Teresa Van
10149274
CPSC 501 Assignment #4

Assignment #4: Report

Algorithm-Based Optimization:

Optimization done by recreating the program using the FFT algorithm:

Regression Tests:

- After making the changes, I ran my unit tests again to ensure no bugs were introduced. The results were the same.

Before and After Tuning:

```
python3 -m unittest UnitTests.py -v
test_maximum_in_normalization (UnitTests.TestFFTConvolve) ... ok
test_normalization_calculation (UnitTests.TestFFTConvolve) ... ok
test_numpy_array_shapes (UnitTests.TestFFTConvolve) ... ok
----------------------------------------------------------------------
Ran 3 tests in 0.000s

OK
```

Timing Measurements:

	Before Optimization	**After Optimization**
User	1303.32s	191.22S
System	0.13s	0.07s
CPU	99%	99%
Total	21:43.69	3:11.31

Teresa Van
10149274
CPSC 501 Assignment #4

Code tunings:

1. Minimizing work inside loops.

- The first code tuning is putting calculations done inside the loop in the function Normalize into a constant before the loop, so that it doesn't have to compute the result every time it loops.

Before:

```
def Normalize(samples, bitsize):
    ...
    temp = 0
    for i in range(((2**bitsize // 2) - 1)):
        temp += (samples / maximum)
    samples = temp
    return samples
```

After:

```
def Normalize(samples, bitsize):
    ...
    temp = 0
    accumulator = samples / maximum
    for i in range(((2**bitsize // 2) - 1)):
        temp += accumulator
    samples = temp
    return samples
```

Regression Tests:

- After making the changes, I ran my unit tests again to ensure no bugs were introduced. The results were the same.

Before and After Tuning:

```
python3 -m unittest UnitTests.py -v
test_maximum_in_normalization (UnitTests.TestFFTConvolve) ... ok
test_normalization_calculation (UnitTests.TestFFTConvolve) ... ok
test_numpy_array_shapes (UnitTests.TestFFTConvolve) ... ok
----------------------------------------------------------------------
Ran 3 tests in 0.000s

OK
```

Timing Measurements:

	Before Tuning	**After Tuning**
User	191.22S	127.63S
System	0.07s	0.06s
CPU	99%	99%
Total	3:11.31	2:07.77

Teresa Van
10149274
CPSC 501 Assignment #4

2. Unrolling

- Unrolled the loop inside Normalize function since calculations done in a loop could be done in a single line.

Before:

```
def Normalize(samples, bitsize):
    if abs(numpy.amax(samples)) > abs(numpy.amin(samples)):
     maximum = abs(numpy.amax(samples))
    else: maximum = abs(numpy.amin(samples))

    temp = 0
    accumulator = samples / maximum
    for i in range(((2**bitsize // 2) - 1)):
        temp += accumulator

    samples = temp
    return samples
```

After:

```
def Normalize(samples, bitsize):
      if abs(numpy.amax(samples)) > abs(numpy.amin(samples)):
       maximum = abs(numpy.amax(samples))
      else: maximum = abs(numpy.amin(samples))

      samples = (samples / maximum * ((2**bitsize / 2) - 1))
      return samples
```

Regression Tests:

- After making the changes, I ran my unit tests again to ensure no bugs were introduced. The results were the same.

Before and After Tuning:

```
python3 -m unittest UnitTests.py -v
test_maximum_in_normalization (UnitTests.TestFFTConvolve) ... ok
test_normalization_calculation (UnitTests.TestFFTConvolve) ... ok
test_numpy_array_shapes (UnitTests.TestFFTConvolve) ... ok
----------------------------------------------------------------------
Ran 3 tests in 0.000s

OK
```

Timing Measurements:

	Before Tuning	**After Tuning**
User	127.63S	2.10S
System	0.06s	0.12s
CPU	99%	99%
Total	2:07.77	0:02.23

Teresa Van
10149274
CPSC 501 Assignment #4

3. Strength Reduction:

- Instead of using float point, I've converted it to a less expensive type such as an integer
- I've changed it so that I'm using integer division instead of floating point to force the result into an integer for (`2**bitsize // 2)`

Before:

```
def Normalize(samples, bitsize):
        if abs(numpy.amax(samples)) > abs(numpy.amin(samples)):
         maximum = abs(numpy.amax(samples))
        else: maximum = abs(numpy.amin(samples))

        samples = (samples / maximum * ((2**bitsize / 2) - 1))
        return samples
```

After:

```
def Normalize(samples, bitsize):
        if abs(numpy.amax(samples)) > abs(numpy.amin(samples)):
         maximum = abs(numpy.amax(samples))
        else: maximum = abs(numpy.amin(samples))

        samples = (samples / maximum * ((2**bitsize // 2) - 1))
        return samples
```

Regression Tests:

- After making the changes, I ran my unit tests again to ensure no bugs were introduced. The results were the same.

Before and After Tuning:

```
python3 -m unittest UnitTests.py -v
test_maximum_in_normalization (UnitTests.TestFFTConvolve) ... ok
test_normalization_calculation (UnitTests.TestFFTConvolve) ... ok
test_numpy_array_shapes (UnitTests.TestFFTConvolve) ... ok
----------------------------------------------------------------------
Ran 3 tests in 0.000s

OK
```

Timing Measurements:

	Before Tuning	**After Tuning**
User	2.10S	2.09S
System	0.12s	0.10s
CPU	99%	99%
Total	0:02.23	00:02.20

Teresa Van
10149274
CPSC 501 Assignment #4

4. Initialize at compile time:

- I've used constants wherever it was possible. Here I've changed it so that the number 16 is a constant (bitsize)

Before:

```
class FFTConvolve():
    def Main(dryRecording, impulseResponse, outputFile):
      bitsize=16
      ...
```

After:

```
bitsize = 16
class FFTConvolve():
    def Main(dryRecording, impulseResponse, outputFile):
      ...
```

Regression Tests:

- After making the changes, I ran my unit tests again to ensure no bugs were introduced. The results were the same.

Before and After Tuning:

```
python3 -m unittest UnitTests.py -v
test_maximum_in_normalization (UnitTests.TestFFTConvolve) ... ok
test_normalization_calculation (UnitTests.TestFFTConvolve) ... ok
test_numpy_array_shapes (UnitTests.TestFFTConvolve) ... ok
----------------------------------------------------------------------
Ran 3 tests in 0.000s

OK
```

Timing Measurements:

	Before Tuning	**After Tuning**
User	2.09S	2.02s
System	0.10s	0.10s
CPU	99%	99%
Total	00:02.20	00:02.12

Teresa Van
10149274
CPSC 501 Assignment #4

5. Exploit Algebraic Identities:

- Here, I've replaced the expensive if statement with a less expensive alternative of using python's built in max function to take the maximum of two values.

Before:

```
if abs(numpy.amax(samples)) > abs(numpy.amin(samples)):
    maximum = abs(numpy.amax(samples))
else: maximum = abs(numpy.amin(samples))
```

After:

```
maximum = max(abs(numpy.amax(samples)), abs(numpy.amin(samples)))
```

Regression Tests:

- After making the changes, I ran my unit tests again to ensure no bugs were introduced. The results were the same.

Before and After Tuning:

```
python3 -m unittest UnitTests.py -v
test_maximum_in_normalization (UnitTests.TestFFTConvolve) ... ok
test_normalization_calculation (UnitTests.TestFFTConvolve) ... ok
test_numpy_array_shapes (UnitTests.TestFFTConvolve) ... ok
----------------------------------------------------------------------
Ran 3 tests in 0.000s

OK
```

Timing Measurements:

	Before Tuning	**After Tuning**
User	2.02s	1.58s
System	0.10s	0.07s
CPU	99%	99%
Total	00:02.12	00:01.65

Teresa Van
10149274
CPSC 501 Assignment #4

Compiler Level Optimization:

Ran using:
`python3 -OO convolve.py guitar.wav l960large_brite_hall.wav output.wav`
Using -O when running python3 optimizes generated bytecode slightly. -OO removes doc strings in addition to the -O optimizations

Regression Tests:

- After making the changes, I ran my unit tests again to ensure no bugs were introduced.
The results were the same. (Although compiler level optimizations are not likely to introduce bugs)

Before and After Tuning:

```
python3 -m unittest UnitTests.py -v
test_maximum_in_normalization (UnitTests.TestFFTConvolve) ... ok
test_normalization_calculation (UnitTests.TestFFTConvolve) ... ok
test_numpy_array_shapes (UnitTests.TestFFTConvolve) ... ok
----------------------------------------------------------------------
Ran 3 tests in 0.000s

OK
```

Timing Measurements:

	Before Optimizing	**After Optimizing**
User	1.58s	1.52s
System	0.07s	0.05s
CPU	99%	99%
Total	00:01.65	00:01.57