(1)

a) The "only if" part is trivial, it follows form the definition of Specid Array.

As for "if" part, let's first prove that

$$A[i,j]+A[i+1,j+1] \leq A[i,j+1]+A[i+1,j]$$
$$\Rightarrow A[i,j]+A[k,j+1] \leq A[i,j+1]+A[k,j],$$

where $i<k$.

Let's proxe it by induction. The base case of $k=i+1$ is given. As for the inductive step, we assume it holds for $k=i+n$ and we want to prove it for $k+1=i+n+1$. if we add the given to the assumption, we get

$$A[i,j]+A[k,j+1] \leq A[i,j+1]+A[k,j]$$
$$A[k,j]+A[k+1,j+1] \leq A[k,j+1]+A[k+1,j]$$
$$\Rightarrow A[i,j]+A[k,j+1]+A[k,j]+A[k+1,j+1] \leq A[i,j+1]+A[k,j]+A[k,j+1]+A$$
$$\Rightarrow A[i,j]+A[k+1,j+1] \leq A[i,j+1]+A[k+1,j]$$

(1)

b) Firstly I looked for fails the given array, when the fail come across, checked for each step, if there are four fails occured, they all have same intersection index. With one movement the array turns special array Implemented in .py file.

c) Firstly, I looked for minimum element in array and I looked for the first position of the element from the left side then return it. Implemented in .py file.

1 d) The divide time is $O(1)$, the conquer part is $T(n/2)$ and the merge part is $O(m+n)$. Thus,

$$T(m) = T(m/2) + cn + dm$$

$$= cn + dm + cn + dm/2 + cn + dm/4 + \dots$$

$$= \sum_{i=0}^{\lg m - 1} cn + \sum_{i=0}^{\lg m - 1} \frac{dm}{2^i}$$

$$= cn \lg m + dm \sum_{i=0}^{\lg m - 1}$$

$$< cn \lg m + 2dm$$

$$= O(cn \lg m + m).$$

(2) I compare the middle elements of arrays arr1 and arr2, let us call these indices mid1 and mid2 respectively, let us assume arr1[mid1] < k, then clearly the elements after mid2 cannot be the required element. We then set the last element of arr2 to be arr2[mid2]. In this way, we define a new subproblem with half the size of one of the arrays. k is 0 indexed, which means if we want a k that's 1 indexed, we have to subtract 1 when passing it to the function.

Time Complexity $O(\log n + \log m)$

③ The given list is divided into two parst. Problem is solved for each part recursively. At the end, the result are combined, The divide part is straightforward In combine part, the sum of left and right subsets are calculated seperatly, using built in sum function at first, if the two subsets are contigous, then the sum of the left and right subsets calculated. If the sum is greater then or equal to both subset sum the two subset are merged. If the sum of elements are calculated from beginning of the left subsets to the right subset.

The complexity of dividing is $O(\log n)$

The complexity of merging is $O(n)$

Therefore the complexity of this problem is $O(n \log n)$

(4) ⇒ Use a color[] array which stores 0 or 1 for every node which denotes opposite colors.

⇒ Call the function from any node

⇒ If the node u has not visited previously then assign !color[u] to color[u] and call function again to visit nodes connected to u.

⇒ If at any point, color[u] is equal to !color[v] then the node is biportite.

⇒ Modify the DFS function such that it returns a boolean value at the end.

Time Complexity O(n)

(5) Firstly, I calculated for each day's profit to new arry, then, find max in this array using divide and conquer, each time divide the array two parts, then check if there are 2 elements, returns greater, if there are more than 2 elements again divide two part to find max profit day.

So each days profit calculating $O(n)$

then find max day is $O(2\log n) = O(\log)$

$$O(n) + O(\log n) = O(n)$$

time Complexity is $O(n)$