

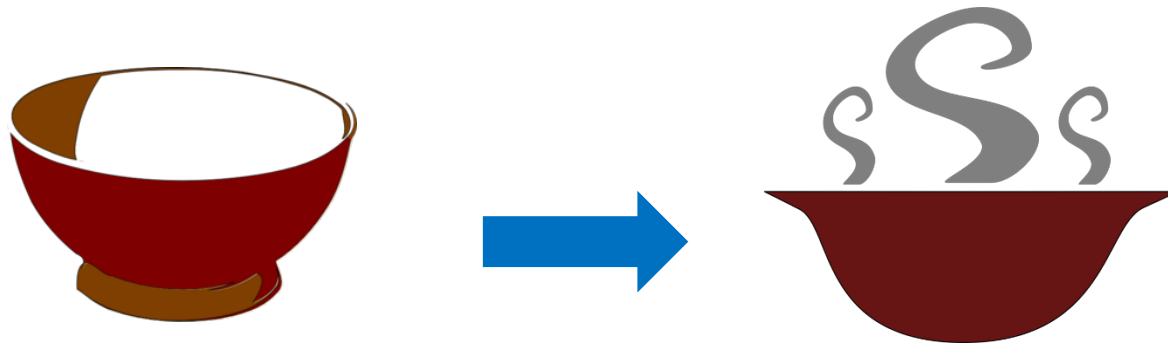
CSE 421

Divide and Conquer

Shayan Oveis Gharan

Boiling Water Example

Q: Given an empty bowl, how do you make boiling water?



A: Well, I fill it with water, turn on the stove, leave the bowl on the stove for 20 minutes. I have my boiling water.

Q: Now, suppose you have a bowl of water, how do you make boiling water?

A: First, I pour water away, now I have an empty bowl and I have already solved this!



Lesson: Never solve a problem twice!

Divide and Conquer Approach

Divide and Conquer

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

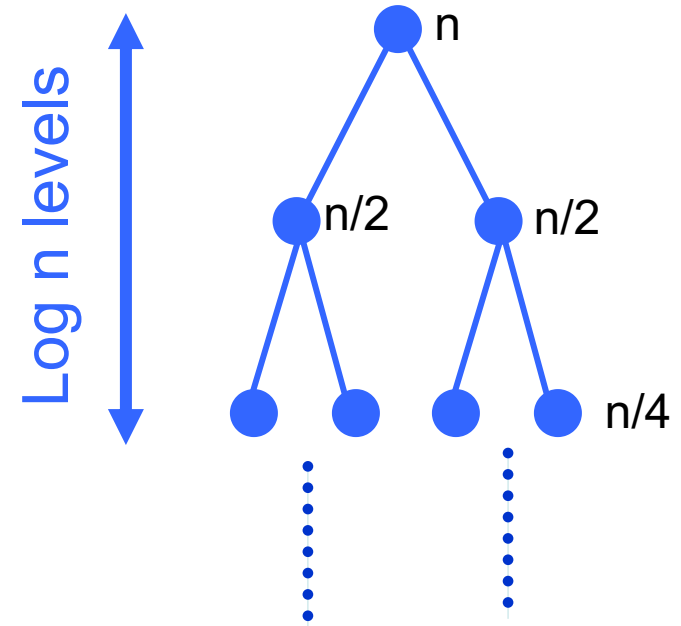
Typically, each sub-problem is **at most a constant fraction** of the size of the original problem

Recursively solve each subproblem

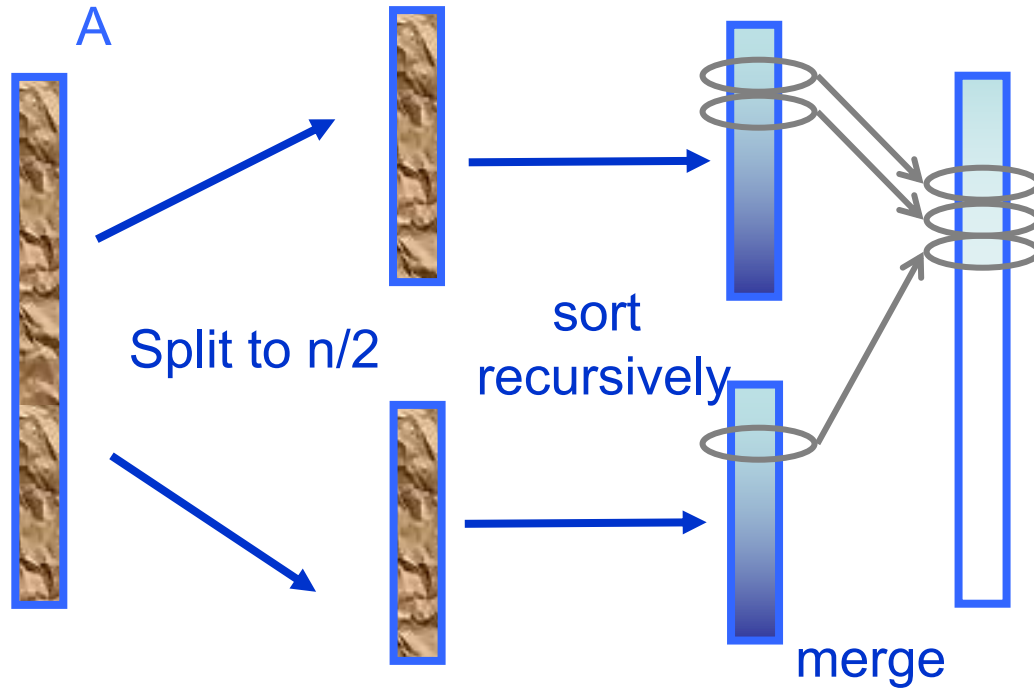
Merge the solutions

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:

$$\begin{aligned}T(n) &= n + T(n - 1) + T(1) \\&= n + n - 1 + T(n - 2) \\&= n + n - 1 + n - 2 + T(n - 3) \\&= n + n - 1 + n - 2 + \dots + 1 = O(n^2)\end{aligned}$$

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 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

$$f(a) \leq 0$$

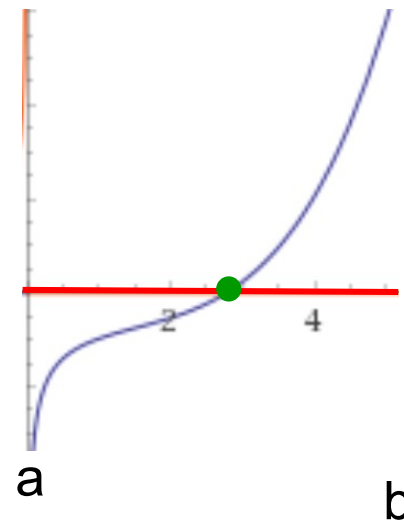
$$f(b) \geq 0$$

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

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!

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



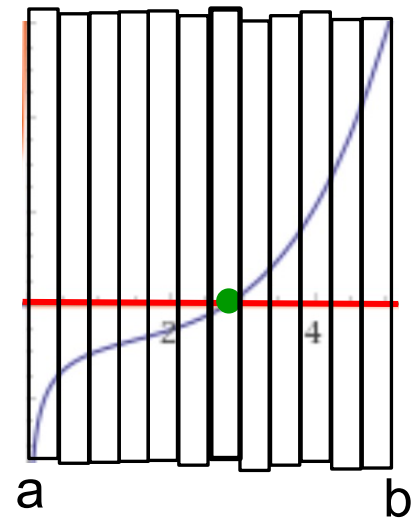
A Naive Approach

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 $O(n) = O\left(\frac{b-a}{\epsilon}\right)$

Can we do faster?



D&C Approach (Based on Binary Search)

Bisection(a, b, ϵ)

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

 return (a)

else

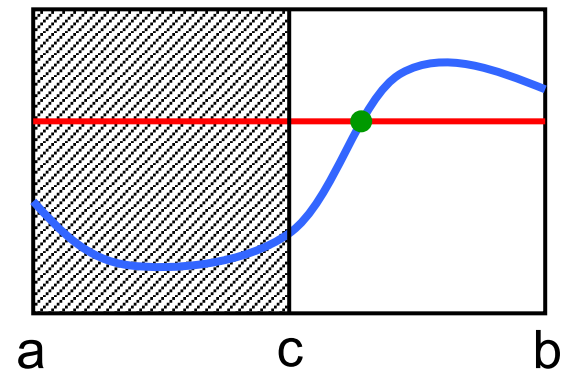
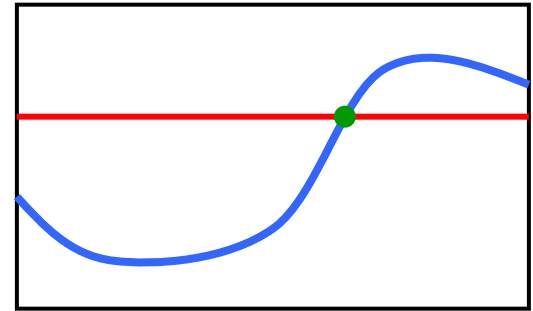
$m \leftarrow (a + b)/2$

 if $f(m) \leq 0$ then

 return(Bisection(c, b, ϵ))

 else

 return(Bisection(a, c, ϵ))



Time Analysis

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

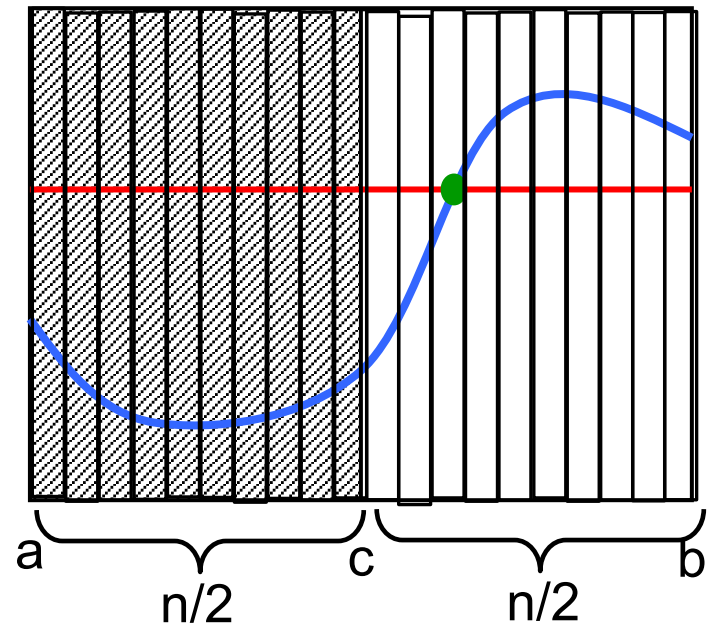
$$\text{And } c = (a + b)/2$$

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

So,

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

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



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

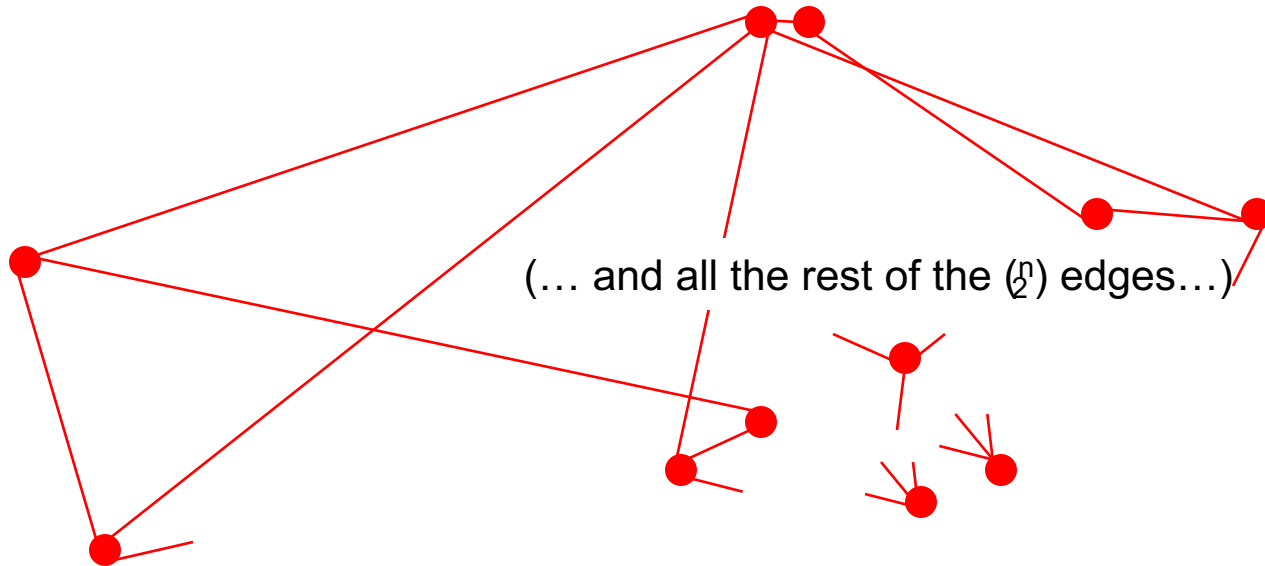
Otherwise, we must have $f(b)f(a + k\epsilon) < 0$ since $f(a)f(b) < 0$ and $f(a)f(a + k\epsilon) \geq 0$. Solve for interval $a + k\epsilon, b$.

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

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!)

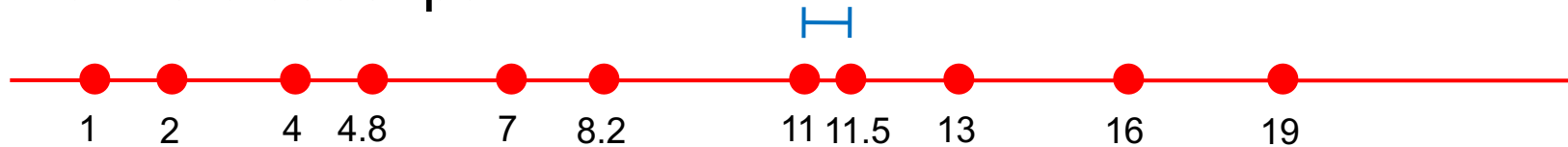


Must look at all n choose 2 pairwise distances, else any one you didn't check might be the shortest.

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

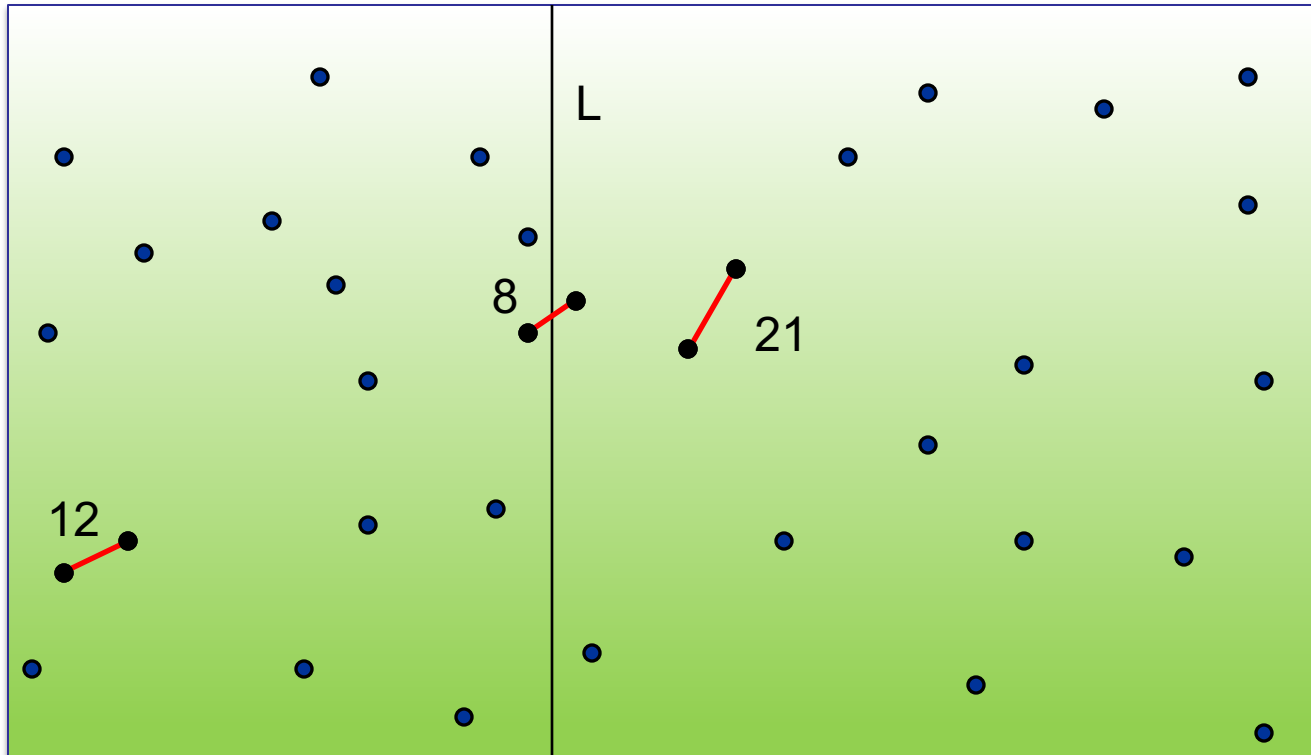
Divide: draw vertical line L with $\approx n/2$ points on each side.

Conquer: find closest pair on each side, recursively.

Combine to find closest pair overall

Return best solutions

← seems like $\Theta(n^2)$?



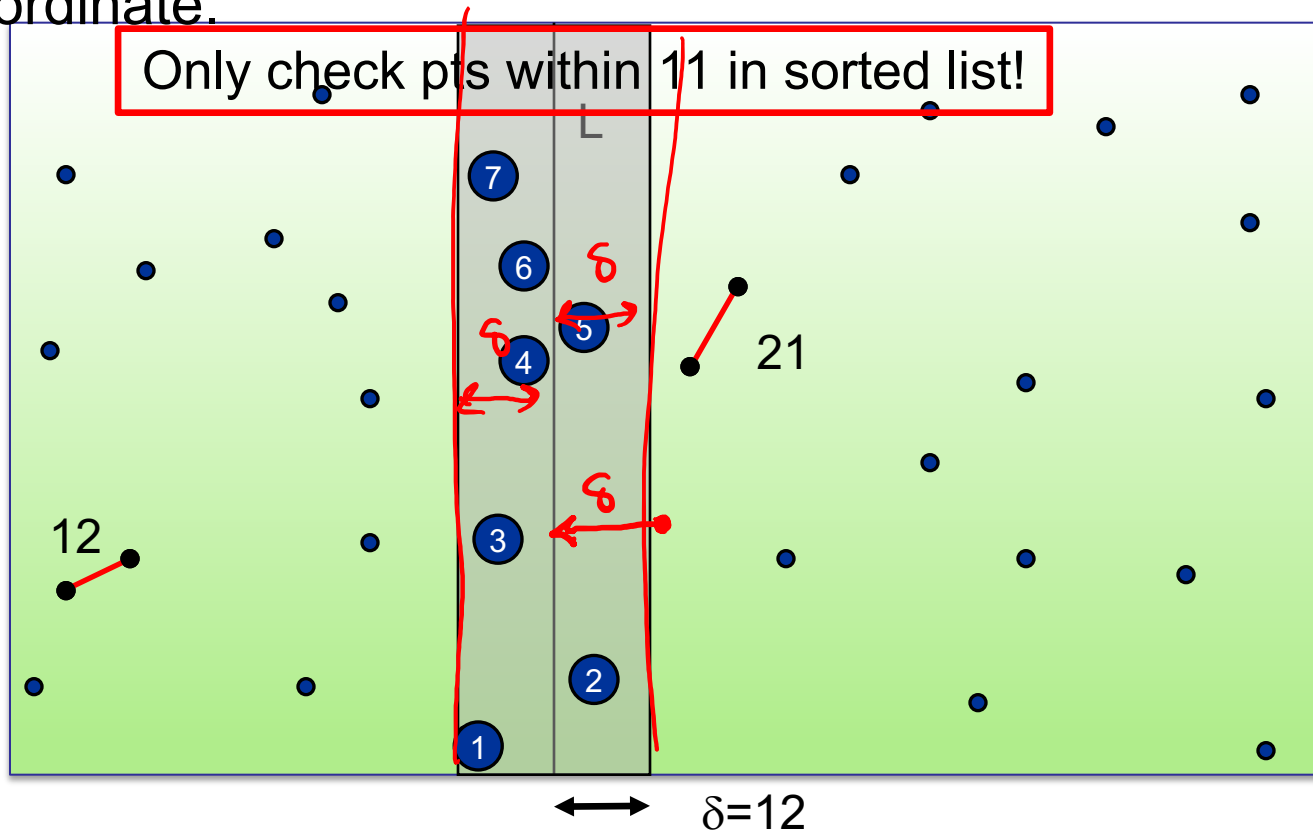
Key Observation

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.

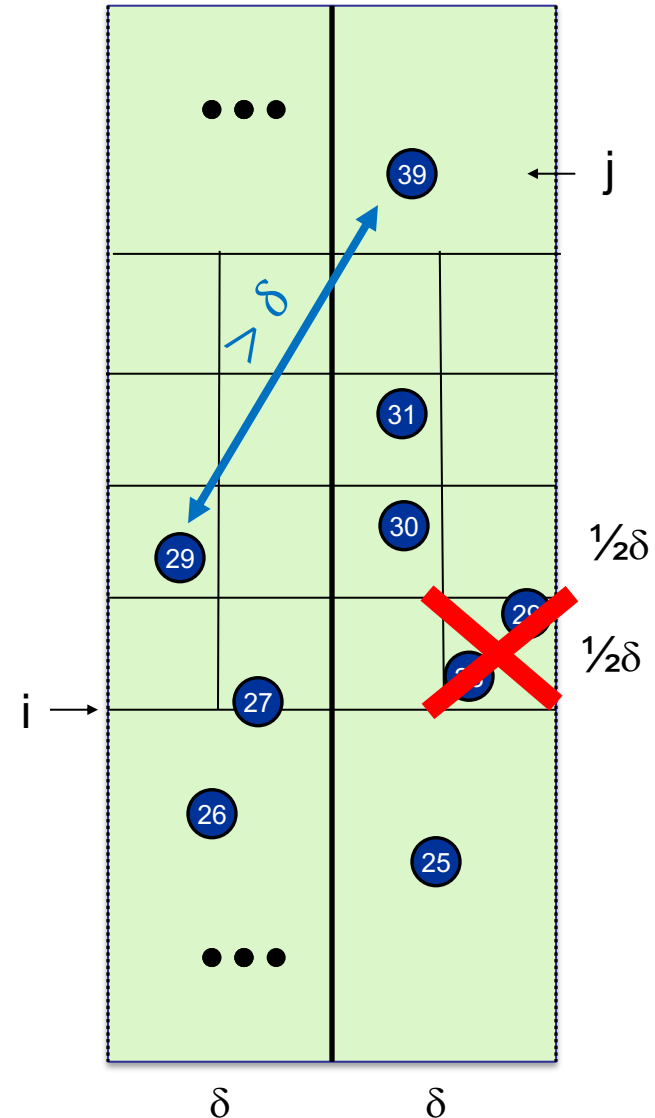
Pf: Such points would be within

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

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( $p_1, \dots, p_n$ ) {  
  if( $n \leq ??$ ) return ??
```

Compute separation line L such that half the points are on one side and half on the other side.

```
 $\delta_1$  = Closest-Pair(left half)  
 $\delta_2$  = Closest-Pair(right half)  
 $\delta$  =  $\min(\delta_1, \delta_2)$ 
```

Delete all points further than δ from separation line L

Sort remaining points $p[1] \dots p[m]$ by y -coordinate.

```
for  $i = 1..m$                                  $i$   
  for  $k = 1..11$   
    if  $i+k \leq m$   
       $\delta = \min(\delta, \text{distance}(p[i], p[i+k]));$ 
```

```
return  $\delta$ .
```

```
}
```

Closest Pair Analysis I

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

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

BUT, that's only the number of distance calculations

What if we counted running time?

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

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.

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