



Lecture 4

Divide and Conquer II: Counting
inversions, counting intersections,
max subarray, maxima set

CS 161 Design and Analysis of Algorithms

Ioannis Panageas

Divide and conquer method (recap)

Steps of method:

- **Divide** input into parts (**smaller problems**)
- **Conquer** (solve) each part recursively
- **Combine** results to obtain solution of original

$$\begin{aligned} T(n) = & \text{divide time} \\ & + T(n_1) + T(n_2) + \dots + T(n_k) \\ & + \text{combine time} \end{aligned}$$

Case study I: Counting inversions

Given numbers A_1, \dots, A_n in an array A , compute the number of **inversions**.

(i, j) is an **inversion**: $A_i > A_j$ and $i < j$.

Example $[18, 29, 12, 15, 32, 10]$ has **9 inversions**:

$(18, 12), (18, 15), (18, 10), (29, 12), (29, 15), (29, 10),$
 $(12, 10), (15, 10), (32, 10)$

Case study I: Counting inversions

- **Solution:** Use Divide and conquer. Tricky part the **combine step**. Run a **modification** of Mergesort that has a **counter** that counts inversions **during merge steps**.
- **Question:** Assume that B_1, \dots, B_k and C_1, \dots, C_l are both sorted. Can you compute the number of inversions of the concatenated sequence $B_1, \dots, B_k, C_1, \dots, C_l$?

Case study I: Counting inversions

- **Question:** Assume that B_1, \dots, B_k and C_1, \dots, C_l are both sorted. Can you compute the number of inversions of the sequence $B_1, \dots, B_k, C_1, \dots, C_l$?

If $B_i > C_j \geq B_{i-1}$ there are

including C_j

$B_1, \dots, B_i, \dots, B_k$

i ↑

$C_1, \dots, C_j, \dots, C_l$

j ↑

Case study I: Counting inversions

- **Question:** Assume that B_1, \dots, B_k and C_1, \dots, C_l are both sorted. Can you compute the number of inversions of the sequence $B_1, \dots, B_k, C_1, \dots, C_l$?

If $B_i > C_j \geq B_{i-1}$ there are

$k - i + 1$ including C_j

$B_1, \dots, B_i, \dots, B_k$

i ↑

$C_1, \dots, C_j, \dots, C_l$

j ↑

Case study I: Counting inversions

If $B_i > C_j \geq B_{i-1}$ there are

$k - i + 1$ including C_j

$B_1, \dots, B_i, \dots, B_k$

i ↑

$C_1, \dots, C_j, \dots, C_l$

j ↑

$B : 1\ 3\ 4\ 9\ 220$

$C : 2\ 3\ 5\ 7\ 8\ 10$

Concatenated: $1\ 3\ 4\ 9\ 220\ 2\ 3\ 5\ 7\ 8\ 10$

↑

↑

8 participates in 2 inversions. $k = 5, i = 4$

Case study I: Counting inversions

Problem: Given two sorted arrays B, C , merge them to a sorted array and count number of inversions **simultaneously**.

Solution: Index i for B , index j for C , index k for A , counter.

While $k < \text{len}(B) + \text{len}(C)$ **do**

If $B[i] \leq C[j]$ **then**

$A[k] \leftarrow B[i]$

$i = i + 1, k = k + 1$

else

$A[k] \leftarrow C[j]$

counter = counter + $\text{len}(B) - i + 1$

$j = j + 1, k = k + 1$

$B : 1\ 3\ 4\ 9\ 22\ 0$

\uparrow
 i

$C : 2\ 3\ 5\ 7\ 8\ 10$

\uparrow
 j

$A :$

counter = 0

Case study I: Counting inversions

Problem: Given two sorted arrays B, C , merge them to a sorted array and count number of inversions **simultaneously**.

Solution: Index i for B , index j for C , index k for A , counter.

While $k < \text{len}(B) + \text{len}(C)$ **do**

If $B[i] \leq C[j]$ **then**

$A[k] \leftarrow B[i]$

$i = i + 1, k = k + 1$

else

$A[k] \leftarrow C[j]$

$\text{counter} = \text{counter} + \text{len}(B) - i + 1$

$j = j + 1, k = k + 1$

$B : 1\ 3\ 4\ 9\ 22\ 0$

i

$C : 2\ 3\ 5\ 7\ 8\ 10$

j

$A : 1$

counter = 0

Case study I: Counting inversions

Problem: Given two sorted arrays B, C , merge them to a sorted array and count number of inversions **simultaneously**.

Solution: Index i for B , index j for C , index k for A , counter.

While $k < \text{len}(B) + \text{len}(C)$ **do**

If $B[i] \leq C[j]$ **then**

$A[k] \leftarrow B[i]$

$i = i + 1, k = k + 1$

else

$A[k] \leftarrow C[j]$

counter = counter + $\text{len}(B) - i + 1$

$j = j + 1, k = k + 1$

$B : 1\ 3\ 4\ 9\ 22\ 0$

i

$C : 2\ 3\ 5\ 7\ 8\ 10$

j

$A : 1\ 2$

counter = 4

Case study I: Counting inversions

Problem: Given two sorted arrays B, C , merge them to a sorted array and count number of inversions **simultaneously**.

Solution: Index i for B , index j for C , index k for A , counter.

While $k < \text{len}(B) + \text{len}(C)$ **do**

If $B[i] \leq C[j]$ **then**

$A[k] \leftarrow B[i]$

$i = i + 1, k = k + 1$

else

$A[k] \leftarrow C[j]$

counter = counter + $\text{len}(B) - i + 1$

$j = j + 1, k = k + 1$

$B : 1\ 3\ 4\ 9\ 22\ 0$

$C : 2\ 3\ 5\ 7\ 8\ 10$

i

j

$A : 1\ 2\ 3$

counter = 4

Case study I: Counting inversions

Problem: Given two sorted arrays B, C , merge them to a sorted array and count number of inversions **simultaneously**.

Solution: Index i for B , index j for C , index k for A , counter.

While $k < \text{len}(B) + \text{len}(C)$ **do**

If $B[i] \leq C[j]$ **then**

$A[k] \leftarrow B[i]$

$i = i + 1, k = k + 1$

else

$A[k] \leftarrow C[j]$

counter = counter + $\text{len}(B) - i + 1$

$j = j + 1, k = k + 1$

$B : 1\ 3\ 4\ 9\ 22\ 0$

$C : 2\ 3\ 5\ 7\ 8\ 10$

i

j

$A : 1\ 2\ 3\ 3$

counter = 7

Case study I: Counting inversions

Problem: Given two sorted arrays B, C , merge them to a sorted array and count number of inversions **simultaneously**.

Solution: Index i for B , index j for C , index k for A , counter.

While $k < \text{len}(B) + \text{len}(C)$ **do**

If $B[i] \leq C[j]$ **then**

$A[k] \leftarrow B[i]$

$i = i + 1, k = k + 1$

else

$A[k] \leftarrow C[j]$

counter = counter + $\text{len}(B) - i + 1$

$j = j + 1, k = k + 1$

$B : 1\ 3\ 4\ 9\ 220$ $C : 2\ 3\ 5\ 7\ 8\ 10$

\uparrow
 i

\uparrow
 j

$A : 1\ 2\ 3\ 3\ 4$

counter = 7

Case study I: Counting inversions

Problem: Given two sorted arrays B, C , merge them to a sorted array and count number of inversions **simultaneously**.

Solution: Index i for B , index j for C , index k for A , counter.

While $k < \text{len}(B) + \text{len}(C)$ **do**

If $B[i] \leq C[j]$ **then**

$A[k] \leftarrow B[i]$

$i = i + 1, k = k + 1$

else

$A[k] \leftarrow C[j]$

$\text{counter} = \text{counter} + \text{len}(B) - i + 1$

$j = j + 1, k = k + 1$

$B : 1\ 3\ 4\ 9\ 220$ $C : 2\ 3\ 5\ 7\ 8\ 10$

\uparrow
 i

\uparrow
 j

$A : 1\ 2\ 3\ 3\ 4\ 5$

counter = 9

Case study I: Counting inversions

Problem: Given two sorted arrays B, C , merge them to a sorted array and count number of inversions **simultaneously**.

Solution: Index i for B , index j for C , index k for A , counter.

While $k < \text{len}(B) + \text{len}(C)$ **do**

If $B[i] \leq C[j]$ **then**

$A[k] \leftarrow B[i]$

$i = i + 1, k = k + 1$

else

$A[k] \leftarrow C[j]$

$\text{counter} = \text{counter} + \text{len}(B) - i + 1$

$j = j + 1, k = k + 1$

$B : 1\ 3\ 4\ 9\ 220$ $C : 2\ 3\ 5\ 7\ 8\ 10$

\uparrow
 i

\uparrow
 j

$A : 1\ 2\ 3\ 3\ 4\ 5\ 7$

counter = 11

Case study I: Counting inversions

Problem: Given two sorted arrays B, C , merge them to a sorted array and count number of inversions **simultaneously**.

Solution: Index i for B , index j for C , index k for A , counter.

While $k < \text{len}(B) + \text{len}(C)$ **do**

If $B[i] \leq C[j]$ **then**

$A[k] \leftarrow B[i]$

$i = i + 1, k = k + 1$

else

$A[k] \leftarrow C[j]$

$\text{counter} = \text{counter} + \text{len}(B) - i + 1$

$j = j + 1, k = k + 1$

$B : 1\ 3\ 4\ 9\ 22\ 0$

\uparrow
 i

$C : 2\ 3\ 5\ 7\ 8\ 10$

\uparrow
 j

$A : 1\ 2\ 3\ 3\ 4\ 5\ 7\ 8$

counter = 13

Case study I: Counting inversions

Problem: Given two sorted arrays B, C , merge them to a sorted array and count number of inversions **simultaneously**.

Solution: Index i for B , index j for C , index k for A , counter.

While $k < \text{len}(B) + \text{len}(C)$ **do**

If $B[i] \leq C[j]$ **then**

$A[k] \leftarrow B[i]$

$i = i + 1, k = k + 1$

else

$A[k] \leftarrow C[j]$

counter = counter + $\text{len}(B) - i + 1$

$j = j + 1, k = k + 1$

$B : 1\ 3\ 4\ 9\ 2\ 2\ 0$ $C : 2\ 3\ 5\ 7\ 8\ 10$

\uparrow
 i

\uparrow
 j

$A : 1\ 2\ 3\ 3\ 4\ 5\ 7\ 8\ 9$

counter = 13

Case study I: Counting inversions

Problem: Given two sorted arrays B, C , merge them to a sorted array and count number of inversions **simultaneously**.

Solution: Index i for B , index j for C , index k for A , counter.

While $k < \text{len}(B) + \text{len}(C)$ **do**

If $B[i] \leq C[j]$ **then**

$A[k] \leftarrow B[i]$

$i = i + 1, k = k + 1$

else

$A[k] \leftarrow C[j]$

$\text{counter} = \text{counter} + \text{len}(B) - i + 1$

$j = j + 1, k = k + 1$

$B : 1\ 3\ 4\ 9\ 2\ 2\ 0$ $C : 2\ 3\ 5\ 7\ 8\ 10$

\uparrow
 i

\uparrow
 j

$A : 1\ 2\ 3\ 3\ 4\ 5\ 7\ 8\ 9\ 10$

counter = 14

Case study I: Counting inversions

Problem: Given two sorted arrays B, C , merge them to a sorted array and count number of inversions **simultaneously**.

Solution: Index i for B , index j for C , index k for A , counter.

While $k < \text{len}(B) + \text{len}(C)$ **do**

If $B[i] \leq C[j]$ **then**

$A[k] \leftarrow B[i]$

$i = i + 1, k = k + 1$

else

$A[k] \leftarrow C[j]$

$\text{counter} = \text{counter} + \text{len}(B) - i + 1$

$j = j + 1, k = k + 1$

$B : 1\ 3\ 4\ 9\ 220$ $C : 2\ 3\ 5\ 7\ 8\ 10$

$i \uparrow$

$j \uparrow$

$A : 1\ 2\ 3\ 3\ 4\ 5\ 7\ 8\ 9\ 10\ 220$

counter = 14

Case study I: Counting inversions

Pseudocode:

ModifiedMergesort($A[1 : n]$)

If $n == 1$ **then**

return $A, 0$

$B, \text{countL} = \text{ModifiedMergesort} (A[1 : \frac{n}{2}])$

$C, \text{countR} = \text{ModifiedMergesort} (A[\frac{n}{2} + 1 : n])$

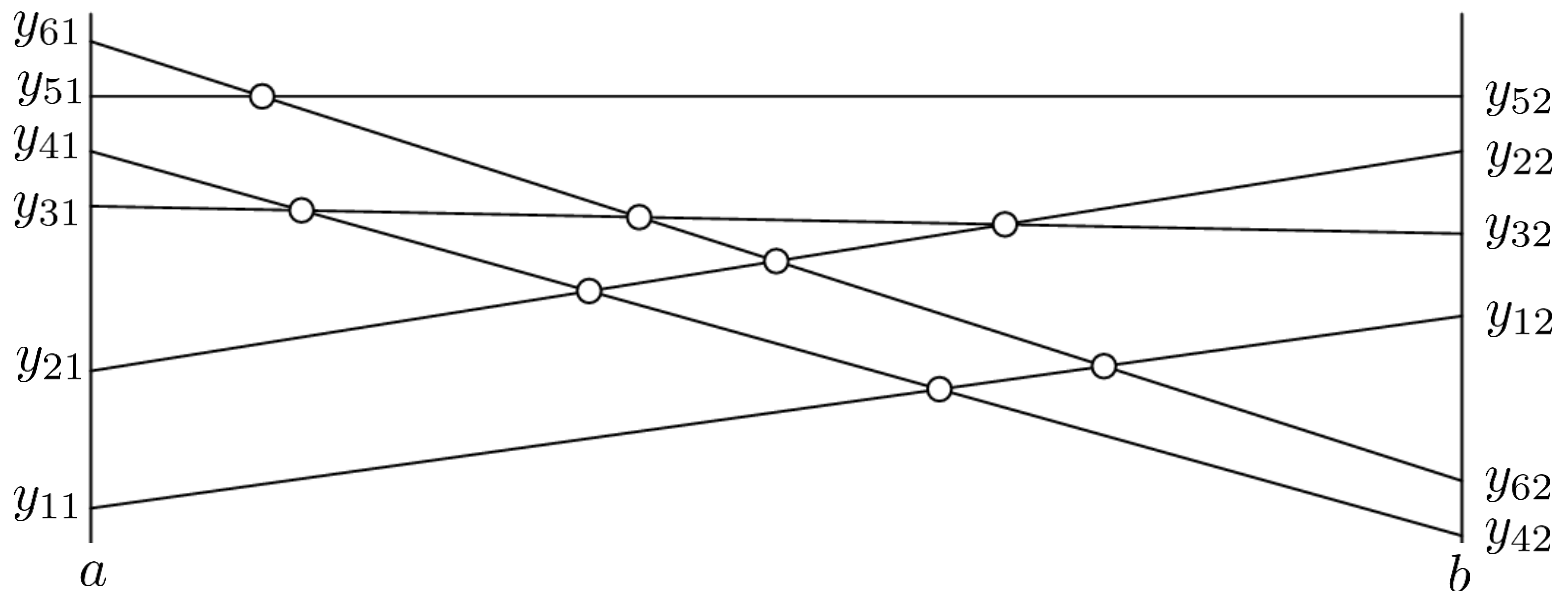
$A, \text{counterM} \leftarrow \text{ModifiedMerge}(B, C)$

return $A, \text{countL} + \text{countR} + \text{counterM}$

Case study II: Counting intersections

Problem: Given n distinct lines in the plane, none of which are vertical and two vertical lines $x = a$ and $x = b$, find the number of intersections. We assume that each line i is described by its endpoints (a, y_{i1}) and (b, y_{i2}) .

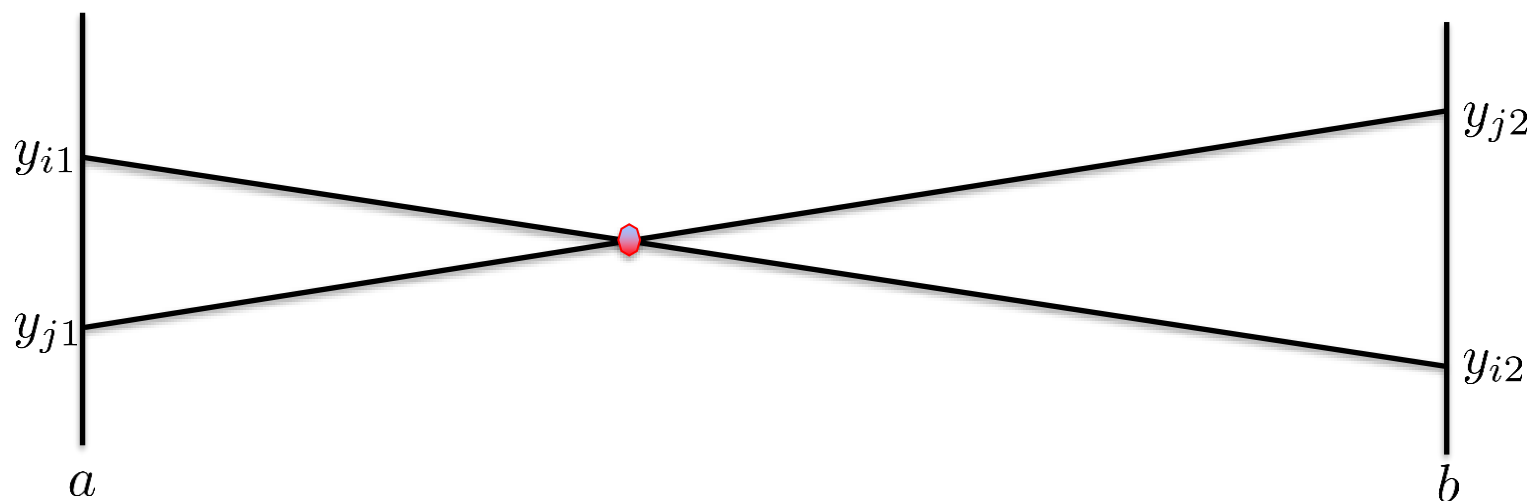
Example: 6 lines (8 intersections)



Case study II: Counting intersections

Problem: Given n distinct lines in the plane, none of which are vertical and two vertical lines $x = a$ and $x = b$, find the number of intersections. We assume that each line i is described by its endpoints (a, y_{i1}) and (b, y_{i2}) .

Question: When two lines i, j intersect?

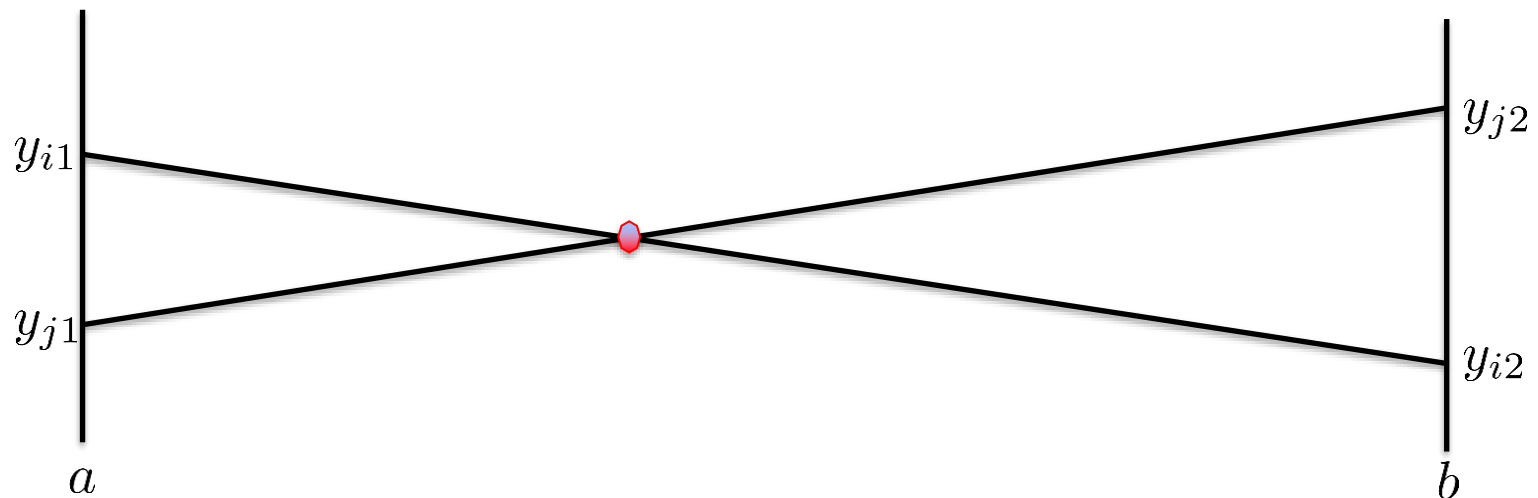


Case study II: Counting intersections

Problem: Given n distinct lines in the plane, none of which are vertical and two vertical lines $x = a$ and $x = b$, find the number of intersections. We assume that each line i is described by its endpoints (a, y_{i1}) and (b, y_{i2}) .

Question: When two lines i, j intersect?

If $y_{i1} > y_{j1}$ then $y_{i2} < y_{j2}$ or If $y_{i1} < y_{j1}$ then $y_{i2} > y_{j2}$



Case study II: Counting intersections

For all pairs i, j with $i < j$, count number of intersections

Pseudocode:

counter $\leftarrow 0$

For $i = 1$ to n **do**

For $j = i + 1$ to n **do**

If $(y_{i1} > y_{j1}$ and $y_{i2} < y_{j2})$ or $(y_{i1} < y_{j1}$ and $y_{i2} > y_{j2})$ **then**

 counter \leftarrow counter + 1

return counter

Case study II: Counting intersections

For all pairs i, j with $i < j$, count number of intersections

Pseudocode:

counter $\leftarrow 0$

For $i = 1$ to n **do**

For $j = i + 1$ to n **do**

If $(y_{i1} > y_{j1}$ and $y_{i2} < y_{j2})$ or $(y_{i1} < y_{j1}$ and $y_{i2} > y_{j2})$ **then**

 counter \leftarrow counter + 1

return counter

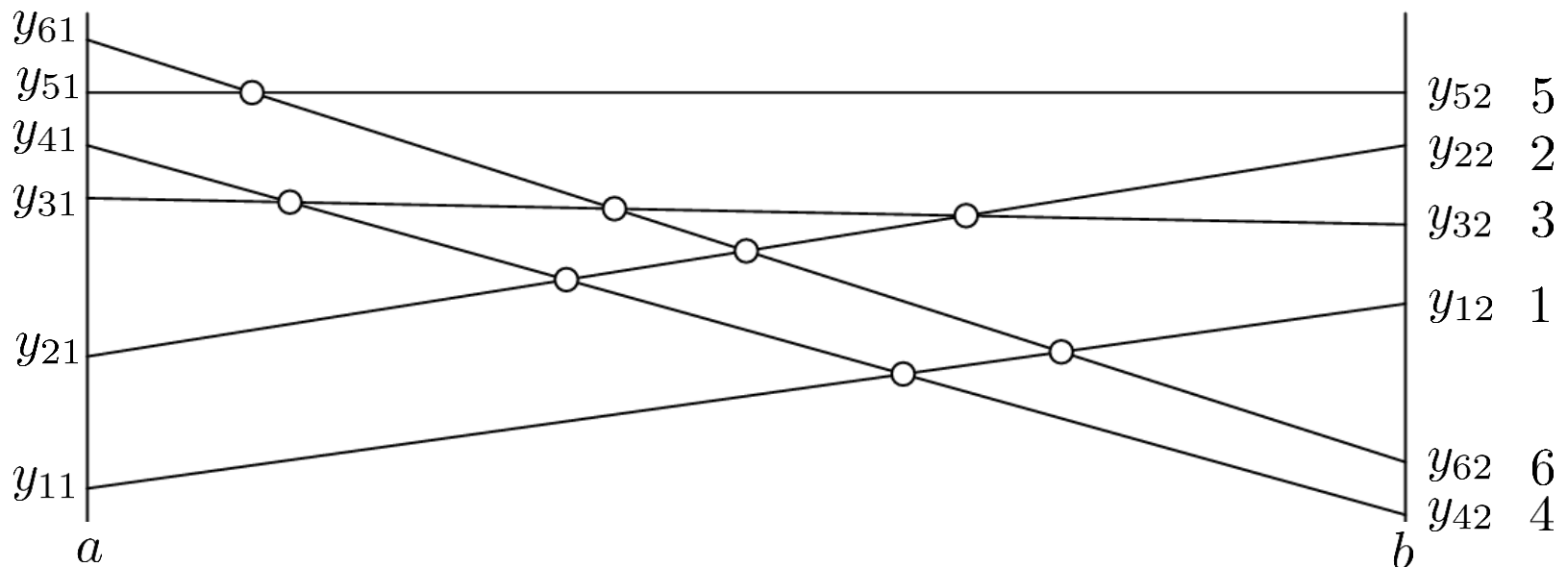
Running time $\Theta(n^2)$

Can we do better?

Case study II: Counting intersections

Idea: Let's sort the lines with respect to y on a . Check the inverse permutation of the indices of the lines on b .

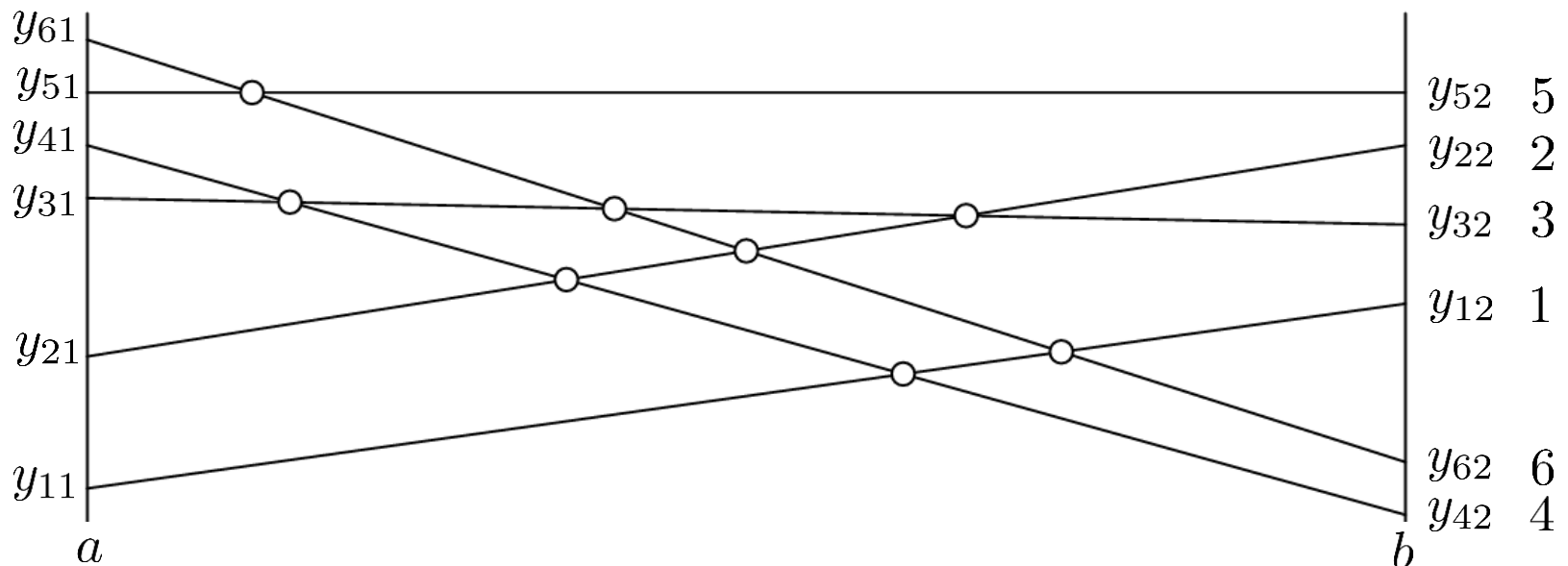
Example: 4, 6, 1, 3, 2, 5



Case study II: Counting intersections

Idea: Let's sort the lines with respect to y on a . Check the inverse permutation of the indices of the lines on b .

Example: 4, 6, 1, 3, 2, 5

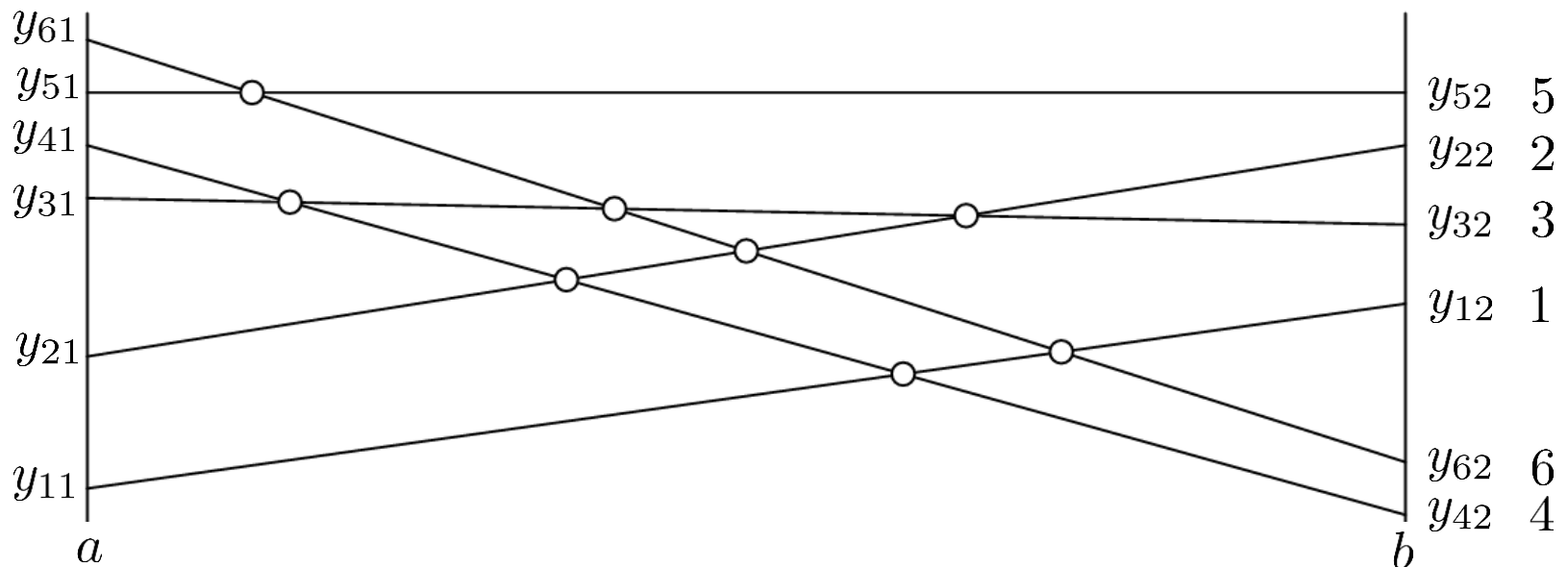


Key observation: Number of inversions is equal to number of intersections. In example (4, 1), (4,3), (4,2), (6,1), (6,3), (6,2), (6,5), (3,2)

Case study II: Counting intersections

Idea: Let's sort the lines with respect to y on a . Check the inverse permutation of the indices of the lines on b .

Example: 4, 6, 1, 3, 2, 5



Solution: Sort the lines with respect to y on a . Run modified mergesort to find number of inversions. Running time $\Theta(n \log n)$.

Case study III: Maximum subarray

Problem (Leetcode, question in interviews): Given an array A of n numbers (positive and negative), find the subarray with the maximum sum.

Example: $A = [-2, -5, 6, -2, -3, 1, 5, -6]$

Case study III: Maximum subarray

Problem (Leetcode, question in interviews): Given an array A of n numbers (positive and negative), find the subarray with the maximum sum.

Example: $A = [-2, -5, 6, -2, -3, 1, 5, -6]$

Solution of example:

$[-2, -5, 6, -2, -3, 1, 5, -6]$ with sum 7.

Case study III: Maximum subarray

For all i, j with $i \leq j$, compute $A_i + A_{i+1} + \cdots + A_j$. Keep the maximum from all sums. Total number of computations is...

Case study III: Maximum subarray

For all i, j with $i \leq j$, compute $A_i + A_{i+1} + \dots + A_j$. Keep the maximum from all sums. Total number of computations is...

Pseudocode:

```
max  $\leftarrow$  0
For  $i = 1$  to  $n$  do
    For  $j = i$  to  $n$  do
        sum  $\leftarrow$  0
        For  $k = i$  to  $j$  do
            sum = sum +  $A[k]$ 
        If sum > max then
            max  $\leftarrow$  sum
return max
```


Case study III: Maximum subarray

For all i, j with $i < j$, compute $A_i + A_{i+1} + \dots + A_j$. Keep the maximum from all sums. Total number of computations is

$$\sum_{i=1}^n \sum_{j=i}^n (j - i + 1) \text{ which is } \Theta(n^3)$$

Pseudocode:

```
max  $\leftarrow$  0
For  $i = 1$  to  $n$  do
    For  $j = i$  to  $n$  do
        sum  $\leftarrow$  0
        For  $k = i$  to  $j$  do
            sum = sum +  $A[k]$ 
        If sum > max then
            max  $\leftarrow$  sum
return max
```

Can we do better?

Case study III: Maximum subarray

Idea 1: Do first preprocessing. Compute partial sums

$S_i = A_1 + \cdots + A_i$ for every i . Running time $\Theta(n)$.

Observe that $A_i + A_{i+1} + \cdots + A_j = S_j - S_{i-1}$

Case study III: Maximum subarray

Idea 1: Do first preprocessing. Compute partial sums

$S_i = A_1 + \cdots + A_i$ for every i . Running time $\Theta(n)$.

Observe that $A_i + A_{i+1} + \cdots + A_j = S_j - S_{i-1}$

Then for all i, j with $i \leq j$, compute the maximum among $S_j - S_{i-1}$.

Case study III: Maximum subarray

Idea 1: Do first preprocessing. Compute partial sums

$S_i = A_1 + \dots + A_i$ for every i . Running time $\Theta(n)$.

Observe that $A_i + A_{i+1} + \dots + A_j = S_j - S_{i-1}$

Then for all i, j with $i \leq j$, compute the maximum among $S_j - S_{i-1}$.

$\text{max} \leftarrow 0$

$S[0] \leftarrow 0$

For $i = 1$ to n **do**

$S[i] \leftarrow S[i - 1] + A[i]$

For $i = 1$ to n **do**

For $j = i$ to n **do**

If $S[j] - S[i - 1] > \text{max}$ **then**

$\text{max} \leftarrow S[j] - S[i - 1]$

return max

$[-2, -5, 6, -2, -3, 1, 5, -6]$

$S = [0, -2, -7, -1, -3, -6, -5, 0, -6]$

$-5+6-2-3+1 = S[6] - S[1] = -3$

Case study III: Maximum subarray

Idea 1: Do first preprocessing. Compute partial sums

$S_i = A_1 + \dots + A_i$ for every i . Running time $\Theta(n)$.

Observe that $A_i + A_{i+1} + \dots + A_j = S_j - S_{i-1}$

Then for all i, j with $i < j$, compute the maximum among $S_j - S_{i-1}$.

$\text{max} \leftarrow 0$

$S[0] \leftarrow 0$

For $i = 1$ to n **do**

$S[i] \leftarrow S[i - 1] + A[i]$

For $i = 1$ to n **do**

For $j = i$ to n **do**

If $S[j] - S[i - 1] > \text{max}$ **then**

$\text{max} \leftarrow S[j] - S[i - 1]$

return max

Running time $\Theta(n^2)$

Can we do better?

Case study III: Maximum subarray

Idea 2: Divide and conquer

$[-2, -5, 6, -2, -3, 1, 5, -6]$

Find max in left half (e.g., green), find max in right half (e.g., black) and **combine/merge**. HOW?

Observe left part has maximum 6 and right part also 6. The solution is 7 though.

Case study III: Maximum subarray

Idea 2: Divide and conquer

$[-2, -5, 6, -2, -3, 1, 5, -6]$

Find max in left half (e.g., green), find max in right half (e.g., black) and **combine/merge**. HOW?

Observe left part has maximum 6 and right part also 6. The solution is 7 though.

Key idea: The solution is either on **left** part, or **right** part or **crosses the midpoint** (has at least one number in both parts).

Case study III: Maximum subarray

Idea 2: Divide and conquer

$[-2, -5, 6, -2, -3, 1, 5, -6]$

Question: How to get the maximum subarray that crosses the midpoint?

Case study III: Maximum subarray

Idea 2: Divide and conquer

$[-2, -5, 6, -2, -3, 1, 5, -6]$

Question: How to get the maximum subarray that crosses the midpoint?

- a) Find the maximum starting from **mid** and **going left**.
 - b) Find the maximum starting from **mid+1** and **going right**.
- Add them up. This can happen in $\Theta(n)$ time using partial sums **S**.
In example above a) is 4 and b) is 3 for a total of 7.

Case study III: Maximum subarray

Pseudocode:

$S[0] \leftarrow 0$ **For** $i = 1$ to n **do**
 $S[i] \leftarrow S[i - 1] + A[i]$

Maxsum($A[1 : n]$)

If $n == 1$ **return** $\max(A[1], 0)$

$\text{maxL} \leftarrow \text{Maxsum}(A[1 : n/2])$

$\text{maxR} \leftarrow \text{Maxsum}(A[n/2 + 1 : n])$

$\text{max1} \leftarrow 0$

$\text{max2} \leftarrow 0$

For $i = n/2$ downto 1 **do**

If $\text{max1} < S[n/2] - S[i - 1]$ **then** $\text{max1} \leftarrow S[n/2] - S[i - 1]$

For $i = n/2 + 1$ to n **do**

If $\text{max2} < S[i] - S[n/2]$ **then** $\text{max2} \leftarrow S[i] - S[n/2]$

return maximum of maxL , maxR and $\text{max1} + \text{max2}$

Case study III: Maximum subarray

Pseudocode:

$S[0] \leftarrow 0$ **For** $i = 1$ to n **do**
 $S[i] \leftarrow S[i - 1] + A[i]$

Maxsum($A[1 : n]$)

$[-2, -5, 6, -2, -3, 1, 5, -6]$

$\text{maxL} = 6, \text{maxR} = 6, \text{max1} + \text{max2} = 4 + 3 = 7$

If $n == 1$ **return** $\max(A[1], 0)$

$\text{maxL} \leftarrow \text{Maxsum}(A[1 : n/2])$

$\text{maxR} \leftarrow \text{Maxsum}(A[n/2 + 1 : n])$

$\text{max1} \leftarrow 0$

$\text{max2} \leftarrow 0$

For $i = n/2$ downto 1 **do**

If $\text{max1} < S[n/2] - S[i - 1]$ **then** $\text{max1} \leftarrow S[n/2] - S[i - 1]$

For $i = n/2 + 1$ to n **do**

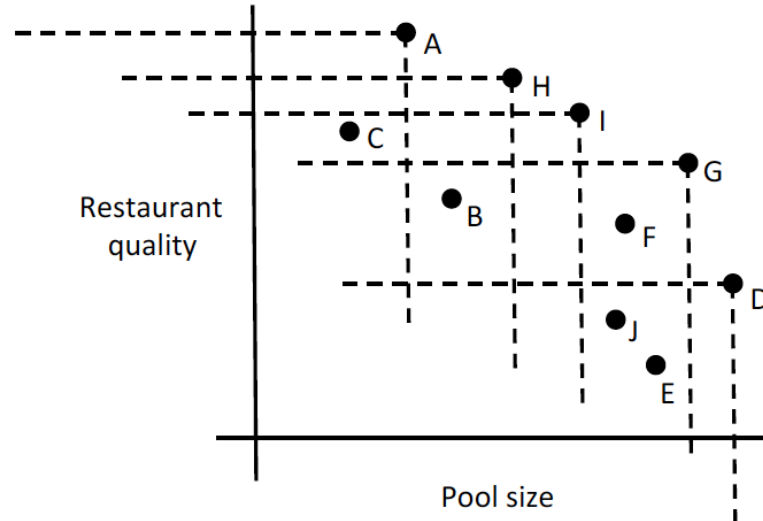
If $\text{max2} < S[i] - S[n/2]$ **then** $\text{max2} \leftarrow S[i] - S[n/2]$

return maximum of maxL, maxR and $\text{max1} + \text{max2}$

Case study IV: Maxima Set

Problem: We are given n points $(x_1, y_1), \dots, (x_n, y_n)$ on the plane. A point (x_i, y_i) is called a **maximum point** if there is no other point (x_j, y_j) that $x_i \leq x_j$ and $y_i \leq y_j$.

Example: x captures pool size and y restaurant quality. 10 hotels



Case study IV: Maxima Set

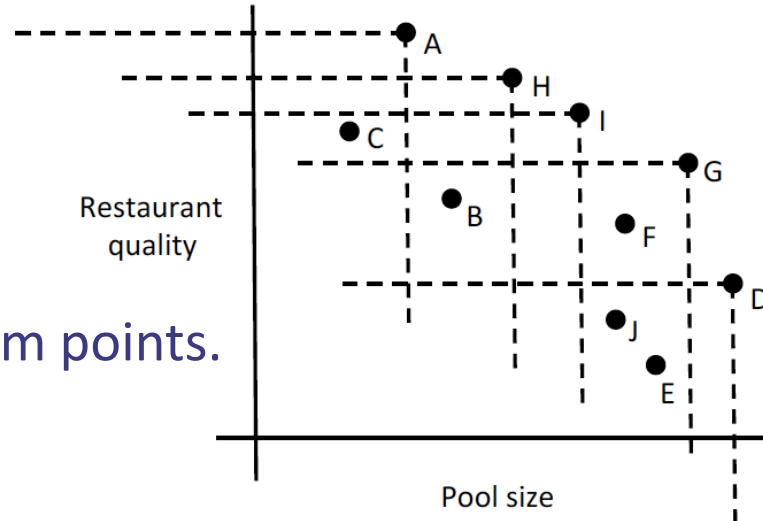
Problem: We are given n points $(x_1, y_1), \dots, (x_n, y_n)$ on the plane. A point (x_i, y_i) is called a **maximum point** if there is no other point (x_j, y_j) that $x_i \leq x_j$ and $y_i \leq y_j$.

Example: x captures pool size and y restaurant quality. 10 hotels

Explanation:

A, H, G, D are maximum points.

C, B, F, J, E are not.



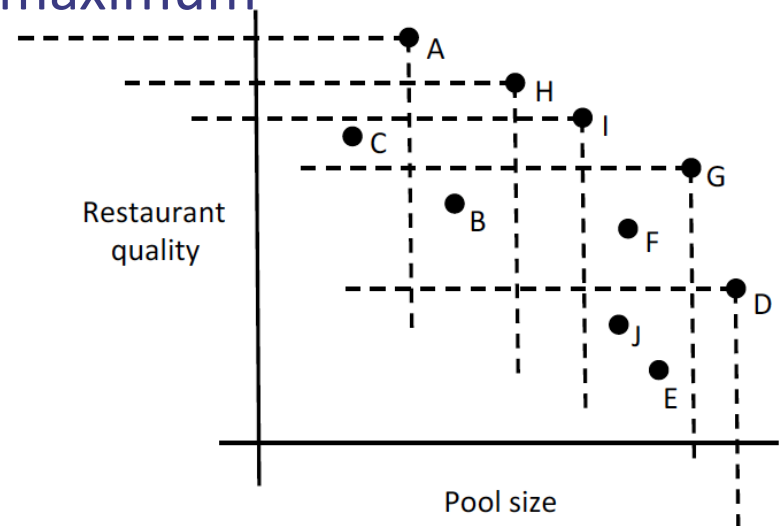
Case study IV: Maxima Set

Problem: We are given n points $(x_1, y_1), \dots, (x_n, y_n)$ on the plane. A point (x_i, y_i) is called a **maximum point** if there is no other point (x_j, y_j) that $x_i \leq x_j$ and $y_i \leq y_j$.

Obvious approach:

For every point (x_i, y_i) , check if it is maximum

To check if it is maximum, you check the condition with all other points.



Case study IV: Maxima Set

Problem: We are given n points $(x_1, y_1), \dots, (x_n, y_n)$ on the plane. A point (x_i, y_i) is called a **maximum point** if there is no other point (x_j, y_j) that $x_i \leq x_j$ and $y_i \leq y_j$.

Pseudocode:

counter $\leftarrow 0$

For $i = 1$ to n **do**

flag $\leftarrow 1$

For $j = i + 1$ to n **do**

If $(x_j > x_i$ and $y_j > y_i)$ **then** flag $\leftarrow 0$

counter \leftarrow counter + flag

return counter

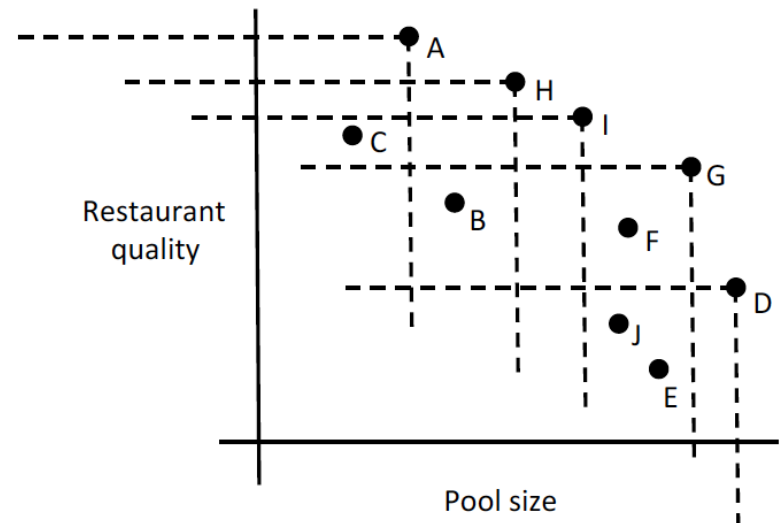
Running time $\Theta(n^2)$

Can we do better?

Case study IV: Maxima Set

Problem: We are given n points $(x_1, y_1), \dots, (x_n, y_n)$ on the plane. A point (x_i, y_i) is called a **maximum point** if there is no other point (x_j, y_j) that $x_i \leq x_j$ and $y_i \leq y_j$.

Idea: Divide and conquer. **Divide** step and **Combine** step is challenging.



Case study IV: Maxima Set

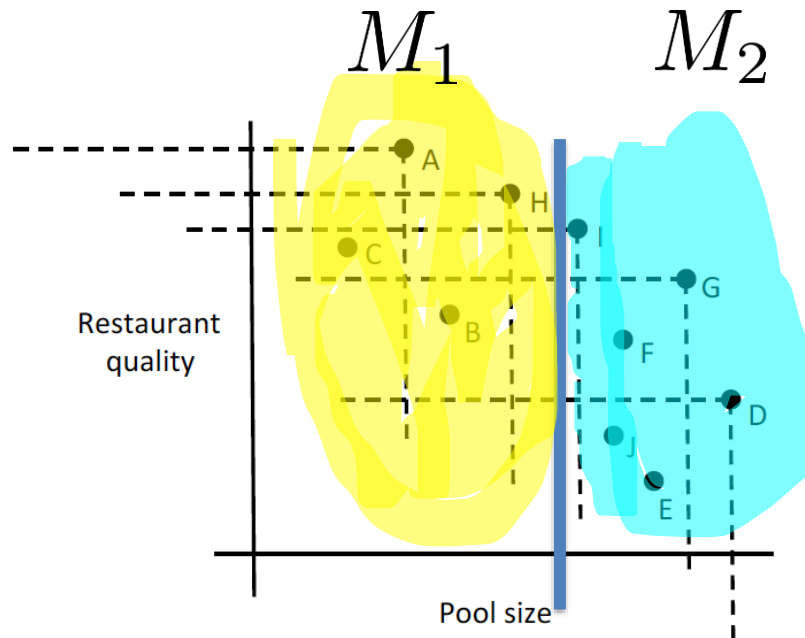
Divide step: It should split the points in two parts of equal size.

How?

Case study IV: Maxima Set

Divide step: It should split the points in two parts of equal size.

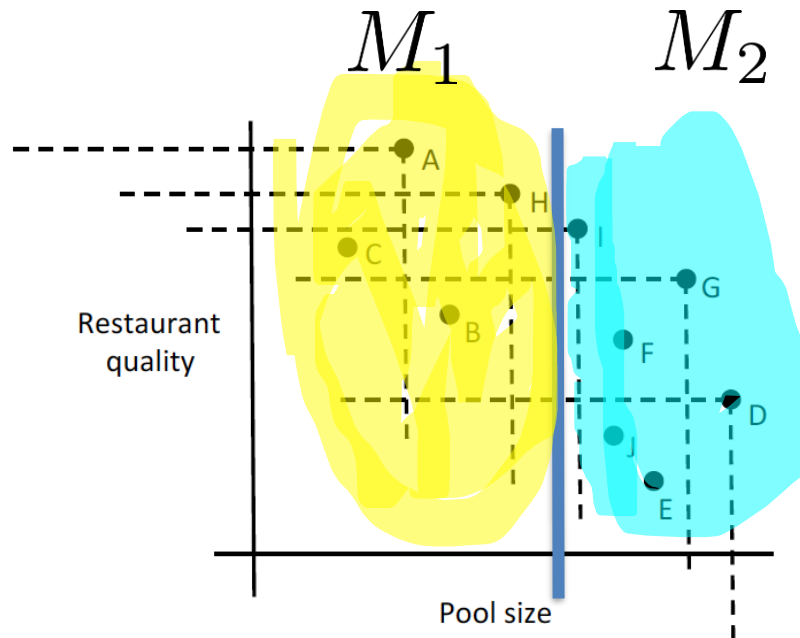
How? Choose the middle (median) point with respect to x coordinates.



Case study IV: Maxima Set

Divide step: It should split the points in two parts of equal size.

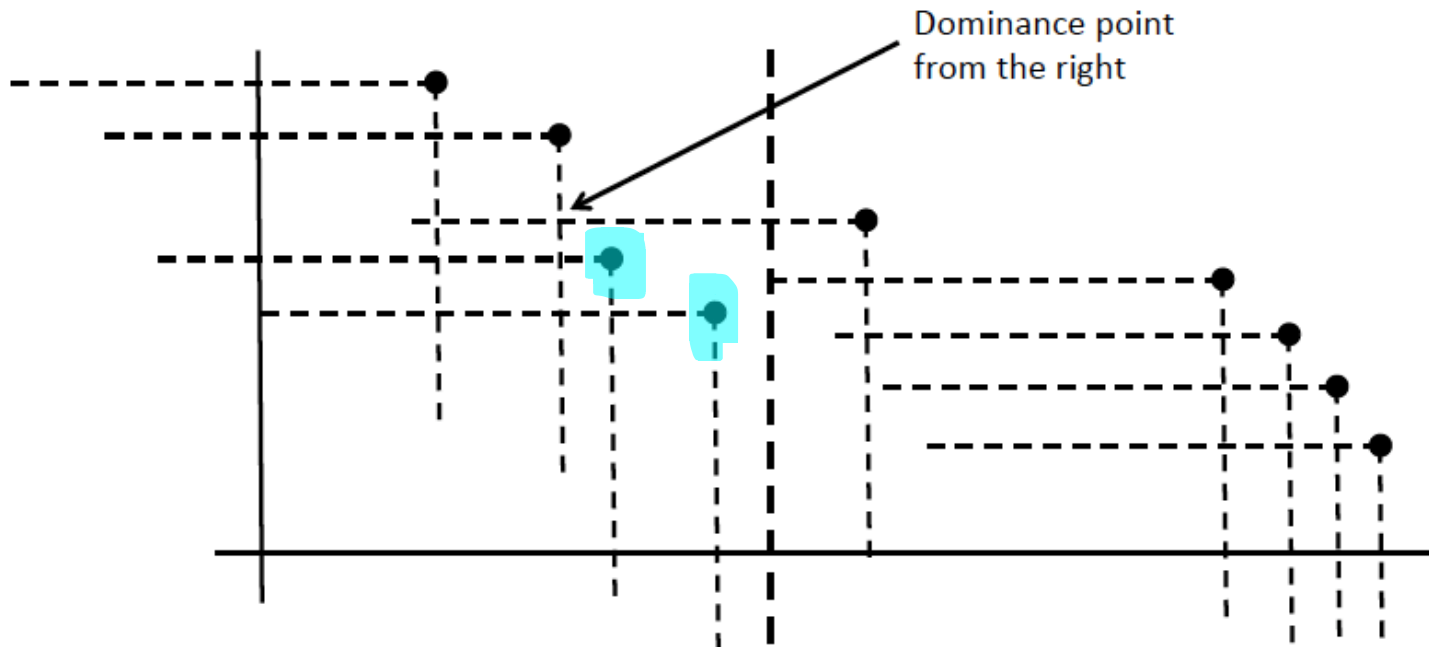
How? Choose the middle (median) point with respect to x coordinates.



Combine step: Return $M_1 \cup M_2$?

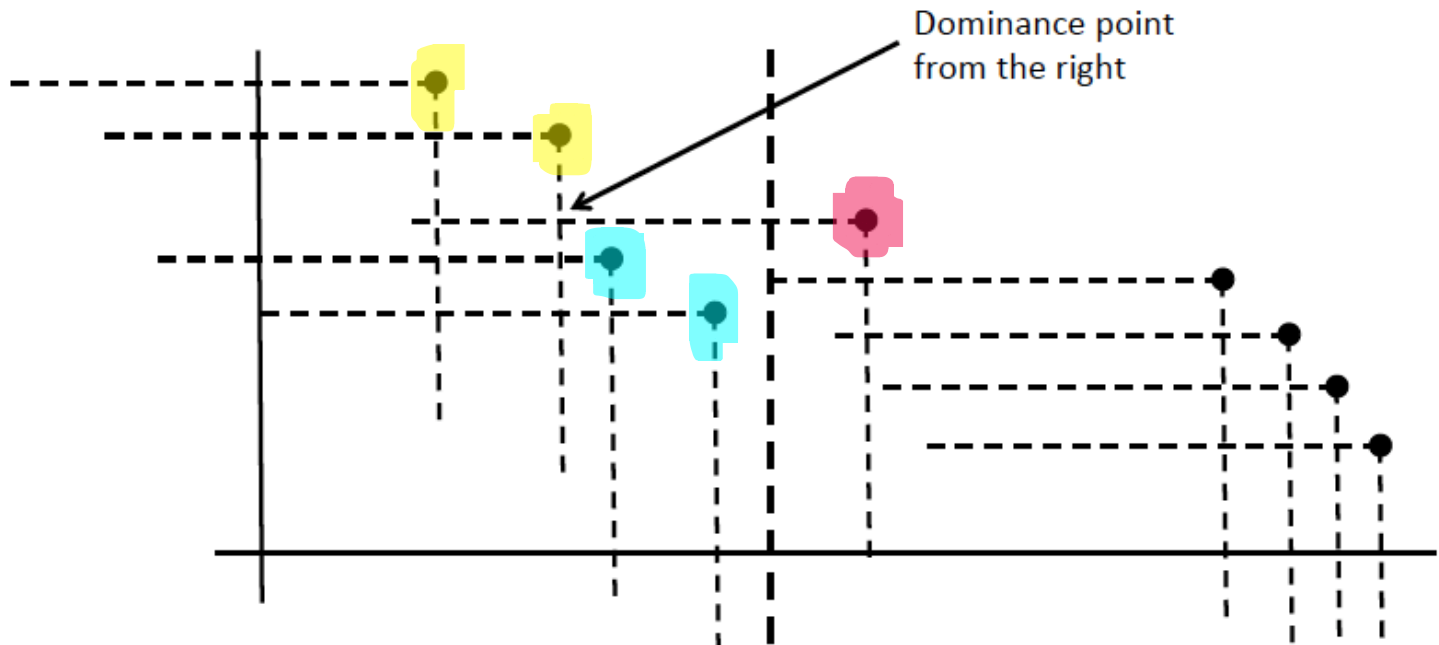
Case study IV: Maxima Set

Combine step: Return $M_1 \cup M_2$? **Wrong:** blue points below of M_1 are not part of the solution



Case study IV: Maxima Set

Combine step idea: M_2 points should part of the solution. From M_1 , the points that are maximum should not be dominated by smallest with respect to x coordinates in M_2



Case study IV: Maxima Set

Pseudocode:

MaximaSet(S, n):

if $n = 1$ **then**

return S

 Let p be the median point in S , by x -coordinates

 Let L be the set of points less than p in S by x -coordinates

 Let G be the set of points greater than or equal to p in S by x -coordinates

$M_1 \leftarrow \text{MaximaSet}(L)$

$M_2 \leftarrow \text{MaximaSet}(G)$

 Let q be the smallest point in M_2

for each point, r , in M_1 **do** by x -coordinates

if $x(r) \leq x(q)$ **and** $y(r) \leq y(q)$ **then**

 Remove r from M_1

return $M_1 \cup M_2$

Case study IV: Maxima Set

Pseudocode:

MaximaSet(S, n):

if $n = 1$ **then**

return S

 Let p be the median point in S , by x -coordinates

 Let L be the set of points less than p in S by x -coordinates

 Let G be the set of points greater than or equal to p in S by x -coordinates

$M_1 \leftarrow \text{MaximaSet}(L)$

$M_2 \leftarrow \text{MaximaSet}(G)$

 Let q be the smallest point in M_2

for each point, r , in M_1 **do** by x -coordinates

if $x(r) \leq x(q)$ **and** $y(r) \leq y(q)$ **then**

 Remove r from M_1

return $M_1 \cup M_2$

Running time is $T(n) = 2T(n/2) + T_{\text{media}}(n) + T_{\text{min}}(n) + \Theta(n)$
 $= 2T(n/2) + T_{\text{media}}(n) + \Theta(n)$

Case study IV: Maxima Set

Pseudocode:

M

Next week we will see how to find the **median** in $\Theta(n)$ time!

This fact will yield $\Theta(n \log n)$ for Maxima Set.

ordinates
in S by x -coordinates

$M_2 \leftarrow \text{MaximaSet}(G)$

Let q be the smallest point in M_2

for each point, r , in M_1 **do** by x -coordinates

if $x(r) \leq x(q)$ **and** $y(r) \leq y(q)$ **then**

 Remove r from M_1

return $M_1 \cup M_2$

Running time is $T(n) = 2T(n/2) + T_{\text{media}}(n) + T_{\text{min}}(n) + \Theta(n)$
 $= 2T(n/2) + T_{\text{media}}(n) + \Theta(n)$