HW 05

Chapter 6.1 Problem 11

> Problem Statement: Anagram detection

> a. Design an efficient algorithm for finding all sets of anagrams in a large file such as a dictionary of English words.

> b. Write a program implementing the algorithm.

Solution (Using presorting)

A. An algorithm that uses presorting to find all the sets of anagrams in a list of words is as follows.

Let's say you have a list of words. `words = ["eat","tea", "bae", "say", "ate"]`

1. First you want to be able to map two words that are anagrams of each other to the same thing. You can do this by sorting the letters in each word. So `"eat" -> "aet"` and `"tea" -> "aet"` Performing this operation is an `O(n*m)`, where `m` is proportional to the average length of each word. Because the number of letts in the average word is samller than the number of words in the english language we can safely assume that `m` will not be the primary factor when determining how long this part of the algorithm takes. The time efficiency of this algorihtm then becomes `O(n)`.

2. The next step of the algorithm would be to sort through the transformed words. The reason we do this is so that all the words that are anagrams of each other end up being next to each other. To do this we can use the default python sorting algorithm is Timsort which worst case operates in `O(n)` time.

3. Now that our mapped words are all sorted we can perform an `O(n)` algorithm that goes through every item, compares it with the next item, and then adds all the sets of anagrams to the main anagram list.

How to run

4. Navigate to the project directory.

5. Run the program.
   `python p11.py`

6. Observe the first 20 anagrams it found.

7. Run the program in interpreter mode.
   `python -i p11.py`

8. You can sort the `anagrams` list and see what the most common anagram is by typing the following.
   ```

> > > a = sorted(anagrams, key=lambda x: len(x), reverse=True)
> > > a[0]
> > > ```
   ```