Section 5: Hashing & Sorting

1. 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 $h(k) = k$.

a) Linear Probing - Insertion

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

Linear Probing - Delete 37, 7, 57

0	8
1	X
2	X
3	
4	
5	
6	
7	X
8	48
9	9

b) Quadratic Probing

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

c) Separate chaining hash table - Use an unsorted linked list for each slot.

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

2. Double Double Toil and Trouble

a) Describe double hashing.

The first hash function determines the location where initially 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.

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.

c) Compare open addressing and separate chaining.

Open Addressing

- Deals with collisions by moving element to a different index
- Can use less memory
- Linear probing: easiest, but has primary clustering, need to resize a lot
- Quadratic probing: secondary clustering, will find a spot if $\lambda < ½$
- Double hashing: low chance of clustering, but need another hash function

Separate Chaining

- Deals with collisions by putting all the elements in a “bucket” at that index
- Easier to implement
- More memory because needs “bucket” data structure
- Average runtime for insert/delete/find: $O(1 + \lambda)$
    - Best: O(1), Worst: O(n)