Dheeraj Satya Sushant Viswanadham
(z5204820)

COMP3121 Assignment 1 – Question 4

4) To start off with, we would need to first obtain the co-ordinates of the four corners of the 4n by 4n square. Then, starting with the top left corner, which we mark as our origin point (0, 0) we can then go through each square and store the number of trees into an array A[x][y] where x and y are the co-ordinates of the larger square from the origin to (x, y). This takes a time complexity of O(n^2) as we are going through each square in the matrix/orchard. Additionally, if we already have overlapping rectangles, then we can simply find our sum by subtracting the overlapping rectangle from the other neighbouring rectangles. Once our array A[x][y] has been generated, we can then find out how many trees are in a given square given its co-ordinates (which has time complexity of O(1)). Essentially, we are going through each square and finding the number of trees in it by using its neighbouring rectangle values.

E.g. given size n square;

A[x][y] = large square

A[x – n][y] = right neighbouring rectangle

A[x][y - n] = left neighbouring rectangle

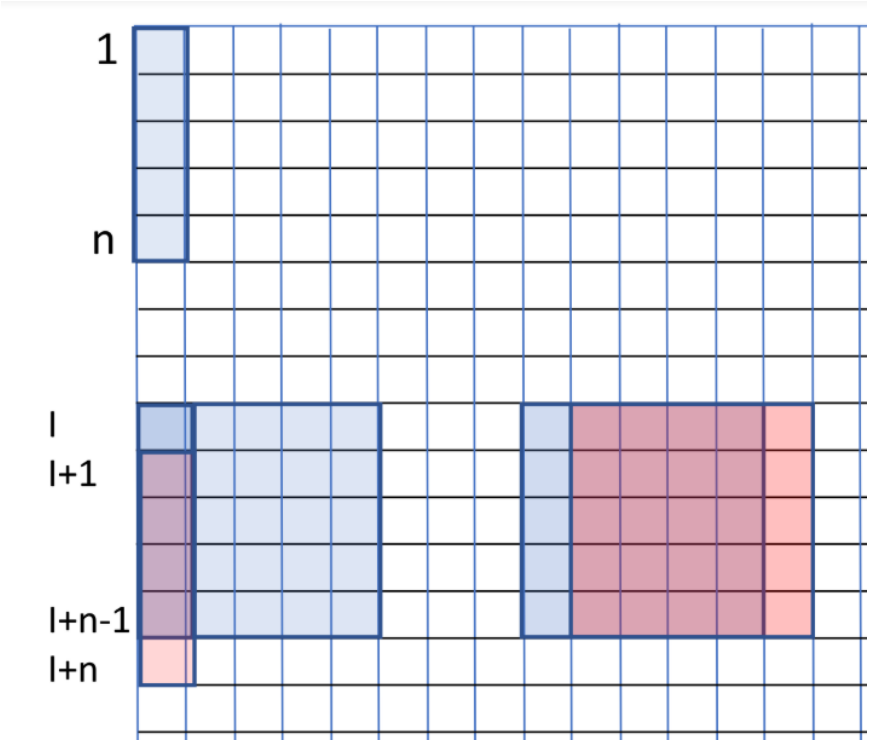
A[x – n][y - n] = overlapping square

Hence: A[x][y] - A[x – n][y] - A[x][y - n] + A[x – n][y - n] = our square of size n by n.

Ultimately, we will have time complexity of **O(n^2)** from the above algorithm.

Note: 'rectangle' is used above as a rectangle is a square but not vice-versa and the neighbouring rectangles do not have to be a square in order to find the sum of the n by n square.

See given diagram as an overall visual example: (note this diagram was emailed by Aleks as a hint)



Dheeraj Satya Sushant Viswanadham
(z5204820)

Example Pseudocode:

```
# Overall runtime of O(n^2):

# total[x][y] will be our total number of trees present between the origin of matrix[0][0] and matrix[x][y]

width = length(matrix);

# for loop to determine whether x is in the range of the length

    # for loop to determine whether y is in the range of the length

        # then we simply do our calculations as explained above i.e.

        total[x][y] = matrix[x][y];

        # if x > 0 then simply add it:

            # e.g. total[x][y] += total[x – 1][y];

        # if y > 0 then simply add it:

            # e.g. total[x][y] += total[x][y - 1];

        # make sure to also subtract any overlapping squares if both x & y > 0:

            # e.g. total[x][y] -= total[x – 1][y – 1];


# Then loop through the matrix/orchard again to find the number of trees of n square such as below:

# for loop to determine whether x is in the range of the length

    # for loop to determine whether y is in the range of the length

        # then we simply do our calculations as explained above given size n square i.e.

        final[x][y] = total[x][y];

        # if x - n is >= to 0 then simply remove right side rectangle:

            # e.g. final[x][y] -= total[x – n][y];

        # if y – n is >= to 0 then simply remove left side rectangle:

            # e.g. final[x][y] -= total[x][y - n];

        # make sure to add the overlapping squares if both (x – n) & (y – n) >= 0

            # e.g. final[x][y] += total[x – n][y – n];

# End
```