Design & Analysis of Algorithms-I

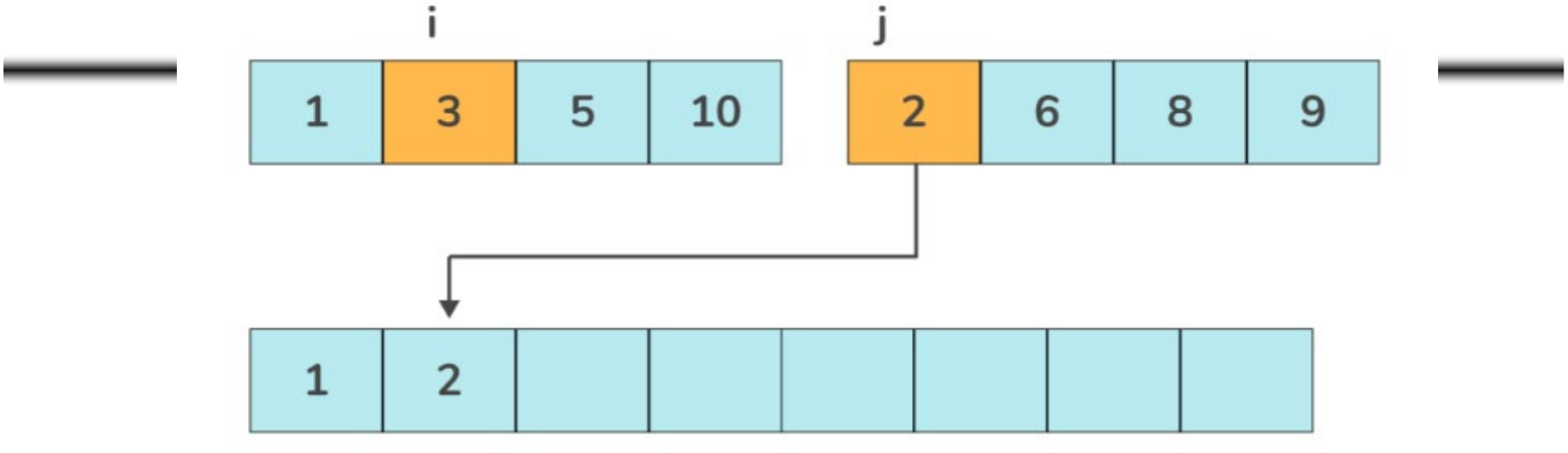
Divide and Conquer Algorithms

Counting Inversions

- The inversion count for any array is the number of steps it will take for the array to be sorted, or how far away any array is from being sorted.
- If we are given an array sorted in reverse order, the inversion count will be the maximum number in that array.
- The inversion count for an array sorted in increasing order will be zero.
- Inversion will be counted if arr[i] is greater than arr[j] where i is lesser than j.

Given an array A of n distinct numbers, an inversion is a pair (i, j) of array indices with $i < j$ and $A[i] > A[j]$. The goal is to count the number of such inversions.

Counting Inversions



Since A[i] > A[j], all elements at index k > i will also be greater than A[j]
Count these all indexes as inversion also, in this case it 3

While iterating through both the halves, if the current element **A[i]** is less than **A[j]**, add it to the sorted list, else increment the count by **mid – i**.

Counting Inversions

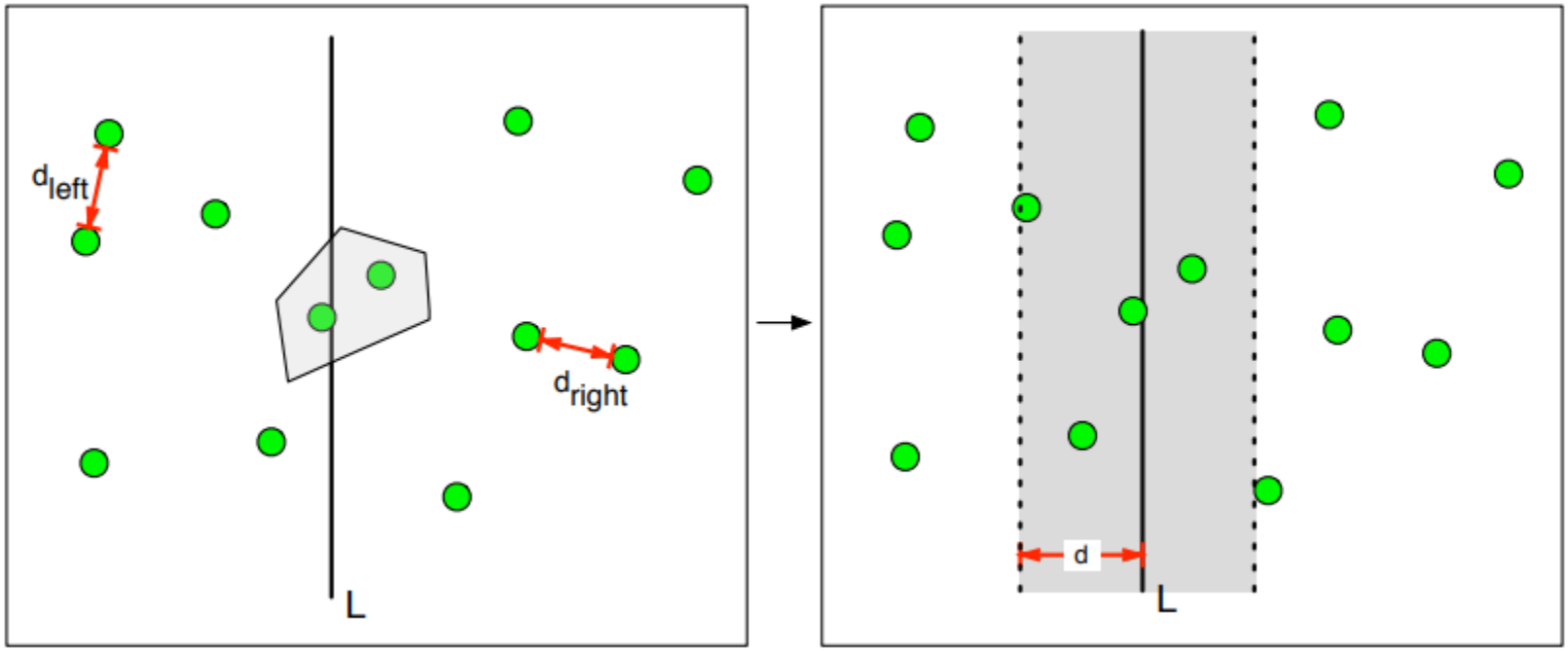
Count Split Inversions:

- Initialize $i = 0$ for A_{left}, $j = 0$ for A_{right}, and $k = 0$ for A.
- Initialize $\text{count} = 0$ to keep track of split inversions.
- While i is less than the length of A_{left} AND j is less than the length of A_{right}:
  - If $A_{\text{left}}[i]$ is less than or equal to $A_{\text{right}}[j]$:
    - Assign $A_{\text{left}}[i]$ to $A[k]$.
    - Increment i by 1.
  - Else:
    - Assign $A_{\text{right}}[j]$ to $A[k]$.
    - Increment the count by the remaining elements in A_{left} (i.e., $\text{length}(A_{\text{left}}) - i$).
    - Increment j by 1.
  - Increment k by 1.
- Copy the remaining elements from A_{left} or A_{right} to A if any are left.

Given an array A of n distinct numbers, an inversion is a pair (i, j) of array indices with $i < j$ and $A[i] > A[j]$. The goal is to count the number of such inversions.

Closest Pair of Points

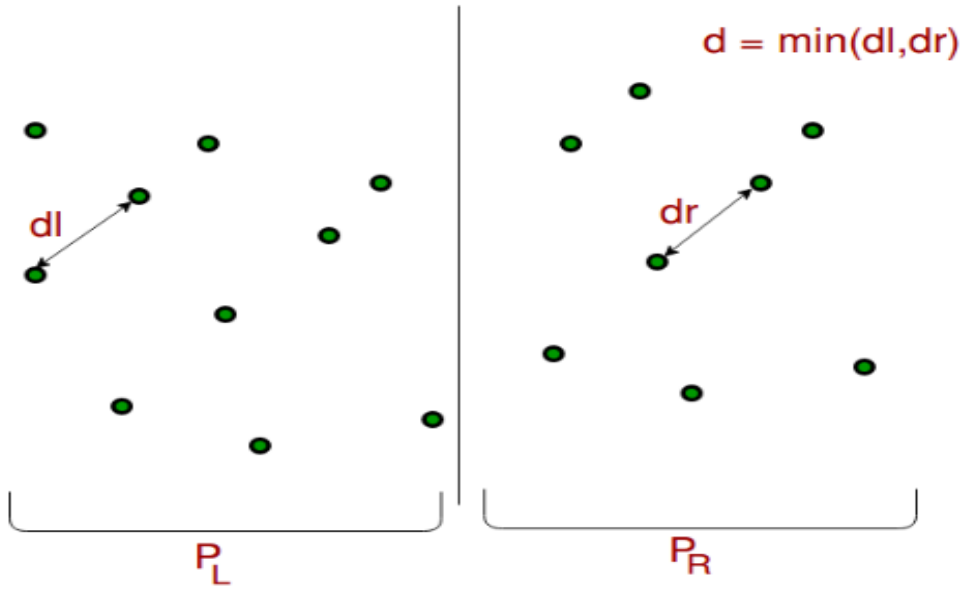
Given n points in the plane, find the pair of points that is the closest together.



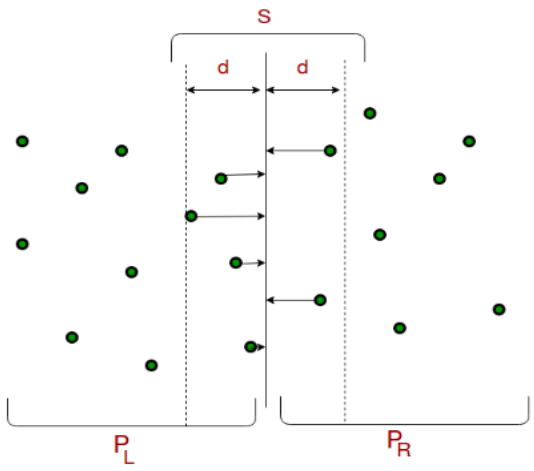
Clearly, we can solve the problem in $O(n^2)$ time, but in fact we can do better. The main idea is to divide the points in half, and recursively find the closest pair of points in each half. The tricky part will be the case when the closest pair of points spans the line that divides the points in half, like the shaded pair below:

Given n points in the plane, find the pair of points that is the closest together.

1. **Determine the Middle Point**:
   - Identify the middle point of the sorted array. This is the point $P[\frac{n}{2}]$.
2. **Divide the Array**:
   - Split the array into two halves:
     - The first half contains points from $P[0]$ to $P[\frac{n}{2}]$.
     - The second half contains points from $P[\frac{n}{2}+1]$ to $P[n-1]$.
3. **Recursive Calculation**:
   - Determine the smallest distance in the first half and label this distance as dl.
   - Determine the smallest distance in the second half and label this distance as dr.
   - Identify the smaller value between dl and dr, and label this as d.



4. **Consider Points Near the Middle**:
   - Visualize a vertical line that passes through the middle point $P[\frac{n}{2}]$.
   - From the entire array, identify points that are within a distance d from this middle line. These points are potential candidates for having a distance smaller than d with some point in the other half of the array.
   - Store all these identified points in a new array named $\text{strip}[]$.



Sort and Scan the Strip:

- First, sort the points in `strip[]` based on their y-coordinates.
- Now, for each point in `strip[]`, compare its distance with the next 6 points in the `strip[]`. The reason we only need to check the next 6 points is due to a geometric observation: If we have a point p and we've drawn a circle of radius d (the current minimum distance) around it, the circle can contain at most 6 points of `strip[]` that can be closer to p.
- If any distance is found smaller than d during this comparison, update d.

Result:

- The smallest distance is d, which is the minimum of distances computed in the left half, right half, and the `strip[]`.

Given n points in the plane, find the pair of points that is the closest together.

ClosestPair $(p_1, p_2, \ldots, p_n)$:

// The points are in sorted order by x.
// We also have the points in a different list ordered by y

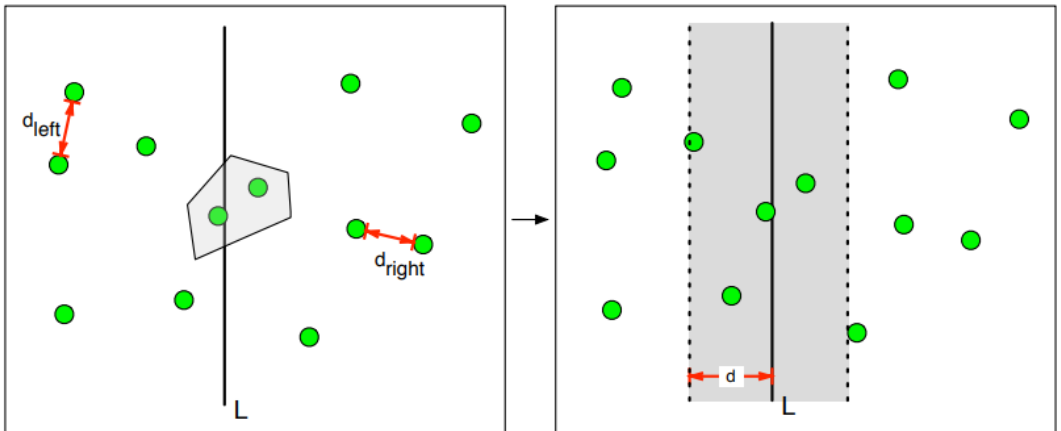
If $n \leq 3$ then solve and return the answer.
Let $m = \lfloor n/2 \rfloor$
Let $\delta = \min(\text{ClosestPair}(p_1, \ldots, p_m), \text{ClosestPair}(p_{m+1}, \ldots, p_n))$

Form a list q of the points (sorted by increasing y) that are within δ of the x coordinate of p_m. Call these points $q_1, q_2 \ldots q_k$

Now we compute d_i, the minimum distance between q_i and all the qs below it, for each $1 \leq i \leq k$ as follows:

$d_i =$ min distance from q_i to $q_{i-1}, q_{i-2}, \ldots, q_j$, where we stop when j gets to 1, or the y coordinate gets too small: $q_j \cdot (0,1) < q_i \cdot (0,1) - \delta$

Return $\min(d_1, \ldots, d_k, \delta)$

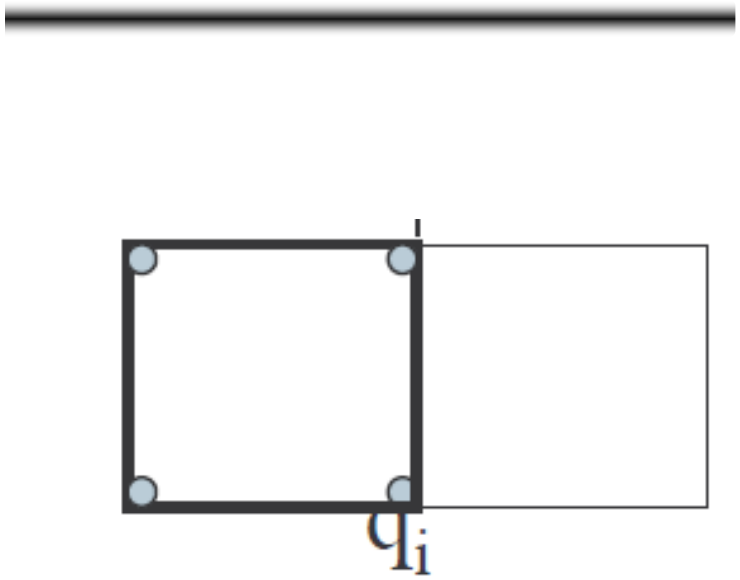


Given n points in the plane, find the pair of points that is the closest together.

Theorem: there are at most 7 q_j's such that $y_i - y_j \leq d$.

Proof:

- Each such q_j must lie either in the left or in the right $d \times d$ square
- Within each square, all points have distance distance $\geq d$ from others
- We can pack at most 4 such points into one square, so we have 8 points total (incl. q_i)



More DC Problems from Syllabus

- finding A^k mod M using DC method
- Finding median (in general k-th smallest element) in a set using DC in expected linear time.