This is questions , You have to Solve in python.

Project Task

The project is about working with and analyzing large texts. As examples of large texts you have the two large text files you used in

Another important part is to understand hashing and binary search trees.

The problem can be divided in four parts:
1. Divide the large texts into a sequence of words.
2. Implement two data structures suitable for working with words as data: a) A hash based set, and b) a binary search tree based table
(dictionary).
3. Use your two data structures to solve certain word related tasks
4. Measure the time to perform certain operations on your table and set implementation.

Part 1 - Divide text into words

As a part of your presentation you should define what you meant by a word and motivate your decision. However, as a rule, a word doesn't
contain any digits, or symbols like ".", ",", "!", "?", etc., and we consider words like "can't" and "John's" as a single word. Furthermore, it might
be a good idea save the words in a separate file to avoid having to repeat the process every time you read the text files.

Part 2 - Implementing data structures

Lecture 9 outlines the basic ideas of two techniques suitable for implementing tables (dictionaries) and sets: 1) Binary search trees, and 2)
Hashing. Your task is to implement a set (suitable for words) based on hashing and a table based on binary search trees. We want you to
follow the simple (non-class based) idea for how to implement a data structure that was used in the linked list example in Lecture 9.
Additional limitations:
The nodes in the binary search tree based table implementation are lists of size 4 (key, value, left-child, right-child).
The hash-based set is built using a Python list to store the buckets where each bucket is another Python list. The initial bucket list size is 10

and rehashing (double the bucket list size) takes place when the number of elements equals the number of buckets.
Furthermore, code skeletons outlining which functions we expect for each data structure are available here. They also contains an example program showing how the various functions can be used.
Notice: You are not allowed to make any changes of the function signatures in the given skeletons. Also, the demo programs should work exactly as outlined in the provided example programs. Your task is simply complete the given code fragments in the skeletons. However, feel free to add additional functions.

Part 3 - Count word related entities

Using your set and table implementation you should be able to handle a few word related questions:
1. By using the set you should be able to count how many unique words that are used in the two given texts files.
2. By using the table you should be able count how many words of a given length a each text has, and to present a histogram (length vs count, plotted using matplotlib).
3. By using the table you should also be able to present a list of the top-10 most frequently used words having a length larger than 4.
Notice: You are not allowed to use Python's set and dictionary classes to solve these problems. However, it might be a good idea to compare your results with a similar program using them. The results should be the same.

Part 4 - Measure time

Measure elapsed time in Python is easy using the function time() in the time module. However, measure the time of very fast operations often mean that you have to repat the experiment very many times, or/and use very large data sets. Finding a suitable experimental setup that highlights the feature you would like to study is often the hardest part.
Tasks:
1. Measure the time to look-up X thousand elements (keys) in tables of different sizes (larger than X) in your binary search tree based table implementation. Measure also the max tree depth at the same sizes. We expect two plots (using matplotlib) showing look-up time vs table size, and max depth vs table size.
2. Adding a new element to a hash table would (in theory) be independent of the table size if it wasn't for rehashing. Try to design an experiment measuring the time it takes to add new elements to your hash set and how it correlates to the rehashing. Measure also

the max bucket size at various set sizes. We expect a few plots (using matplotlib) showing how the time to add (say) 1000 unique
elements differs at various table sizes due to the rehashing mechanism, we also expect a plot showing how the max bucket size
changes with the set sizes.
We strongly suggest that you use the words in the text file eng_news_100K-sentences.txt as your data in the experiments. The file contains
about 2 million words, out of which about 80.000 are unique.

Suggestion for how to get started

For a team with three members we suggest the following start:

1: Handle Part 1: Read arbitrary text and divide it in a sequence or words. Suggestion: Store the words (one on each line) in a
separate file for future use
2: Implement word_set.py based on hashing.
3: Implement table.py (key-value pairs) based on binary search trees. (This is the hardest part in the whole project!)
1: Try to solve exercises in Part 3 using Python's set and dictionary rather than your own implementations.
...