

Note: Your TA probably will not cover all the problems. This is totally fine, the discussion worksheets are deliberately made long so they can serve as a resource you can use to practice, reinforce, and build upon concepts discussed in lecture, readings, and the homework.

1 Pareto optimality

Given a set of points $P = \{(x_1, y_1), (x_2, y_2) \dots (x_n, y_n)\}$, a point $(x_i, y_i) \in P$ is Pareto-optimal if there does not exist any $j \neq i$ such that $x_j > x_i$ and $y_j > y_i$. In other words, there is no point in P above and to the right of (x_i, y_i) .

- (a) Design a $O(n \log n)$ -time divide-and-conquer algorithm that given P , outputs all Pareto-optimal points in P .

Hint: Split the array by x -coordinate. Show that all points returned by one of the two recursive calls is Pareto-optimal, and that you can get rid of all non-Pareto-optimal points in the other recursive call in linear time.

- (b) Analyze the runtime of your algorithm.

2 Maximum subarray sum

Given an array A of n integers, the maximum subarray sum is the largest sum of any contiguous subarray of A (including the empty subarray). In other words, the maximum subarray sum is:

$$\max_{1 \leq i \leq j \leq n} \sum_{k=i}^j A[k]$$

For example, the maximum subarray sum of $[-2, 1, -3, 4, -1, 2, 1, -5, 4]$ is 6, the sum of the contiguous subarray $[4, -1, 2, 1]$.

- (a) Design an $O(n \log n)$ -time divide-and-conquer algorithm that finds the maximum subarray sum.
Hint: Split the array into two equally-sized pieces. What are the types of possibilities for the max subarray, and how does this apply if we want to use divide and conquer?

- (b) Briefly justify the correctness of your algorithm.
Hint: Use induction!

- (c) Prove that your algorithm runs in $O(n \log n)$.

3 Monotone matrices

A m -by- n matrix A is *monotone* if $n \geq m$, each row of A has no duplicate entries, and it has the following property: if the minimum of row i is located at column j_i , then $j_1 < j_2 < j_3 < \dots < j_m$. For example, the following matrix is monotone (the minimum of each row is bolded):

$$\begin{bmatrix} \mathbf{1} & 3 & 4 & 6 & 5 & 2 \\ 7 & 3 & \mathbf{2} & 5 & 6 & 4 \\ 7 & 9 & 6 & 3 & 10 & \mathbf{0} \end{bmatrix}$$

- (a) Give an efficient (i.e., better than $O(mn)$ -time) algorithm that finds the minimum in each row of an m -by- n monotone matrix A .

Hint: monotonicity suggests that a binary-search-esque approach may be effective.

- (b) Provide a brief proof of correctness.

Hint; use induction!

- (c) Analyze the runtime of your algorithm. You do not need to write a formal recurrence relation; an informal summary is fine.

Challenge: rigorously analyze the runtime via solving a recurrence relation using the subproblem $T(m, n)$.