Integer Multiplication - 20200808058

```
static long largeMultiply(long a, long b) {
    long[] df = new long[32]; // Array to store multiplication results
    Stack<Boolean> addStack = new Stack<>(); // Stack to track binary
    while (b > 1) {
        if (b % 2 == 0)
            addStack.push(true); // Push true onto the stack if b is even
        else {
            addStack.push(false); // Push false onto the stack if b is odd
            b--; // Subtracted to make the number b even
        }
        b = b / 2; // Update b by dividing it by 2
    }
    df[0] = a; // Initialize df array with the first element as a
    int i = 0; // Variable i for array index tracking
    while (!addStack.isEmpty()) {
        if (!addStack.pop())
            df[i + 1] = 2 * df[i] + df[0];
        else
            df[i + 1] = 2 * df[i];
        i++;
    }
    return df[i];  // Result is the last element of the df array
}
```

Explanation of Algorithm:

Firstly, an array is determined for the multiplication operation performed in each step, then a stack is used to hold the true and false values. In the a*b operation, the number b is taken and started to be divided by two. Then, if the remainder is 0 when divided by two, true is added to the stack, otherwise false is added and 1 is subtracted to make the number b even and this cycle continues until the number b is equal to 1. Then the number a is set to df[0] and the stack is popped with for loop. If the value in the stack is false, df[i + 1] = 2*df[i] + df[0] is done. If true, df[i + 1] = 2*df[i] is done. By doing this, it goes from bottom to top. Then the last processed element in the array is returned as a result. For example, in this case, the df array of 7*30 is as follows:

[7,21,49,105,210] = 210

The code was coded by **referring** to Tabulation examples on other topics on the internet and the bottom-up computation on slide 10, pages 98 and 136.

Possible Issues:

Inefficiency: According to the articles written on the internet, dynamic programming is not recommended for this problem because dynamic programming does not provide any gain in time or space complexity.

Array Size: The df array is fixed at size 32, which might not be sufficient for all cases. The size should be dynamically determined based on the binary representation of b.

Time Complexity: The time complexity is **O(log b)**, this is because we create a binary representation by dividing b by 2 until it equals 1.

Space Complexity: The space complexity is **O(log b)** because of the space used by addStack. The df array has a fixed size of 32, which is constant. The size of the stack depends on the number of bits of b, because it stores boolean values for whether the remainder after each division of b by two is equal to 0.