The main idea of this programming assignment is that the bloom filter will hold strings. When we add a string, its reference to indices will be set to 1. If we add it again, it will increase to 2. Since different strings can map to the same index, returning the indices for an string will yield different values (2, 3, 7). We will average them to get an estimate.

Problem:

BloomFilter Class

table ← empty list of numbers

<u>createTable algorithm:</u>

```
    Creates new BloomFilter
    Input: size - the size of the table
    Output: none

    table ← new list of bits of size size
    for each element in table
        element ← 0
    end for
```

Runtime is O(n) where n is the size of the table

<u>Add/Increment algorithm:</u>

```
    Adds string to BloomFilter
    Input: o
    Output: none

    firstIndex ← h1(o)
    secondIndex ← h2(o)
    thirdIndex ← h3(o)
    table[firstIndex] ← table[firstIndex] + 1
    table[secondIndex]← table[secondIndex] + 1
    table[thirdIndex] ← 1table[thirdIndex] + 1
```

Runtime is O(n) where n is the length of the string. This is due to the call of h3, which has the same runtime

<u>h1 algorithm:</u>

```
Hashes string
Input: string
Output: an index in table

return (hash value of string) % size of table
```

Constant runtime

<u>h2 algorithm:</u>

```
Hashes string
Input: string
Output: an index in table

temp ← string
temp ← temp but in reverse order
return (hash value of temp) % size of table
```

Constant runtime

<u>h3 algorithm:</u>

```
Hashes string
Input: string
Output: an index in table

temp ← string but in reverse order
Combination  ← empty list

for i in temp.length*2
    if i % 2 == 0
        combination ← ith letter of string
    else
        combination ← ith letter of temp
    end if
end for

return (hash value of combination) % size of table
```

Runtime is O(n) where n is the length of the string

<u>Count Algorithm:</u>

```
    Return frequency of string
    Input: o - string
    Output: frequency of string

    firstIndex ← h1(o)
    secondIndex ← h2(o)
    thirdIndex ← h3(o)
    If firstIndex or secondIndex or thirdIndex == 0
        return 0
    End if
    Return average of firstIndex, secondIndex, thirdIndex
```

Constant runtime

<u>New algorithm:</u>

```
    Clear table
    Input: none
    Output: none

    for each element in table
        element ← 0
    end for
```

Runtime is O(n) where n is the size of the table

Example:

BloomFilter filter = new BloomFilter(10);

filter.add(“192.12.235.36”) - adds string

return filter.count(“192.12.235.36”) - returns the average of 1, 1 and 1 = 1.

filter.new() - clears table and sets everything to 0