Comp 3761 Efficiency of Sample Problems Lab 2

Due date:

This file contains Lab 2. You must submit your answers to the D2L Dropbox "Lab 2" prior to the indicated due date.
Lab 2 requires Java programming. You can work in pairs (but you must still submit your own work to D2L).

Note that late assignments will not be graded.

You need to hand in the following to D2L:

- Print screen of your output
- Java code
- A short report outlining your procedure, results, and conclusion.

Please do not zip or compress your submissions. D2L allows you to upload multiple files

Grading:

- Applying technique #1 for solving problem [3 mark]
- Applying technique #2 for solving problem [3 mark]
- Applying technique #3 for solving problem [3 mark]
- Report and conclusion [1mark]

1- An anagram is a rearrangement of the letters of one word to form another word. For example, the words tea and eat are anagrams. Both words must have the same number of letters. Capitalization is ignored. Here are some additional examples:

Algorithm	logarithm
Conversationalists	conservationalists

There are numerous techniques for finding out if two words are anagrams. For example:

Technique #1:

```
for each letter in word 1
    search word 2 for the letter
    if found, delete the letter from word 2
```

Technique #2:

```
sort word 1
sort word 2
use a linear compare of the 2 sorted words
```

Technique #3:

Use the fact that an anagrams have the same number sum of distinct of letters in each word. Given that, write an algorithm to compare if two words are anagrams of each other by comparing the sum of distinct letters.

```
    So, for anagrams Stressed = Desserts the sum for each charecter is:
        d = 1
        e = 2
        r = 1
        s = 3
    words that are anagrams of eachother have the same sum for each distinct character.
```

hint: you can use an array to hold the sums of individual characters then index into the sum using the ascii value of a character. Using the example above of “Stressed” we know the asci value for ‘d’ is 100. So to increment the amount of d’s we observe in an array we can do A[100] = +1; or A[asci(d)] = +1;

Your challenge today (this week) is not simply to devise an algorithm to determine if two words are anagrams. Instead we are going to look for the word (or words) that have the largest number of anagrams, in the English language dictionary.

Furthermore, you are going to apply all of the three techniques to the above problem in an attempt to find the fastest anagram checking technique and measure the efficacy based on execution time. You will compare the efficiency of three anagram techniques.

Input to your program(s): the words in the file named Dict, posted on D2L.
Output from your program(s): for each technique you test, you must output (to the console)

a. the word(s) with most anagrams
b. the time (in seconds) that it took to find (a)

For example, I might output (from one of three techniques): Note: this is not the correct answer!

```
Technique #1: [dog] has 2 anagrams 445.789 secs
```

I am looking for you to approach this exercise as a scientific experiment. As with all experiments we start by asking some questions. In this case they could be:

Which technique is fastest for finding the word (or words) that have the largest number of anagrams?
What is the overall efficiency class for each technique?(like O(n), O(nlogn),..)