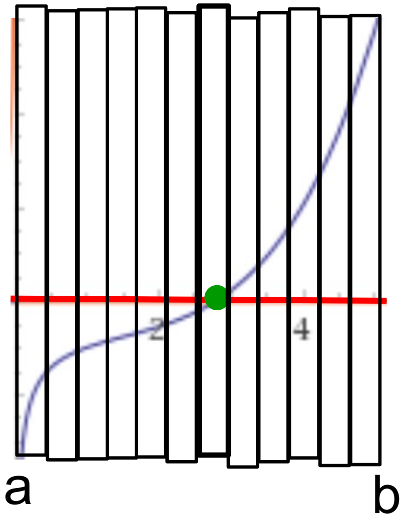
Suppose we want ϵ approximation to a root.

Divide [a,b] into $n = \frac{b-a}{\epsilon}$ intervals. For each interval check

$$f(x) \leq 0, f(x+\epsilon) \geq 0$$

This runs in time $O(n) = O(\frac{b-a}{\epsilon})$

Can we do faster?



D&C Approach (Based on Binary Search)

Bisection(a,b, ε)

if $(b - a) < \epsilon$ then

return (a)

else

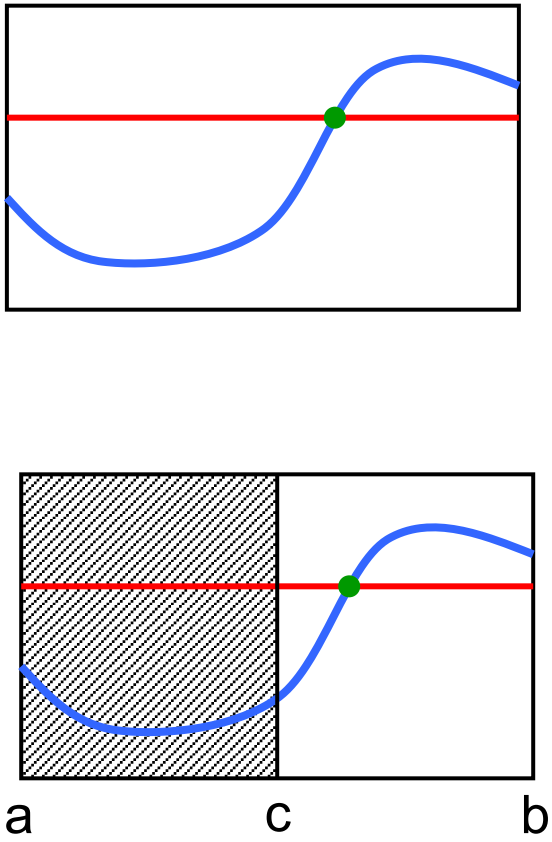
$m \leftarrow (a + b)/2$

if $f(m) \leq 0$ then

return(Bisection(c, b, ε))

else

return(Bisection(a, c, ε))

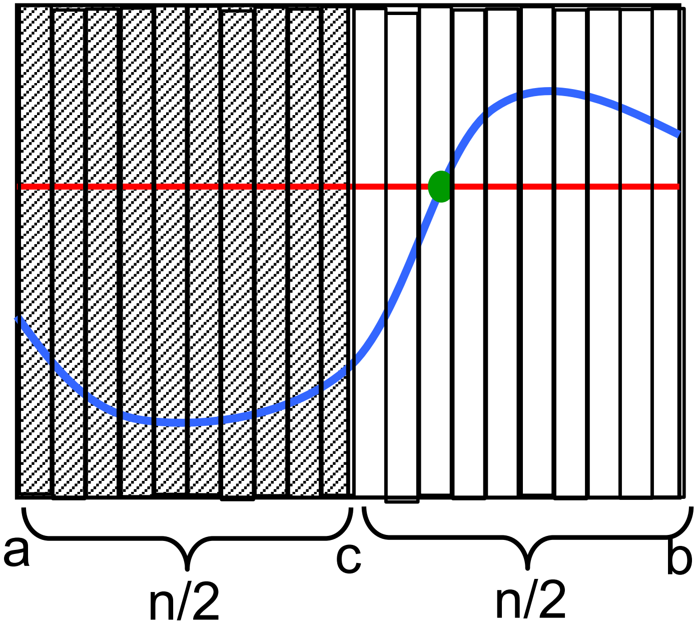


Time Analysis

Let $n = \frac{a-b}{\epsilon}$

And $c = (a+b)/2$

Always half of the intervals lie to the left and half lie to the right of c



So,

$$T(n) = T\left(\frac{n}{2}\right) + O(1)$$

i.e., $T(n) = O(\log n) = O(\log \frac{a-b}{\epsilon})$

Correctness Proof

P(k) = "For any a, b such that $k\epsilon \leq |a - b| \leq (k+1)\epsilon$ if $f(a)f(b) \leq 0$, then we find an ϵ approx to a root using $\log k$ queries to f"

Base Case: P(1): Output $a + \epsilon$

IH: Assume P(k).

IS: Show P(2k). Consider an arbitrary a, b s.t.,

$$2k\epsilon \leq |a - b| < (2k+1)\epsilon$$

If $f(a + k\epsilon) = 0$ output $a + k\epsilon$.

If $f(\mathrm{a})f(a + k\epsilon) < 0$, solve for interval $a, a + k\epsilon$ using log(k) queries to f.

Otherwise, we must have $f(b)f(a + k\epsilon) < 0$ since $f(a)f(b) < 0$ and $f(a)f(a + k\epsilon) \geq 0$. Solve for interval $a + k\epsilon, b$.

Overall we use at most $\log(k) + 1 = \log(2k)$ queries to *f*.

Master Theorem

Suppose $T(n) = a\,T\left(\frac{n}{b}\right) + cn^k$ for all $n > b$. Then,

- If $a > b^k$ then $T(n) = \Theta\left(n^{\log_b a}\right)$
- If $a < b^k$ then $T(n) = \Theta\left(n^k\right)$
- If $a = b^k$ then $T(n) = \Theta\left(n^k \log n\right)$

Works even if it is $\left\lceil \frac{n}{b} \right\rceil$ instead of $\frac{n}{b}$.

We also need $a \geq 1, b > 1$, $k \geq 0$ and $T(n) = O(1)$ for $n \leq b$.

Master Theorem

Suppose $T(n) = a\,T\left(\frac{n}{b}\right) + cn^k$ for all $n > b$. Then,

- If $a > b^k$ then $T(n) = \Theta\left(n^{\log_b a}\right)$
- If $a < b^k$ then $T(n) = \Theta\left(n^k\right)$
- If $a = b^k$ then $T(n) = \Theta\left(n^k \log n\right)$

Example: For mergesort algorithm we have

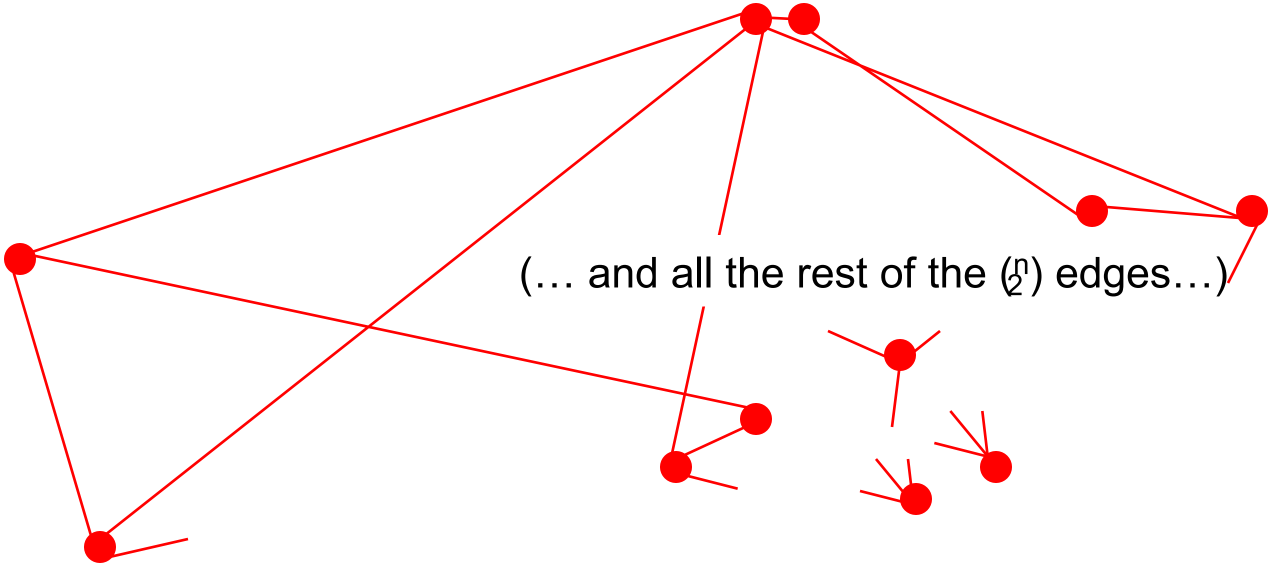
$$T(n) = 2T\left(\frac{n}{2}\right) + O(n).$$

So, $k = 1, a = b^k$ and $T(n) = \Theta(n \log n)$

Finding the Closest Pair of Points

Closest Pair of Points (non geometric)

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.

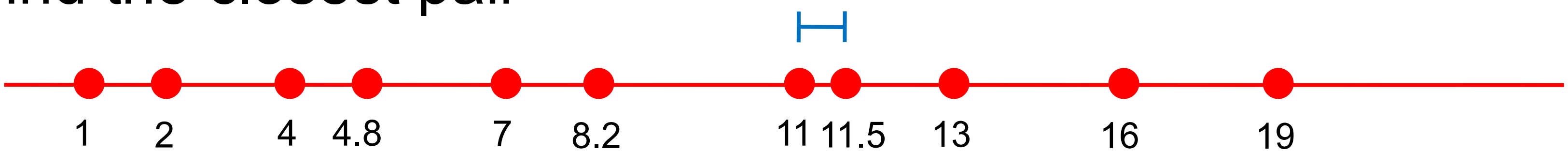
i.e., you have to read the whole input

Closest Pair of Points (1-dimension)

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

e.g., given 11, 2, 4, 19, 4.8, 7, 8.2, 16, 11.5, 13, 1

find the closest pair



Fact: Closest pair is adjacent in ordered list

So, first sort, then scan adjacent pairs.

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-dimensions)

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.

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

Assumption: No two points have same x or y coordinates.

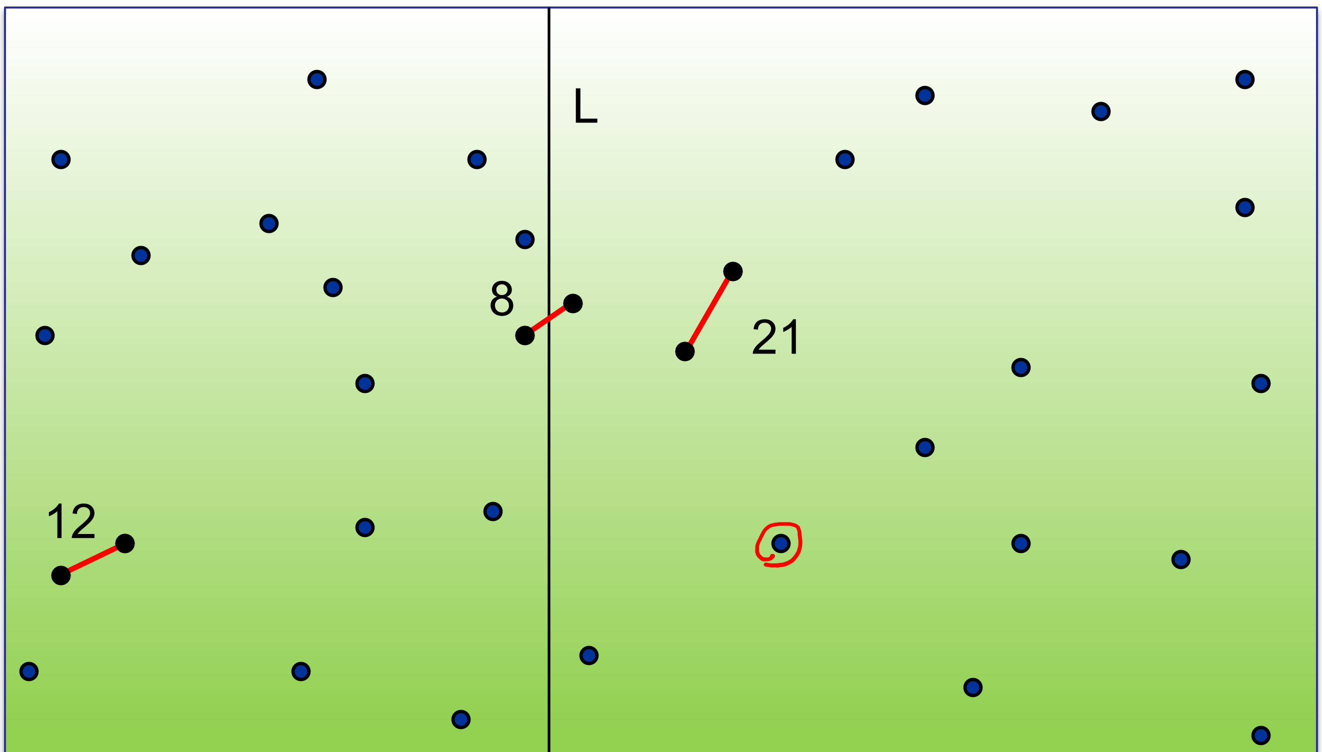
A Divide and Conquer Alg

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

Conquer: find closest pair on each side, recursively.

Combine to find closest pair overall

Return best solutions ← seems like $\Theta(n^2)$?



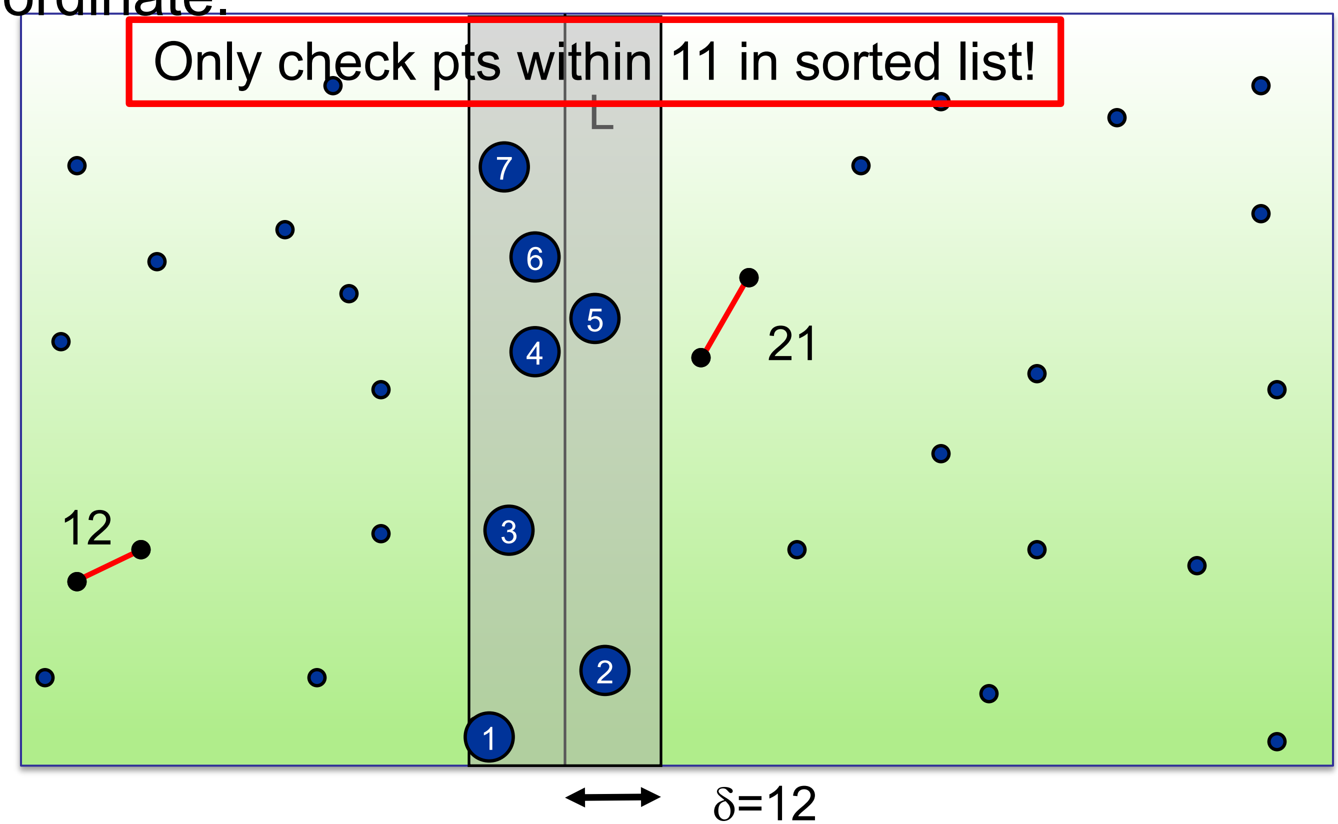
Key Observation

Suppose δ is the minimum distance of all pairs in left/right of L.

$$\delta = \min(12{,}21) = 12.$$

Key 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.



Almost 1D Problem

Partition each side of L into $\frac{\delta}{2} \times \frac{\delta}{2}$ squares

Claim: No two points lie in the same $\frac{\delta}{2} \times \frac{\delta}{2}$ box.

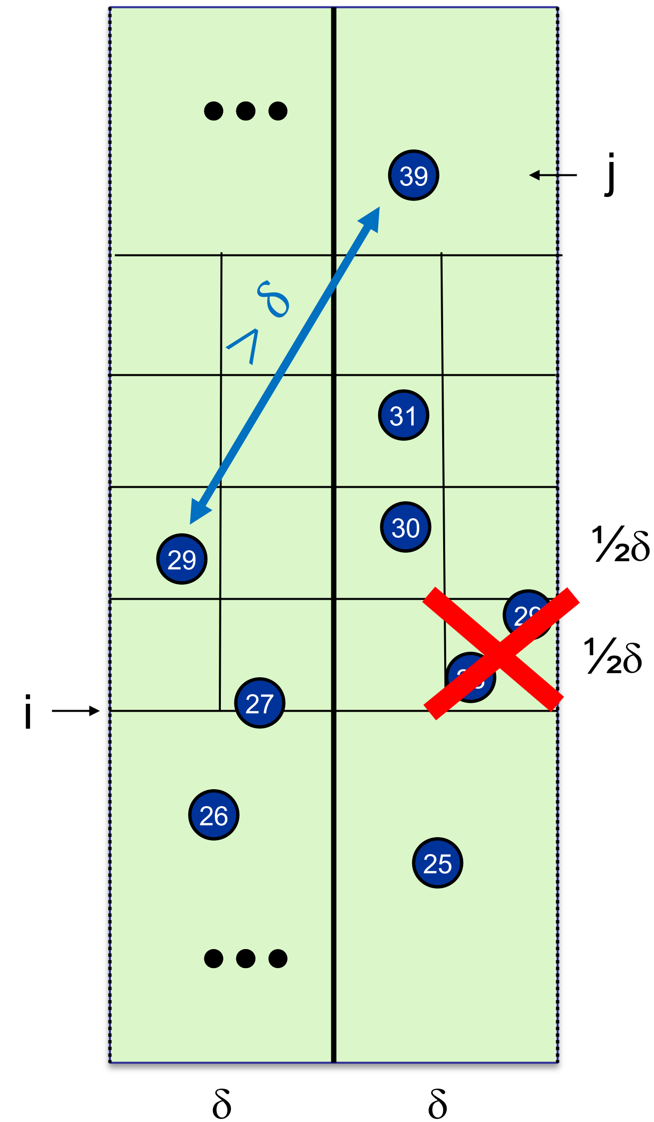
Pf: Such points would be within

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

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

Claim: If $|i - j| > 11$, then the distance between s_i and s_j is $> \delta$.

Pf: only 11 boxes within δ of $y(s_i)$.



Closest Pair (2Dim 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
      for k = 1…11
       if i+k <= m
           δ = min(δ, distance(p[i], p[i+k]));

   return δ.
}
```

Closest Pair Analysis I

Let D(n) be the number of pairwise distance calculations in the Closest-Pair Algorithm when run on n ≥ 1 points

$$D(n) \le \begin{cases} 1 & \text{if } n = 1 \\ 2D\left(\frac{n}{2}\right) + 11\,n & \text{o.w.} \end{cases} \Rightarrow D(n) = \mathrm{O}(n\log n)$$

BUT, that's only the number of distance calculations

What if we counted running time?

$$T(n) \le \begin{cases} 1 & \text{if } n = 1 \\ 2T\left(\frac{n}{2}\right) + O(n\log n) & \text{o.w.} \end{cases} \Rightarrow D(n) = \mathrm{O}(n\log^2 n)$$

Can we do better? (Analysis II)

Yes!!

Don’t sort by y-coordinates each time.

Sort by x at top level only.

This is enough to divide into two equal subproblems in O(n)

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

Sort points by y-coordinate by merging two pre-sorted lists.

$$T(n) \leq \begin{cases} 1 & \text{if } n = 1 \\ 2T\left(\frac{n}{2}\right) + O(n) & \text{o.w.} \end{cases} \Rightarrow D(n) = O(n \log n)$$