Andy Ratsirarson – Lab 3 Analysis

Introduction
This is an analysis of Lab3, writing a program that encode/decode message using HuffmanEncoding. The results are printed on screen and in an output file.

Design:
The main data structures used are Min Heap as priority queue implemented as an array and Binary tree implemented as Linked List. Since the algorithm requires a lot of enqueue/dequeue without the need of it to be partially ordered, Min Heap is a perfect candidate as a priority queue. It gives O(logn) for both dequeue and enqueue. As far as the Huffman Tree, binary tree is the natural representation of the structure. It gives a Theta(logn) to go down to the leaf in average.

Performance:
Unfortunately, based on the frequency table provided, there was no gain in compression. In fact, the original message was smaller in size than the encoded message. However, when I changed the frequency table based on Adele's song Hello, then encoded partial lyrics in her other albums. I saw a 25% gain of size in average.

What can be improved?
There are a lot of improvements that can be done in this program. For instance, for code lookup (from the table built from the frequency table), although I am retrieving the corresponding code for a character at O(1), I allocated unnecessary space to hold the Code table because I am using ASCII decimal representation of letters as index. This could have been done better if I used a proper dictionary data structure which would have given me O(1) for retrieval and O(n) memory allocation.
Also, code structure could have been better handled, such as allowing user to input the type of algorithm to encoding/decoding the message.

Conclusion
Huffman Encoding is a good encoding and compression algorithm but very dependent on the frequency table provided and the message to be encoded in terms of character frequency used. If the message to encode is fairly uniformly distributed, ASCII will be a better candidate as ASCII encoding is based on 8 bits uniform across all character. Changing the procedure for handling break ties obviously changes the encoding result but surprisingly with little impact. I accidentally reversed the break ties procedure by the bigger node to be on the left and the lesser on the right, and it made only a difference in few characters.
Through this programming assignment, I have read quite a lot about Encryption/encoding algorithm, and was able to learn much. And it also allowed me to use multiple data structures appropriate to the task and thinking about tradeoffs. For instance, when picking the structure to lookup the code based on the character provided, I had to decide between fast lookup vs memory consumption.