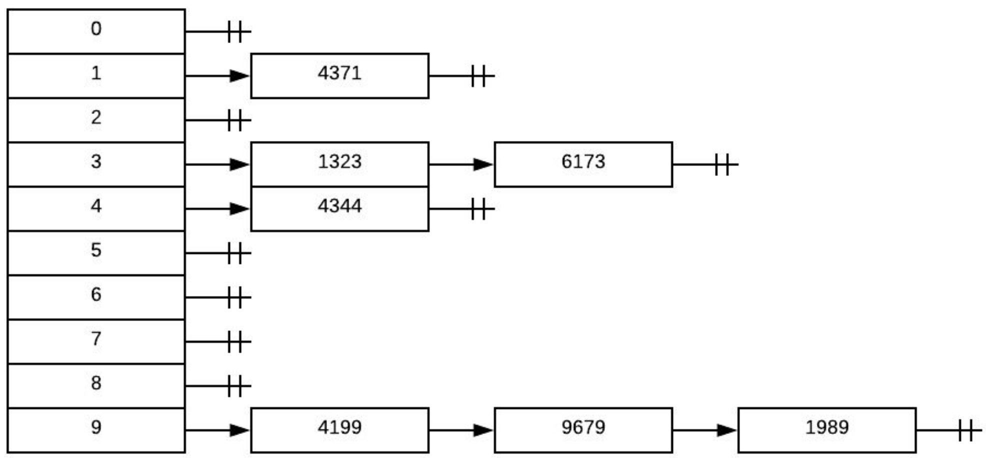
Andrew DiBella
Hw4

Hashing

1) Given input {4371, 1323, 6173, 4199, 4344, 9679, 1989} and a hash function $h(x) = x \bmod 10$, show the resulting:
   a. Separate chaining hash table.
   b. Hash table using linear probing
   c. Hash table using quadratic probing.
   d. Hash table with second hash function $h2(x) = 7 - (x \bmod 7)$.

a)



b)

0	9679
1	4371
2	1989
3	1323
4	6173
5	4344
6	
7	
8	
9	4199

c)

0	9679
1	4371
2	
3	1323
4	6173
5	4344
6	
7	
8	1989
9	4199

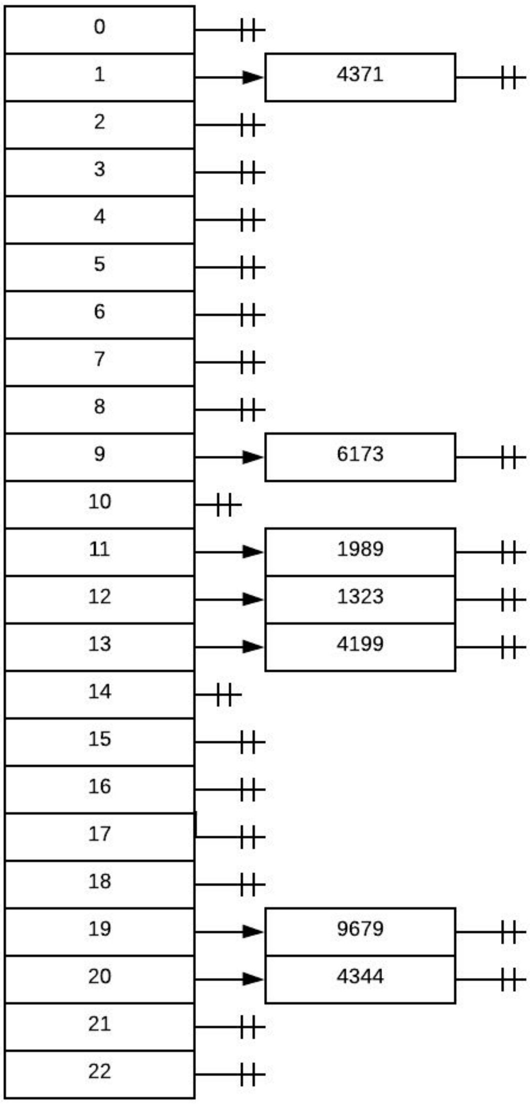
d)

Since the original table size is 10 (not a prime number), using double hashing would not be implemented correctly due to collisions. The second hash function is ideal but the table size and first hash function causes a collision when trying to insert 6173 into the hash table. There is first a collision at index three using the h1 and a second collision at index 1 using h2.

0	
1	4371
2	
3	1323
4	
5	
6	
7	
8	
9	

2) To rehash the original you must double the size of the original table which would be 20. To prevent what already occured in question 1 part d I am going to choose a new hash size of 23 because it is at least double and also the closest prime number to 20. The new hash function is now h(x) = x mod 23

a)



b,c,d) After rehashing to a prime number their is no chance for collision in the hash table for these inputs so for the remaining parts 1 table will suffice for all three because they are identical.

0	
1	4731
2	
3	
4	
5	
6	
7	
8	
9	6173
10	
11	1989
12	1323
13	4199
14	
15	
16	
17	
18	
19	9679
20	4344
21	
22	

3)

Table Size	Linear Probing with hash_string_1	Quadratic Probing with hash_string_1	Linear Probing with hash_string_2	Quadratic Probing with hash_string_2	Linear Probing with hash_string_3	Quadratic Probing with hash_string_3
1753	175967	54285	2901	2026	990	980
1786	175967	30399	3176	2148	978	974
2039	175967	52641	2423	2051	1007	997
2048	175967	52871	2791	2055	977	976
3067	175967	49621	2392	1876	942	943
3072	175967	50226	2765	1888	941	939
4093	175967	47664	2168	1743	906	906
4096	175967	49200	2271	1779	917	916
5119	175967	47334	1910	1703	915	915
5120	175967	47512	1850	1679	904	903
8191	175967	46826	1830	1665	889	889
8192	175967	46720	1820	1641	895	895

Linear Probing:

```
class Hash {
 public static void main(String [] args){

   Scanner input = new Scanner(System.in);
   int tableSize = 8192;
   int probes = 0;
   Map<Integer, String> map = new HashMap<>();

   while (input.hasNext()){
     String next = input.nextLine();


     int index = hash_string_3(next, tableSize);
     while (map.get(index) != null){
       index++;
       probes++;
     }
     map.put(index, next);
     probes++;

   }
   System.out.println(probes);
```

```
}
```

Quadratic Probing:

```
import java.util.HashMap;
import java.util.Map;
import java.util.Scanner;

class Hash {
 public static void main(String [] args){

   Scanner input = new Scanner(System.in);
   int tableSize = 1753;
   int probes = 0;
   Map<Integer, String> map = new HashMap<>();

   while (input.hasNext()){
     String next = input.nextLine();

     int index = hash_string_1(next,tableSize);
       int start = index;
       int count = 1;
       do{
         if (map.get(index) == null){
             map.put(index, next);
             probes++;
             break;
           }
           probes++;
           index = (index + count * count++) % tableSize;
       } while (index != start);
   }
   System.out.println(probes);
 }
```

Hash functions:

```
static int hash_string_1(String key, int tableSize) {
   return (key.length() * key.length() * 4) % tableSize;
 }
  static int hash_string_2(String key, int tableSize) {
   return    (key.charAt(0) +
        27 * key.charAt(1) +
       729 * key.charAt(2)) % tableSize;
 }
  static int hash_string_3(String key, int tableSize) {
   int hash = 0;
   int prime = 31;
   for (int i = 0; i < key.length();i++){
     hash = (prime * hash + key.charAt(i));
   }
   return hash % tableSize;
 }
```

The different hash functions are the main reason for the amount of collisions. Using prime numbers for multipliers for the probing process caused less collisions. A bad hash function like hash_function_1 will cause a collision any time the characters of the string are the same regardless of the order. It was unexpected for hash string one to produce the same number of probes for each table size. I hope there wasn't an error on my end for the linear probing but I may have over simplified it. Quadratic probing was more efficient and caused less probes than linear probing. However the difference between linear and quadratic probing was also dependent on the table size. Overall, a good hashing algorithm depends on the all three factors: a good hash function, the table size, and the type of probing used. Deciding which would be more effective depends on the data you are given.