1. a) Proving $A[i,j] + A[i+1, j+1] \le A[i, j+1] + A[i+1, j]$

We know the rule is $A[i,j] + A[k, l] \le A[i, l] + A[k, j]$
For solving this problem, we'll use induction method and base case is $k = i+1$.

$$A[i,j] + A[k, j+1] \le A[i, j+1] + A[k, j]$$
$$A[k,j] + A[k+1, j+1] \le A[k, j+1] + A[k+1, j]$$

$$A[i,j] + A[k,j] + A[k, j+1] + A[k+1, j+1] \le A[i, j+1] + A[k, j+1] + A[k,j] + A[k+1, j]$$

$$A[i,j] + A[k+1, j+1] \le A[i, j+1] + A[k+1, j]$$

b) Some 2D array is given, but order of just two elements are wrong. We know the rule:

$$A[i,j] + A[i+1, j+1] \le A[i, j+1] + A[i+1, j]$$

We apply the rule all the elements in the array.
If there is inconsistency between elements, then we look for replacement.
After when we replace the elements there is no wrong, and every elements okey the rule, it means done.

For example;

$$arr = \begin{matrix} 10 & 17 & 28 & 13 & 23 \\ 17 & 22 & 16 & 29 & 23 \\ 24 & 28 & 22 & 34 & 24 \\ 11 & 13 & 6 & 17 & 7 \\ 45 & 44 & 32 & 37 & 23 \\ 36 & 33 & 19 & 21 & 6 \\ 75 & 66 & 51 & 53 & 34 \end{matrix} \xrightarrow[28,\ 13]{\text{we swap}} \begin{matrix} 10 & 17 & 13 & 28 & 23 \\ 17 & 22 & 16 & 29 & 23 \\ 24 & 28 & 22 & 34 & 24 \\ 11 & 13 & 6 & 17 & 7 \\ 45 & 44 & 32 & 37 & 23 \\ 36 & 33 & 19 & 21 & 6 \\ 75 & 66 & 51 & 53 & 34 \end{matrix}$$

$10 + 22 \le 17 + 17$ ✓

$28 + 29 \le 13 + 16$ ✗

$28 + 16 \le 13 + 29$ ✓

c) Finding leftmost minimum element in each row by divide and conquer, we can use tournament method.

→ First we check if array has 1 single element or not. If so, we return that element.

→ Then we check if array has 2 elements. If so, we control and return minimum element.

→ Last but not least, we divide the array into two parts. And do recursive call both parts. We return which one is smaller.

d) This algorithm solves problems by dividing them into 2 subproblems of half the size, recursively solving each subproblem, and then combining the solution in constant time.

$T(n) = 2T(n/2) + O(1)$

We can use Master Theorem for solving it.

$a=2 \quad b=2 \quad d=0$

$(\log_a b = \log_2 2 = 1) > (d=0) \rightarrow$ case 1 $\rightarrow O(n)$

2. For solving this problem we can follow these steps:

→ First, we look size of these two arrays. If one of it's length is 0, we return other array.

→ Then we find middle points of arrays. We compare the middle elements of theese two arrays.

→ If arrA[midA] > arrB[midB], we don't look to after middle element of arrB or vice versa.

→ In this method, we divide the problem half of it's size.

→ Let's assume sizes of arrays are i and j.
The worst-case running time complexity would be $O(\log i + \log j)$. Because we divide theese two arrays seperately.

3. For solving this problem we can follow these steps:

→ We divide the array into two arrays

→ Then we calculate

1. maximum contiguous subset sum in left half
2. maximum contiguous subset sum in right half
3. maximum contiguous subset sum in midpoint

→ 1. and 2. steps are recursive.

→ Worst-case runnig time $Q(n \log n)$. Because the program has recurrence relation and can be express as:

$T(n) = 2T(n/2) + Q(n)$

We apply Master Theorem's 2. rule.

$a=2 \quad b=2 \quad d=1$

$\log_2 2 = 1 \longrightarrow Q(n \log n)$

Scanned by CamScanner

5- For solving this problem we can follow these steps:

→ First we find the gain array by substract of price and cost arrays

→ And we check if all elements of the gain array are lesser or equal 0 to report if there is no day to make money.

→ Then we apply divide and conquer method find the index of the largest element in gain array.
We divide the array into two parts and same thing will apply both left and right parts.

→ We take the index, because we need to find the day of maximum gain, not the maximum gain.

→ Worst-case running time is $O(n)$.
Recurrence relation is:
$T(n) = 2T(n/2) + 2$
Apply Master Theorem → case 1 → $O(n)$

4- For solving this problem we can follow these steps;

→ First of all, in question wants to solve decrease and conquer aproach. So I chose DFS for solving it.

→ In DFS we follow the first path as far as we can go

→ If we reach a dead end, we will go back to last explored edge

→ While we're doing these, we use recursion

→ Worst-case running time is $O(n)$.