***** Explanation_3******

- The Hufmann problem used class Node with attributes frequency, char, left, right.
- the implementation keep a dictionary of occurrence of each character.
- for key in dict(characters) the code uses min heap data structure to designated appropriate nodes to the characters.
- the root nodes are popped from the min heap and merged(add the freq and create new arbitrary node) till the length of heap is 1
- This 1 node object is a encoded tree.
- this data is then passed to the decoding function to decode the given data.

Methods used:
—> count_occurence(): Counts the frequency of keys and create a dictionary[key=char] and value = freq. Time complexity is O(n)
—> create_min_heap(): give key and value create min heap by pushing the node in heap. Time complexity is O(long)
—> merge_low_freq_nodes(): Pops head and assign left and right node. Time complexity is O(1)
—> code_key(): assigns code to the character. since I look through all the node in tree Time complexity is O(n)

Time Complexity:
The time complexity is O(nlogn)
Time complexity for huffman_ecoding method —> nlogn
Time complexity for huffman_decoding method —> O(l) where l is the length of code.

Space complexity:
O(number of characters in the sentence)