403: Algorithms and Data Structures

Lower Bound for Sorting
&
The closest pair in 2D

Fall 2016

UAlbany

Computer Science

So far: Sorting

Algorithm	*Time*	*Space*
• **Insertion**	$O(n^2)$	in-place
• **Merge**	$O(n \log n)$	2nd array to merge
• **Heapsort**	$O(n \log n)$	in-place
• **Quicksort**	from $O(n \log n)$ to $O(n^2)$	in-place

Can we do better than $O(n \log n)$?

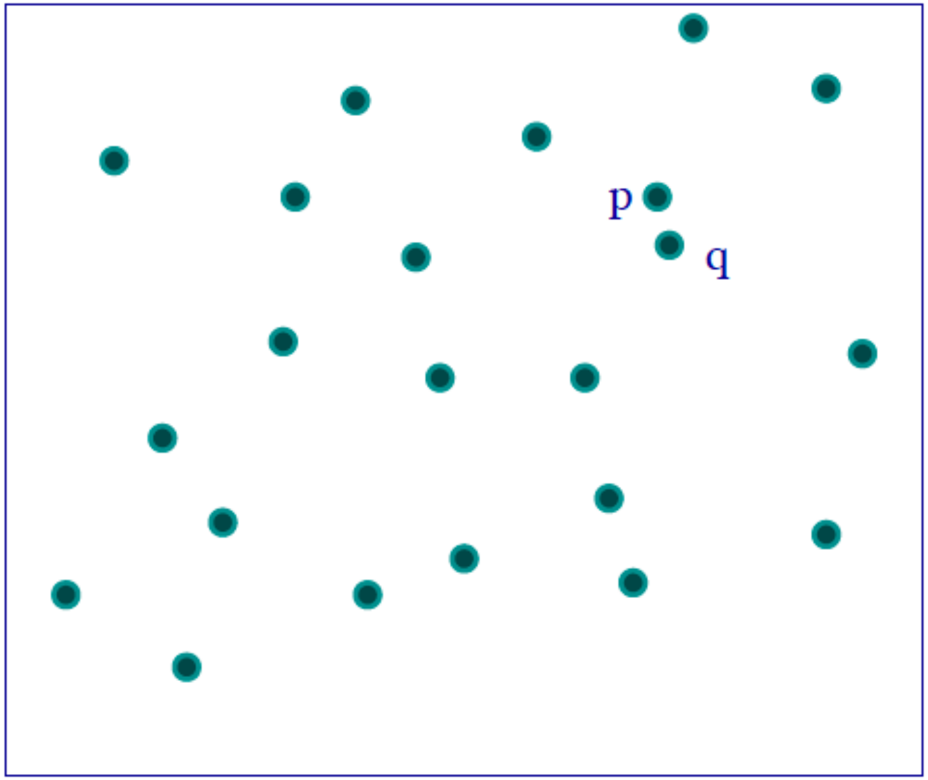
<u>**Spoiler: Not if we do comparisons only**</u>

Lower bound on comparison sorts

- Any algorithm performing only comparisons runs in nlogn)
- We will prove this using the concept of **decision trees**

Closest pair in 2D

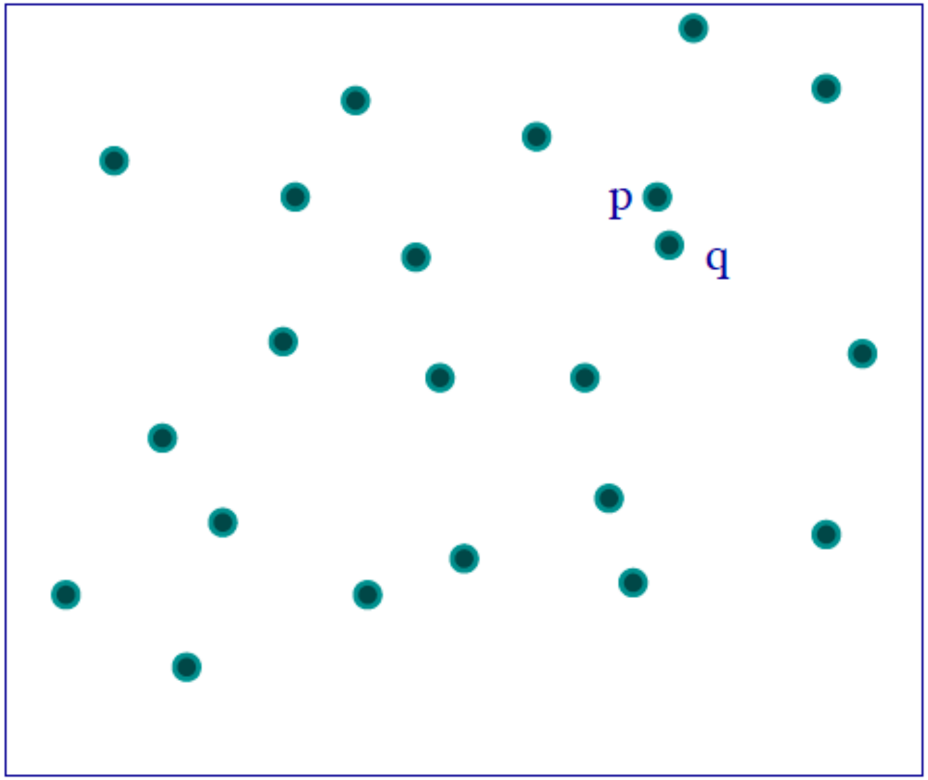
- Given n points in 2-dimensions, find two whose mutual distance is smallest.
- Euclidean distance



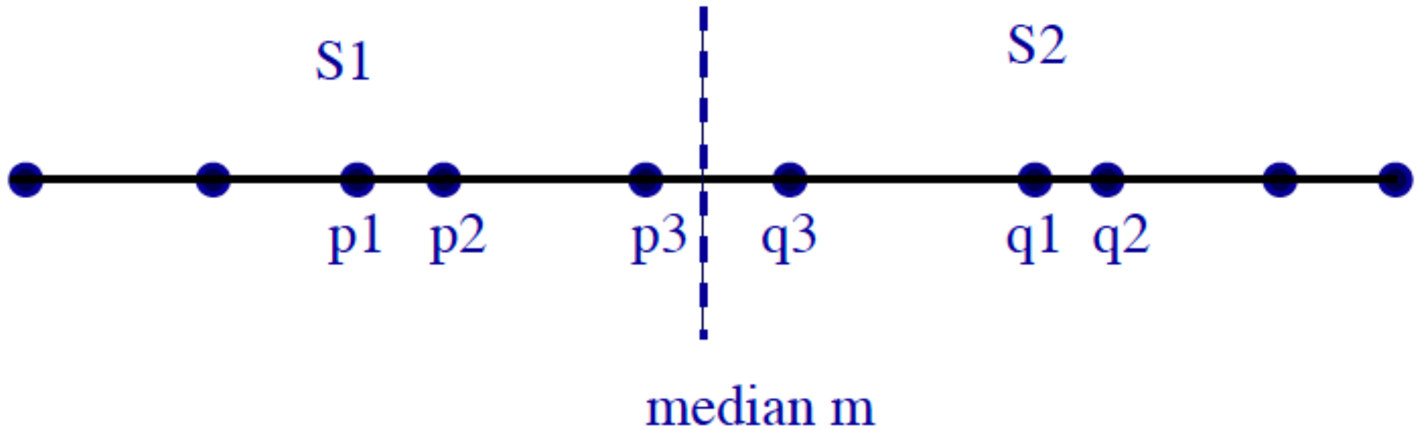
$$d(p,q) = \sqrt{(x_p - x_q)^2 + (y_p - y_q)^2}$$

Closest pair in 2D

- Brute force?
  - Consider all pairs
- Complexity?
  - $O(n^2)$

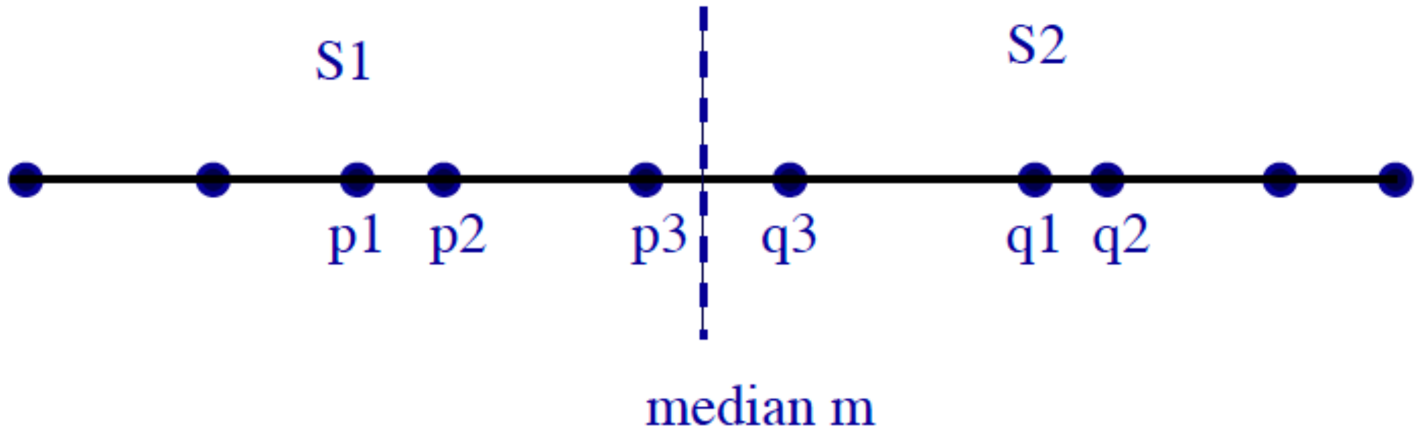


Divide-And-Conquer (1D)



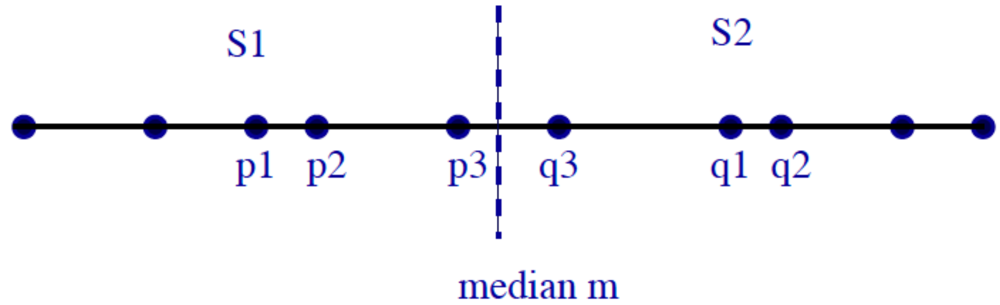
- We can simply sort and consider consecutive pairs O(nlogn)
  - Does not generlize to 2D

Divide-And-Conquer (1D)



- DIVIDE: split array in two equal parts
- CONQUER: recursively find closest pair in parts
- COMBINE:
  - Let **d** be the smallest separation in **S1** and **S2**
  - If **dist(p3,q3)<d** return **dist(p3,q3)** else **d**

Divide-And-Conquer (1D) Pseudo code



Closest-Pair-1D(S)

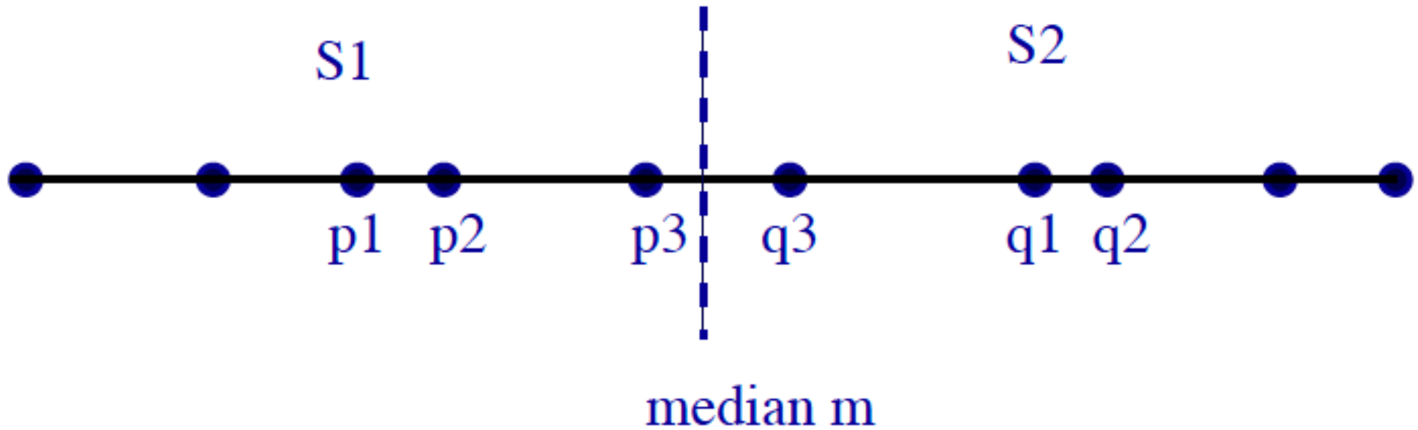
If |S|= 1, output d = infinity

If |S|= 2, output d= |p2-p1|

Otherwise, do the following steps:

1. Let m = median(S)
2. Divide S into S1, S2 at m.
3. d1 = Closest-Pair-1D(S1).
4. d2 = Closest-Pair-1D(S2).
5. d12 is minimum distance across the cut.
6. Return d = min(d1; d2; d12)

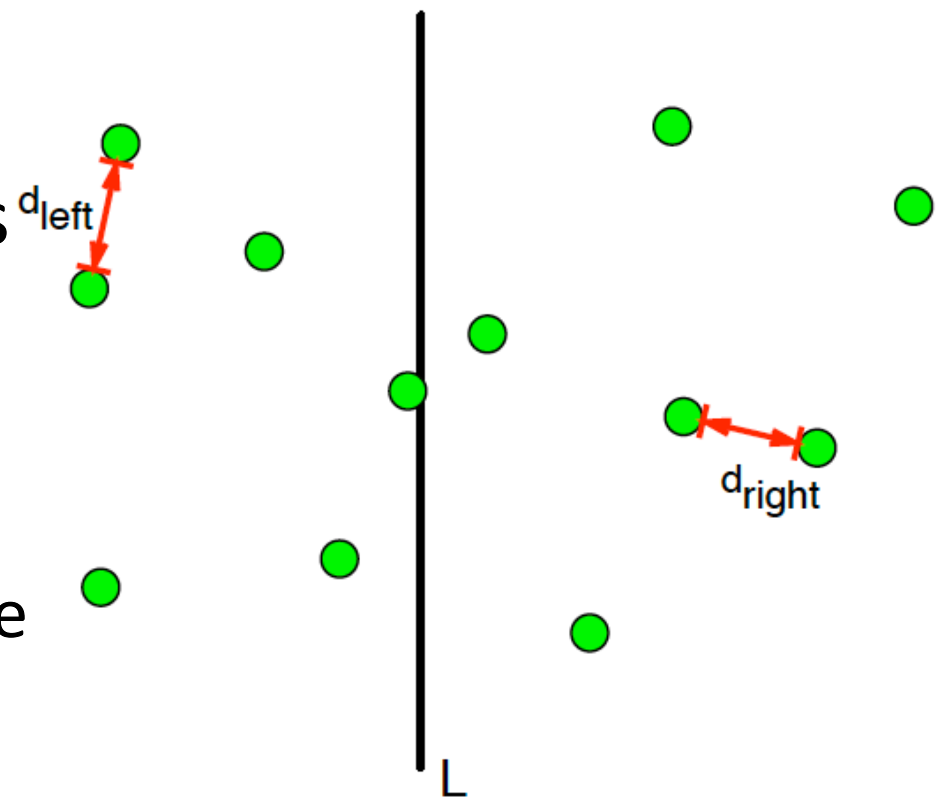
Divide-And-Conquer (1D)



- Key observation: If m is the dividing coordinate, then p3, q3 must be within d of m.
  - **p3** must be the rightmost point in **S1**
  - **q3** must be the leftmost point in **S2**
  - **Hard to generalize to 2D**
- **How many points of S1 can be in (m-d,m]?**

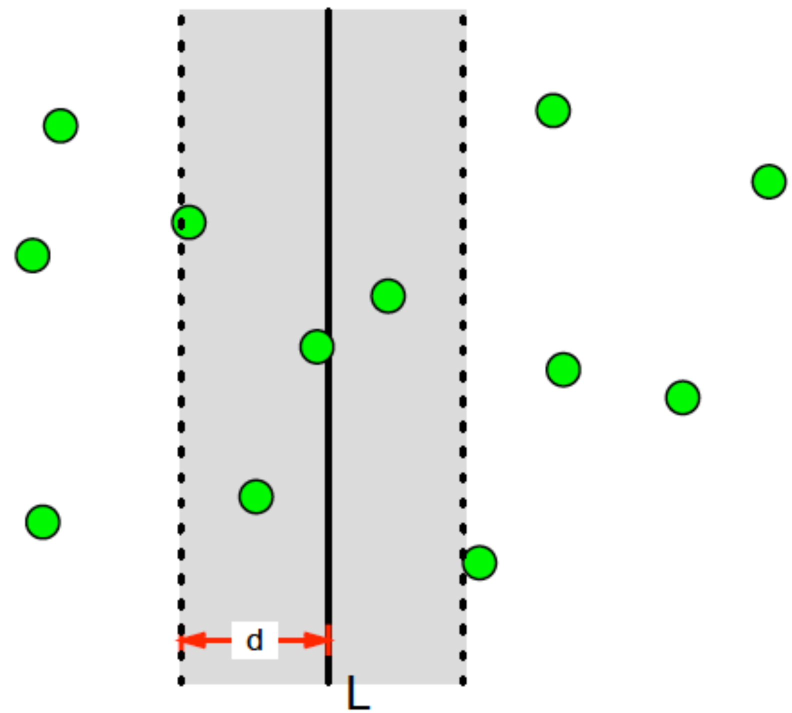
Divide-And-Conquer (2D)

- DIVIDE: split points in two equal parts with line L
- CONQUER: recursively find closest pair in parts
- COMBINE:
  - $d = \min(d_{left}, d_{right})$
  - d is the answer unless L split points that are close



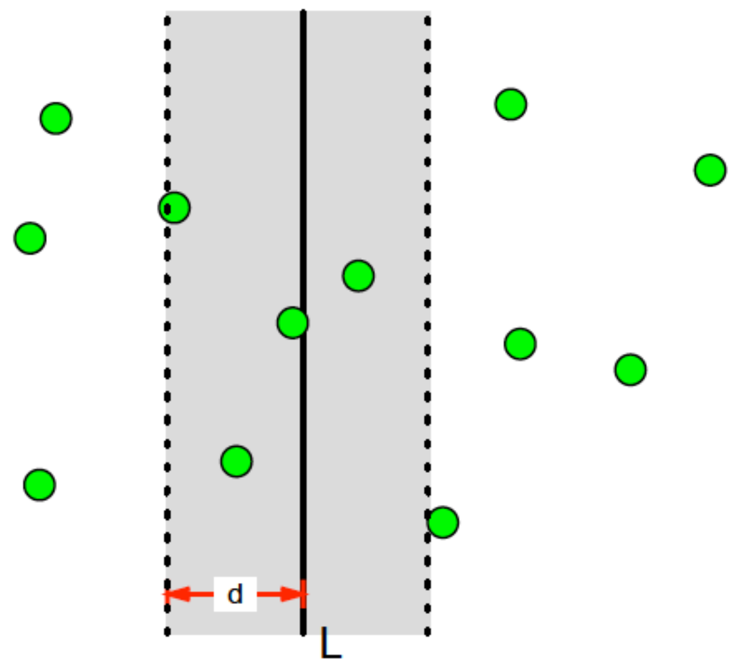
Region near L

- If there is a pair (p,q) within distance d split by L, then both p and q must be within d from L
- Let Sy be an array of points in the region sorted by y coordinate
- size of Sy might be O(|S|)
  - Cannot check all pairs

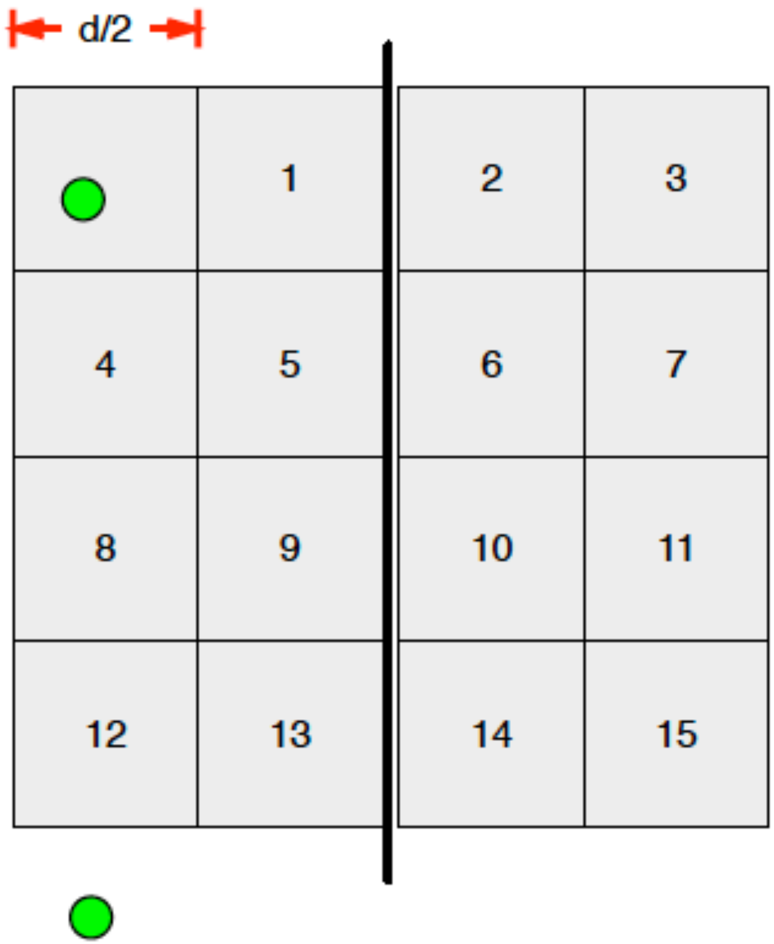


Special structure in Sy

- Let: **Sy=p1,p2...pm**, then if **dist(p_i,p_j)<d** then **j-i<=15**
- close-by points are closeby in the array

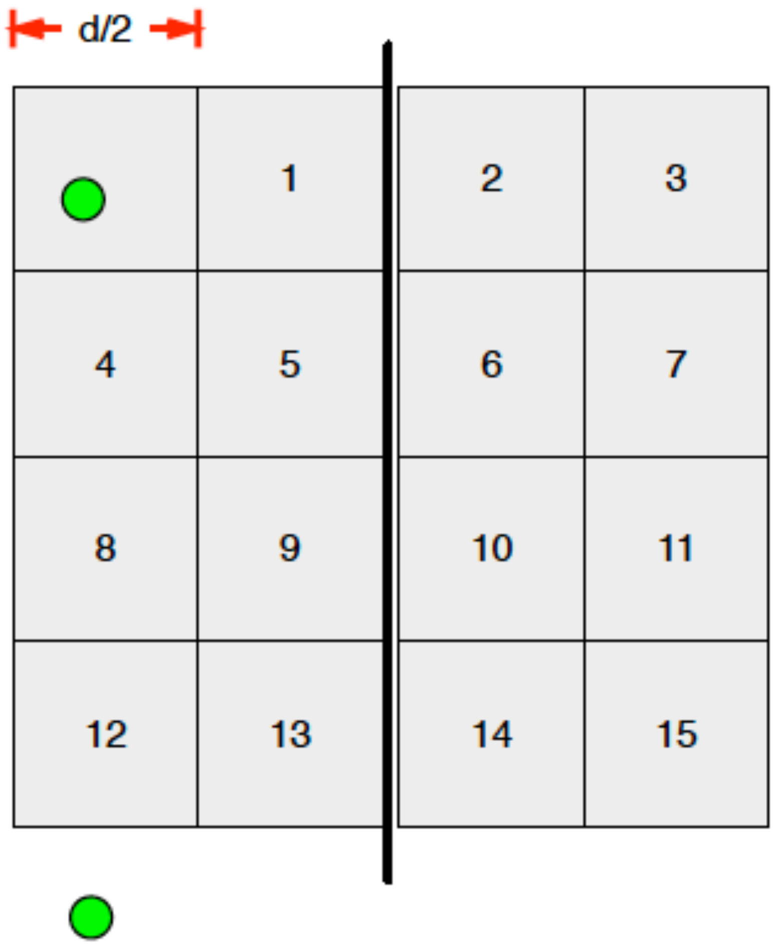


Proof: close points within 15 positions



- Divide the region in squares of side d/2
- How many points in each box?
- At most 1
  - Each box in contained in one half
  - No 2 points in a half are closer than d

Proof: close points within 15 positions



- Suppose 2 points separated by 15 indices
- At least 3 full rows separate them
- Height of 3 rows >3d/2 > d
- Points are farther than d from each other

Divide and Conquer(2D)

```
ClosestPair(ptsX, ptsY)
    if (size(ptsX)<2) return null
    if (size(ptsX)==2) return ptsX
    m=median of x coordinates
    Prepare subsets to the left of m: ptsX-left, ptsY-left and to the right of m: ptsX-right,
    ptsY-right  // They should be sorted but you should not use sorting (see book chapter)
    pair-left = ClosestPair(ptsX-left, ptsY-left)
    pair-right= ClosestPair(ptsX-right, ptsY-right)
    d = min of distances between pair-left and pair-right
    res = pair among pair-left and pair-right of the smaller distance
    ptsWithinD: an array of points within distance d from m, sorted by y coordinates
    for i=1…ptsWithinD.length
        for j=i+1…min(ptsWithinD.length,i+15)
            if dist(ptsWithinD[i], ptsWithinD[j])<d
                res = (ptsWithinD[i], ptsWithinD[j])
                b= dist(ptsWithinD[i], ptsWithinD[j])
    return res
```

DIVIDE

CONQUER

COMBINE: O(n)

Analysis

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