Question 1)

Part A)

Algorithm Idea: We start out by creating an array that will store the sum of the product of each value. This will have a size of n. Then, we will use a for loop that will go through each row of matrix A, given that the size is n. It will start with at 1 and after finding each row, we will have another for loop that will take each value for the column of the matrix and each value for the vector, given that both of them are size n. We will find the sum of the product of the individual value of matrix A and vector x. What I mean is, I will do the following equation: ((An1\*xn1)+ (An2\*xn2)+ … + (Ani-1\*xni-1) + (Ani\*xni)), where i is the position of the number inside the column of the matrix and the position in the vector. This should produce a runtime of O(n2)

Algorithm Detail:

Given matrix A of size n x n – A

Given vector x of size n - x

ans = new int [n]; <- creating vector

for (i = 0; i < n; i++) <- (used to look through each row of matrix)

for ( j = 0; j < n; j++) <- (used to look at the value for the column of matrix and vector)

ans[i] += A[i][j] \* x[j] <- take previous ans[i] and add it to the product of A[j] and x[j]

Part B)

I can argue that my algorithm runs in Ω(n2). When we first look at the algorithm, we can see that we have to run the first for loop in n times. Then, with the second for loop, we run it again n times. When we do a nested for loop, we have the multiple the two run times together, which becomes n2. In order for something to run in Ω(n2), the run time you are dealing with must be greater or equal to n2. Similar with O(n2), the run time you are dealing with must be less than or equal to n2. Since the algorithm above runs in n2, it equals to both the O and Ω, and since it equals to both O and Ω, it becomes Θ.

Source: CSE 331 support page and TA recitation notes week 4.

Question 2)

Part A)

Algorithm Idea: We first create a matrix Ur by first making a for loop that takes in variable j. While j is less than or equal to n, we will have a if statement with 4 conditions. The first condition is, if j == n and i == n-1, then we break out of the loop, else if j == n and i != n-1, then increase i by one, else if j < i, Ur[i][j] 0, else Ur[i][j] will be r[i]. Once we have the matrix, we now multiple each column in the matrix by the vector. We will assume that the vector is a single column vector. We will add each iteration of the product of the columns to find the final column.

Algorithm Detail:

Given vector r – size n

Given vector x – size n

m = new int[n][n]

ans = new int[n]

i = 0;

for (j = 0; j <= n; j++)

if( j == n && i == n-1)

break;

else if ( j == n && i != n-1)

i++;

else if( j < i)

m[i][j] = 0;

else

m[i][j] = r[i];

for ( k = 0; k < m[n]; k++){

ans[k] += m[k] \* x[k]

Part B)

I can argue that my algorithm runs in Ω (n). At the first step of the algorithm, we have a for loop, which runs in n times. After the first for loop is done, we have another for loop, which runs in n times as well. When having two for loops that aren’t nested, we add the two loops together, which becomes n + n = 2n. Since 2 is a constant, we can ignore the 2. This means that our algorithm runs in n time. In order for something to run in Ω(n), the run time you are dealing with must be greater or equal to n. Similar with O(n), the run time you are dealing with must be less than or equal to n. Since the algorithm above runs in n, it equals to both the O and Ω, and since it equals to both O and Ω, it becomes Θ.