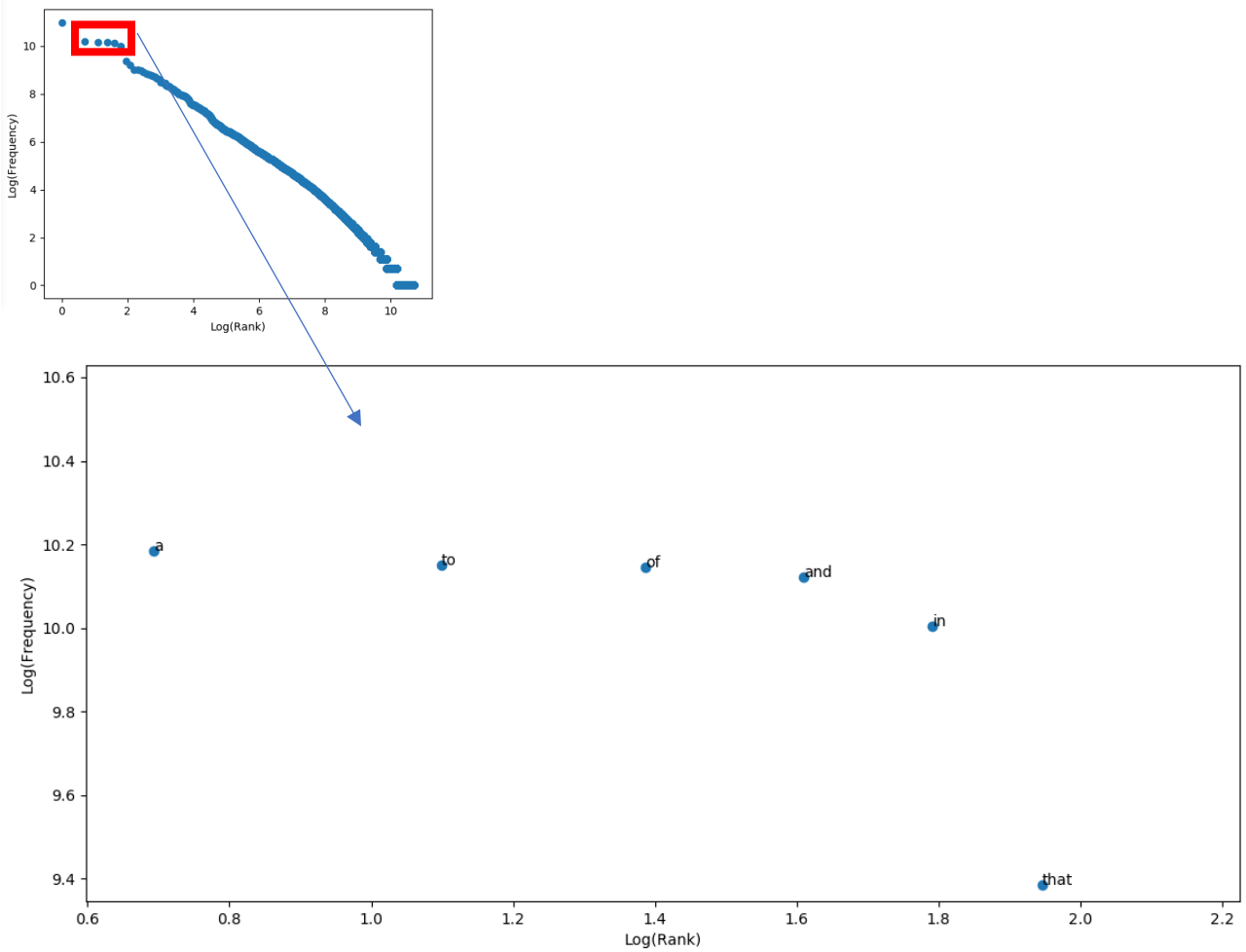
Question 1

a. Word can be defined as a string of characters. Counting word as a stream of characters would not be a good option, in my opinion. We should do text preprocessing. Lets call the space delimited string as a token. The following techniques can be applied during preprocessing-
    1. Case folding. Each word should be converted in the same case. With this we might lose sense of difference in "**US'** and "**us**" but will give us the right count of occurrences of words like "**the**" which can appear in the beginning of the sentence.
    2. Remove punctuations. Special characters like hash, period, quotation mark, braces, brackets, comma and exclamation at the end or the beginning of the token should be removed. Example "**'he said'**" and **",he said**" should be treated as two words **he** and **said** only. And tweets like "**#BUCK**" and "**buck**" should be counted as same word "**buck**".
    3. Advanced level processing can be to remove apostrophe that appears in the second last position from the end of the token and expand it as two separate tokens. This is a subjective matter of choice whether this type of preprocessing is appropriate or not. We can process some of the common contractions example "**can't**," can be counted as two words "can" and "not" and "**I'm**" can be counted as "I' and "am". The token terminating with **<string>n't** can be formed as a rule to split as string+not.
    4. Tokens which have colons can be split in two words. Example "**example:a, b, c**" should be split as "example" and "a".
    5. A highly advanced tokenizer for English text can consider lemmatization as well. Example changing "**singing**" to "sing" but leaving "**speaker**" intact. We can also consider incorporating porter stemming algorithm but this will be adverse for cases where we wanted to count the words "speak" and "speaker" as two separate words.

b.

ZipfLaw.py

c. On running the code, following is the plot –



This plot is almost linear and hence conforms to the Zipf's Law.
The code handles both csv and txt files.
For the csv file, the requirement is there must be 2 columns rank and frequency and the first line must be a header.
The tokens(space delimited strings) are first converted to lower case.
Punctuations and other special characters are pruned from the token.
A common contraction is handled. Example can't => can + not, don't => do + not
Whereas contractions like ain't will be split to ai+not . although this not grammatically correct but the tradeoff of advantage is evident in more frequent words like don't and can't.

Question 2.

a.

```
current = S
for (idx, obs) in enumerate(input):
  next = set()
  for s in current:
    if s in M and obs in M[s]:
      next = next.union(M[s][obs])
  current = next
```

Proving correctness –
After each iteration, we get closer to the goal. The goal is to determine if the input string is a valid string recognized by the FSA. Starting with a finite set of starting states. we scan each input character and transition from the current state to the next state if the current state has a possible/ valid transition rule for the scanned input character. This new set of reachable states is saved in the set named next. Before scanning the next character, the new states in the next are copied in current and looped in a similar fashion. thus, we keep moving forward in the FSA using the mappings between current state, input character and the next state.

Loop invariant –
set of states in current, that is , the set of states we can possibly reach after scanning i characters of the input. Current always contains the states of the FSA reached after scanning i character of the input provided we begin with possible initial states of the FSA.
Before any character of the input is scanned, that is i=0, the current has the initial set of the starting states possible for the given FSA.
After scanning i^{th} input character, the current contains set of states reachable from the possible set of states that were there in current after processing input till $(i-1)^{th}$ character.
Thus after scanning the last character of the input, the set of states in the current will be last final set of states reachable from the given input in the given FSA. After the last character of the input has been scanned, the set of states in the current won't change.
Now if the final set of states in the current contains atleast of one of the states recognized as final state by the FSA, we will determine that the input is recognized by the FSA.

Question 3.
$F(n) = 3*n^2 + 5$
Order of the function is $O(n^2)$.
Proof:
Let $g(n) = n^2$
$F(n) = O(g(n))$
there exists N and C such that for all n>N , $|f(n)| <= C|g(n)|$
if we take **N = 4 and C = 4,**
for n = 4,
$f(n) = 3*(4)^2+5 = 3*16+5 = 48+5 = 53$
$g(n) = (4)^2 = 16$
$Cg(n) = 4*16 = 64$
Thus we can clearly see,
$|f(n)| <= C|g(n)|$ would satisfy for C=4 and all N>4.

Question 4.
Complexity of the code:
Max(O(n), O(m*p))
Where
n => length of the input file
m=> length of the input string
p=>length of the current set with maximum number of states or the maximum number of states reachable from a given state given a particular input character in the FSA

while processing the input, for each character, we determine the set of possible states we can transition to from the given current state and the input observation. Thus complexity of the code depends on length of the input times the maximum length of the current set, that is , case for the input character i and the current state set(states we can be at after reading input till i-1 characters), the maximum length of the next possible reachable states.