3.2 a) Derive the asymptotic time complexity depending on the number of bits n for a brute-force implementation of multiplication.

A and B are binary bit arrays. Result will be a binary bit array equal to the product of A and B. The pseudocode submitted down is just a simple way of multiplying (brute-force) by adding A with A, B times and storing in Result.

Pseudocode:

```
void Multiply(A, B); // Function to Multiply a and b
void Add(Result, A); // Function to Add a, b times and store it in result
Result = [0]; // Assigning result to 0
for( i = 0; i<length.B; i++) // Counting the digits in b[]
    if( B[i] == 1) // If the last one is 1
        Result = Add(result, A); // Add a, i times and store in result variable
A.append(0); // Add a 0 in the end of a
return Result;  // Return result
```

As the *for* loop goes only one to count the digits of *B[]* the asymptotic time complexity of this algorithm is: ***T(n) = O(n)***

3.2 b) Derive a Divide & Conquer algorithm for the given problem by splitting the problem into two subproblems. For simplicity you can assume n to be power of 2.

Pseudocode:

```
void Multiply(A, B); // Function to multiply A and B
if(length.A == 1) // Checking the length of A
    return (A[0]*B[0]); // If 1 just Multiply
Ahigh = A[n/2:n]; // By dividing the A [array] of bits in low and high
Allow = A[0:n/2];
Bhigh = A[n/2:n]; // By dividing the B [array] of bits in low and high
Blow = A[0:n/2];
x = Multiply(Ahigh, Bhigh); // Multiplying function
y = Multiply(Alow, Blow);
z = Multiply(Alow, Bhigh);
w = Multiply(Alow, Blow);
y = Binary_addition(y, z);  // Adding the binary digits in the end
x = Shifting(x, n); // Shifting n digits
y = Shifting(y, n/2);
return Binary_addition(x, y, w); // Returning the result
```

3.2 c) Derive a recurrence for the time complexity of the Divide & Conquer algorithm you developed for subpoint b).

Time complexity of this algorithm is *$T(n) = 4T(n/2) + O(n)$* as we are doing 4 functions with the n/2 digits (function are for multiplying, adding and shifting).