CS2383 – Fall 2024
Assignment 7 – Hash Tables

Due: Thursday Nov. 28, 10am (class time)

IMPORTANT: individual work please!

Tasks:

1. For each of the type of hash tables below (as a symbol table), draw them after inserting each of the following keys in that sequence (starting with an empty structure): 41, 25, 10, 60, 33, 75, 5, 50, 2, 7, 28, 80, 45, 6, and 31. Use the simple hash function h(k) = k % m.

    a) Separate Chaining with table of size m=10 – note: for this one, just show the final table
    b) Open addressing – linear probing – start with table of size m=10, and double the size of the table each time the load factor becomes larger than 0.5
    c) Open addressing – quadratic probing – with table of size m=25 (no resizing)
    d) Open addressing – double hashing using $h_2(k) = 13 – (k \% 13)$ – with table of size m=25 (no resizing)

2. The goal of this question is to do an empirical runtime analysis of Hash Tables (HashMap), and compare this with Red-Black Trees (TreeMap). First code the following (in Java):
    - Ask the user for the size of the input (n).
    - Create an array of n Integer objects (not just “int”), containing the values from 1 to n in that sequence.
    - Randomly shuffle this array. Use the shuffling algorithm in p.32 of your textbook. You can use the “nextInt” method in the Random class (in the Java API) rather than “StdRandom.uniform(…)”.
    - Create a HashMap and a TreeMap. Use Integers as both the key and the value. For the HashMap, use the static method “newHashMap” to create it, passing the value n in parameters. This way, the performance will not include the costly table resizing.
    - Do the following with the HashMap, and take actual runtime of this entire block of code:
        - Insert each of the elements from the shuffled array, in sequence, into the HashMap. Use the same object for both the key and the value.
        - Then search each of the elements in the shuffled array, in that sequence, in the HashMap
    - Redo the same as the last point above, this time with the TreeMap. Get a separate runtime value.

    With the code you prepared above, do an empirical runtime analysis, using different values of n. Note that there will be a lot of variations: for each value of n, run the program at least 5 times and take the average time (report all those times in your submission). Identify the Big-Oh for the HashMap and for the TreeMap. Compare with the expected runtime using the theoretical runtimes for those operations in the textbook or in the slides. Which one is faster: HashMap or TreeMap? Why do we still need both structures?

Submission: submit everything on paper, including a printout of your code in Question #2.