Prajwal Pokharel
UT EID : pp23366
Project 2
EE 360C

1. The first method that I implemented was constructing Huffman tree. While implementing this method first I created an arraylist of tree nodes. I iterated through string literals and assigned each tree nodes value to be equal to the value of string literals, and I set the frequency of the tree node to be the frequency of each string literal. This took O(n) time. Then I created a priority queue for tree nodes using comparator which took O(nlogn) time. After that I created a Huffman tree with the help of priority queue using recursion, and this took O(nlogn) time.
   Finally, I populated my hashmap using string literal as key, and Huffman code as my value which I found by traversing Huffman tree. This took O(nlogn) time as well. Overall my time complexity for this method was O(nlogn).

2. The second method that I implemented was encode. In this method, I traversed through each string literal in the message, and I looked for its corresponding value in the hashmap. After finding the value I concatenated all the values. Looking value in the hashmap took constant time, concatenating took constant time, and traversing through the message took O(n) time. Thus, my overall time complexity for this method was O(n) time.

3. The third method that I implemented was decode. In this method, I traversed through the message, and while traversing depending upon the bits I got I went left or right in the Huffman tree starting from the root of the Huffman tree which took constant time. When I found a leaf, I read the value and concatenated that to returning decoded message, and I started from root again to get another string literal. Since I traversed through the binary code once, my overall running time for this algorithm was O(n).

4. We are using Huffman tree which is a binary tree to encode the given message. All the letters in the message appear as leaves in Huffman tree. For any letter x in the message, in order to encode we follow the path that goes from node to the leaf which is labeled x. While traversing if the path goes from node to left child we write 0 and if the path goes to right child we write 1. Finally, the resulting string bits gives the encoding for x. Now, the proof which shows that my encoding is prefix free is as follows:

   Proof by Contradiction:
   Suppose letter x has an encoding which is a prefix of the encoding of letter y in my Huffman Tree. Then the path from root to x is the prefix of the path from root to y. This also means that x lies in the path from root to y. This is a contradiction since this shows that x would not be a leaf in my Huffman tree, and as I mentioned before all the letters appear as leaves in the Huffman Tree. Thus, this shows that my encoding is prefix free.