CMPS 2200 Recitation 03

Name (Team Member 1):__________________________________
Name (Team Member 2):__________________________________

This recitation includes part of assignment 02.

Now that you have some practice solving recurrences, let's work on implementing some algorithms. In lecture, we discussed a divide and conquer algorithm for integer multiplication. This algorithm takes as input two n-bit strings $x = \langle x_L, x_R \rangle$ and $y = \langle y_L, y_R \rangle$ and computes the product xy by using the fact that $xy = 2^{n/2} x_L y_L + 2^{n/2}(x_L y_R + x_R y_L) + x_R y_R$. Use the `main.py` to implement one algorithm for integer multiplication: a divide and conquer algorithm that runs in quadratic time. Please refer to Eqs (15) and (16) https://nbviewer.org/github/allan-tulane/cmps2200-slides/blob/main/module-02-recurrences/recurrences.ipynb

The computation we must implement is:

$

$$
\begin{aligned}
x \cdot y = & \quad (2^{n/2} x_L + x_R)(2^{n/2} y_L + y_R) && (1)\\
= & \quad 2^n (x_L \cdot y_L) + 2^{n/2}(x_L \cdot y_R + x_R \cdot y_L) + (x_R \cdot y_R) && (2)\\
& && (3)
\end{aligned}
$$

$

First, we'll define our entry point, which calls a helper function `_quadratic_multiply`. This returns a BinaryNumber, which we convert to a decimal value for testing purposes:

```
def quadratic_multiply(x, y):
    # this just converts the result from a BinaryNumber to a regular int
    return _quadratic_multiply(x,y).decimal_val
```

We'll also use two helper functions to split the binary vector and convert binary vectors to int are:

```
def binary2int(binary_vec):
    if len(binary_vec) == 0:
        return BinaryNumber(0)
    return BinaryNumber(int(''.join(binary_vec), 2))

def split_number(vec):
    return (binary2int(vec[:len(vec)//2]),
            binary2int(vec[len(vec)//2:]))
```

and here is how we can do a bit shift needed for the 2^n part:

```python
def bit_shift(number, n):
    # append n 0s to this number's binary string
    return binary2int(number.binary_vec + ['0'] * n)
```

The implementation of `_quadratic_multiply` will do the following:

1. Obtain `xvec` and `yvec`, the `binary_vec` values of `x` and `y`
2. Pad `xvec` and `yvec` so they are the same length by adding leading 0s if necessary (e.g., if `xvec`=1 and `yvec`=10, then change `xvec` to 01. This will ensure our splitting and recombining will work properly.
3. Base case: If both x and y are ≤ 1, then just return their product.
4. Otherwise, split `xvec` and `yvec` into two halves each. Call them `x_left` `x_right` `y_left` `y_right`.
5. Now you can apply the formula above directly. Anywhere there is a multiply, call `_quadratic_multiply`
6. Use `bit_shift` to do the 2^n and $2^{n/2}$ multiplications.
7. Finally, you have to do three sums to get the final answer. For this assignment, you can just use the `decimal_val`s of each number to do this, though keep in mind that binary addition is a $O(n)$ operation, assuming n bits per term.