1. Hash Table (25 pts;10+4+11)

a) A hash table of initial table capacity 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 is 4, (b) the load factor threshold is 1.5, (c) there must be exactly one rehash, when the threshold is exceeded. Assume the rehash doubles the capacity of the table.

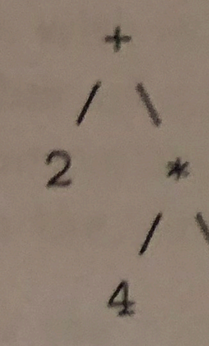
Show the hash table just before the rehash, as well as the final hash table with all entries in it. The worst case and average case requirements apply ONLY to the final hashtable. What is the average number of comparisons for a successful search in the final hashtable? Show your work.

b) What is the worst case big O running time to find a key in a hash table of capacity N and load factor threshold t, that has n keys? (Assume it takes $O(1)$ time to compute the hash code for any key.) Detail the process and derive the answer.

c) A movie app maintains a list of movie names with number of views. For the purpose of this exercise, assume both movie names and number of views are unique. While most of the time users look up movies to find number of views, occassionally they want to find the rank of a movie by views. Suppose you store the movies in a hash table. Which of movie name and number of views would you use for key, and which for value? What would be the worst case big O running time to find the rank of a movie, if there are n movies?

2. Binary Tree/Huffman Coding (25 pts, 12+6+7)

a) 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 reference to the root of its expression tree. You may NOT implement helper methods. (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) {
```

b) Given a document with 32 distinct characters, with equal number of occurences of each, what would be the ratio of the length of the Huffman coded document to the original, if each character in the input was coded in 8 bits? Show your work.

c) Suppose a character string of length n is encoded to k bits using Huffman coding. Consider decoding this back to the original character string. Briefly describe the decoding process. How many units of time (NOT big O) would the decoding take? Derive your answer, starting with specifying what unit time operation(s) you are counting.

3. BST/AVL Tree (12+7+6)

a) Suppose each binary search tree node is modified by adding a field that stores the number of nodes in its LEFT subtree, to help answer the question: what is the k-th SMALLEST key in the binary search tree? ($k = 1$ means smallest, $k = 2$ means second smallest, etc.)

```
public class EnhancedBSTNode {
    int key;
    EnhancedBSTNode left, right;
    int leftSize;  // number of nodes in left subtree
    ...
}
```

Implement the following recursive method to return the k-th smallest key in an enhanced BST. The method should throw a `NoSuchElementException` if k is out of range. You may NOT use helper methods.

```
public static int
kthSmallest(EnhancedBSTNode root, int k) throws NoSuchElementException {
```

b) Suppose an insertion into an AVL tree results in a Case 1 rotation being applied at node X. Show that the rotation can be done in $O(1)$ time in the worst case.

c) Consider two binary search trees A (with x nodes) and B (with y nodes). They are NOT AVL trees, just plain BSTs. Assume there are no duplicates in the entire set of items in A and B. Describe the fastest algorithm to output the combined set of items in A and B in sorted (ascending) order. You can use additional array space ($O(1)$ time to allocate), but no other data structure. Derive the worst case big O running time of your algorithm.