(a) For a brute-force multiplication implementation, we need to perform n multiplications for each digit in the first number, resulting in a total of n^2 multiplications. Since each multiplication takes O(n) time, the total time complexity of the brute-force implementation is O(n^3).

(b) We can split the two n-bit integers a and b into two n/2-bit integers, a1, a2, b1, and b2. We can then recursively compute three products: a1 \* b1, a2 \* b2, and (a1 + a2) \* (b1 + b2). The final product is the sum of these three products, shifted appropriately. The algorithm is as follows:

function multiply(a, b, n):

if n == 1:

return a \* b

else:

a1, a2 = split(a, n)

b1, b2 = split(b, n)

p1 = multiply(a1, b1, n/2)

p2 = multiply(a2, b2, n/2)

p3 = multiply(a1+a2, b1+b2, n/2)

return (p1 << n) + ((p3 - p1 - p2) << n/2) + p2

(c) The recurrence for the time complexity of the Divide & Conquer algorithm is:

T(n) = 3T(n/2) + O(n)

(d) To solve the recurrence using the recursion tree method, we can draw a tree where each node represents the time required for a recursive call at a given level. The number of nodes at each level is 3^i, where i is the level number, and each node has a size of O(n). Therefore, the total time required at each level is O(n \* 3^i). Summing over all levels, we get:

T(n) = O(n) + O(n \* 3) + O(n \* 3^2) + ... + O(n \* 3^(log n))

= O(n \* (3^(log n+1) - 1) / (3 - 1))

= O(n \* (3^(log n+1) - 1))

= O(n \* 3^(log n))

= O(n^log 3)

Therefore, the time complexity of the Divide & Conquer algorithm is O(n^log 3).

(e) To validate the result using the master theorem, we can compare the recurrence to the standard form:

T(n) = a T(n/b) + f(n)

where a = 3, b = 2, and f(n) = O(n). The master theorem states that the time complexity of the algorithm is:

O(n^log\_b a) if f(n) = O(n^(log\_b a - ε)) for some ε > 0

O(f(n)) if f(n) = Ω(n^(log\_b a + ε)) for some ε > 0 and a f(n/b) ≤ c f(n) for some constant c < 1 and sufficiently large n

In this case, f(n) = O(n), which falls under the second case, since:

log\_b a = log\_2 3 ~= 1.585

log\_b a < 2

f(n) = O(n^1.585)