CSE 421

Divide and Conquer / Closest Pair of Points

Yin Tat Lee

Stuff

This Friday. A 20-min course assessment done by the college.

I won't be fired ☺ even if I got terrible review ☹.

So, you can say ANYTHING.

It is only used to help me to improve the teaching.

HW4 and HW5 is out. (You will like HW5.)

Midterm: Apr 30 (Monday)

It covers from Lecture 1 to Lecture 12 (Friday).

There is a midterm review for you to ask questions.

I will be there and I will find an expert in grading (TA) to help me.

Removing weight Distinction Assumption

Suppose edge weights are not distinct, and Kruskal's algorithm sorts edges so

$$c_{e_1} \leq c_{e_2} \leq \cdots \leq c_{e_m}$$

Suppose Kruskal finds tree T of weight $c(T)$, but the optimal solution T^* has cost $c(T^*) < c(T)$.

Perturb each of the weights by a very small amount so that

$$c'_{e_1} < c'_{e_2} < \cdots < c'_{e_m}$$

If the perturbation is small enough, $c'(T^*) < c'(T)$.

However, this contradicts the correctness of Kruskal's algorithm, since the algorithm will still find T, and Kruskal's algorithm is correct if all weighs are distinct.

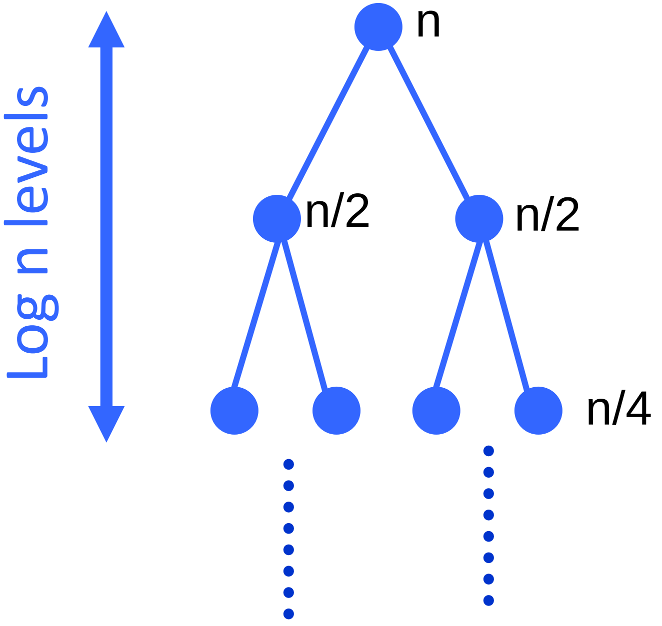
∎

Divide and Conquer Approach

Divide and Conquer

We reduce a problem to several subproblems.

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



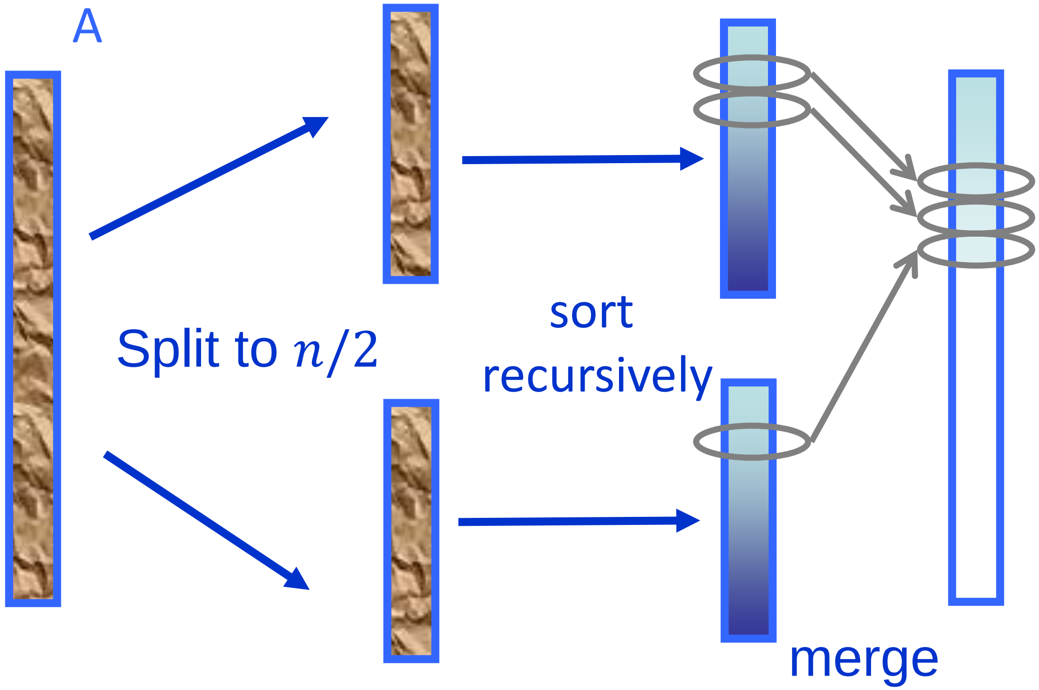
Recursively solve each subproblem

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:

$$T(n) = n + T(n-1) + T(1)$$
$$= n + n - 1 + \ T(n-2)$$
$$= n + n - 1 + n - 2 + T(n-3)$$
$$= n + n - 1 + n - 2 + \cdots + 1 = O(n^2)$$

Reinventing Mergesort

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

Reinventing Mergesort

- "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

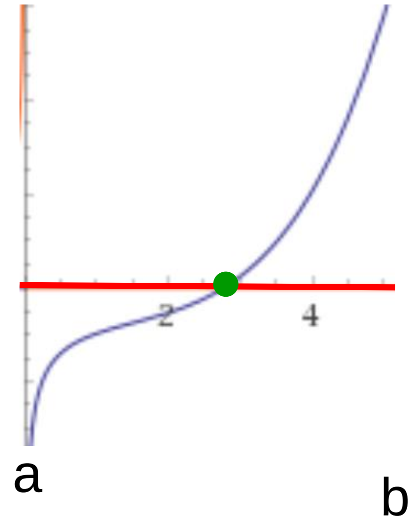
Given a continuous function f and two points $a < b$ such that

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

Find an approximate root of f (a point c where $f(c) = 0$).

f has a root in $[a, b]$ by intermediate value theorem

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



Note that roots of f may be irrational,
So, we want to approximate the root with an arbitrary precision!

A Naive Approach

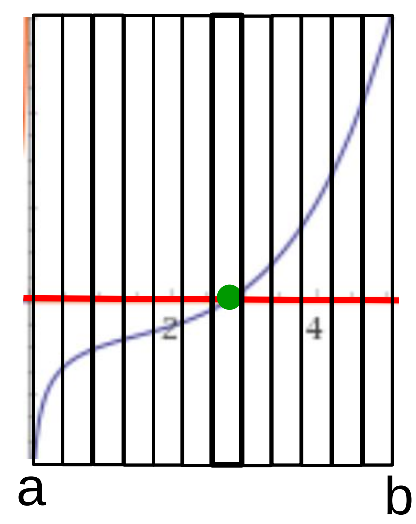
Suppose we want ϵ approximation to a root.

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

Can we do faster?



Divide & Conquer (Binary Search)

Bisection (a, b, ε)

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

return a;

else

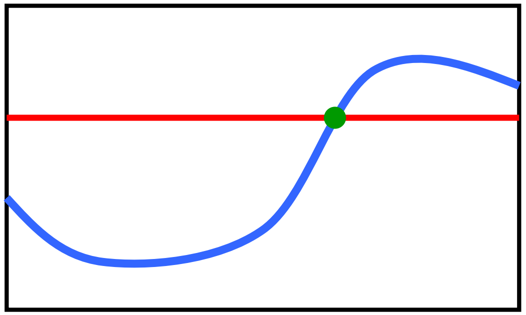
$m \leftarrow (a + b)/2$;

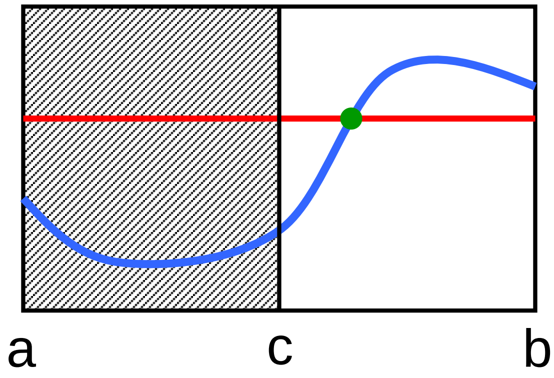
if $f(m) \leq 0$ then

return Bisection(c, b, ε);

else

return Bisection(a, c, ε);

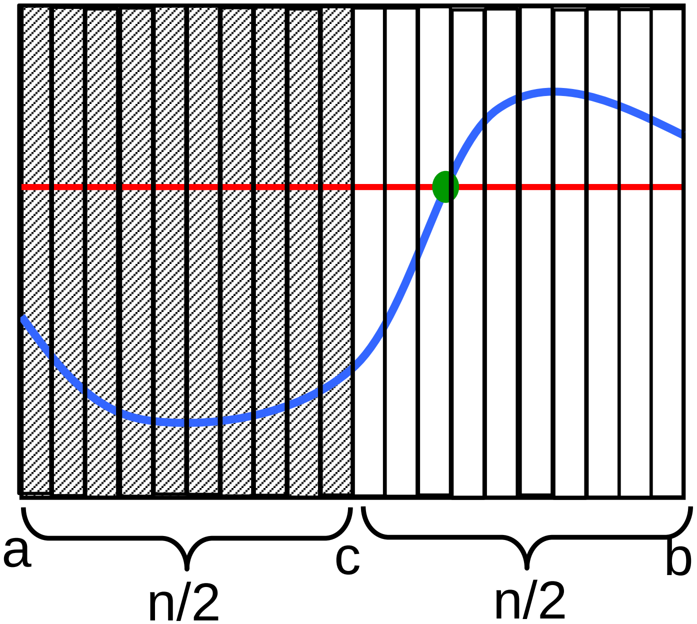




Time Analysis

Let $n = \frac{a-b}{\epsilon}$ be the # of intervals 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., $T(n) = O(\log n) = O(\log(\frac{a-b}{\epsilon}))$

Shameless plug: How about k dimension?

Yes, binary search still works for some functions. It takes $O(k^3 \log(\frac{k}{\varepsilon}))$ ☹.

Fast Exponentiation

Fast Exponentiation

- Power(a, n)

  Input: integer $\boldsymbol{n} \geq \mathbf{0}$ and number $\boldsymbol{a}$

  Output: $\boldsymbol{a}^{\boldsymbol{n}}$

- Obvious algorithm

  $\boldsymbol{n} - \mathbf{1}$ multiplications

- Observation:

  if $\boldsymbol{n}$ is even, then $a^n = a^{n/2} \cdot a^{n/2}$.

Divide & Conquer (Repeated Squaring)

```
Power(a,n) {
   if (n = 0)
       return 1
   else if (n is even)
       Compute x = Power(a,n/2)
       return x • x
   else
       Compute x = Power(a,(n − 1)/2)
       return x • x • a
}
```

Time (# of multiplications):

$T(n) \le T(\lfloor n/2 \rfloor) + 2$ for $n \ge 1$

$T(0) = 0$

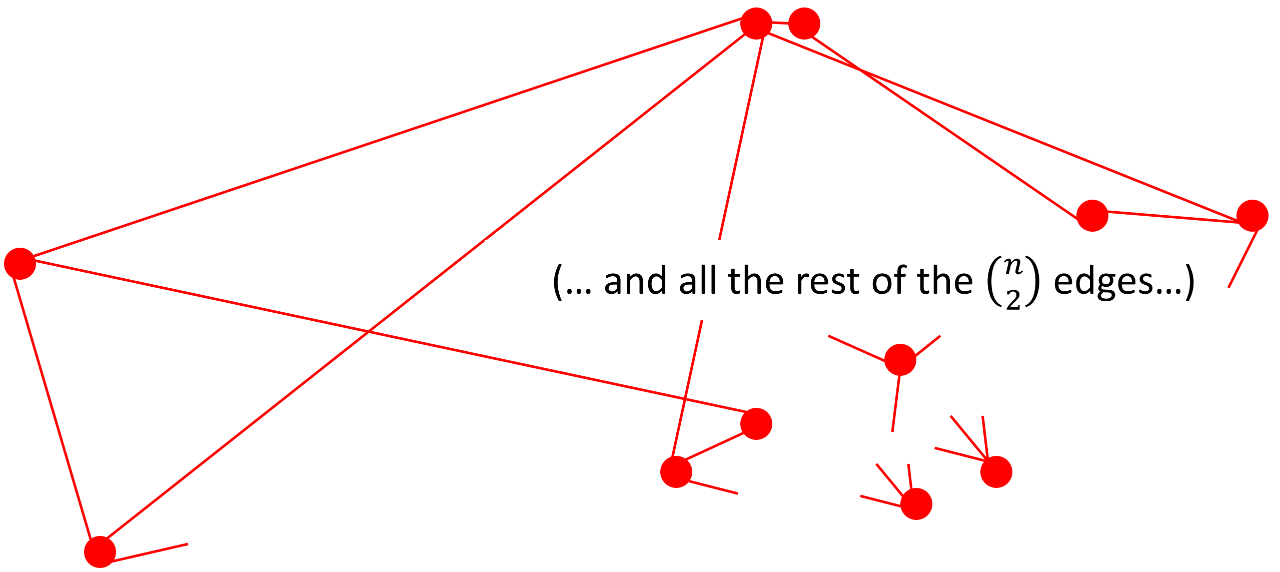
Solving it, we have

$$T(n) \le T(\lfloor n/2 \rfloor) + 2 \le T(\lfloor n/4 \rfloor) + 2 + 2$$
$$\le \cdots \le T(1) + \underbrace{2 + \cdots + 2}_{\log_2(n) \text{ copies}} \le 2 \log_2 n.$$

Finding the Closest Pair of Points

Closest Pair of Points (general metric)

Given n points and arbitrary distances between them, find the closest pair.



Must look at all $\binom{n}{2}$ pairwise distances, else any one you didn't check might be the shortest.

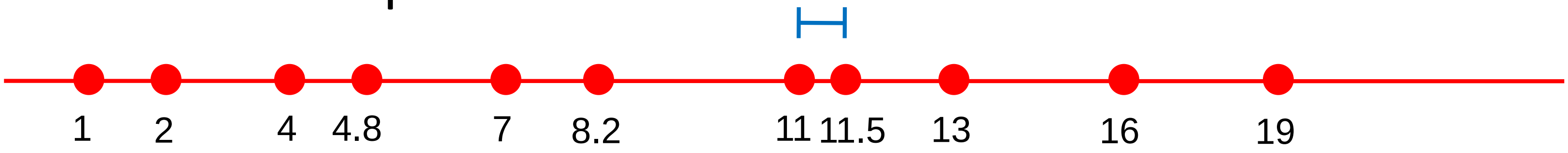
i.e., you have to read the whole input.

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 calculate 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.

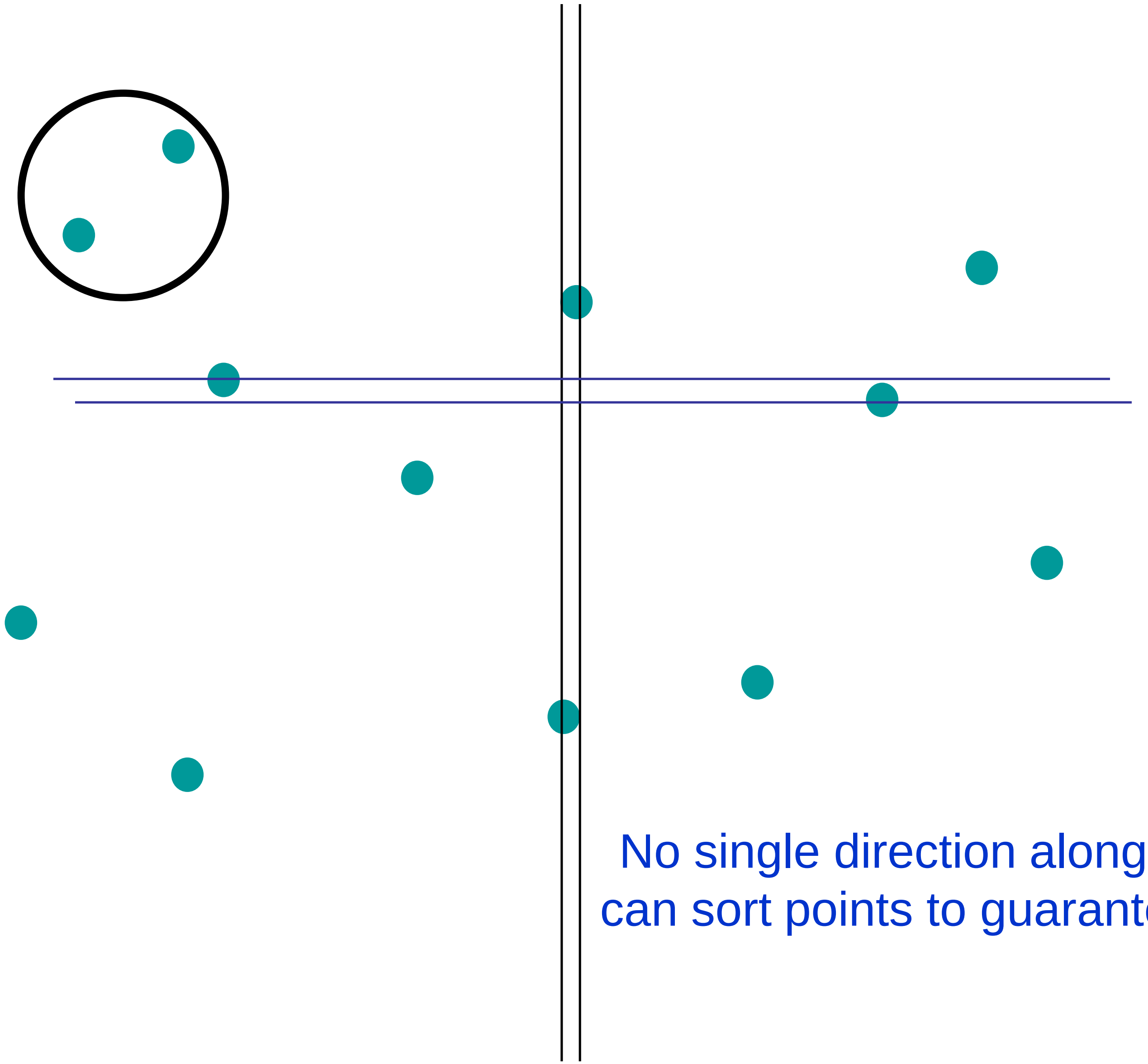
Fundamental geometric primitive.

- 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 in $\Theta(n^2)$ time.

Assumption: No two points have same x coordinate.

Closest Pair of Points (2-dimensions)



No single direction along which one can sort points to guarantee success!

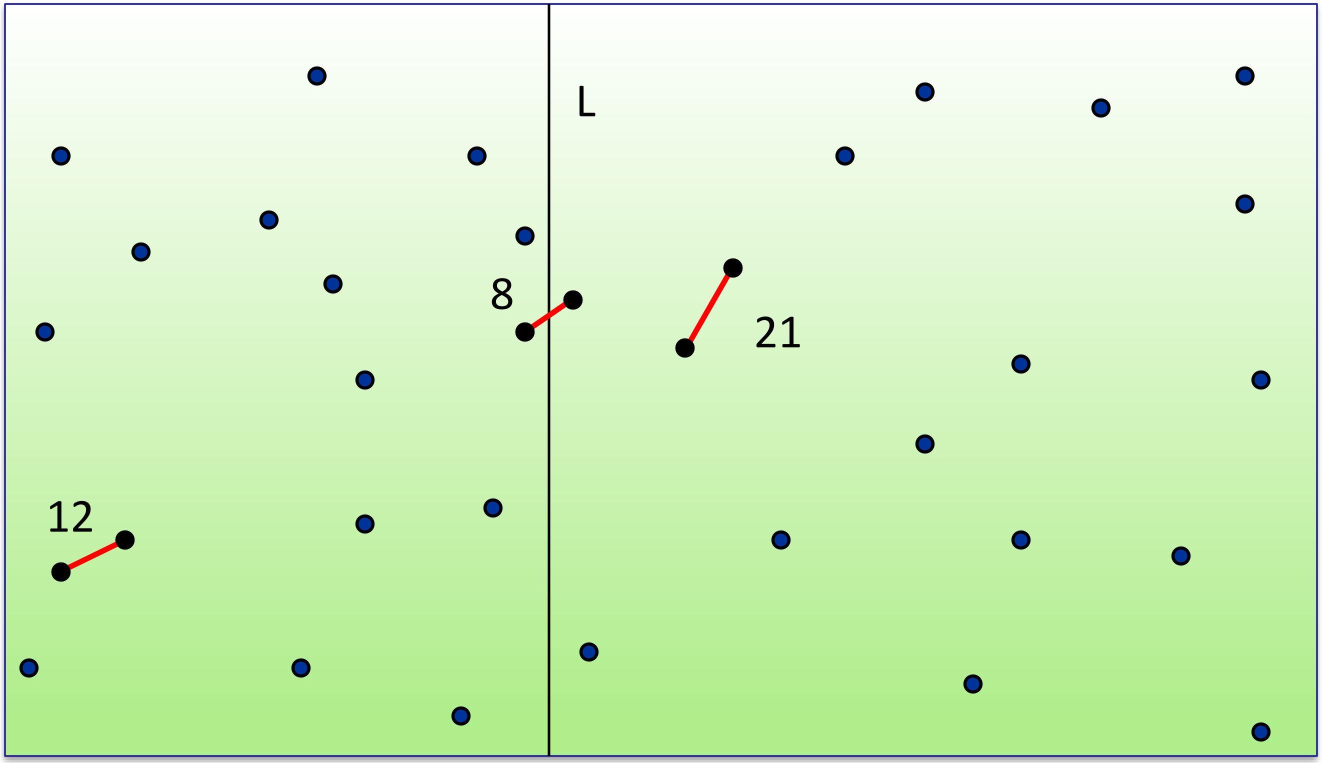
Divide & Conquer

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 ← How ?

Return best solutions



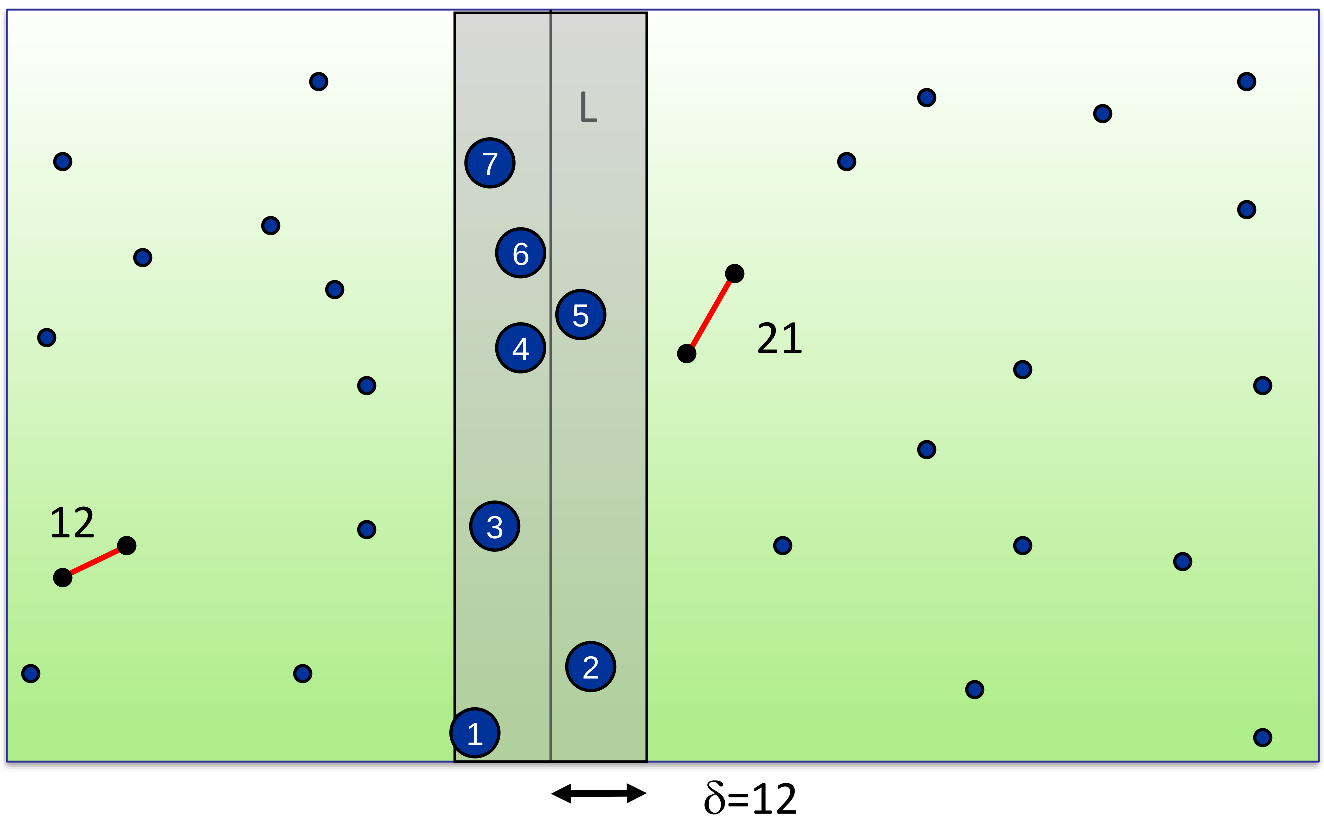
Key Observation

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.

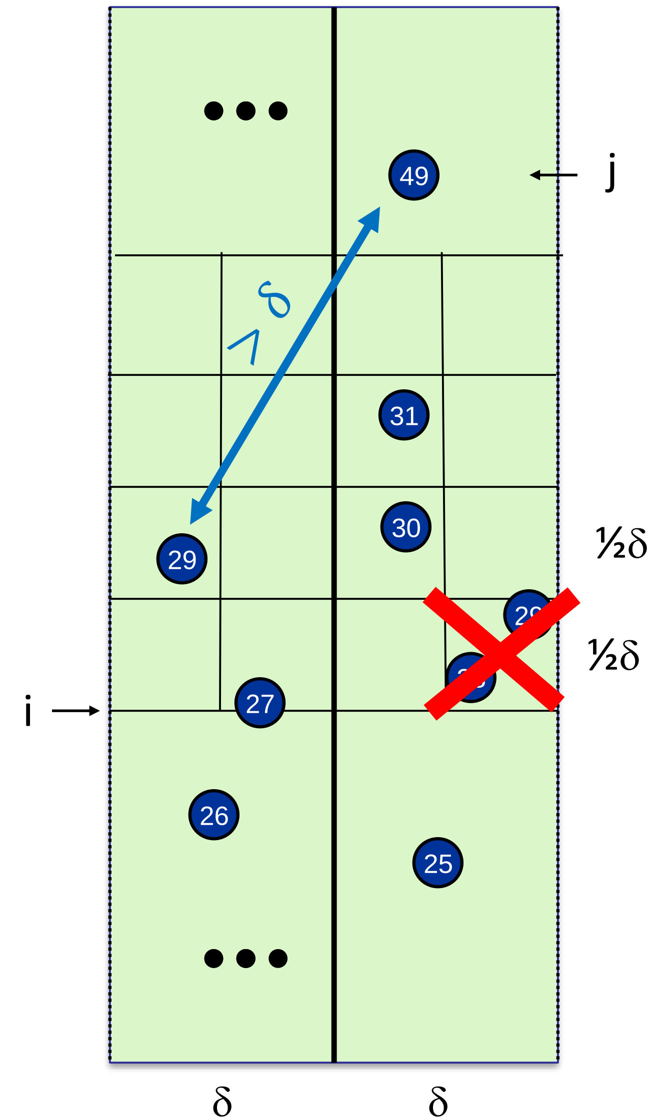
Proof: Such points would be within

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

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 $> \delta$.

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



Closest Pair (2 dimension)

```
Closest-Pair(p_1, p_2, ..., p_n) {
    if(n ≤ 2) return |p_1 - p_2|

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

    δ_1 = Closest-Pair(left half)
    δ_2 = Closest-Pair(right half)
    δ  = min(δ_1, δ_2)

    Delete all points further than δ from separation line L

    Sort remaining points p[1]…p[m] by y-coordinate.

    for i = 1, 2, ..., m
        for k = 1, 2, ..., 11
          if i + k ≤ m
             δ = min(δ, distance(p[i], p[i+k]));

    return δ.
}
```

Closest Pair Analysis

Let $D(n)$ be the number of pairwise distance calculations in the Closest-Pair Algorithm

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

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

What if we counted running time?

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

Closest Pair (2 dimension) Improved

```
Closest-Pair(p_1, p_2, ..., p_n) {
    if(n ≤ 2) return |p_1 − p_2|

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

   (δ_1, p_1) = Closest-Pair(left half)
   (δ_2, p_2) = Closest-Pair(right half)
    δ         = min(δ_1, δ_2)
    p_sorted  = merge(p_1, p_2)   (merge sort it by y-coordinate)

    Let q be points (ordered as p_sorted) that is δ from line L.

    for i = 1, 2, ..., m
        for k = 1, 2, ..., 11
          if i + k ≤ m
              δ = min(δ, distance(q[i], q[i+k]));

    return δ and p_sorted.
}
```

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

$$\Rightarrow D(n) = O(n \log n)$$