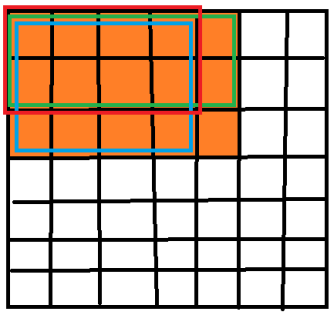
Question 4

Idea is to preprocess the orchard to allow us to calculate the total number of trees in a square in constant time. This can be done by using an array where arr[i][j] stores the sum of trees in the square between (0,0) and (i,j).

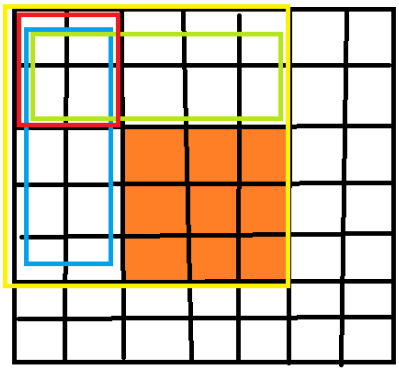
Building this array requires us to go through each cell in the orchard which takes n^2 time. We can also calculate the sum in constant time by utilising the cell's left and upper neighbouring cell values. We add the two neighbouring sums together and minus off the overlapping middle part to obtain the required sum.

Orange rectangle = Blue rectangle + green rectangle - overlapping red rectangle



Finally now that we have built our sum array, we can find the total number of trees of any sized rectangle from any index in constant time. In this question, the rectangle will be a square of size n. Again we will have to go through each cell in the orchard which is n^2 time. For each cell, we can compute how many trees by considering its neighbouring sums. The total will be sum[i][j] which is the large rectangle minus the left (sum[i][j - n]) minus the upper (sum[i - n][j]) and plus the overlapping top left corner (sum[i - n][j - n]). Overall runtime will be n^2.

Orange square = Yellow square - green rectangle - blue rectangle + overlapping red square



SAMPLE CODE

```
#!/usr/bin/python3
import random

def main():
	# TESTING

	grid = [[1,2,3,4,5,6,7,8] for i in range(8)]
	print("MAX TREES =", q4(grid))

	for row in grid:
		random.shuffle(row)
	print("MAX TREES =", q4(grid))

def q4(grid, squareSize=None):
	# sums[i][j] = sum of all trees from square between grid[0][0] and grid[i][j]
	# this is done in (4n)^2 time
	size = len(grid)
	sums = [[0 for i in range(size)] for j in range(size)]

	for i in range(size):
		for j in range(size):
			sums[i][j] = grid[i][j]

			if i > 0: # not first row, we add the upper rectangle
				sums[i][j] += sums[i - 1][j]

			if j > 0: # not first col, we add left rectangle
				sums[i][j] += sums[i][j - 1]


			# minus the middle overlap if there is one
			if i > 0 and j > 0:
				sums[i][j] -= sums[i - 1][j - 1]

	# Using sums[][] arr, we can loop through grid again in (4n)^2 time
	# we can calculate the sum of each n//4 sized square in constant time
	# total runtime is n^2
	if not squareSize:
		squareSize = size//4

	maxTrees = 0
	res = [[0 for i in range(size)] for j in range(size)] # debugging

	for i in range(squareSize - 1, size):
		for j in range(squareSize - 1, size):
			total = sums[i][j]

			# trim off excess
			if i - squareSize >= 0:
				total -= sums[i - squareSize][j]

			if j - squareSize >= 0:
				total -= sums[i][j - squareSize]

			# if we double minus the top corner square off, we need to add it back on
			if i - squareSize >= 0 and j - squareSize >= 0:
				total += sums[i - squareSize][j - squareSize]

			res[i][j] = total
			maxTrees = max(maxTrees, total)

	return maxTrees

if __name__ == '__main__':
	main()
```