Problem Set 5
ISE 407 – Computational Methods in Optimization
Due: November 8, 2019
Dr. Ralphs

1. This problem is to compare different dictionary implementations in Python, including Python's own built-in implementation. First, obtain my dictionary implementations in Python from Github:

   ```
   git clone https://github.com/tkralphs/PyDict.git
   ```

   In addition, you should choose one or two classic books that are out of copyright and find a plain text copy on-line. Project Gutenberg is a good place to browse. The book should have at least 10K words.

   (a) Develop a Python script that takes the name of a book file and a search term as an argument and counts the number of occurences of each word in the text by inserting them into a dictionary. The value for each key should be the number of occurences. Be sure you strip punctuation and other characters away before insertion so that you truly get just the words themselves. Also, make sure you ignore case. Your script should run from the command-line as follows.

   ```
   my_script.py --file file.txt --word x
   ```

   Running your script with

   ```
   my_script.py --file file.txt --sort n
   ```

   should sort the list of words by frequency of occurence in the text and print out the `n` most frequently occuring words (along with their frequencies).

   (b) Count the number of comparisons required to insert all the words in your text into the dictionary for both open addressing and chaining. Make a graph showing the number of comparisons required as a function of the load factor for each implementation. What does this say about the time-space tradeoff for each?

   (c) Make a similar graph of the time it takes to insert all the words as a function of load factor. What does this say about the time-space tradeoff for each?

   (d) Compare the two Python-based hash table implementations to Python's own built-in implementation in terms of time to insert.

   (e) Analyze the distribution of the words in the table for both open addressing and chaining. Is the hash function a good one?

2. Consider the following scheme for sorting a list of n numbers. For simplicity, let's assume $n = 2^k$ for some k (the numbers may have more than k bits).

   - We first insert the items into a hash table with chaining, with the hash function being the first k bits of the number (this is not a good hash function in general, but serves a purpose here).

- Next, we sort the individual lists of items in each slot in the hash table with insertion sort.
- Finally, we just concatenate all of these sorted lists together.

(a) Why does this algorithm sort correctly?

(b) What is the worst case running time of this algorithm and with what kinds of inputs is it realized?

(c) Assuming the numbers are completely random, what running time would you expect in practice? Explain.

3. Consider the following sorting algorithm.

```
def fsort(A, beg = None, end = None):
    if beg == None:
        beg = 0
    if end == None:
        end = len(A) - 1
    if beg >= end:
        return
    if A[beg] > A[end]:
        A[beg], A[end] = A[end], A[beg]
    fsort(A, beg+1, end-1)
    if A[beg] > A[beg+1]:
        A[beg], A[beg+1] = A[beg+1], A[beg]
    fsort(A, beg+1, end)
```

(a) Formally show that this sorting algorithm is correct (hint: use induction).

(b) Write a recurrence for the running time of this algorithm. Is this algorithm efficient?