4. Hash... Browns?

For the following scenarios, insert the following elements in this order: 7, 9, 48, 8, 37, 57. For each table, TableSize = 10, and you should use the primary hash function $h(k) = k1 \bmod m0$. If an item cannot be inserted into the table, please indicate this and continue inserting the remaining values.

(a) Linear Probing

0	
1	
2	
3	
4	
5	
6	
7	
8	
9	

Solution:

0	8
1	37
2	57
3	
4	
5	
6	
7	7
8	48
9	9

(b) Quadratic Probing

0	
1	
2	
3	
4	
5	
6	
7	
8	
9	

Solution:

0	
1	37
2	8
3	
4	
5	
6	57
7	7
8	48
9	9

(c) Separate chaining hash table - Use a linked list for each bucket. Order elements within buckets in any way you wish.

0	
1	
2	
3	
4	
5	
6	
7	
8	
9	

Solution:

0	
1	
2	
3	
4	
5	
6	
7	57 $\rightarrow$37 $\rightarrow$7
8	8 $\rightarrow$48
9	9

5. Double Double Toil and Trouble

(a) Describe double hashing.

Solution:

The first hash function determines the original location where we should try to place the item. If there is a collision, then the second hash function is used to determine the probing step distance as 1*h2(key), 2*h2(key), 3*h2(key) etc. away from the original location.

(b) List 2 cons of quadratic probing and describe how one of those is fixed by using double hashing.

Solution:

In quadratic probing, 1) if the table is more than half full (load factor = 0.5) then you are not guaranteed to be able to find a location to place the item, 2) suffers from secondary clustering (items that initially hash to the same location resolve the collision identically).

Assuming a good second hash function is used, double hashing does not suffer from 1). Assuming a good second hash function is used, double hashing avoids secondary clustering because items that initially hash to the same location resolve the collision differently, which decreases the likelihood that two elements will hash to the same index after initial collision.

Name:_____

1. Suppose we sort an array of numbers, but it turns out every element of the array is the same, e.g., `{17, 17, 17, ..., 17}`. (So, in hindsight, the sorting is useless.)

   (a) What is the asymptotic running time of insertion sort in this case?

   (b) What is the asymptotic running time of selection sort in this case?

   (c) What is the asymptotic running time of merge sort in this case?

   (d) What is the asymptotic running time of quick sort in this case?

   Solution:

   (a) $O(n)$

   (b) $O(n^2)$

   (c) $O(n \log n)$

   (d) $O(n^2)$

7) **[10 points] Sorting**

You are given a list of AVL trees. The keys in the AVL trees are ages of people. Each AVL tree represents the ages for people in a different community. Your task is to sort the AVL trees such that tree X comes before tree Y if and only if:

- The minimal age in tree X is less than the minimal age in tree Y, **or**
- The minimal ages are the same, but the maximal age in tree X is less than the maximal age in tree Y

Otherwise, ties are broken arbitrarily. You may assume that:

- There are *k* trees
- Each tree has *n* keys in it
- The range of ages is fixed (0-127)

a) [5 points] Describe in a few sentences or numbered steps how you could use Mergesort to sort these trees efficiently in the worst case. What is the running time in terms of k and n?

1. **Find the minimum and maximum age for each tree. Time O(klogn), since each tree is balanced.**
2. **Perform a mergesort where the comparison is based on using the min element for each tree, with the max element as a tie breaker. Time is O(klogk).**

Total worst case time O(klogn + klogk)

Note: Another way of doing this was to just find the min and max values when doing the comparison in mergesort which would be O(k lg k * lg n).

Running Time: **O(klogn + klogk)**

b) [5 points] Describe in a few sentences or numbered steps how you could use ideas from Radixsort to sort these trees efficiently in the worst case. What is the running time in terms of k and n?

1. **Find the minimum and maximum as above. Worst case O(klogn).**
2. **Perform a BinSort using the <u>maximum</u> age of each tree (It is critical that you sort on the least significant – the part I care about least, first!). Time O(k+128) = O(k)**
3. **BinSort on the <u>minimum</u> age of each tree. Time O(k).**

Total time O(k + klogn) = O(klogn).

Running Time: **O(klogn)**