Project Report

In my project, I use a plenty of algorithms to achieve the required tasks. Following are my explanation of them one by one:

Task1:

First, I use a Scanner to read the input file, and store the three numbers from the first line. (That will use 3 space complexity.) Then, for the second line of the input file, which is the first line of the matrix, I store the (0, 1) numbers of this line to an array's first line. (The array has 2 lines, that will use 2*M space complexity.) The principle of judging whether the number is 0 or 1 is to judge if the corresponding number in the input file equals to C. Then, for every other line of the input file, store what I will calculate followed to the second line of the array. When I meet the numbers equal to C, I choose the minimum of the left, upper and diagonal of this number to plus one, otherwise choose 0 to store in the array. By doing this, every plus one is made when this makes a required square. Use a parameter named borderSideLength to store the maximum side length of the square. Then, every time I meet a number equals to C, I compare the number which I calculated in the array to the borderSideLength, if it's bigger, update the borderSideLength, and store the corresponding row and column of this number. After every row calculated, copy the second line of array to the first line, and repeat. At last, after doing things mentioned above, which looped M * N times, we have the maximum side length and the square's lower right corner's coordinate X2, Y2. Then we can calculate the upper left corner's coordinate X1, Y1. (That will use M * N time complexity) Thus, task1 is finished, using $\Theta (N \cdot M)$ time complexity and O(M) space complexity.

Task2:

In this task, I use divide and conquer. First do the same things as task 1 to store M, N, and C. Then, create an array[M] which has only one line and all values are 0. (That will use M space complexity) For every line in the input file, if the number equals to C, the corresponding value in the array plus one, otherwise make the value 0. Then, this task becomes an easier task: find the largest rectangle of the histogram. To do this, divide this

array into three parts: left, right, and medium. Finding the largest rectangle of each part and keep doing this to conquer. After finding the largest rectangle, store the largest area, the lower right corner's Y2, and the row of the matrix. At last we can calculate all the coordinates needed. Divide and conquer uses $\Theta (M \cdot \log(M))$ time for every line of the matrix, and we have N lines in the matrix, so the total time is $\Theta (N \cdot M \cdot \log(M))$. And we only used an array[M], and the number of other spaces is all constant, so the space complexity is O(M).

Task3:

In this task, I do the same thing as task2 to store M, N, C, and change the task to find the largest rectangle area of the histogram. This time I use a stack to do the easier task. The stack is to store the largest height of the numbers, and for every number in the array, whenever the peek of the stack's height is smaller than that number, we push the index into the stack. Otherwise, if the peek of the stack's height is bigger, pop from the stack and find the biggest area of the histogram from the array[0] to this number. Store the corresponding coordinates and calculate others. Because I use a stack for every line, taking $\Theta (M)$ time, the total time complexity is $\Theta (N \cdot M)$. We only used an array[M], and the number of other spaces is all constant, so the space complexity is O(M).

Task4:

To achieve the time complexity of this task, I use the main idea of task2. But the histogram cannot be built like task2. First, save the input file into an array, and find the minimum and the maximum number of the input matrix. Then, from the minimum number, the numbers of [min, min + C] should be in the required rectangle, and for the number which is 1 bigger than the minimum, [min + 1, min + 1 + C], etc. So, just add another loop outside the row loop, which takes a parameter t from minimum to (maximum – C). And change the plus one principle of the array to that the number is between t and (t + C). The space complexity will be $O (M \cdot N)$. I can reduce it by O(M) by not storing the matrix but using System.in will make the input file only can be read by once. Because task2 has a time complexity of $\Theta (N \cdot M \cdot \log(M))$, and I add another loop which will run (maximum – C – minimum + 1) times, the total time complexity will be $\Theta (C \cdot N \cdot M \cdot \log(M))$.