Explanation of Algorithm:

**embedIntoImage:**

A PrimeIterator is created using (2\*length of message\*Log(length of message)) which gets the amount that is needed to generate the amount of primes needed to be used.

When the PrimeIterator is created it makes a Boolean array the size of the message length+1, assigns the currentPrime to 0, and makes all Boolean values from 2 to length of message true.

A character array is created using the characters in the message (after assigning them to their uppercase values.

A loop is entered as long as PrimeIterator has a prime left and the number of characters hidden in the image is less than the length of the message.

To check if the PrimeIterator has a prime left it checks whether anything in the array past the Current Prime being used + 1 is true. If it finds a true value, it returns true.

A prime number that will be used to find the y and the x of the image that is needed to be changed is found.

PrimeIterator finds this by taking the current prime being used and finding the first value that is true after the current prime. (ex: if current prime is 2 the next prime will be 3 and that index will be true). After it finds the next prime it runs through a loop which assigned each index of Next Prime\*Number Incrementing to message length and assigns all those as false. Then returns the next prime

Y is found by taking the prime and dividing it by the width of the message.

X is found by taking the prime and subtracting Y\*the width of the message.

The pixel is color is grabbed using X and Y

The character to be put into the color is found by taking the array and using the characters already hidden in the message as the index, representing that character to as integer and subtracting 32 from it.

The change to red is calculated as (Pixel Red Color & Bit-mask(11111100))+(Character to be Put in Pixel & Bitmask(110000) Bit-Shifted Right by 4. The same with green except for the second part after the “+” is (Character to be Put in Pixel & Bitmask(001100) Bit-Shifted Right by 2. The same with blue except for the second part after the “+” is (Character to be put in Pixel & Bit-mask(000011).

Create a color using the value changes of red, green, and blue.

Assign the color to the pixel at X and Y.

**retrieveFromImage:**

A PrimeIterator is created using (width of the picture \* height of the picture) which gets the amount that is needed to generate the amount of primes needed to be used.

When the PrimeIterator is created it makes a Boolean array the size of the message length+1, assigns the currentPrime to 0, and makes all Boolean values from 2 to length of message true.

A loop is started that goes for as long as there is a prime to still be used.

To check if the PrimeIterator has a prime left it checks whether anything in the array past the Current Prime being used + 1 is true. If it finds a true value, it returns true.

A prime number that will be used to find the y and the x of the image that is needed to be changed is found.

PrimeIterator finds this by taking the current prime being used and finding the first value that is true after the current prime. (ex: if current prime is 2 the next prime will be 3 and that index will be true). After it finds the next prime it runs through a loop which assigned each index of Next Prime\*Number Incrementing to message length and assigns all those as false. Then returns the next prime

Y is found by taking the prime and dividing it by the width of the message.

X is found by taking the prime and subtracting Y\*the width of the message.

The pixel is color is grabbed using X and Y.

The hidden red character is found by taking the Red of the Pixel and bit-masking it with (000011) so to only get the first 2 bits of the integer. The same is done for green and blue.

The red is then bit-shifted to the left by 4. The same is done with the green except it is only bit-shifted to the left by 2;

The values of red, green, and blue are added together and then a value of 32 is added to them to get the character representation of the integer. The character is then concatenated to a string.

At the end of the loop the string created is returned.

Help from Students:

* Drew Hurdle: Informed me of nlogn to get the number needed to reach the primes I need. Before that point I was using width of picture \* height of picture which was really inefficient.
* Drew Hurdle also helped me with some problems in the bit-masking and bit-shifting in regards to bit order of operations.
* David Ericson: Told me how he decided to instead of calculate all the primes when Prime Iterator is created, to only calculate primes as needed when the next() function is called. This saves on time and any overhead previously used when all the primes are found at the beginning because sometimes not all the primes are used. Before this I was calculating all the primes at the beginning.

Mathematical Analysis:

messageLength<=((Width\*height)/Log(width\*height))/2

O(n^messageLength\*((messageLength(messageLength+lastPrime)/2)+(messageLength\*Log(MessageLength)\*( messageLength\*Log(MessageLength)+lastPrime)/2

O(n^messageLength)

n^messageLength=((width\*height)/Log(width\*height))/2

((width\*height)/Log(width\*height))/2<width\*height

(width\*height)/Log(width\*height)<width\*height

(n^messageLength<width\*height)

O(width\*height(width\*height+lastPrime)/2)+( width\*height\*Log(width\*height)\*( width\*height \*Log(width\*height)+lastPrime)/2

O(width\*height(width\*height+lastPrime)/2)

O((2\*(width\*height))/2)

O(width\*height==width\*height)

O(width\*height)

Empirical Analysis:

My approach was to find out how long it takes to write the message into the picture, and to find how long it takes to read in the message from the picture. Keeping track of the time it takes to write will tell me if it gets slower the larger the message is or if gets increasingly faster. Keeping track of the time it takes to read will tell me how much of a difference between Widths and Heights of pictures.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Width** | **Height** | **Length of Message** | **Time to Write** | **Time to Read** | **Total Time** |
| 100 | 100 | 50 | 3.873ms | 25.244ms | 29.117ms |
| 100 | 100 | 100 | 5.023ms | 24.471ms | 29.494ms |
| 100 | 100 | 300 | 9.116ms | 22.897ms | 32.013ms |
| 200 | 200 | 50 | 4.307ms | 26.475ms | 30.782ms |
| 200 | 200 | 100 | 5.291ms | 25.987ms | 31.278ms |
| 200 | 200 | 300 | 9.438ms | 21.871ms | 31.309ms |
| 300 | 300 | 50 | 3.920ms | 25.683ms | 29.603ms |
| 300 | 300 | 100 | 4.93ms | 26.086ms | 31.016ms |
| 300 | 300 | 300 | 9.314ms | 22.806ms | 32.12ms |