The algorithm I wrote is able to compress the absolute duck out of an image (at least according to the benchmarks) but takes about a year (read as: 30 seconds to maybe 3 minutes your mileage may vary). However, it is worth mentioning that the compression is lossless. Currently in order to decompress an image (or any image similar to it like the sub-image that was discussed in the requirements) you must first compress the uncompressed version. This still works for the sub-image requirement that the project described.

The algorithm essentially goes through each color value that exists for each pixel, tallies up how many times they occur, and applies the huffman algorithm to the results. In other words, the color that appears the most will have the lowest number of bits assigned to it. After that, I go through each pixel, take its color value, and write the bits (BITS) to a byte array. If the color of one pixel is the same as the previous one, a single 0 bit is written instead. Finally, the byte array itself is written to the output file and saved.

For decompression, the bytes are read in with their leading zeros and then iterated through to find the colors that have already been stored in our dictionary. This process is comparatively quick.

For compression the time complexity is the complexity of the huffman algorithm + storing items in the dictionary twice for each color which comes out to O(nlogn + 2n) where n is the number of unique colors in the pixel.