Farid Rajabi Nia fr2md

Prelab:
I am using priority queue for creating the tree using heap structure. The heap was given and I just changed it to accommodate node pointers. The way that making tree works is that first I find out the frequency of each letter from the input file. We are only concerned with the ascii values between 32 and 127. Based on the frequency of each letter stored in an array, they have been prioritized on the queue. By repeating the process of extracting the least frequent from the heap one at a time, attaching them to a carrier node which is called T in the slides, I have created the Hoffman encoder tree. Following lab 2 link list, I have created two different classes hoffNode and hoffTree. In hoffNode class, I am creating a node with four parameters.

Time complexity:
For prelab, I have while loop to get input from the file. Its running time is linear. I have used heap structure with worst case running time of log n, array with linear running time, creating a Hoffman tree worst case running time of linear. However, I have a while loop and I am traversing on my heap to find the symbols and that is my worst case scenario of running time which is close to n^2.

Space Complexity:
For the space complexity, I should consider the number of nodes created in each structure which is my heap and my Hoffman tree. The worst case space complexity is the maximum number of nodes is n/2. My each node contains one character, one integer, one string and two pointers. (1+4+8+8+8=29 bits for each node). In total, I have an array with the max input size of 95 integers. I have a tree with at most 95 leaves.

Inlab:
Decoding it was rather easier. I used my node class and a helper class called treeBuilder to backtrack what I did in the encoding section of the lab. Following the instructions, to go left and right in cases of encountering 0 and 1, I have recreated another Hoffman tree. Based on the tree, and the assigned characters read from the input file, I have determined the frequency of each letter, starting from the most frequent one. I am not a big fan of recursive functions since they are risky in terms of run time efficiency. With the worst case running time of exponential. In the provided code for inlab, the cin functions parse the characters by space so it did not recognize space as a character. It messed my code up but I figure out a nice trick to take care of that. However, it took me almost two hours to figure it out and fix it and I submitted my lab 1 hour and 13 minutes late.

Time complexity:
For inlab, I am using a while loop to read the file and create a tree. The running time of the while loop with the recursive function to create a tree. The running time of my while loop is linear but the worst case for my recursion might be exponential. But in this case, the number of recursion is the same as the number of unique characters. So the total run time of that while is linear

(number of input from the screen). I have two more loops with linear complexity at the end of the file. So the total runtime should be linear at worst case.

Space complexity:
My node size is still the same and I am creating a tree again in the lab so my worse case of maximum number of nodes is the same as above. (max number of nodes is in my tree n/2. My each node contains one character, one integer, one string and two pointers. (1+4+8+8+8=29 bits for each node)). I have an array of integers in this section as well.