a) **Approach &Pseudocode**: Let the length of the input string, str, be n. Let maxwordlength be the length os maxword in the dictionary.

   I. First, I constructed a ArrayList<String> of the dictionary words.

   II. Then I created a DPtable[n][maxwordlength]. The (i,j)th entry in this table will be a 1 if str(i:i+j+1) is a word in the dictionary

   III. Then I recursively called getsentences(str,i) on str. This function looks for words starting from the ith position in the str using the ith row in the DP table. Once it finds a word, it calls getsentences(str-str(i+j+1)). This function keeps track of the words in the sentence prior to i. It prints all the words once it reaches the end of the str

b) **Time Complexity**: The worst case time complexity is when the dictionary contains all possible words in the string i.e 2^n words. The complexity of lookup for ArrayList is $O(2^n)$. All the elements in the DPtable are 1 and maxwordsize=n. In this case, time to fill the DP table is $O(n*n*2^n)$. Each row in the table indicates that getsentences will be called O(n) times and this is same for all rows. Number of times u end up at start of any row is max of n*n. So, the number of times getsentences is called is O(n*n* n) and lookup in each case is $O(2^n)$. So the total complexity is $O(n^3 2^n)$. **Its exponential in worst case**.

c) **Correctness**: The algorithm looks for all possible combinations of words. It starts at the beginning and looks for words upto size maxwordlength and calls the recursive function on rest of the string. I only prints sentences when the end of the str is reached. If the last remaining letters in the str don't make a word, then the algo doesn't print that sentence

d) Loop Invariant: The algorithm finds breaks in the input str such that the substring before the break and substring after the break make meaningful sentences. So, the loop invariant is substring[0,break-1] + substring[break,end] make a sentence.