Faculty of Science

Course:	CSCI 3070U: Analysis and Design of Algorithms
Assignment:	#2
Topic:	Recurrences, heaps, heapsort, dynamic programming

Note: This assignment is to be completed individually. This means you should not work in teams. These restrictions are in place because the idea of this assignment is to prepare you to do well on the upcoming midterm, which will include questions related to this assignment.

Part 1

Read the first 20 or so slides of the document from Stanford NLP group on Minimum Edit Distance (MED):

http://www.stanford.edu/class/cs124/lec/med.pdf

Using this technique, implement this algorithm using Dynamic Programming in an approved programming language. Include some testing code to try it out on a few string pairs:

- spoof/stool
- podiatrist/pediatrician
- blaming/conning

Part 2

For this part of the assignment, you will implement a radix sort procedure for sorting numbers between 0 (inclusive) and 1,000,000 (exclusive) (i.e. 6-digit numbers). Your procedure will use a modified version the counting sort implementation that you developed in tutorial #5 in order to sort by each digit (modified to sort by digit, of course).

You may find the following code (which determines the value of an arbitrary digit) useful:

```
function getDigit(num, digit) {
      working = num / 10digit-1;
      return workingmod 10;
}
```

Part 3

Write a greedy algorithm to solve the fractional knapsack problem, as described below:

> Given a number of items with weight and value, select a percentage (0.0-1.0) for each item(s) to include in your knapsack such that the weight capacity (W) of the knapsack is not exceeded, and the total value (V, the sum of the value of all included items) is maximal.

One way that you could implement this problem is to pass 2 arrays into your procedure: 1) weights, where weights[i] is the weight of element i, and 2) values, where values[i] is the value of element i.

Part 4

Write an implementation of Huffman coding, which is a greedy implementation of assigning prefix codes. This will require a program that does the following:

1. Read through a simple ASCII text file (passed as an argument), determining the frequency of each character in the file
2. Use these frequencies to calculate the optimal prefix codes for each character (Huffman)
    - Print a complete table of prefix codes for the characters in the document (print only characters with frequencies > 0)
3. Normally, you would then encode each character using its prefix code
    - However, actually implementing this requires some trick bit manipulation (which, for some languages, is difficult to do)
    - Instead, calculate the length of the entire document before and after using the prefix codes

How to Submit

Put your source code answers to part 1-4 into a ZIP file, called `Assignment2_FirstNameLastName_StudentNum.zip` (do not use .rar, .7z or other archival formats) and submit this file to Blackboard.