2a.

Base case: When n == 0, the function returns 1; when n == 1, the function returns a.

Divide step: Recursively divide n in half. The divide step worst case running time is log2(n).

Combine step: Multiply the result of function call with n/2 as the parameter by two if n is even; Multiply the result of function call with n/2 as the parameter by two and then plus a if n is odd.

Algorithm: Recursively divide n to its half until reaching the base case of n == 0 or n == 1. At each recursion, multiply the next recursion call by two if n is even, or multiply the next recursion call by two and then plus a if n is odd.

Correctness:

I will prove it by induction.

When n == 0, the function returns 1. When n == 1, the function returns a. Assume the function is correct for all n < k. At n == k, if n is even, the result would be function(a, k/2) \* function(a, k/2), which is correct according to the assumption since k/2 < k; if n is odd, the result would be function(a, k/2) \* function(a, k/2) + a, which is correct according to the assumption since k/2 < k and a^n when n == 1 is a. In summary, for all n, the algorithm is correct.

Running Time:

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

According to the tree method, there are log2(n) levels, and each level has one branch. In total, the running time is O(1 \* log2(n)).

4．

1. According to master theorem case 1, f(n) = n = O(n^(log23(37) - e)) for some constant e > 0, T(n) = Θ(𝑛log23(37))