Q2.

(a) We denote the problem of finding the best cave in each row of a matrix $n \times m$ as $T(n, m)$. We just scan the middle row $\lfloor \frac{n}{2} \rfloor$ and find the index of the max number in that row, denote as j. Then we know the best caves in rows above middle row must be at indices no larger than j and the best caves in rows below middle row must be at indices no less than j. Thus we just need to find caves in upper left matrix $T(n - \lfloor \frac{n}{2} \rfloor - 1, j)$ and lower right matrix $T(n - \lfloor \frac{n}{2} \rfloor - 1, m - j + 1)$, both sub-matrices contain the jth column but exclude the middle row of the original matrix. The pseudocode is as follows:

Algorithm 1 $FindBestCaves(A, startRow, endRow, startCol, endCol, M)$

Require: A is an matrix, each row of A contains distinct numbers

```
1: if startRow>endRow then
2:    return
3: end if
4: if startCol==endCol then
5:    for row = startRow to endRow do
6:       M[row] = startCol
7:    end for
8:    return
9: else
10:   middleRow = endRow + floor((startRow − endRow + 1)/2)
11:   maxCol = startCol
12:   for col = startCol+1 to endCol do
13:      if A[middleRow][col]¿A[middleRow][maxCol] then
14:         maxCol = col
15:      end if
16:   end for
17:   M[middleRow] = maxCol
18:   FindBestCaves(A, startRow, middleRow − 1, startCol, maxCol, M)
19:   FindBestCaves(A, middleRow + 1, endRow, maxCol, endCol, M)
20: end if
21: return
```

(b) If we find the best cave in jth row is at index i, $M[j] = i$, then we know $M[k] \leq i \forall k < j$ so the best caves in the rows above jth row lie in the first i columns. Thus to find the best caves in the rows above jth row we only need to search the sub-matrix $A[1 : j - 1, 1 : i]$. Similarly we know $M[l] \geq i \forall l > j$, so to find the best caves in the rows below jth row we only need to search the sub-matrix $A[j + 1 : n, i : n]$. So on and so forth, we can find all the best caves.

(c) We divide solving a $n \times m$ matrix by splitting the rows half by half so the row number of each subproblem on the same recurrence level are the same. However the columns number varies, and the best case is when we have a $n \times 1$

matrix we know immediately the best caves in these n rows and no more divide of this subproblem is needed. Thus the worst case is we have to divide each subproblem into two until the last level. Denote $T(n)$ as solving a matrix of $n \times n$, we have the recurrence:

$$T(n) = 2T(\frac{n}{2} + 1) + O(n)$$

(d) $a = 2, b = 2, c = 1 = \log_b a,\ f(n) = O(n \log^k n), k = 0$ according to master theorem we have $T(n) = \Theta(n^{\log_b a} \log^{k+1} n = \Theta(n \log n)$. The worst-case running time complexity of solving a problem of size n is $\Theta(n \log n)$.