CS 112 Spring 2020 – Test 2

Name: _____ Net Id: _____

Write the answer to each question directly under the question. You may use as much space as needed to write your answer, and your answer can span multiple pages. Do not change the order of the questions. Type all your text answers (do not handwrite). If you need to draw something, you may draw it in Word itself. Or, you may draw it on a separate piece of paper (NEATLY!), take a picture, and insert it in this document where it should go in your answer. When you are finished, convert this document to PDF, and submit to Gradescope.
It would help if you wrote your answers in a different color than black.

Q1. Huffman Coding [15 pts]

1a) [5 pts, NO PARTIAL CREDIT]

Given the following set of character-probability pairs:

(D,0.1), (U,0.15), (R,0.15), (S,0.27), (A,0.33)

Show the final Huffman tree. (No need to show any of the steps of the tree building process).

1b) [5 pts, NO PARTIAL CREDIT]

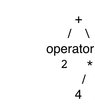
Choose any 5 characters and come up with probabilities for them that would result in the tallest possible Huffman tree for these characters. Show the probabilities as well as the Huffman tree.

1c) [5 pts]

A text document has 32 distinct characters, with equal number of occurrences of each, and each character stored in 8 bits. What would be the ratio of the length of the Huffman coded document to the original? Detail how you arrived at your answer - draw pictures if you are able to explain better that way. An answer without reasoning will get 1 point. An incorrect answer will get no credit, the reasoning will be ignored.

Q2. Expression Tree [13 pts]

You are given an expression tree that contains only integer constants and the binary operators '-', '+', '*' and '/'. Here's an example, along with the tree node class:




```
public class ETNode {
    String data; // integer constant, or binary operator
    ETNode left, right;
    ...
}
```

Complete the following method to evaluate an expression given a pointer to the root of its expression tree. You may NOT implement helper methods, and you may NOT use arrays or any other data structure. Your code should work exclusively with the tree structure.
(Note: `Integer.parseInt(String)` returns the int value represented by the string parameter. `Character.isDigit(char)` returns true if the `char` parameter is a digit, false otherwise.)

```
// Evaluates an expression tree given its root,
//returns 0 if tree is empty
public static double evaluate(ETNode root) {
    // complete this method
```

Q3. Hash Table [17 pts]

3a) [9 pts]

A hash table of initial table capacity (array length) 5 is set up to store integers. The hash code is the integer itself, which is directly mapped to the table using **integer mod table_capacity**. Come up with the smallest dataset of distinct non-negative integers that can be inserted in the hash table, with the following conditions: (a) after inserting all the integers in the dataset, the average number of comparisons for successful search is *strictly less than* 2, and the worst case number of comparisons for successful search is 4, (b) the load factor threshold is 1.5, and (c) there must be exactly one rehash, when the threshold is exceeded.

Assume that when an item is about to be inserted, the load factor is computed before actually inserting the item into the hash table. If rehashing is done, then the item is inserted after the rehash. Assume the rehash doubles the capacity of the table.

Show the hash table entries just before the rehash, as well as the final hash table with all entries in it. The worst case and average case requirements apply ONLY to the final hash table. What is the average number of comparisons against the integers in the final hash table for successful search?

3b) [4 pts]

As in the LittleSearchEngine assignment, consider a hash table that stores frequencies (number of occurrences) of words in a set of documents. Words are the keys, and for each word, the associated value is an array list of (document name, frequency) pairs, in *descending order of frequencies*.

Now suppose you are given a list of 50 words. You are asked to find all documents in the hash table in which one or more of these words each occurs with a frequency of **f** or more (**f** is an integer parameter). The final result should be a list of documents without duplicates.
(In other words, if word1 occurs with a frequency >= **f** in doc1, and word2 also occurs with a frequency >= **f** in doc1, then doc1 should only be reported once in the final result.) This non-duplication can be achieved by storing the resulting document names in a hash set.

Describe an algorithm for how you would do this (no Java code, write steps in plain English). Your algorithm should not look at more documents in the hash table than is absolutely necessary to find those with frequencies >= **f** for the given words.

3c) [4 pts]

This part pertains to the algorithm you came up with in Q3b above.
Assuming that a total of **n** document names appear in the hash table, and there are **k** document matches (duplicates included, of which **u** are unique) over all the 50 words combined, derive the big O running time of your algorithm.

For each big O component of the running time, specify whether it is worst case or expected case. You don't have to add up the components to a single big O.
Show your work - just big O answers without derivation will get no credit.
Also, your big O derivation will not get credit if your algorithm in 3b) is incorrect.