Compress: First, it parses through the codebook that was given and creates a linked list with each token and their code. It then parses through the file and checks every token at a time. When it looks for each token in the file, it actually returns the token along with the delimiter after it. So if the token was “button” and there was a tab after it, it would return “button\t” (“\t” and “\n” would be two characters, not the single value of what it represents). It then extracts the delimiter after the fact and puts both of them in the new .hcz file. It does this by going through the linked list of the codebook and if it finds a match with the token then it writes the code into the .hcz file. This goes on until it reaches the end of the file.
The space complexity for compress is O(n), with n being the amount of tokens.
The time complexity for compress is O(n+m), with n being the number of unique tokens, and m being the total number of tokens in the file.

Decompress: First, it parses through the codebook that was given and creates a linked list with each token and their code. It then goes through the .hcz file one character at a time. It writes the new character at the end of a number that is building up with every character. It continuously checks if there is a match with any code in the linked list of the codebook, and if it finds a match then it writes the token into the new file and the number is reset to nothing, starting again with the next character. It does this until it reaches the end of the file.
The space complexity for decompress is O(n), with n being the number of tokens in the codebook.
The time complexity for decompress is O(m*n), with n being the number of unique tokens in the file, and m being the amount of characters in the file.

Build: First, it goes through a file and creates a linked list that has every token and how many times it occurs within the file (or files if it is recursive). It then uses the Huffman algorithm to find the codes for every token and creates an AVL tree with all of the

tokens and their code. It then recursively goes through the AVL tree and writes every token with their code into the HuffmanCodebook file one line at a time.

The time complexity of Build is O(n*m + nlogn), with n being the amount of unique tokens and m being the total amount of tokens.

The space complexity of Build is also O(n), with n being the number of unique tokens.