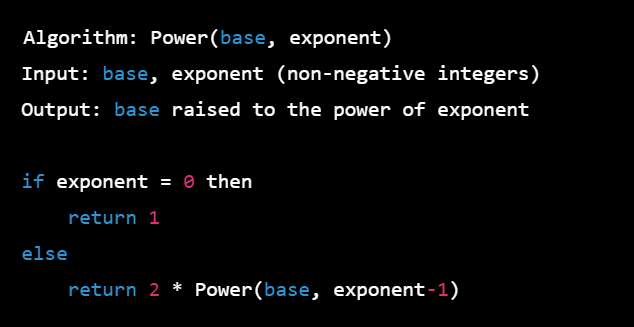
The recursive algorithm for computing 2^n based on the given formula is as follows:



In this algorithm, we make use of the given formula 2^n = 2^(n-1) + 2^(n-1) and recursively compute 2^(n-1) until we reach the base case where n=0 and 2^0 = 1.

To set up the recurrence relation for the number of additions made by the algorithm, let A(n) be the number of additions made by the algorithm to compute 2^n. We can see that for n > 0, the algorithm makes one addition to compute 2^n from the result of 2^(n-1), which is computed recursively. So we have:

Text

Description automatically generated

Solving the recurrence relation:

Text

Description automatically generated

Therefore, the number of additions made by the algorithm to compute 2^n is 2^(n-1) for n > 0.

Drawing the tree of recursive calls for the algorithm:

Shape

Description automatically generated

From the tree, we can see that there are a total of 15 recursive calls made by the algorithm to compute 2^4.

While the recursive algorithm is correct, it is not an efficient algorithm for computing 2^n as it requires a large number of recursive calls, which leads to a large amount of overhead in terms of stack space used by the function calls. A more efficient algorithm for computing 2^n would be the iterative method of repeated squaring, which can compute 2^n in O(log n) time complexity.

Top of Form