CS 401

Master Theorem / Closest Points

Xiaorui Sun

Divide and Conquer

Divide: We reduce a problem to several subproblems.

Typically, each sub-problem is at most a constant $c < 1$ fraction of the size of the original problem

Conquer: Recursively solve each subproblem

Combine: Merge the solutions

Examples:

- Mergesort, Counting Inversions, Binary Search

Master Theorem

Master Theorem

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

c: absolute constant

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

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 $n \leq b$.

Question

Consider the following recurrence. Which case of the master theorem?

$$T(n) = \begin{cases} 0 & \text{if } n \le 1 \\ T\left(\left\lfloor \frac{n}{5} \right\rfloor\right) + T\left(n - 3\left\lceil \frac{n}{10} \right\rceil\right) + \frac{11}{5}n & \text{if } n > 1 \end{cases}$$

- A. $T(n) = \Theta(n)$
- B. $T(n) = \Theta(n \log n)$
- C. $T(n) = \Theta(n^2)$
- D. Master theorem not applicable

Master Theorem

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

$$T(n) = \begin{cases} \Theta(n^k) & \text{if } a < b^k \\ \Theta(n^k \log n) & \text{if } a = b^k \\ \Theta(n^{\log_b a}) & \text{if } a > b^k \end{cases}$$

Akra–Bazzi theorem
Wiki!

How to use master theorem?

For a divide and conquer algorithm

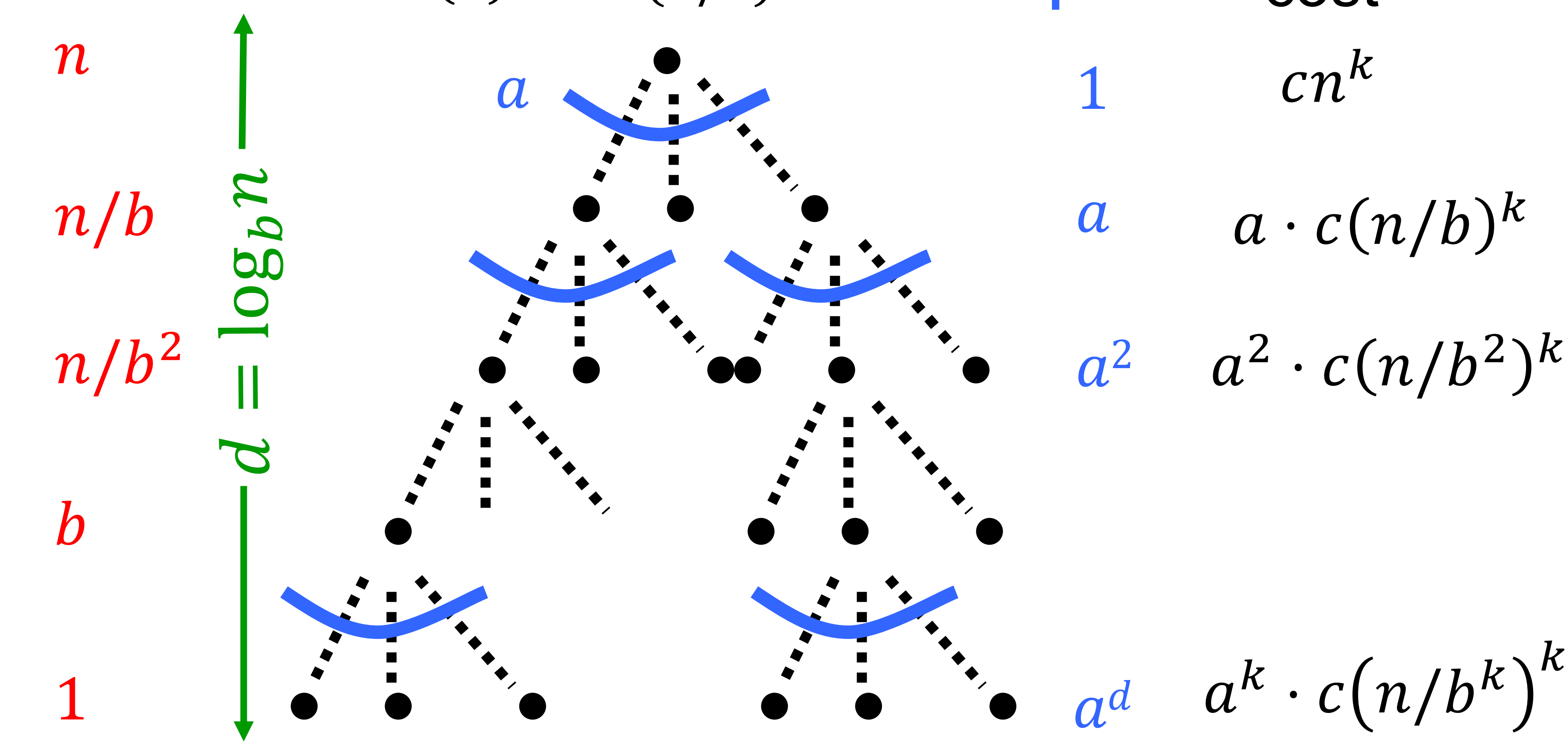
- a: number of subproblems
- b: ratio of problem size / subproblem size
- $c \cdot n^k$: running time of divide and combine step

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

Example: Mergesort have two subproblems of half size of the original problem, and the cost of divide and combine step is O(n), so T(n) = 2 T(n / 2) + c n, which implies T(n) = Θ(n log n)

Understand Master Theorem

$T(n) = aT(n/b) + cn^k$



$$T(n) = \sum_{i=0}^{d=\log_b n} a^i c \left(\frac{n}{b^i}\right)^k$$

Understand Master Theorem

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

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

# of problems increases slower than the decreases of cost. Top term dominates.

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

# of problems increases faster than the decreases of cost Bottom term dominates.

$$T(n) = \sum_{i=0}^{d=\log_b n} a^i c \left(\frac{n}{b^i}\right)^k$$

A Useful Identity

Theorem: $1 + x + x^2 + \cdots + x^d = \frac{x^{d+1}-1}{x-1}$

Proof: Let $S = 1 + x + x^2 + \cdots + x^d$

Then, $xS = x + x^2 + \cdots + x^{d+1}$

So, $xS - S = x^{d+1} - 1$

i.e., $S(x-1) = x^{d+1} - 1$

Therefore, $S = \frac{x^{d+1}-1}{x-1}$

O_x means the hidden constant depends on x

Corollary:

$$1 + x + x^2 + \cdots + x^d = \begin{cases} O_x(1) & \text{if } x < 1 \\ d+1 & \text{if } x = 1 \\ O_x\left(x^{d+1}\right) & \text{if } x > 1 \end{cases}$$

Solve: $T(n) = aT\left(\frac{n}{b}\right) + cn^k$

Corollary:

$$1 + x + x^2 + \cdots + x^d = \begin{cases} \Theta_x(1) & \text{if } x < 1 \\ \Theta(d) & \text{if } x = 1 \\ \Theta_x\left(x^{d+1}\right) & \text{if } x > 1 \end{cases}$$

Going back, we have

$$T(n) = \sum_{i=0}^{d=\log_b n} a^i c \left(\frac{n}{b^i}\right)^k = cn^k \sum_{i=0}^{d=\log_b n} \left(\frac{a}{b^k}\right)^i$$

Hence, we have

$$T(n) = \Theta\left(n^k\right) \begin{cases} 1 & \text{if } a < b^k \\ \log_b n & \text{if } a = b^k \\ \left(\frac{a}{b^k}\right)^{\log_b n} & \text{if } a > b^k \end{cases}$$

Solve: $T(n) = aT\left(\frac{n}{b}\right) + cn^k$

$$T(n) = \Theta(n^k)\begin{cases} 1 & \text{if } a < b^k \\ \log_b n & \text{if } a = b^k \\ \left(\frac{a}{b^k}\right)^{\log_b n} & \text{if } a > b^k \end{cases}$$

For $a < b^k$, we simply have $T(n) = \Theta(n^k)$.

For $a = b^k$, we have $T(n) = \Theta(n^k \log_b n) = \Theta(n^k \log n)$.

For $a > b^k$, we have $T(n) = \Theta\left(n^k \left(\frac{a}{b^k}\right)^{\log_b n}\right) = \Theta(n^{\log_b a})$.

$$b^{k \log_b n} = \left(b^{\log_b n}\right)^k = n^k$$

$$a^{\log_b n} = (b^{\log_b a})^{\log_b n} = (b^{\log_b n})^{\log_b a} = n^{\log_b a}$$

Finding the Closest Pair of Points

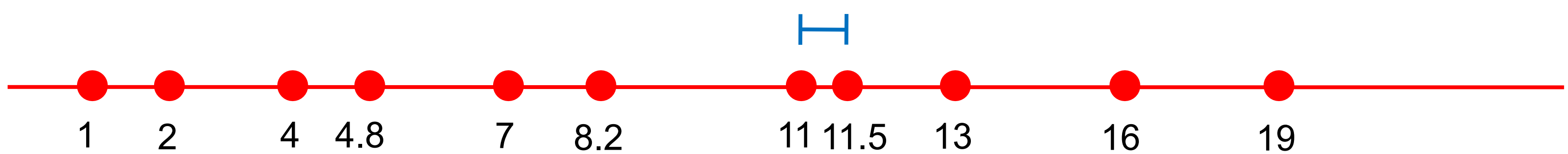
Closest Pair of Points (1-dim)

Given n points, find the closest pair.

Brute force: Check all $\binom{n}{2}$ pairwise distances

1-dim case: 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

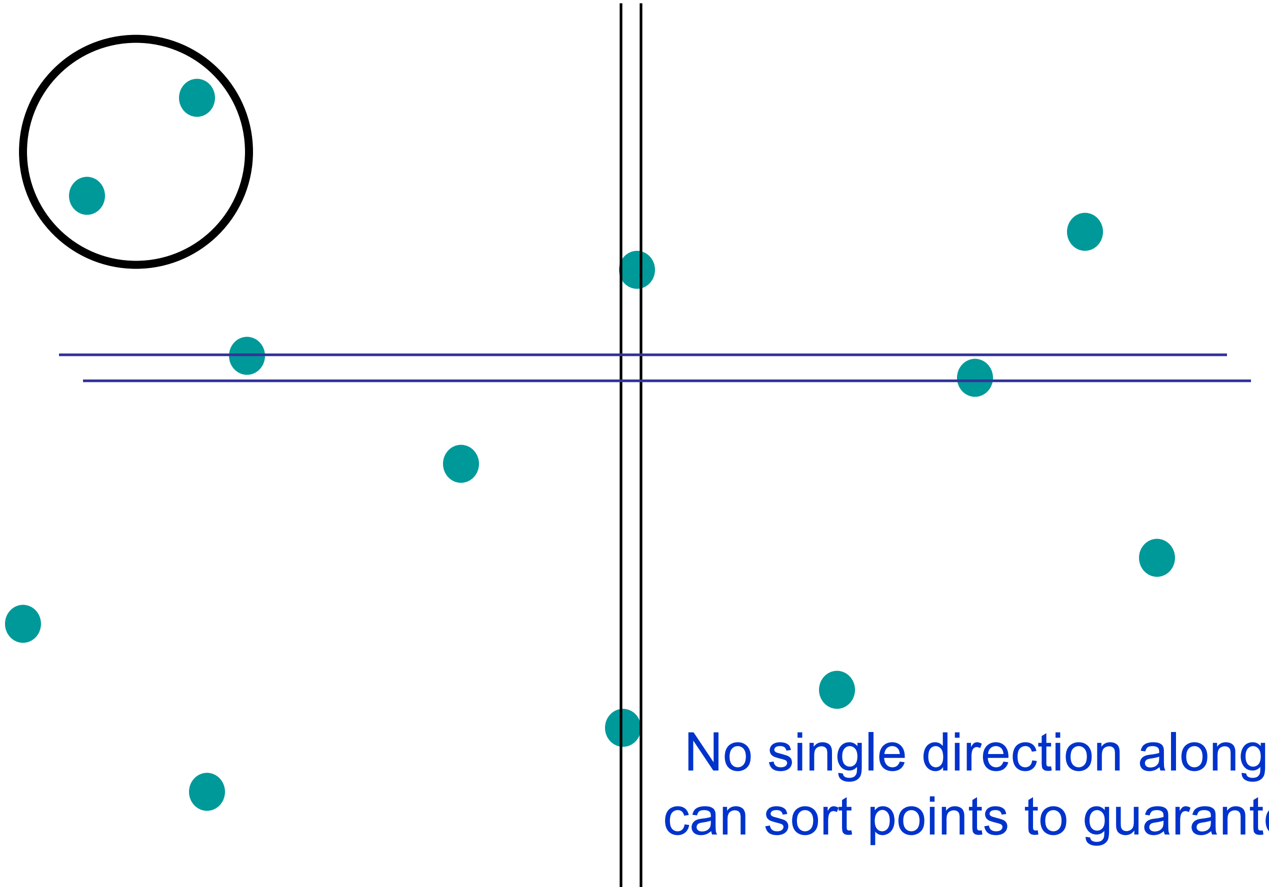
So, first sort, then scan adjacent pairs.

Time $O(n \log n)$ to sort, if needed, Plus $O(n)$ to scan adjacent pairs

Closest Pair of Points (2-dim)

Given n points in the plane, find a pair with smallest Euclidean distance between them.

Idea: make use of 1-dim algorithm (but not in a simple way)



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

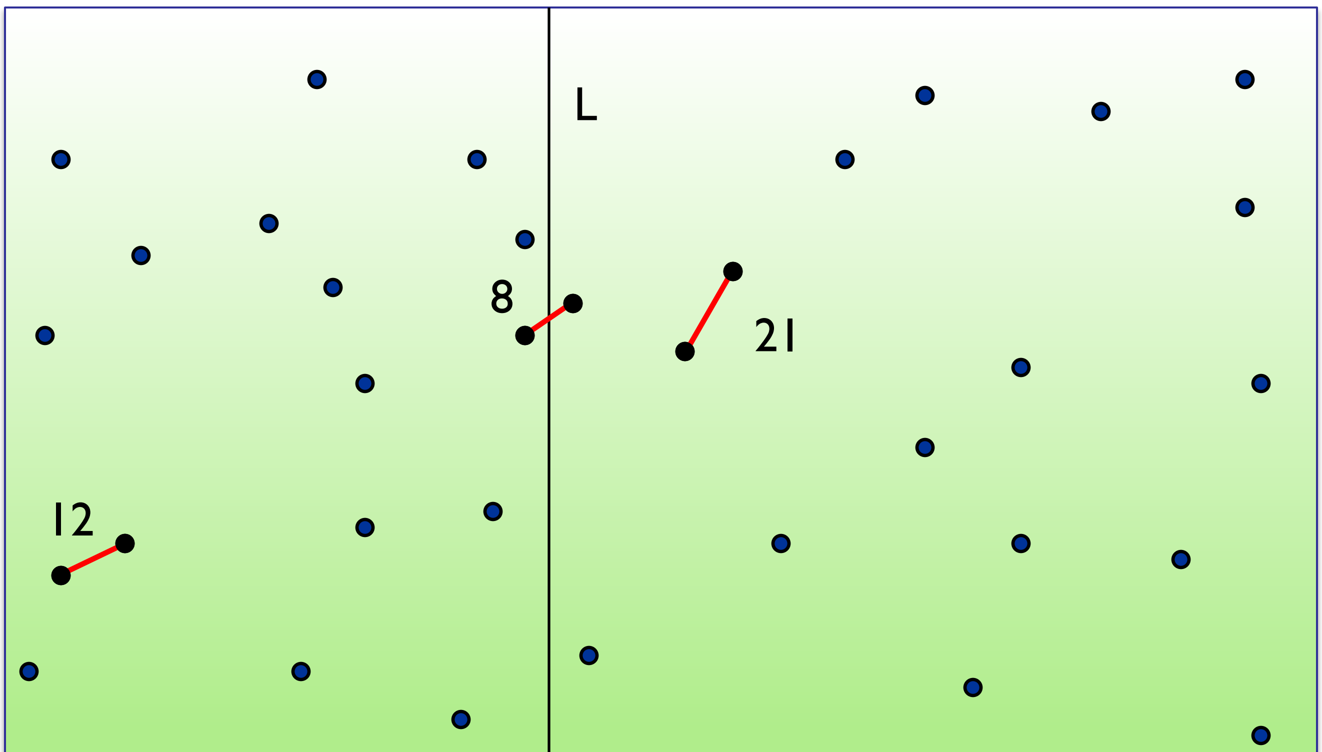
Divide & Conquer

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



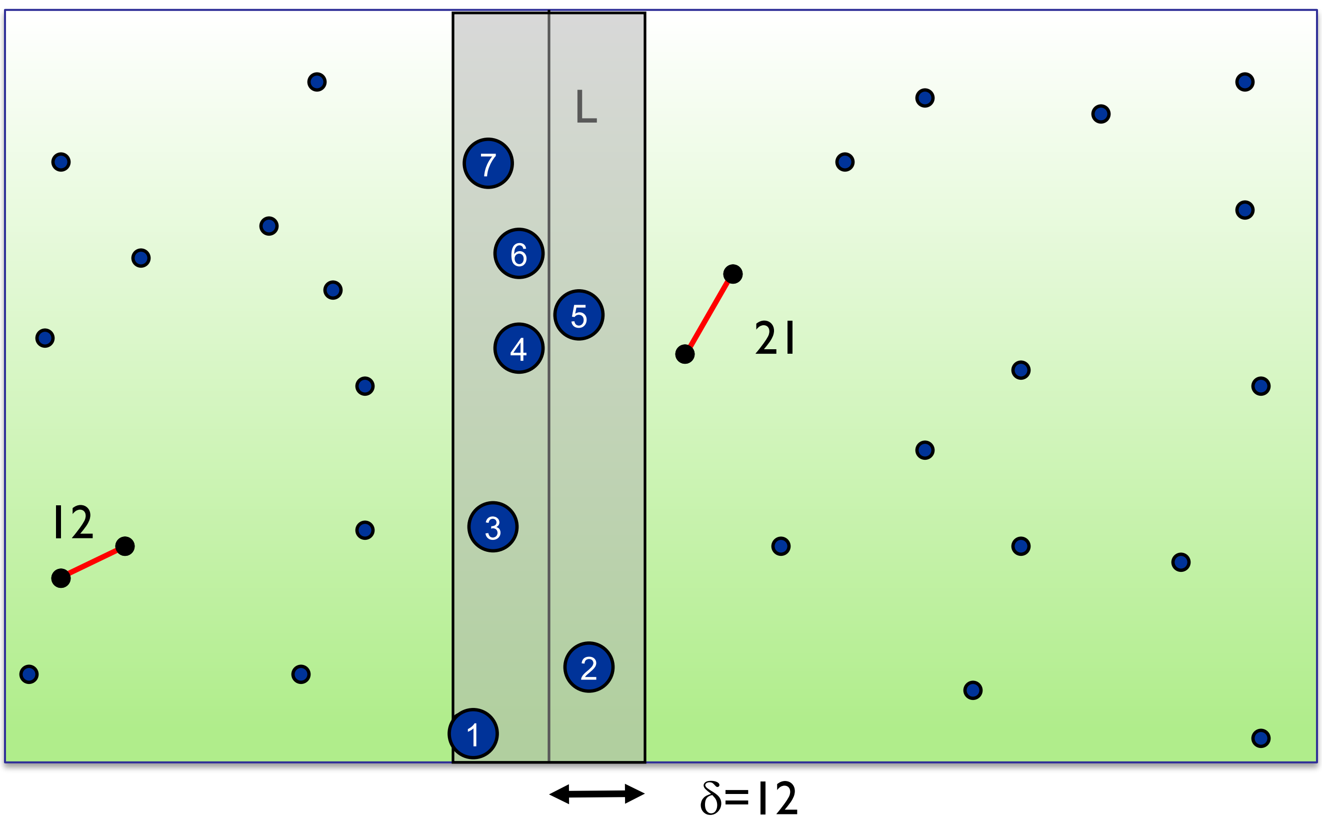
Key Observation

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.

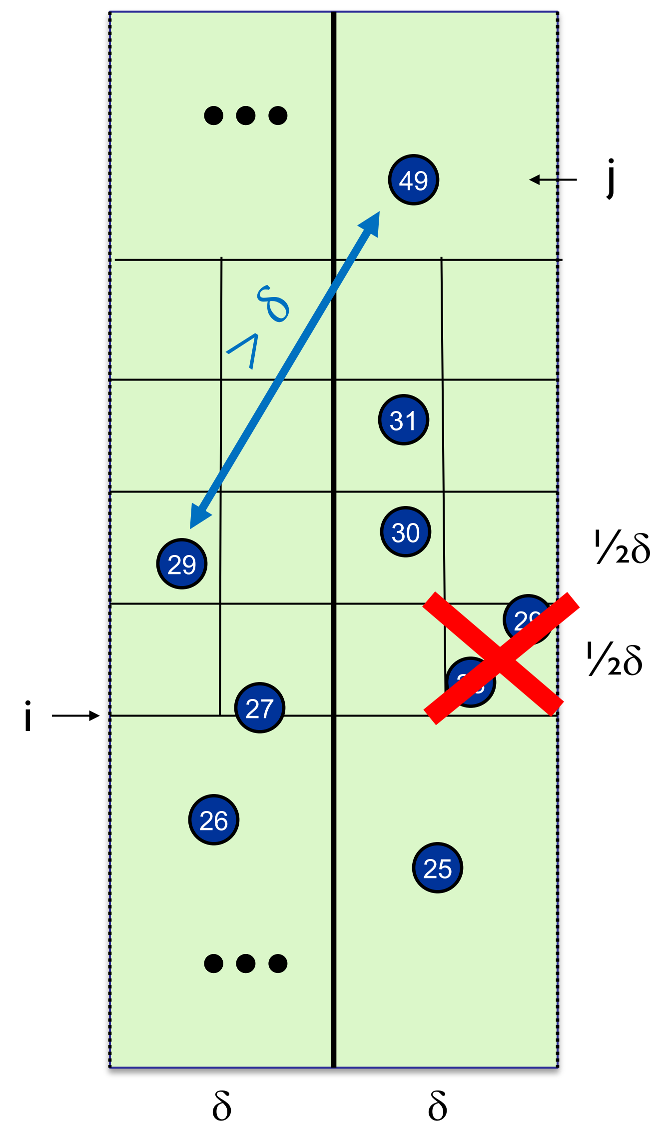
Proof: Such points would be within

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

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 distance(p_1, p_2)

    Compute separation line L such that half the points
    are on one side and half on the other side.                    O(n log n)

    δ_1 = Closest-Pair(left half)
    δ_2 = Closest-Pair(right half)
    δ  = min(δ_1, δ_2)                                               O(1)

    Delete all points further than δ from separation line L          O(n)

    Sort remaining points p[1]…p[m] by y-coordinate.                 O(n log n)

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

    return δ.
}
```

Closest Pair Analysis

Running time?

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

Can we do better?

Closest Pair (2 dimension) Improved

```
Closest-Pair(p_1, p_2, ..., p_n) {
    if (n ≤ 2) return distance(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, Q_1) = Closest-Pair(left half)
   (δ_2, Q_2) = Closest-Pair(right half)
    δ        = min(δ_1, δ_2)
    Q_sorted = merge(Q_1, Q_2)   (merge sort it by y-coordinate)

    Let S be points (ordered as Q_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(S[i], S[i+k]));

    return δ and Q_sorted.
}
```

Assume: input sorted by x-coordinate (O(n log n) overhead initially)

$O(1)$

$O(1)$

$O(n)$

$O(n)$

$O(n)$

Input sorted by y-coordinate

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

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

Summary

Closest pair in 2-dimension: Given n points in the plane, find a pair with smallest Euclidean distance between them.

Brute Force: Check all pairs of points in $\Theta(n^2)$ time.

Divde and Conquer:

- 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

Exercise: Remove the assumption of “no two points have same x coordinate”?