**Algorithm Explanation**

* Embedding a message into an image
* *Input*
  + *A clean image to embed the message into*
  + *A string of characters, message, to be embedded into the image*
* *Output*
  + *An image with a secret message embedded into its prime indexed pixels*

1. Store the height and width of the image in pixels
2. Convert the message to all capital letters
3. Initialize a list of prime numbers up to a max number, the max number being the pixel height multiplied by the pixel width, minus one
   1. Initialize a list of all numbers from 2 to the max number - this is a list of the candidates that could potentially be prime
   2. A Number i is set to a value of 2
   3. If the number in the candidate list at index i is 0, go to step 3.8, else go to step 3.4
   4. A number j is set to i squared
   5. If j is less than or equal to the max number, move to step 3.6, else move to step 3.8
   6. The number in the candidate list at index j is set to 0, indicating that it has been eliminated as a candidate
   7. Increment j by i and move to step 3.5
   8. Increment i by 1. If i is now less than or equal to the square root of the max number rounded down, then move to step 3.3, else, move to step 3.9
   9. Iterate through each number in the candidates list, and if it is not equal to 0, add it to the list of primes.
4. Iterate through each number in the list of prime numbers for the length of the message, iterating through each character in the message as well
   1. Take the ascii decimal value of the current message character
   2. Subtract 32 from the ascii value
   3. Convert this decimal value to a binary string of length 6, filling any extra slots to the left of the binary value with 0’s
   4. Retrieve the color at the pixel whose index is the current prime number
      1. Calculate the coordinates of the pixel by modding the index by the image width to get the x position, and by dividing the index by the width to get the y position.
   5. Get the individual red, green, and blue values from the pixel’s color as decimal values from 0 to 255.
   6. Convert the r, g, and b values into their respective binary equivalents, stored as a string.
   7. Take the last two values in each of these binary strings, replacing them with the first, second, and third pair of binary characters from the message character’s binary string.
   8. Replace the currently indexed pixel’s colors with a new color using the altered r g and b values.
   9. If this was the last character of the message, move to step 5, else iterate to the next prime number and message character, and go back to step 4.1
5. Return the newly altered image.

* Retrieving an embedded message from an image
* *Input*
  + *An image with a secret message imbedded within its prime indexed pixels*
* *Output*
  + *A message retrieved from the image*

1. Store the height and width of the image in pixels
2. Initialize a list of prime numbers up to a max number, the max number being the pixel height multiplied by the pixel width, minus one
   1. Initialize a list of all numbers from 2 to the max number - this is a list of the candidates that could potentially be prime
   2. A Number i is set to a value of 2
   3. If the number in the candidate list at index i is 0, go to step 2.8, else go to step 2.4
   4. A number j is set to i squared
   5. If j is less than or equal to the max number, move to step 3.6, else move to step 2.8
   6. The number in the candidate list at index j is set to 0, indicating that it has been eliminated as a candidate
   7. Increment j by i and move to step 2.5
   8. Increment i by 1. If i is now less than or equal to the square root of the max number rounded down, then move to step 2.3, else, move to step 2.9
   9. Iterate through each number in the candidates list, and if it is not equal to 0, add it to the list of primes.
3. Initialize an empty string to store message as we build it
4. Iterate through each number in the list of primes
   1. Retrieve the color at the pixel whose index is the current prime number
      1. Calculate the coordinates of the pixel by modding the index by the image width to get the x position, and by dividing the index by the width to get the y position.
   2. Get the individual red, green, and blue values from the pixel’s color as decimal values from 0 to 255.
   3. Convert the r, g, and b values into their respective binary equivalents, stored as a string.
   4. Take the last two characters of the binary strings and concatenate all 6 into one string
   5. Convert this binary string into decimal form, then add 32 to it.
   6. Convert this to the corresponding ascii character
   7. Add this character to the end of the full message string
   8. If this was the last prime number in the list of primes, move onto step 5. Else, take the next prime number and go back to step 4.1
5. Return the message string

* **Outside help**
* The only help I received from outside sources was asking for clarification on the PrimeSuspects / Steganog class name, to which the student answered that they were pretty sure they were both referencing the same class and we could probably use either one.
* **Performance**
* If I were to disregard the performance of the Sieve technique used to generate the list of prime numbers, I could then focus solely on my actual algorithm to find its performance. Additionally, because I don’t know the efficiency of the many smaller operations used throughout the program, for instance convenience and conversion methods used for things like string substringing and concatenation, I will ignore those factors in my evaluation. Finally, the performance would vary based on whether or not the program is being run to embed a message, or to retrieve a message.
  + Embedding a message – Algorithm performance is focused on the message being encoded
    - Because the length of the message is known, the main loop that causes the algorithm’s basic operation only runs once per every character in the message, up to the number of prime indexed pixels contained within the image.
    - T(n) = n, where n is the number of characters in the message, and n < (width\*height)/(log(width\*height)-1)
    - T(n) O(n)
  + Retrieving a message – Algorithm performance is focused on the image being decoded
    - In this case, because the length of the embedded message is not known, the algorithm’s loop runs once for every prime indexed pixel within the image.
    - The number of prime numbers under the image’s width\*height (width \* height equaling the total number of pixels in the image, and therefore the maximum prime index) can be approximately calculated with x/(logx-1), where x = (width\*height).
    - T(n) = n/(logn-1), where n is the number of pixels the image contains, or image width\*height
    - Or T(n) = (1/n)(logn-1)
    - T(n) O(logn)
* **Empirical performance analysis**
* I did two separate analyses, one for embedding a message into the image, and one for retrieving the message.
* To setup the message embedding tests, I used a single message of 50 characters, and different images of a solid grey background with a constant height of 100 pixels and a varying width. I recorded the time in milliseconds before the program ran the embedding process, then calculated the time elapsed when it was done.
* The setup for the message retrieval tests was almost identical, except that I used those images I embedded with messages to then test the message retrieval.
* Message embedding

|  |  |  |  |
| --- | --- | --- | --- |
| * Image Width | * Image Height | * Message Length | * Time (milliseconds) |
| * 100 | * 100 | * 50 | * 4 |
| * 200 | * 100 | * 50 | * 3 |
| * 300 | * 100 | * 50 | * 2 |
| * 400 | * 100 | * 50 | * 3 |
| * 500 | * 100 | * 50 | * 2 |
| * 600 | * 100 | * 50 | * 2 |
| * 700 | * 100 | * 50 | * 2 |
| * 800 | * 100 | * 50 | * 3 |
| * 900 | * 100 | * 50 | * 2 |
| * 1000 | * 100 | * 50 | * 2 |

* Message retrieval

|  |  |  |  |
| --- | --- | --- | --- |
| * Image Width | * Image Height | * Message Length | * Time (milliseconds) |
| * 100 | * 100 | * 50 | * 16 |
| * 200 | * 100 | * 50 | * 16 |
| * 300 | * 100 | * 50 | * 19 |
| * 400 | * 100 | * 50 | * 19 |
| * 500 | * 100 | * 50 | * 27 |
| * 600 | * 100 | * 50 | * 20 |
| * 700 | * 100 | * 50 | * 38 |
| * 800 | * 100 | * 50 | * 34 |
| * 900 | * 100 | * 50 | * 29 |
| * 1000 | * 100 | * 50 | * 56 |

* **The original secret message**
* WE THE PEOPLE OF THE UNITED STATES OF AMERICA IN ORDER TO FORM A MORE PERFECT UNION, ESTABLISH JUSTICE, INSURE DOMESTIC TRANQUILITY, PROVIDE FOR THE COMMON DEFENSE, PROMOTE THE GENERAL WELFARE, AND SECURE THE BLESSINGS OF LIBERTY TO OURSELVES AND OUR POSTERITY, DO ORDAIN AND ESTABLISH THIS CONSITITUTION FOR THE UNITED STATES OF AMERICA.