# i206 Spring 2013: Assignment 5

**Name**_____

## Question 1: Practice Algorithm Analysis: Spell Checking

These days every word processor has a spell-checking program. There are a lot of clever ways to do spell checking and do it efficiently. In this question we will look at a few ways to do it. The question will describe an algorithm for doing the spell check and your job is to write python code to implement the algorithm and to do the big O analysis of the algorithm.

In each case, assume you have a dictionary of size ***m*** (meaning it has ***m*** unique words in it) and a document of length ***n*** (meaning there are ***n*** words in it; many of these words will repeat, such as "the"). You can use your own document; be sure it is in text format. There is a link to a dictionary with ~65,000 words on the website at:

http://blogs.ischool.berkeley.edu/i206s13/files/2013/02/wordlist.txt.zip

For this assignment, assume that ***m*** is very close to ***n*** in size.

You want to make sure that every word in the document is spelled correctly. Open the document up, read in all the text, separate it into words (break boundaries at white space, remove all punctuation). The dictionary is structured so that there is one word per line. Your job for each of the algorithms below is to:

i. Analyze the order (big O) of this algorithm, assuming that both n and m can get very large, to encompass millions of words each. To do this, analyze each line of pseudocode in terms of the worst case running time and combine this into your big O analysis. Show your work.
ii. Write the python code for this pseudocode. Test it too!

Here are the three algorithms:

a) Algorithm A: Go through the words in the document sequentially. For each word, look it up in the dictionary. Go through the dictionary sequentially too – i.e. when you look it up in the dictionary, start from the first word in the dictionary and check if the word matches. The algorithm in pseudocode (not the smartest algorithm in several ways) is:

```
words = readWordsFromDocument(document)
dictionary = readWordsFromDocument(dictionaryWords)
```

```
for word in words:
     found = false
     for dictWord in dictionary:
          if word == dictWord:
               found = true
     if found != true:
          print "Word " + word + " not found in
          dictionary."
```

b) Algorithm B: This time you first sort the dictionary. You again go through the document sequentially, but you look up the word in the dictionary in sorted order using **binary search**. Assume that the running time for the python **sort** algorithm (which you are free to use) is O(n log n). Please write your own binary search code although you are free to look at and use samples from the web. Here is the pseudocode:

```
words = readWordsFromDocument(document)
dictionary = readWordsFromDocument(dictionaryWords)
dictionary = Sort(dictionary)

for word in words:
     if BinarySearch(word, dictionary) != true:
          print "Word " + word + "not found in
dictionary"
```

c) Algorithm C: Use python's built in dictionary function for storing the words of the dictionary. Take it on faith for now that each access to the dictionary is O(1), a constant. (They are implemented using hash tables which have this O(1) property for lookup; we'll go over that later in the course.) Assume that the code createDictionaryDataStructure runs in time 0(n).

```
words = readWordsFromDocument(document)
dictionary = readWordsFromDocument(dictionaryWords)
dictionary = createDictionaryDataStructure(dictionary)

for word in words:
  if DictionaryLookup(word, dictionary) != true:
     print "Word " + word + "not found in dictionary"
```

# i206 Spring 2013: Assignment 5