CSCI3160: Tutorial 3

Problem 1

- *O*(*n*log*n*)-time algorithm for finding the number of inversions.
- **Problem 2**
 - *O*(*n*log*n*)-time algorithm to solve the dominance counting problem.

- Problem: Given an array A of n distinct integers, count the number of inversions.
- \Box An inversion is a pair of (i, j) such that
 - $1 \le i < j \le n$.
 - A[i] > A[j].

Example: Consider A = (10, 3, 9, 8, 2, 5, 4, 1, 7, 6). Then (1, 2) is an inversion because A[1] = 10 > A[2] = 3. So are (1, 3), (3, 4), (4, 5), and so on. There are in total 31 inversions.

Let: A = (10, 3, 9, 8, 2, 5, 4, 1, 7, 6)

- $A_1 = (10, 3, 9, 8, 2), A_2 = (5, 4, 1, 7, 6).$
- The counts of inversions in A_1 and A_2 are known by solving the "counting inversion" problem recursively on A_1 and A_2 .

- Let: A = (10, 3, 9, 8, 2, 5, 4, 1, 7, 6)
 - $A_1 = (10, 3, 9, 8, 2), A_2 = (5, 4, 1, 7, 6).$
 - The counts of inversions in A₁ and A₂ are known by solving the "counting inversion" problem recursively on A₁ and A₂.
- □ We need to count the number of crossing inversion (i, j) where *i* is in A_1 and *j* in A_2 .

- Let: A = (10, 3, 9, 8, 2, 5, 4, 1, 7, 6)
 - $A_1 = (10, 3, 9, 8, 2), A_2 = (5, 4, 1, 7, 6).$
 - The counts of inversions in A_1 and A_2 are known by solving the "counting inversion" problem recursively on A_1 and A_2 .
- □ We need to count the number of crossing inversion (i, j) where *i* is in A_1 and *j* in A_2 .

□ Binary search

• Sort A_1 and A_2 , and conduct n/2 binary searches ($O(n \log n)$).

- Let: A = (10, 3, 9, 8, 2, 5, 4, 1, 7, 6)
 - $A_1 = (10, 3, 9, 8, 2), A_2 = (5, 4, 1, 7, 6).$
 - The counts of inversions in A_1 and A_2 are known by solving the "counting inversion" problem recursively on A_1 and A_2 .
- □ We need to count the number of crossing inversion (i, j) where *i* is in A_1 and *j* in A_2 .
- □ Binary search
 - Sort A_1 and A_2 , and conduct n/2 binary searches ($O(n \log n)$).
 - Let f(n) be the worst-case running time of the algorithm on n numbers.
 - ✓ $f(n) \le 2f(\lceil n/2 \rceil) + O(n \log n)$
 - ✓ which solves to $f(n) = O(n\log^2 n)$.

Counting inversions: a faster algorithm

□ Strategy: ask a harder question, and exploit it in the conquer phase.

Counting inversions and sorting

- □ Strategy: ask a harder question, and exploit it in the conquer phase.
- Given an array A of n distinct integers, output the number of inversions and produce an array to store the integers of A in ascending order.

Counting inversions and sorting

- □ Strategy: ask a harder question, and exploit it in the conquer phase.
- Given an array A of n distinct integers, output the number of inversions and produce an array to store the integers of A in ascending order.
- $\Box A = (10, 3, 9, 8, 2, 5, 4, 1, 7, 6)$
 - $A_1 = (2,3,8,9,10), 8 \text{ invs}; A_2 = (1,4,5,6,7), 4 \text{ invs}.$

Counting inversions and sorting

- □ Strategy: ask a harder question, and exploit it in the conquer phase.
- Given an array A of n distinct integers, output the number of inversions and produce an array to store the integers of A in ascending order.
- $\Box A = (10, 3, 9, 8, 2, 5, 4, 1, 7, 6)$
 - $A_1 = (2,3,8,9,10), 8 \text{ invs}; A_2 = (1,4,5,6,7), 4 \text{ invs}.$
- Exploit subproblem property
 - Subarrays A_1, A_2 are sorted
 - Count crossing inversions in O(n) time.
 - \blacktriangleright Merge 2 sorted arrays in O(n) time.

□ Let S_1 and S_2 be two disjoint sets of *n* integers. Assume that S_1 is stored in an array A_1 , and S_2 in an array A_2 . Both A_1 and A_2 are sorted in ascending order. Design an algorithm to find the number of such pairs (a, b) satisfying the following conditions:

- ✓ $a \in S_1$,
- ✓ $b \in S_2$,
- ✓ a > b.
- ✓ Your algorithm must finish in O(n) time.

□ Method

• Merge A_1 and A_2 into one sorted list A.

Let: A = (10, 3, 9, 8, 2, 5, 4, 1, 7, 6)

•
$$A_1 = (2,3,8,9,10), A_2 = (1,4,5,6,7)$$

$$A_1 \qquad \boxed{2} \qquad \boxed{3} \qquad \boxed{8} \qquad \boxed{9} \qquad \boxed{10}$$

$$A_2 \qquad \boxed{1} \quad \boxed{4} \quad \boxed{5} \quad \boxed{6} \quad \boxed{7}$$

□ We will merge them together and in the meantime maintain the count of crossing inversions.



- Ordered list produced: Nothing yet
- The count of crossing inversions : 0



- Ordered list produced: 1
- The count of crossing inversions : 0



- Ordering produced: 1, 2
- The count of crossing inversions : 0 + 1 = 1.

Last count Newly added: (2,1) is a crossing inversion



- Ordering produced: 1, 2, 3
- The count of crossing inversions : 1 + 1 = 2.

Last count Newly added: (3,1) is a crossing inversion.



• Ordering produced: 1, 2, 3, 4

• The count of crossing inversions : 2

Last count



- Ordering produced: 1, 2, 3, 4, 5
- The count of crossing inversions : 2

Last count



Ordering produced: 1, 2, 3, 4, 5, 6
The count of crossing inversions : 2.

Last count



Ordering produced: 1, 2, 3, 4, 5, 6, 7
The count of crossing inversions : 2 Last count



٦

- Ordering produced: 1, 2, 3, 4, 5, 6, 7, 8
- The count of crossing inversions : 2 + 5 = 7.

K



- Ordering produced: 1, 2, 3, 4, 5, 6, 7, 8, 9
- The count of crossing inversions : 7 + 5 = 12.

Last count Newly added count: (9,1), (9,4), (9,5), (9,6), (9,7)

۲



٦

- Ordering produced: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
- The count of crossing inversions : 12 + 5 = 17.

Last count Newly added count: #integers from A_2 already in the ordered list produced

۲

Counting inversions

□ Analysis

Let f(n) be the worst-case running time of the algorithm on n numbers.

Then

- $f(n) \le 2f([n/2]) + O(n),$
- which solves to $f(n) = O(n \log n)$.

Problem

• Give an *O*(*n*log*n*)-time algorithm to solve the dominance counting problem discussed in the class.

□ Point dominance definition

- Denote by N the set of integers. Given a point p in twodimensional space N², denote by p[1] and p[2] its x- and ycoordinates, respectively.
- Given two distinct points p and q, we say that q dominates p if $p[1] \le q[1]$ and $p[2] \le q[2]$.



□ Let *P* be a set of n points in \mathbb{N}^2 . Find, for each point $p \in P$, the number of points in *P* that are dominated by *p*.



Divide: Find a vertical line l such that P has $\lfloor n/2 \rfloor$ points on each side of the line. (k-selection, O(n) time).



Divide:

- P_1 = the set of points of *P* on the left of *l*.
- P_2 = the set of points of *P* on the right of *l*.



Divide:

• Solve the dominance counting problem on P_1 and P_2 separately.



On P_1 , we have obtained: $(p_1, 0), (p_2, 1), (p_3, 0), (p_4, 2)$. On P_2 , we have obtained: $(p_5, 0), (p_6, 1), (p_7, 0), (p_8, 0)$.

Divide:

- Solve the dominance counting problem on P_1 and P_2 separately.
- It remains to obtain, for each point *p* ∈ *P*₂, how many points in *P*₁ it dominates.

Example: p_2 p_4 p_6 p_4 p_5 p_5

 p_3

 p_7

 p_8

On P_1 , we have obtained: $(p_1, 0), (p_2, 1), (p_3, 0), (p_4, 2).$ On P_2 , we have obtained: $(p_5, 0), (p_6, 1), (p_7, 0), (p_8, 0).$

Review: Binary search

- Sort P_1 by y-coordinate. ($O(n \log n)$)
- Then, for each point $p \in P_2$, we can obtain the number of points in P_1 dominated by p using binary search. ($O(n \log n)$)



 P_1 in ascending of y-coordinate: p_3, p_1, p_4, p_2 .

How to perform binary search to obtain the fact that p_5 dominates 2 points in P_1 ?

• Search using the y-coordinate of *p*₅.

Dominance counting: a faster algorithm

Ask a harder question:

- Output the dominance counts and sort *P* by y-coordinate.
- □ Scan the point from P_1 by y-coordinate in ascending order, and scan P_2 in the same way synchronously.
 - Merge the following two sorted arrays, based on y-coordinates and obtain the number of points in P_1 dominated by p.

•
$$P_1 = (p_3, p_1, p_4, p_2)$$

•
$$P_2 = (p_8, p_7, p_5, p_6)$$



□ Scan the points from P_1 by y-coordinate in ascending order. Do the same on P_2 .

•
$$P_1 = (p_3, p_1, p_4, p_2)$$

•
$$P_2 = (p_8, p_7, p_5, p_6)$$





$$\square P_1 = (p_3, p_1, p_4, p_2)$$

$$\square P_2 = (p_8, p_7, p_5, p_6)$$

- $\Box P = ()$
 - All the points will be stored in this array in ascending order of y-coordinate.
 - To be produced by merging P_1 and P_2 .

$$\square P_{1} = (p_{3}, p_{1}, p_{4}, p_{2})$$

$$\square P_{2} = (p_{8}, p_{7}, p_{5}, p_{6})$$

$$\square \text{ count} = 0$$

$$\square \overline{P} = ()$$



$$\square P_1 = (p_3, p_1, p_4, p_2)$$
$$\square P_2 = (p_8, p_7, p_5, p_6)$$
$$\square \text{ count} = 0$$
$$\square \overline{P} = (p_8)$$

• p_8 dominates 0 point in P_1 .



$$\square P_{1} = (p_{3}, p_{1}, p_{4}, p_{2})$$

$$\square P_{2} = (p_{8}, p_{7}, p_{5}, p_{6})$$

$$\square \text{ count} = 0$$

$$\square \overline{P} = (p_{8}, p_{3})$$



$$\square P_{1} = (p_{3}, p_{1}, p_{4}, p_{2})$$

$$\square P_{2} = (p_{8}, p_{7}, p_{5}, p_{6})$$

$$\square \text{ count} = 0$$

$$\square \overline{P} = (p_{8}, p_{3}, p_{1})$$



$$\square P_{1} = (p_{3}, p_{1}, p_{4}, p_{2})$$

$$\square P_{2} = (p_{8}, p_{7}, p_{5}, p_{6})$$

$$\square \text{ count} = 2$$

$$\square \overline{P} = (p_{8}, p_{3}, p_{1}, p_{7})$$

• p_7 dominates 2 point in P_1



$$\square P_{1} = (p_{3}, p_{1}, p_{4}, p_{2})$$

$$\square P_{2} = (p_{8}, p_{7}, p_{5}, p_{6})$$

$$\square \text{ count} = 4$$

$$\square \overline{P} = (p_{8}, p_{3}, p_{1}, p_{7}, p_{5})$$

• p_5 dominates 2 point in P_1



$$\square P_{1} = (p_{3}, p_{1}, p_{4}, p_{2})$$

$$\square P_{2} = (p_{8}, p_{7}, p_{5}, p_{6})$$

$$\square \text{ count} = 4$$

$$\square \overline{P} = (p_{8}, p_{3}, p_{1}, p_{7}, p_{5}, p_{4})$$







$$\Box P_{1} = (p_{3}, p_{1}, p_{4}, p_{2}).$$

$$\Box P_{2} = (p_{8}, p_{7}, p_{5}, p_{6}).$$

$$\Box \text{ count} = 8$$

$$\Box \overline{P} = (p_{8}, p_{3}, p_{1}, p_{7}, p_{5}, p_{4}, p_{2}).$$

$$\Box \text{ Current time complexity: } O(n).$$

, *p*₆).

□ Analysis

- Let f (n) be the worst-case running time of the algorithm on n points.
- $f(n) \le 2f([n/2]) + O(n),$
- which solves to $f(n) = O(n \log n)$.