1. You are given an $n \times n$ matrix $A$ where every row is in sorted order and every column is in sorted order.

   Design an efficient (better than $\Omega(n^2)$) divide-and-conquer algorithm to search for an element $x$ in the matrix, and analyze your algorithm’s runtime complexity with respect to $n$.

2. You are playing a (one-player) game that starts out with $n$ characters arranged in a line, each of whom has a different number of strength points. Some of the characters belong to the red team, and the rest belong to the blue team. In each turn, you pick two adjacent characters to battle each other. The one with more points is the winner (in the case of a tie, the character on the left wins). The winner becomes stronger by absorbing the loser’s points, and the loser is removed from the game. The game is over when there’s only one character left.

   Given each character’s starting points $p_i$ and team affiliation $t_i$, design a dynamic programming algorithm that finds a sequence of turns that results in a red team victory (or determines that this is impossible). Analyze your algorithm’s runtime complexity.

3. $n$ people stand in a line, each holding a stack of numbered cards that is already in sorted order. The first person passes her stack to the second person, who uses a MergeSort-style merge operation to combine it with his own. Then, he passes the combined stack on to the third person, who repeats the process. In the end, the $n$th person holds a fully sorted stack of all the cards.

   These $n$ people are wondering if standing in a different order could speed up their procedure. For each person $i$, you are given the size $a_i$ of their stack of cards. Merging two stacks of size $x$ and $y$ takes $(x + y)$ time. Design a greedy algorithm that finds the optimal ordering of the people, and prove your algorithm’s correctness.