Finding closest pair of points

**Problem**
Given a set of points \(\{p_1, \ldots, p_n\}\) find the pair of points \(\{p_i, p_j\}\) that are closest together.
Brute force gives an $O(n^2)$ algorithm: just check every pair of points.

Can we do it faster? Seems like no: don’t we have to check every pair?

In fact, we can find the closest pair in $O(n \log n)$ time.

What’s a reasonable first step?
Split the points with line $L$ so that half the points are on each side.

Recursively find the pair of points closest in each half.
Merge: the hard case

Let \( d = \min\{d_{\text{left}}, d_{\text{right}}\} \).

- \( d \) would be the answer, except maybe \( L \) split a close pair!
If there is a pair \( \{p_i, p_j\} \) with \( \text{dist}(p_i, p_j) < d \) that is split by the line, then both \( p_i \) and \( p_j \) must be within distance \( d \) of \( L \).

Let \( S_y \) be an array of the points in that region, sorted by decreasing \( y \)-coordinate value.
• Let $S_y$ be an array of the points in that region, sorted by decreasing $y$-coordinate value.

• $S_y$ might contain all the points, so we can’t just check every pair inside it.

**Theorem**

Suppose $S_y = p_1, \ldots, p_m$. If $\text{dist}(p_i, p_j) < d$ then $j - i \leq 15$.

In other words, if two points in $S_y$ are close enough in the plane, they are close in the array $S_y$. 

Divide the region up into squares with sides of length \( d/2 \):

How many points in each box?

At most 1 because each box is completely contained in one half and no two points in a half are closer than \( d \).
Divide the region up into squares with sides of length $d/2$:

How many points in each box?

At most 1 because each box is completely contained in one half and no two points in a half are closer than $d$. 
Suppose 2 points are separated by $> 15$ indices.

- Then, at least 3 full rows separate them (the packing shown is the smallest possible).
- But the height of 3 rows is $> 3d/2$, which is $> d$.
- So the two points are farther than $d$ apart.
Therefore, we can scan $S_y$ for pairs of points separated by $< d$ in linear time.

ClosestPair(Px, Py):
    if $|Px| == 2$: return dist(Px[1], Px[2]) // base

    $d_1 = \text{ClosestPair(FirstHalf(Px, Py))}$ // divide
    $d_2 = \text{ClosestPair(SecondHalf(Px, Py))}$
    $d = \min(d_1, d_2)$

    $Sy = \text{points in Py within } d \text{ of } L$ // merge
    For $i = 1, \ldots, |Sy|$: For $j = 1, \ldots, 15$:
        $d = \min(\text{dist}(Sy[i], Sy[j]), d )$
    Return $d$
Total Running Time:

- Divide set of points in half each time: $O(\log n)$ depth recursion
- Merge takes $O(n)$ time.
- Recurrence: $T(n) \leq 2T(n/2) + cn$
- Same as MergeSort $\implies O(n \log n)$ time.