Etude 10:Arithmetic

Introduction:

This Etude is focused on taking a given input of numbers, then using '+' and '*' operators on them in a given order of operations ('left to right' or 'BEDMAS') to reach a given value.

Implementation

The original implementation I went with was a program that compiled operations from left to right. However, this proved impossible, as it was implemented using arrays, which got recursively larger as there needed to be 2 arrays (Multiplication and addition) for each prior array(Multiplication and addition). Looking back on it, I could have used Linked lists to resolve the problem.
However, the implementation I ended up using was, based on dynamic programming principles (Namely the factory line problem), was a recursively calling solution that worked from the front of the problem backwards.
If I need to find if [All values of an array after operations] = x, and the final value in the array is 4, then there are 3 possibilities:

*[All values-1] * 4=x |||| Does [All values-1]= x/4?*
[All values-1]+4 =x |||| Does [All values-1]= x-4?
[All values] != x;

To find the answers to the first 2 , the same solution can be applied to [all values-1], replacing x with the modified number, and replacing 4 with the next latest number in the array.

This proved robust for a left to right style.
BEDMAS proved more complicated originally, but I realized that it only affected multiplication:
BEDMAS 1+2+3= 6 LEFT TO RIGHT 1+2+3=6.
This added an extra 2 cases that would be checked if the style of 'N' was used to check the answer.

*[all other values] * ([n-1]*[n])=x/([n-1]*[n])*
Or
[all other values + ([n-1][n]) = x-([n-1]*[n]).*

The break case for the recursion is once the loop reaches the first element in the array. If it is the required value for the prior loops to be true, then the entire recursive tree is true, and the program can work from start to finish, adding whatever operator is relevant to the test.

Issues/Problems Encountered:

The largest issue I found was testing division. As integers were used to calculate required values, odd numbers gave incorrect solutions due to be rounded up. This was resolved by checking every required division value matched a equal calculation cast to double.

Another issue was that originally, the program first added all values together. As only multiplication or addition could be used, I assumed this would make the program more efficient, as all values only added together seemed like the minimum, and if it was above the needed value, it should be impossible.
This turned out to be incorrect for obvious reasons (1*2 is less than 1+2), and was removed as such.

External Resources Used:

https://www.geeksforgeeks.org/assembly-line-scheduling-dp-34/