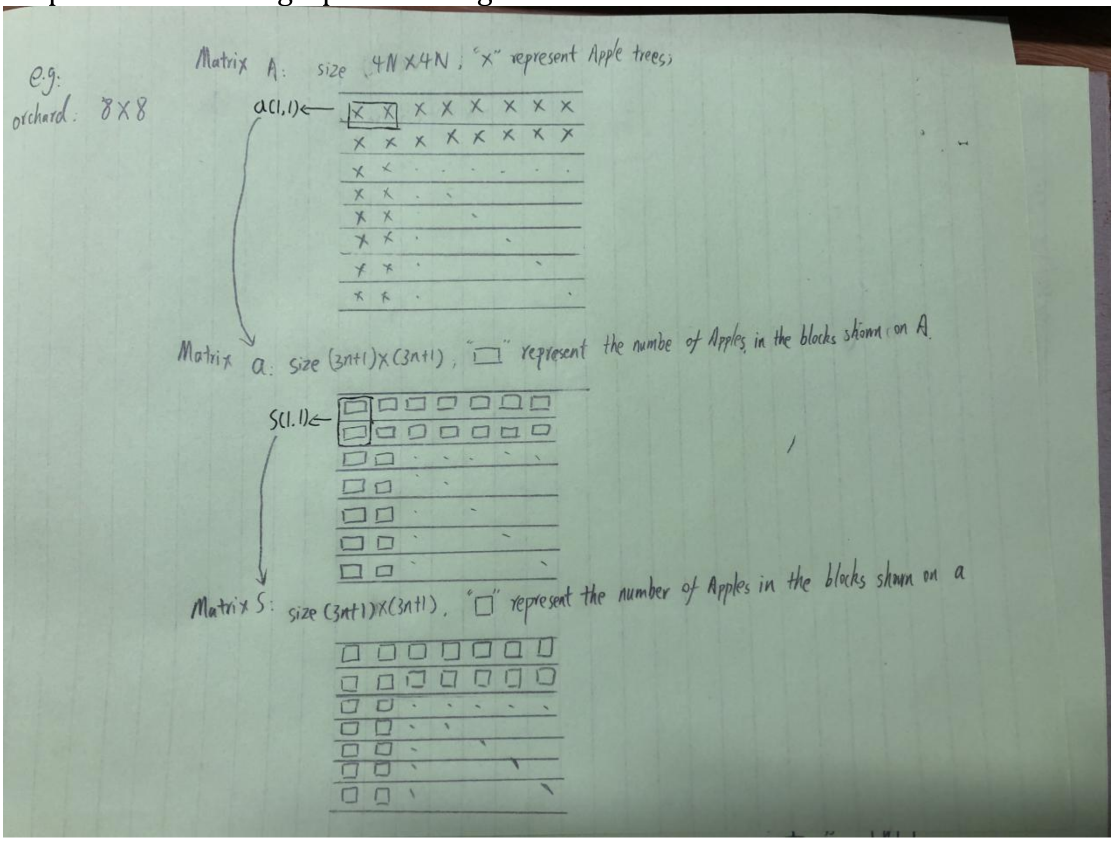
Given: An Orchard with size of 4N*4N, we can purchase apples form N*N
Aim: chose such a square which contains the largest total number of apples and which runs in time O(n^2)

Step 1:

Set the apples on each tree into a matrix that is A(0,0)~A(4N,4N), the rest steps shown on the graph following:



Step 2:

The size of each square we choose is N*N, then on each sides of the orchard we can choose 4N-N+1 squares. So that we can manipulate the N*N blocks into another matrix "a" which has size of (3N+1 * 3N+1) each element represents the apples in N*N square. For example the a(1,1) means the all apples in a 1*N block that starts from A(1,1) to A(1,1+n); a(2,2) means all apples in a 1*N block that starts from A(1,2) to A(1,2+n), ... a(1, 3N+1) means all apples in a 1*N block that starts from A(1, 2n+1) to A(1, 3n+1). So in one column the totally number of apples can be represented as a(i,j) = a(i,j) + a(i,j+1) + a(i,j+2) + ... + a(i,j+n-1).

Step 3:

We want to calculate all possible rectangle blocks of apples when row number is fixed. This can be achieved by calculate the number of apples in a(1,1) we will calculate n-1 times. Then from a(1,2) to a(1,3n+1) only need 2 times for each. Because the a(1,2) is equivalent to a(1,1) - A(1,1) + A(1,n+1). So totally calculated n - 1 + 2 * 3n = 7n-1 times which is O(n), then if we want to calculate from a(1,1) to a(3n+1, 3n+1) we need to calculate O(n) * (3n-1), the time complexity is O(n^2)

Step 4:

We want to calculate all possible square blocks of apples when column number is fixed. This can be achieved by calculate the number of apples in s(1,1) we will calculate n-1 times. Then from s(1,1) to s(3n+1,1) only need 2 times for each. Because the s(2,1) is equivalent to s(1,1) - a(1,1) + a(n+1, 1). So totally calculated n - 1 + 2 * 3n = 7n-1 times which is O(n), then if we want to calculate from s(1,1) to s(3n+1, 3n+1) we need to calculate O(n) * (3n-1), the time complexity is O(n^2)

Step 5:

The total complexity is 2*O(n^2), Go through all these possible combinations find out the largest combination.

Conclusion:

The algorithm can pick the biggest within O(n^2) time complexity.