Algorithm Theory

Chapter 1
Divide and Conquer

Part II:
Comparing Orders & Closest Pair of Points

Fabian Kuhn

Comparing Orders

- Many web systems maintain user preferences / rankings on things like books, movies, restaurants, …

- Collaborative filtering:
  - Predict user taste by comparing rankings of different users.
  - If the system finds users with similar tastes, it can make recommendations (e.g., Amazon)

- A key problem: compare two rankings
  - Intuitively, two rankings (of movies) are more similar, the more pairs are ordered in the same way
  - Label the first user's movies from 1 to n according to ranking
  - Order labels according to second user's ranking
  - How far is this from the ascending order (of the first user)?

Number of Inversions

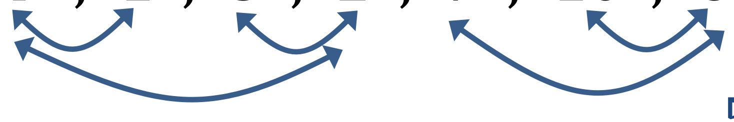
Formal problem:

- **Given**: array $A = [a_1, a_2, a_3, \dots, a_n]$ of n elements

- **Objective**: Compute number of inversions I

$$I := \left|\{0 \leq i < j \leq n \mid a_i > a_j)\}\right|$$

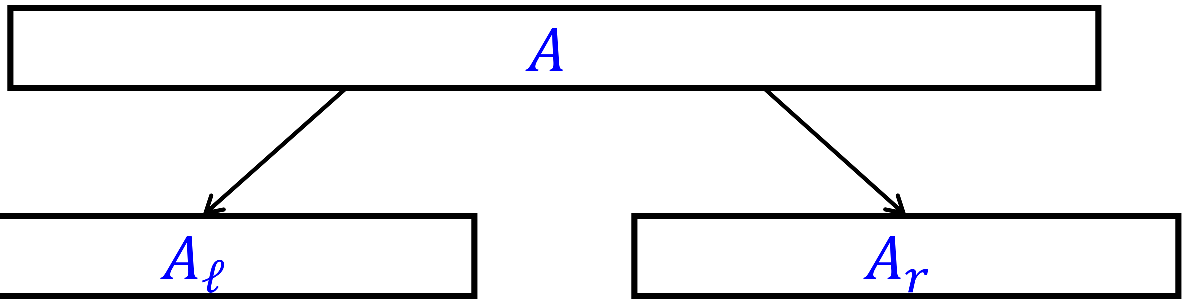
- **Example**: $A = [\ 4\ ,\ 1\ ,\ 5\ ,\ 2\ ,\ 7\ ,\ 10\ ,\ 6\ ]$



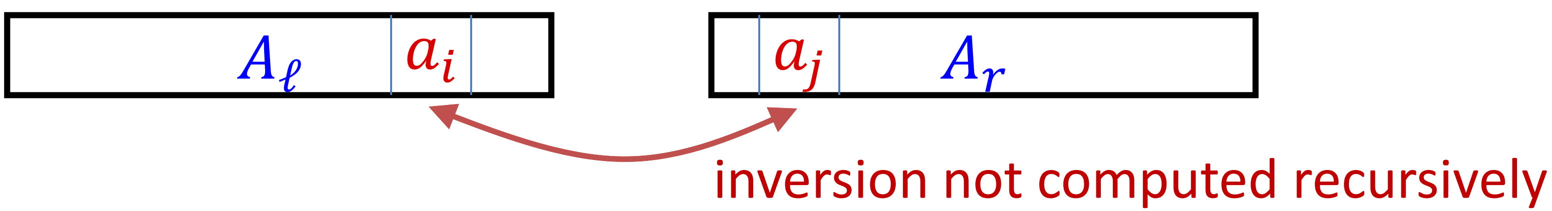
5 inversions

- **Naïve solution**:
  - Go through all pairs and check if it is an inversion
  - Time = $O(\#\text{pairs}) = O(n^2)$

Divide and Conquer Solution



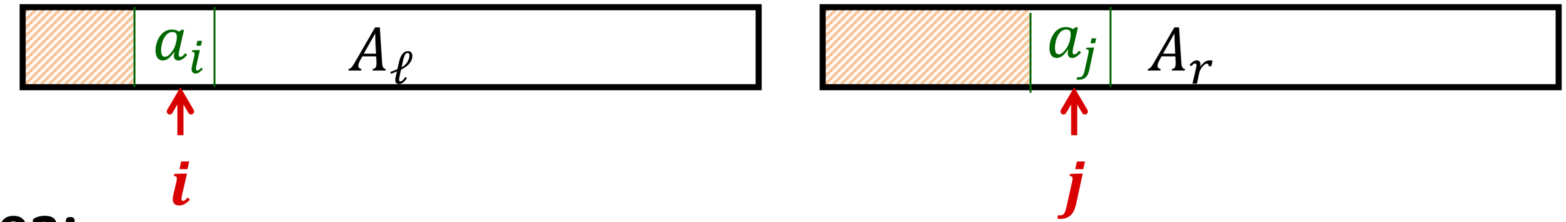
1. Divide array into 2 equal parts A_ℓ and A_r
2. Recursively compute #inversions in A_ℓ and A_r
3. Combine: add #pairs $a_i \in A_\ell$, $a_j \in A_r$ such that $a_i > a_j$



Combine: Count #pairs $a_i \in A_\ell$ and $a_j \in A_r$ for which $a_i > a_j$

Combine Step

Assume A_ℓ and A_r are sorted



Idea:

- Maintain pointers i and j to go through the sorted parts
- While going through the sorted parts, we count the number of inversions between the parts

Invariant:

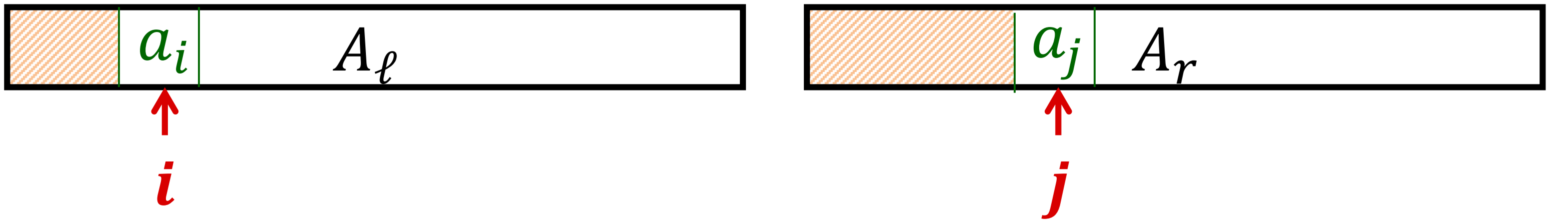
- At each point in time, all inversions involving some element left of i (in A_ℓ) or left of j (in A_r) have been counted
  - and all others still have to be counted...

Guaranteeing Sorted Order:

- While going through the parts, also merge the parts into one sorted order (like in Mergesort).

Combine Step

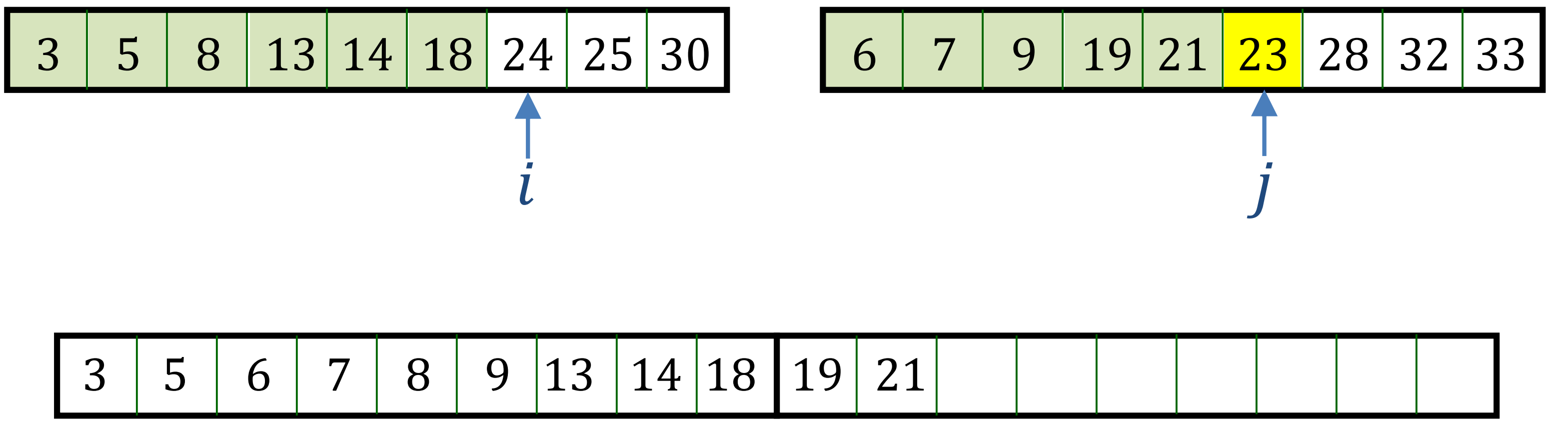
Assume A_ℓ and A_r are sorted



- Pointers i and j, initially pointing to first elements of A_ℓ and A_r
- If $a_i \leq a_j$:
  - a_i is smallest among the remaining elements
  - No inversion of a_i and one of the remaining elements
  - Do not change count
- If $a_j < a_i$:
  - a_j is smallest among the remaining elements
  - a_j is smaller than all remaining elements in A_ℓ
  - Add number of remaining elements in A_ℓ to count
- Increment pointer, pointing to the smaller element

Combine Step: Example

- Assume A_ℓ and A_r are sorted



- **Count:** $0 + 7 + 7 + 6 + 3 + 3 + 3$

Comparing Orders : Summary

- We need sub-sequences in sorted order
- Combine step is like merging in merge sort

- **Idea**: Solve sorting and #inversions at the same time!
  1. Partition A into two equal parts A_ℓ and A_r
  2. Recursively compute #inversions and recursively sort A_ℓ and A_r
  3. Merge A_ℓ and A_r to sorted sequence, at the same time, compute number of inversions between elements a_i in A_ℓ and a_j in A_r

Time for divide and combine: $O(n)$

- Need to go over all $n/2$ indices in A_ℓ and all $n/2$ indices in A_r once.

Number of Inversion: Analysis

Recurrence relation:

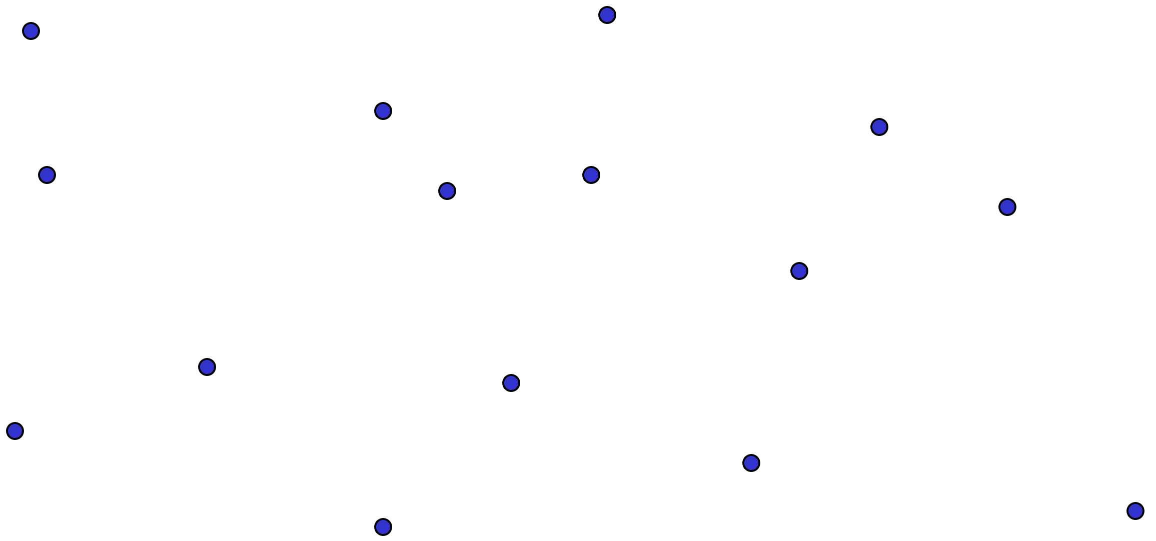
$$T(n) \leq 2 \cdot T(n/2) + c \cdot n, \qquad T(1) \leq c$$

Same recurrence relation as for mergesort:

$$\boldsymbol{T(n) = O(n \cdot \log n)}$$

Geometric divide-and-conquer

Closest Pair Problem: Given a set S of n points, find a pair of points with the smallest distance.



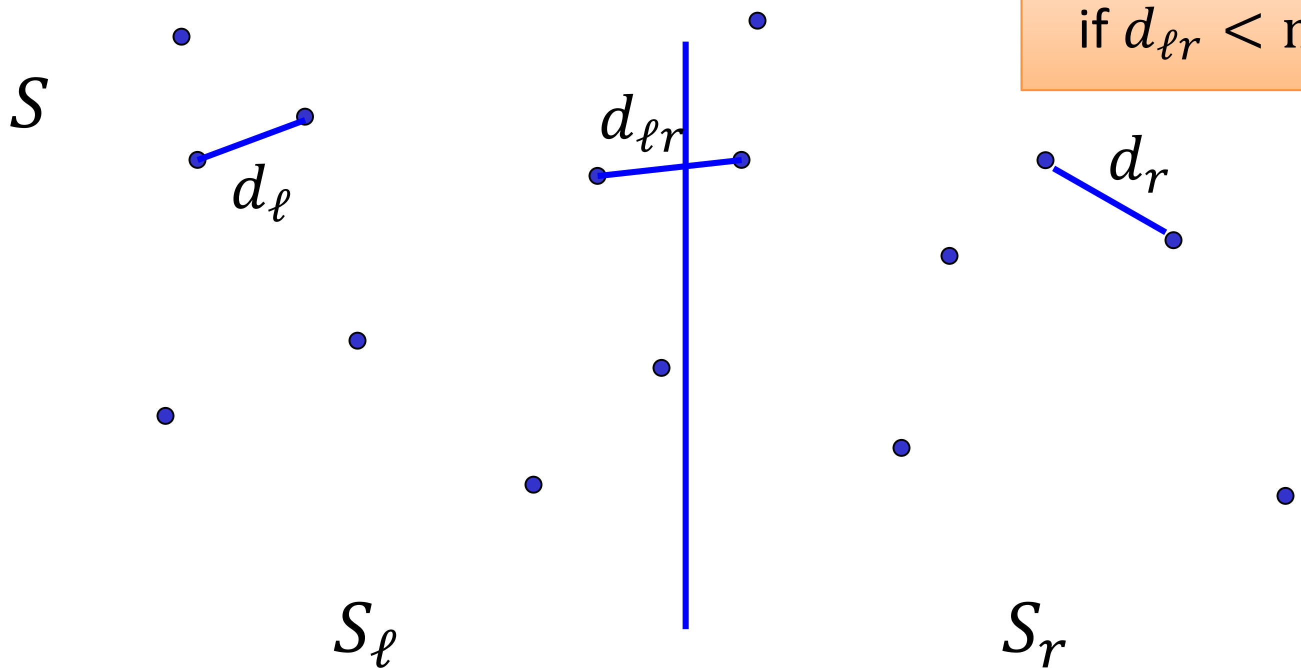
Naïve solution:

- Go over all pairs of points, compute distance, take minimum
- Time: $O(n^2)$

Divide-and-Conquer Solution

0. Sort points by x-coordinate
1. **Divide:** Divide S into two equal sized sets S_ℓ und S_r.
2. **Conquer:** $d_\ell = \text{mindist}(S_\ell)$ $\quad d_r = \text{mindist}(S_r)$
3. **Combine:** $d_{\ell r} = \min\{d(a,b) \mid a \in S_\ell, b \in S_r\}$
   return $\min\{d_\ell, d_r, d_{\ell r}\}$

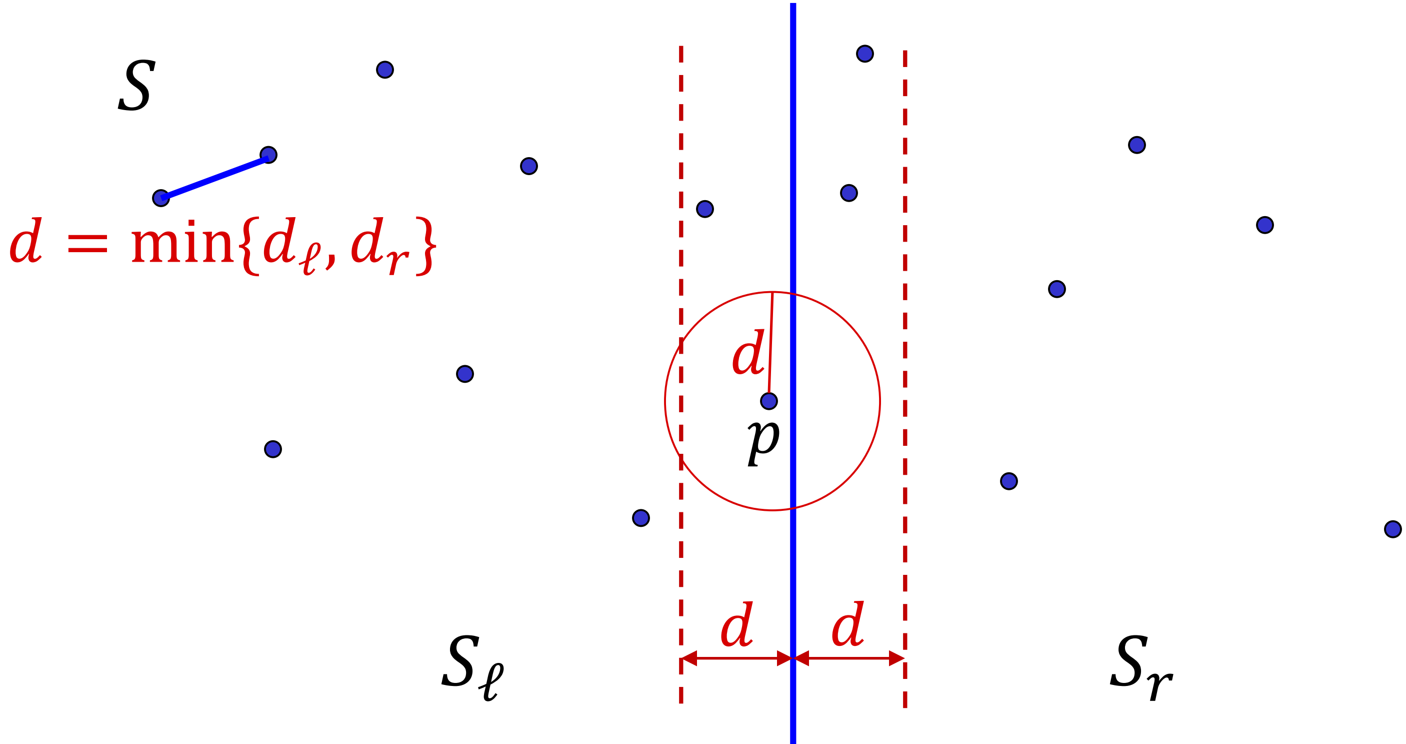
Remark: only need $d_{\ell r}$ if $d_{\ell r} < \min\{d_\ell, d_r\}$



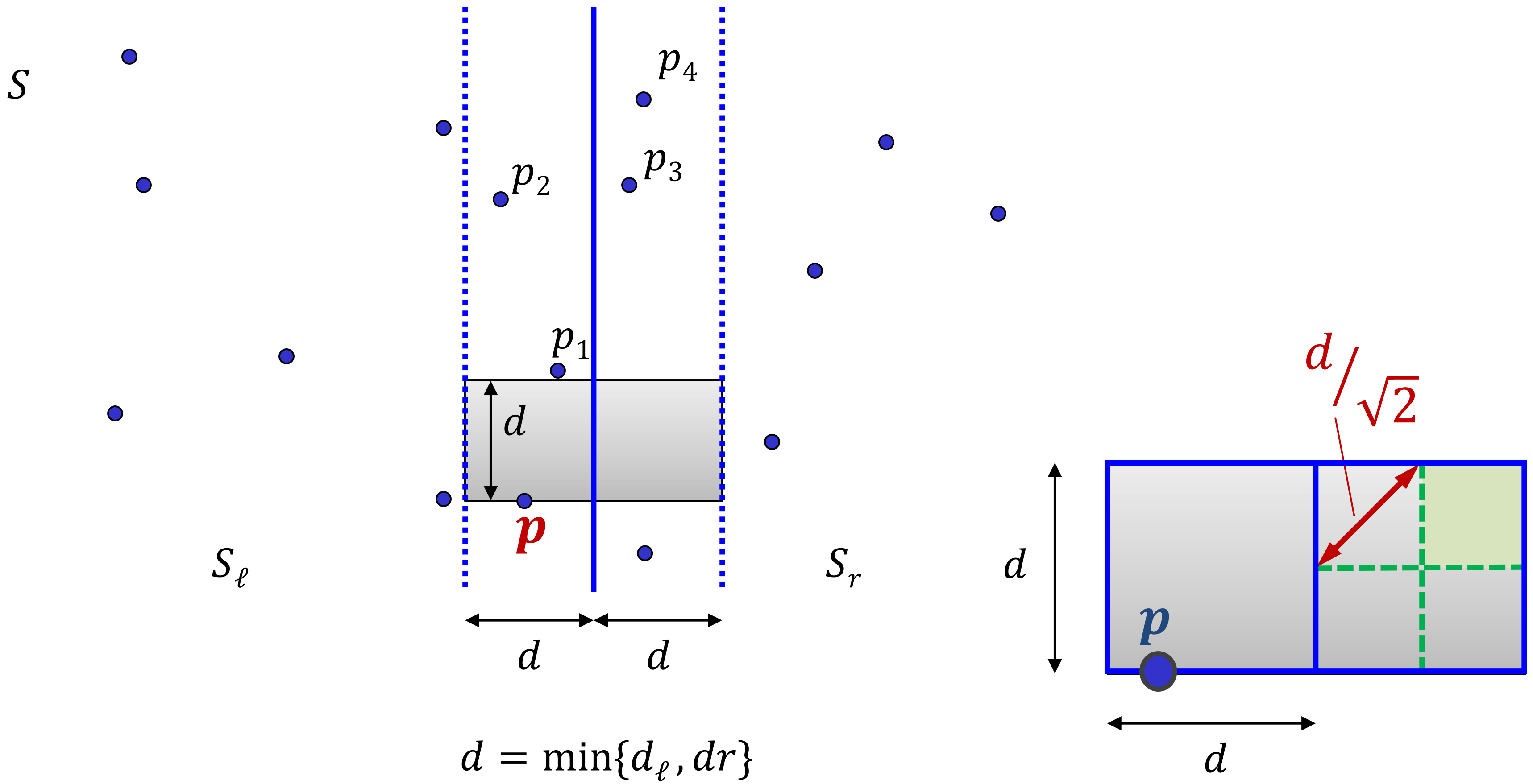
Divide-and-conquer solution

1. **Divide:** Divide S into two equal sized sets S_ℓ und S_r.
2. **Conquer:** $d_\ell = \text{mindist}(S_\ell)$ $d_r = \text{mindist}(S_r)$
3. **Combine:** $d_{\ell r} = \min\{d(a,b) \mid a \in S_\ell, b \in S_r\}$
   return $\min\{d_\ell, d_r, d_{\ell r}\}$

Computation of $d_{\ell r}$ if $d_{\ell r} < \min\{d_\ell, d_r\}$



Combine step



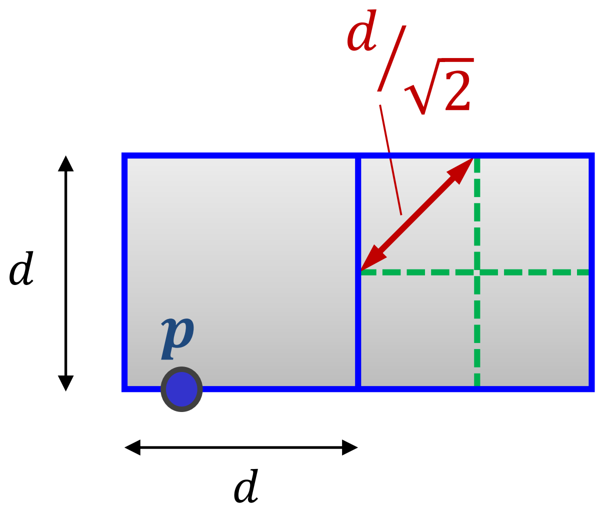
$d = \min\{d_\ell, dr\}$

Combine step

1. Consider only points within distance $\leq d$ of the bisection line, in the order of increasing y-coordinates.

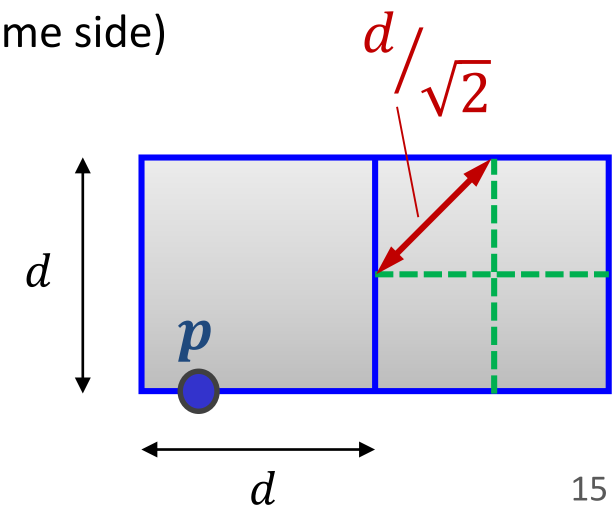
2. For each point p consider all points q on the other side which are within y-distance less than d

3. There are at most 4 such points.



Implementation

- Initially sort the points in S in order of increasing x-coordinates
- While computing closest pair, also sort S according to y-coord.
  - Partition S into S_ℓ and S_r, solve and sort sub-problems recursively
  - Merge to get sorted S according to y-coordinates
  - Center points: points within x-distance $d = \min\{d_\ell, d_r\}$ of center
  - Go through center points in S in order of incr. y-coordinates
    - Each point only has to be compared to 7 next center points in the sorted order of all center points
      (when including the center points on the same side)



Running Time

Recurrence relation:

$$T(n) = 2 \cdot T(n/2) + c \cdot n, \qquad T(1) \leq c$$

Solution:

- Same as for computing number of number of inversions, mergesort (and many others…)

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