pseudocode

```
* aCountRow (char[] row, int start, int end)
    if end>start:
        middle = ⌊(end+start)/2⌋
        if row[middle] = 'a':
            if row[middle+1] = 'b':        } if we are at rightmost 'a'
                return middle+1
            if start = middle:             → if the entire array is filled with 'a'
                return middle+2
            start = middle                 → check upper half now
            return aCountRow (char[]row, int start, int end)
        if row[middle] = 'b':
            if row[middle-1] = 'a':        } if we are at leftmost 'b'
                return middle
            end = middle                   → check lower half now
            return aCountRow (char[]row, int start, int end)

    else:
        return 0

* a count (char[][]mat):
    result = 0
    dimension = mat.length
    for i=0 to i<dimension:
        result += aCountRow (mat[i], 0, dimension-1)

    return result
```

Time and space complexity

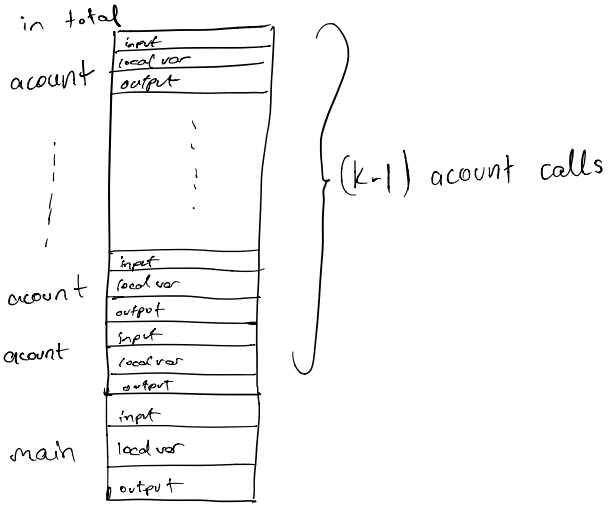
*In my algorithm, the goal is to find either the rightmost 'a' or the leftmost 'b' by reducing the row size to its half each time. If the row is n-dimensional and if it takes k number of halving to reduce the row size to 1, then:

$$\frac{n}{2^k} = 1 \quad \Rightarrow \quad k = \log_2 n$$

Before each halving, the algorithm does constant number of operations, call it c_1. Then, the total number of operations for each row is approximately $c_1 \log n$.

And the total number of operations for the entire matrix is approximately $c_1 n \log n$ since we repeat acount for n times. Therefore, the time complexity is $O(n \log n)$

* For each row after I call acount first time, since it's recursive I call it k-2 times more before I return the last acount call I call acount (k-1) times in total



Above, you can see the stack right before the last acount call returns an integer. Here, we have k function calls with each having constant number of variables. After the last acount call returns an integer, the memories allocated to each returned acount is freed. Therefore, for each row, the space complexity is $O(\log n)$ and it is also $O(\log n)$ for acountrow because since each returned acount is returned, stack can have at most $c \cdot \log n$ number of functions in it. Therefore the space complexity is $O(\log n)$.

I have completed this assignment individually, without support from anyone else. I hereby accept that only the below listed sources are approved to be used during this assignment:

(i) Course textbook,

(ii) All material that is made available to me by the professor (e.g., via Blackboard for this course, course website, email from professor / TA),

(iii) Notes taken by me during lectures.

I have not used, accessed or taken any unpermitted information from any other source. Hence, all effort belongs to me.

Umur Berkay Karakaş