CS 300 - Data Structures

Homework 3 – 5/12/2018

Atilla Alpay Nalcaci - 19510

Part I: Hash Table Class Implementation

While implementing the hash table code which has been initially gathered from lecture notes, various changes had been made regarding `insert`, `remove`, `findPos` and `find` functions in order to modify the code properly (e.g. along the lines of **linear probing**) with respect to the tasks and achievements of this homework. Hash function that is used in the program is x modulo size (x % probe_Tot). Also, after a couple of unsuccessful attempts on implementing the hash table header in template format, the hash table class had been created without a template.

Statistics of hash table, which are successful/unsuccessful insertions, deletions and locations (find), are stored in six different arrays. The structure of arrays (namely initialize) is modified where each array stores total number of probes and total number of transactions of its corresponding operation.

a. *insert* Function

```
157 int HashTable::insert ( const int & x )
158 {
159     int temp = 1;
160
161     // Insert x as active
162     int currentPos = findPos( x, temp );
163
164     if( isActive(currentPos) )
165         return ( -1*temp );
166
167     array[currentPos].element = x;
168     array[currentPos].info = ACTIVE;
169     current++;
170
171     return temp;
172 }
```

`insert` function has been modified so that it will return the temporary probe value. The temporary probe value is acquired from the `findPos` function. You can find detailed explanation regarding `findPos` function at Part (d).

b. *remove* Function

```
124 int HashTable::remove( const int & x )
125 {
126     int temp = 1;
127
128     int currentPos = findPos( x, temp );
129     if( !isActive( currentPos ) )
130         return ( -1*temp );
131
132     array[currentPos].info = DELETED;
133     current--;
134
135     return temp;
136 }
```

`remove` function has also been implemented to return the temporary probe value. For an unsuccessful operation, the `int temp` value becomes positive for succesful and negative for unsuccesful operations of the hash table.

c. *find* Function

```
142 int HashTable::find ( const int & x ) const
143 {
144         int temp = 1;
145
146         int currentPos = findPos( x, temp );
147         if ( isActive(currentPos) )
148                 return temp;
149
150         return ( -1*temp );
151 }
```

As in `insert` and `remove` functions, `find` function also returns the temporary probe value that it has acquired from the `findPos` function.

d. *findPos* Function

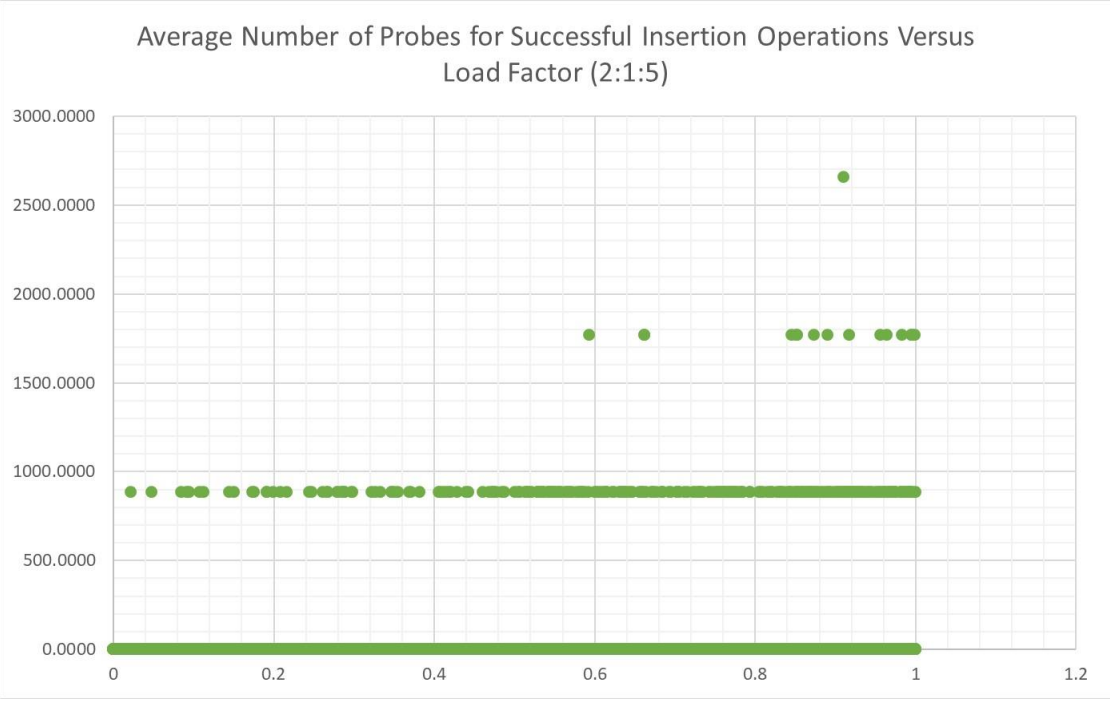
```
96  int HashTable::findPos( const int & x, int & y ) const
97  {
98      int currentPos = rehash( x );
99
100     while( array[currentPos].info == ACTIVE && array[currentPos].element != x )
101     {
102         currentPos++;
103         y++;
104
105         if ( currentPos >= array.size() )   // perform the mod
106             currentPos -= array.size();     // if necessary
107     }
108
109     return currentPos;
110 }
```

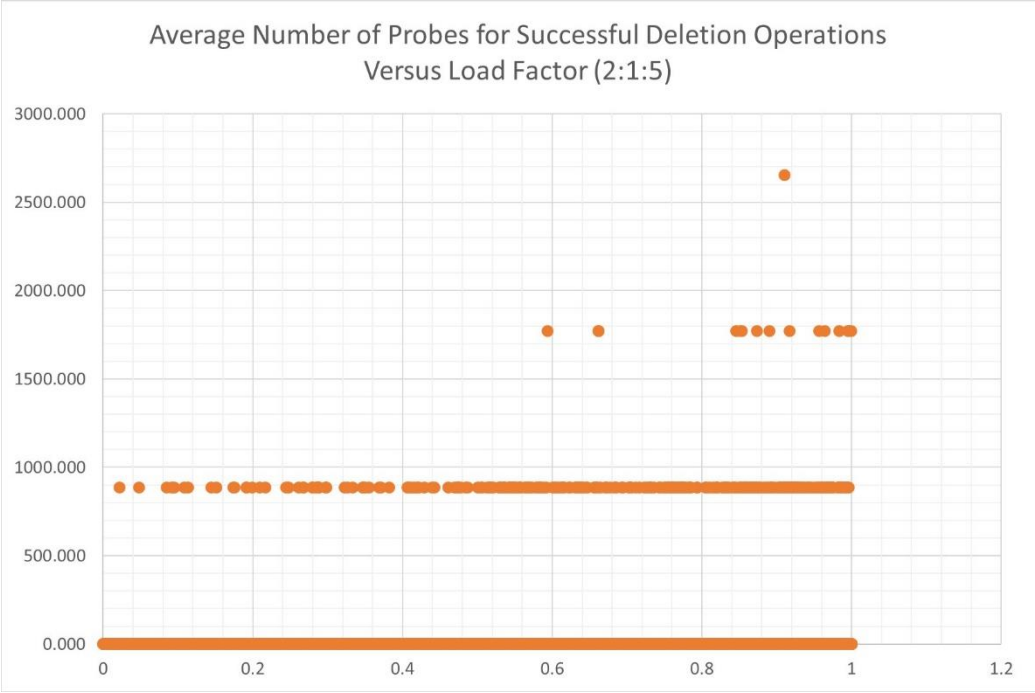
Lastly, `findPos` function has been implemented in order to sustain **hash-table implementation with linear probing** properly. The loop condition has been modified so that the index investigation before any operation becomes dynamic, meaning that hash table will always be aware of any removed index values while searching for a position.

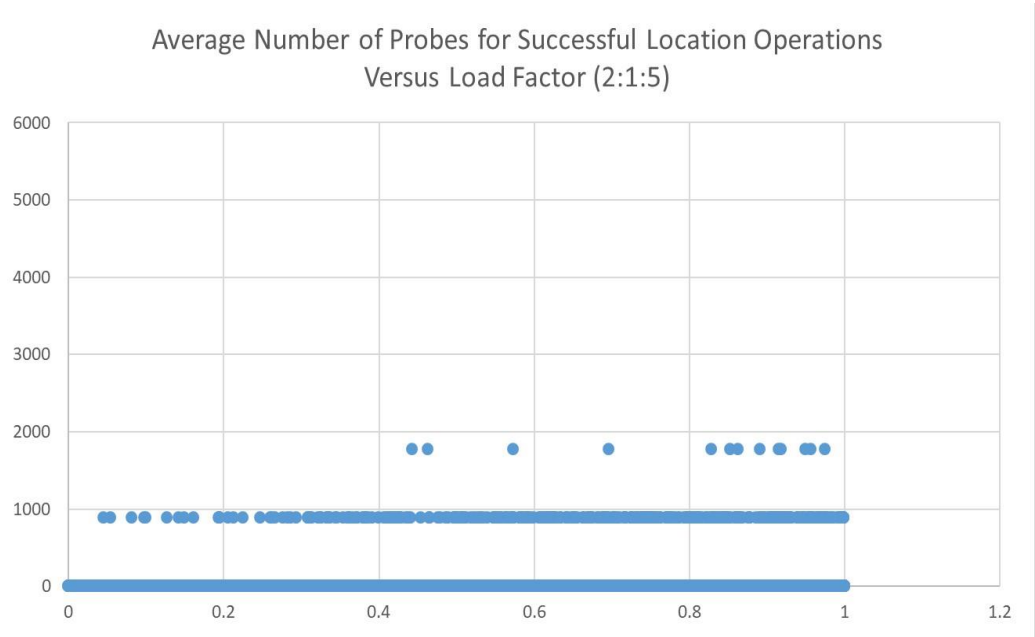
Part II: Output Organization & Graph Plotting:

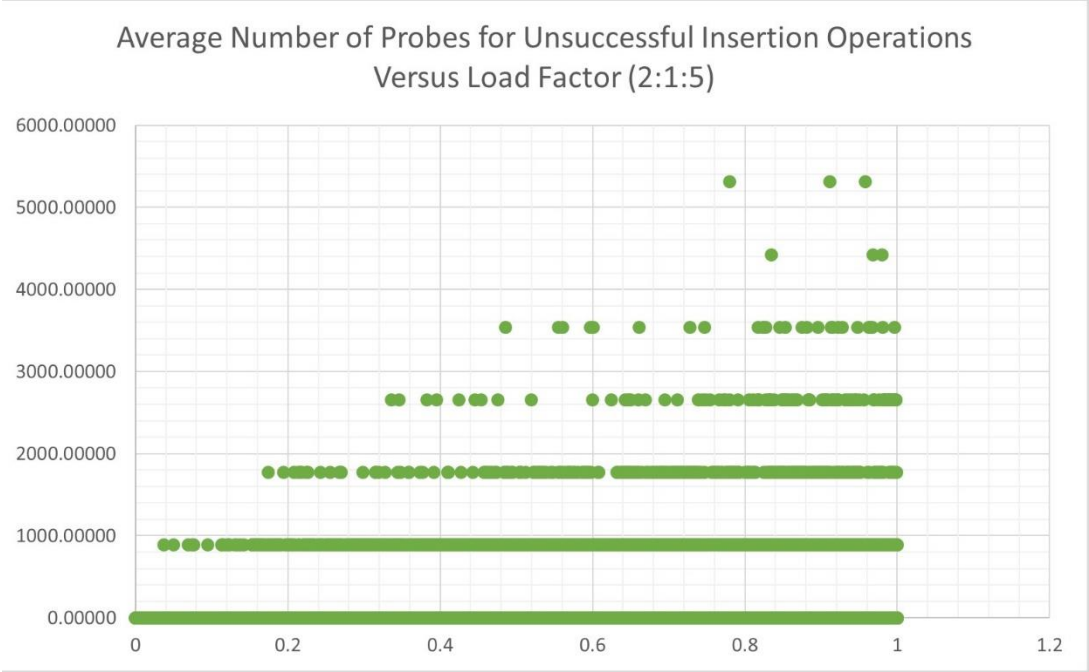
Below you can find the scatter plotted graphical interpretations of each category (namely successful insertion, deletion, find and unsuccessful insertion, deletion, find).

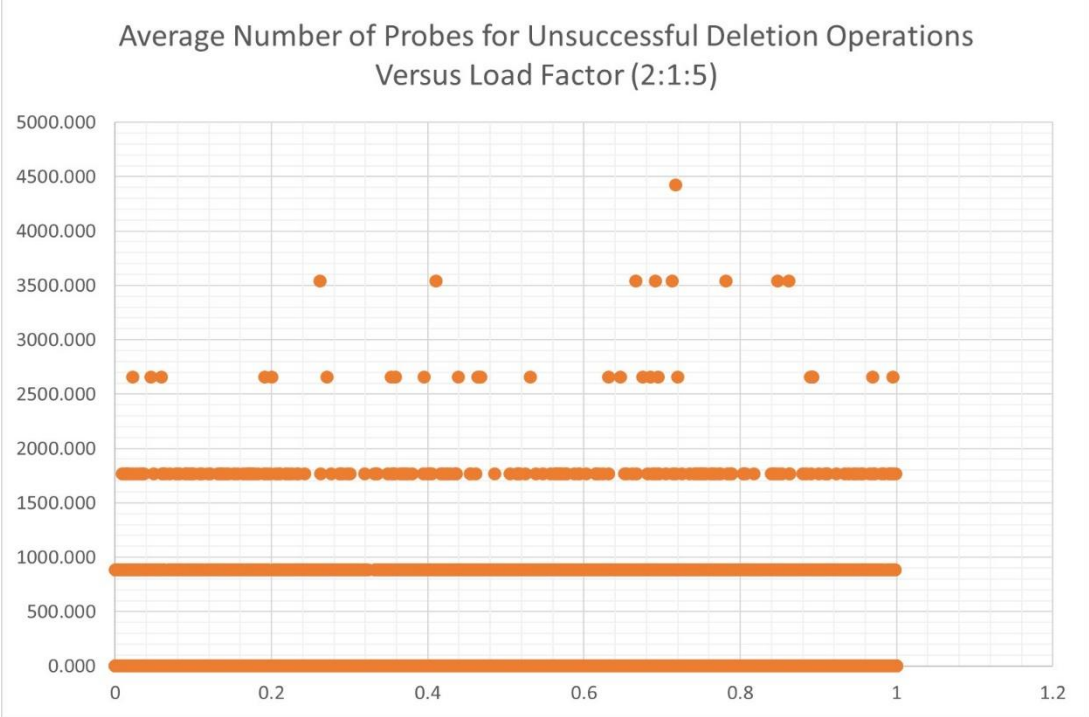
a. Probability (2:1:5)

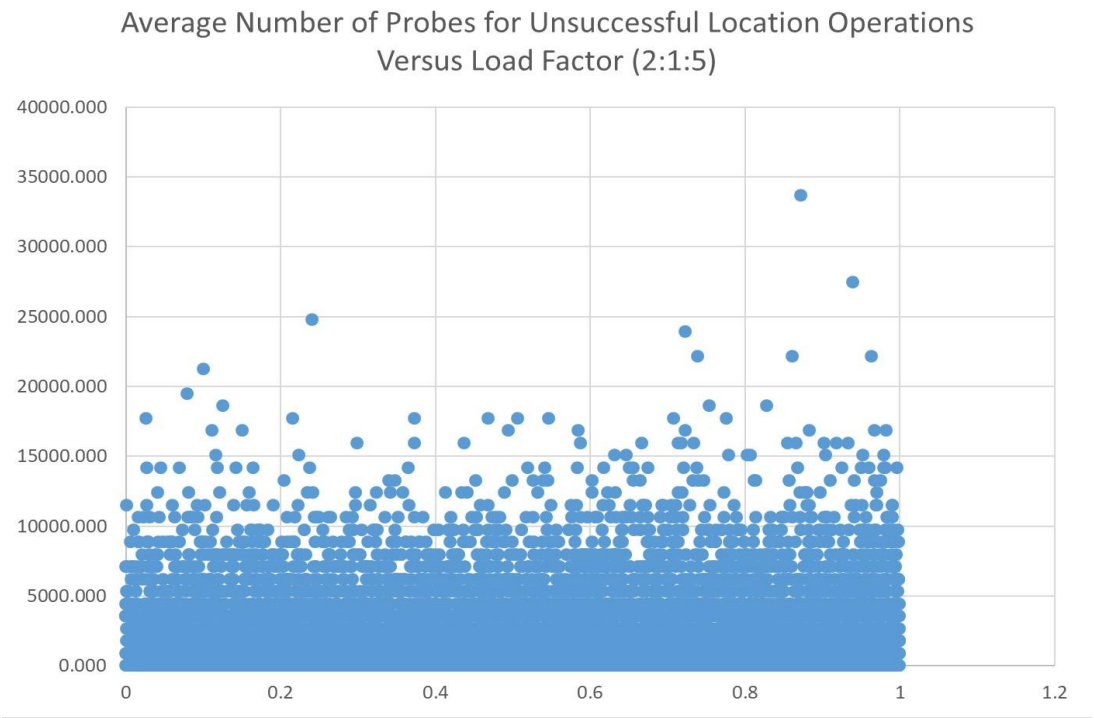



Average Number of Probes for Successful Deletion Operations
Versus Load Factor (2:1:5)
3000.000
2500.000
2000.000
1500.000
1000.000
500.000
0.000
0
0.2
0.4
0.6
0.8
1
1.2


Average Number of Probes for Successful Location Operations
Versus Load Factor (2:1:5)
6000
5000
4000
3000
2000
1000
0
0
0.2
0.4
0.6
0.8
1
1.2

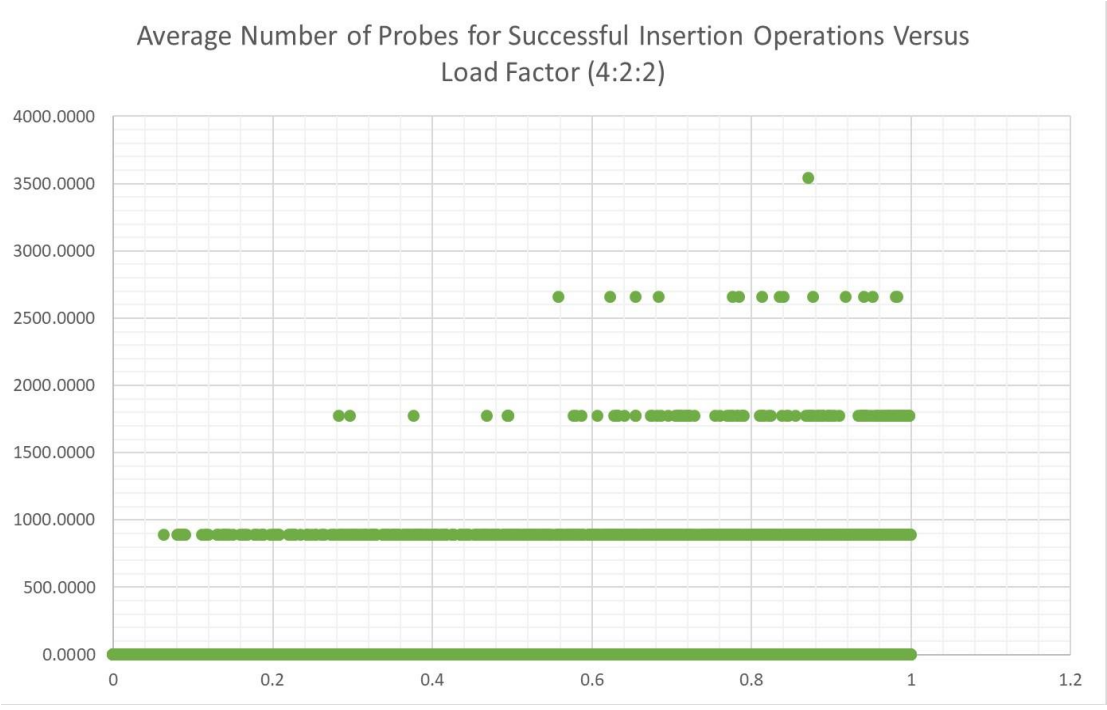

Average Number of Probes for Unsuccessful Insertion Operations Versus Load Factor (2:1:5)
6000.00000
5000.00000
4000.00000
3000.00000
2000.00000
1000.00000
0.00000
0
0.2
0.4
0.6
0.8
1
1.2

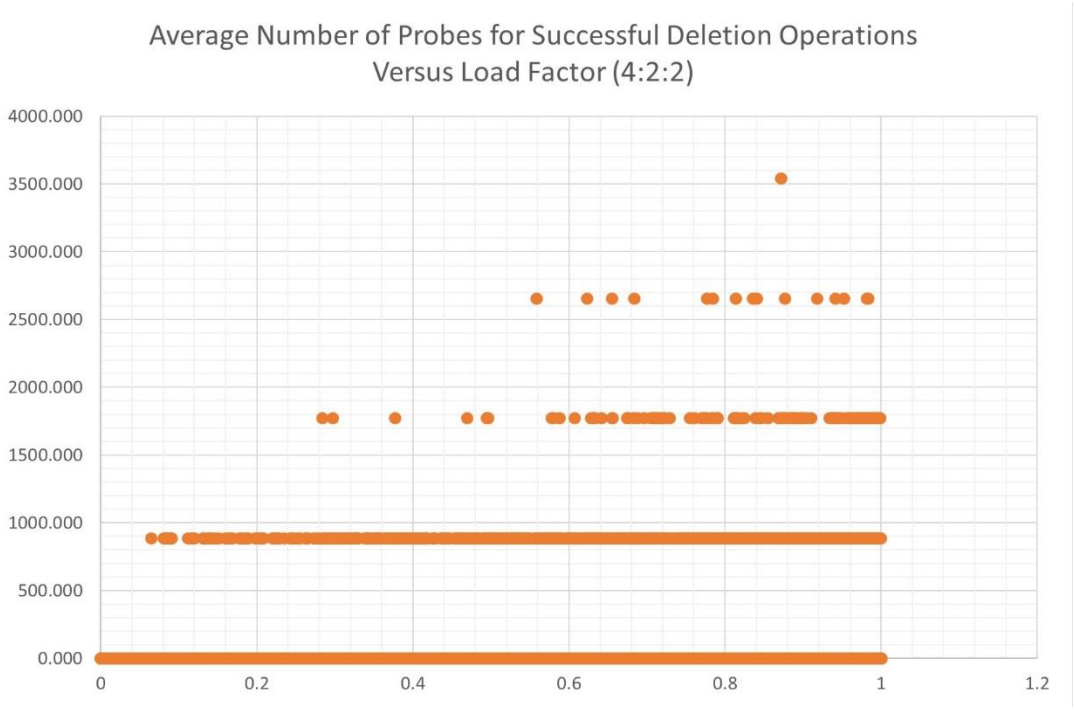

Average Number of Probes for Unsuccessful Deletion Operations Versus Load Factor (2:1:5)
5000.000
4500.000
4000.000
3500.000
3000.000
2500.000
2000.000
1500.000
1000.000
500.000
0.000
0
0.2
0.4
0.6
0.8
1
1.2

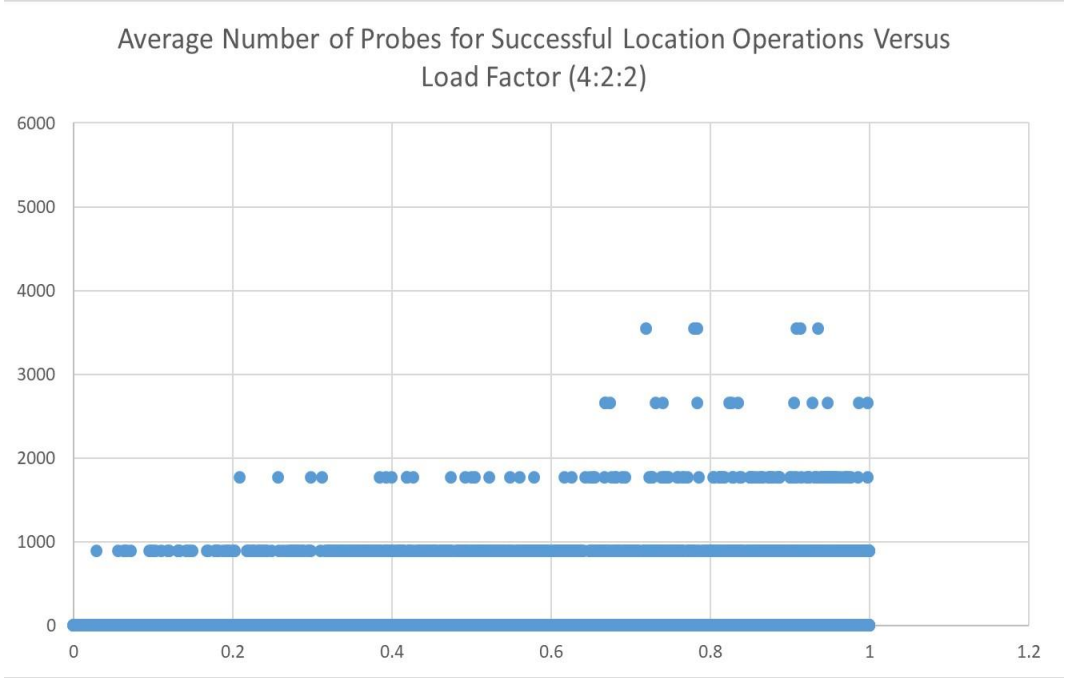


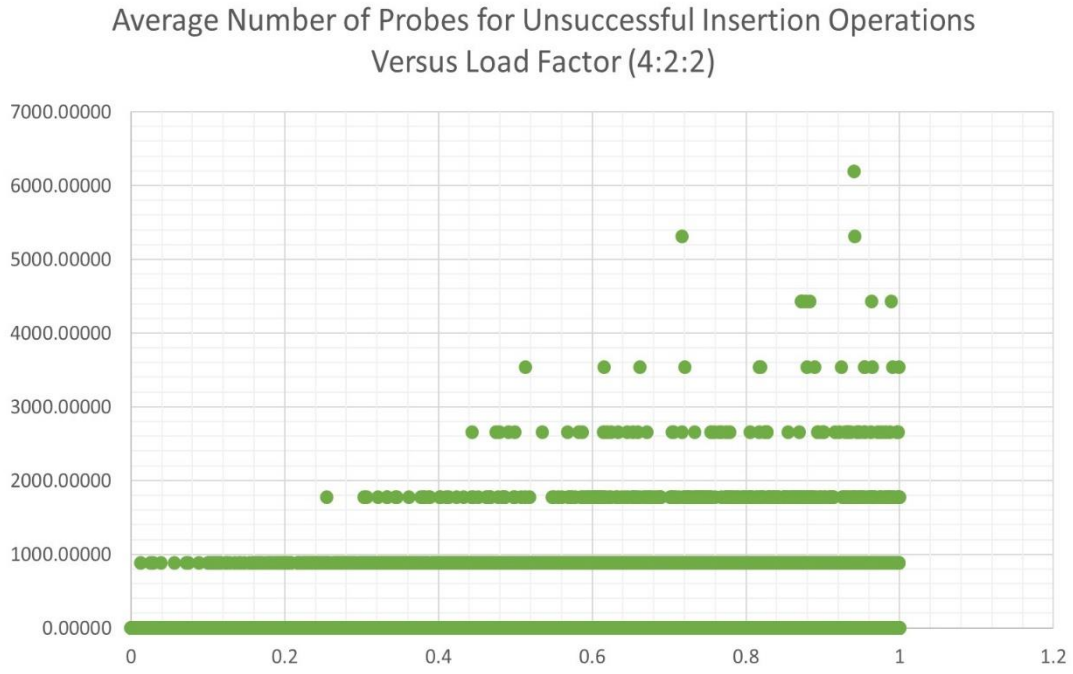
---

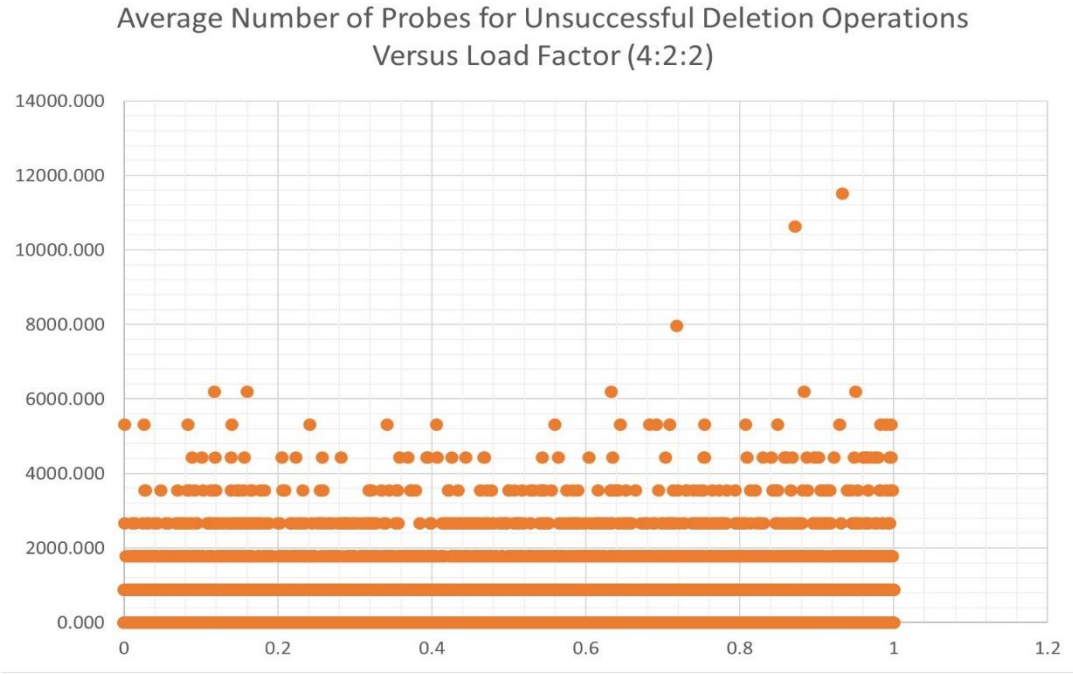
b. Probability (4:2:2)

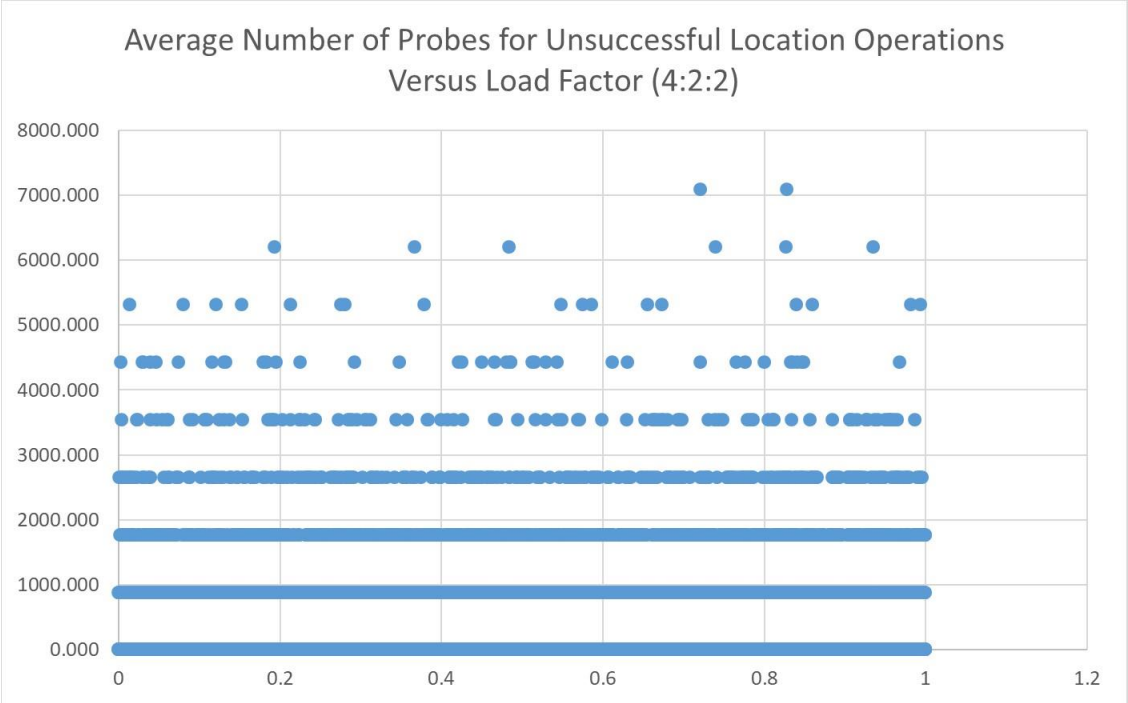



Average Number of Probes for Successful Deletion Operations Versus Load Factor (4:2:2)
4000.000
3500.000
3000.000
2500.000
2000.000
1500.000
1000.000
500.000
0.000
0
0.2
0.4
0.6
0.8
1
1.2


Average Number of Probes for Successful Location Operations Versus Load Factor (4:2:2)
6000
5000
4000
3000
2000
1000
0
0
0.2
0.4
0.6
0.8
1
1.2

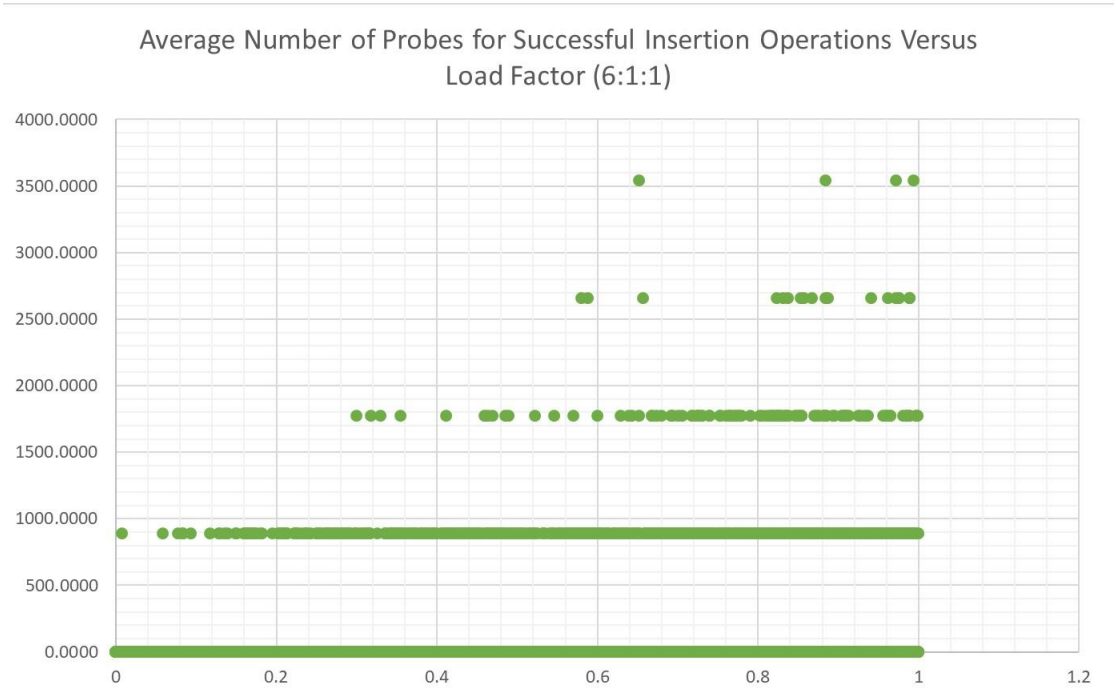

Average Number of Probes for Unsuccessful Insertion Operations
Versus Load Factor (4:2:2)
7000.00000
6000.00000
5000.00000
4000.00000
3000.00000
2000.00000
1000.00000
0.00000
0
0.2
0.4
0.6
0.8
1
1.2

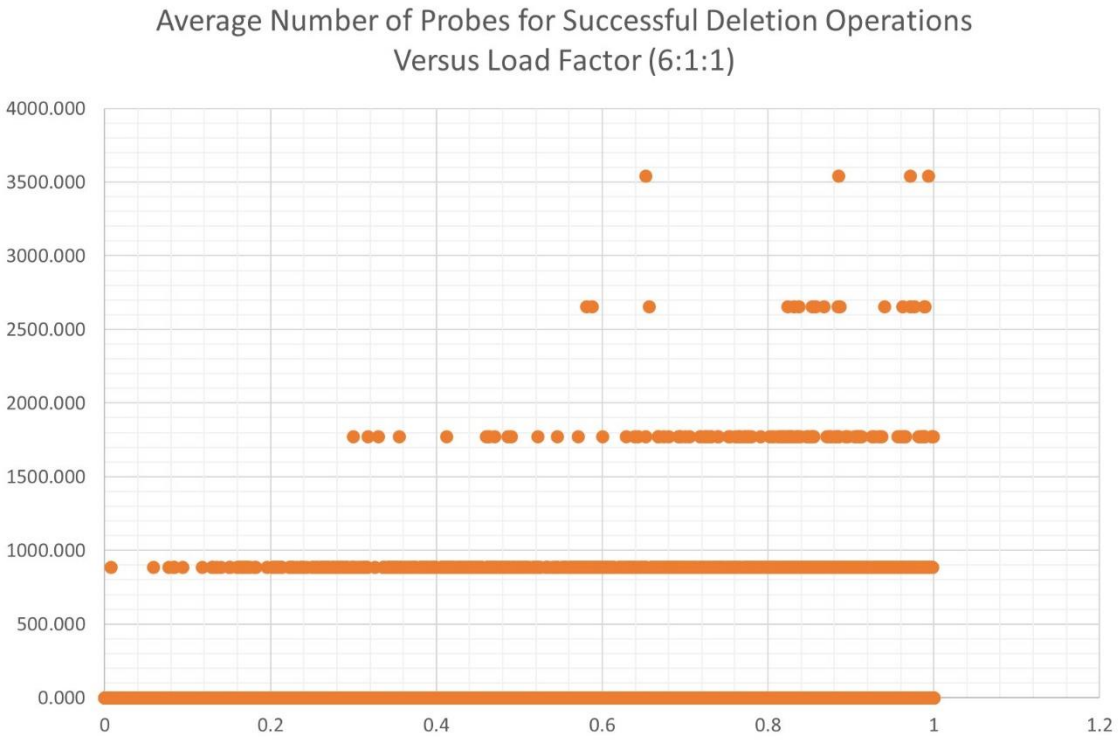

Average Number of Probes for Unsuccessful Deletion Operations
Versus Load Factor (4:2:2)
14000.000
12000.000
10000.000
8000.000
6000.000
4000.000
2000.000
0.000
0
0.2
0.4
0.6
0.8
1
1.2

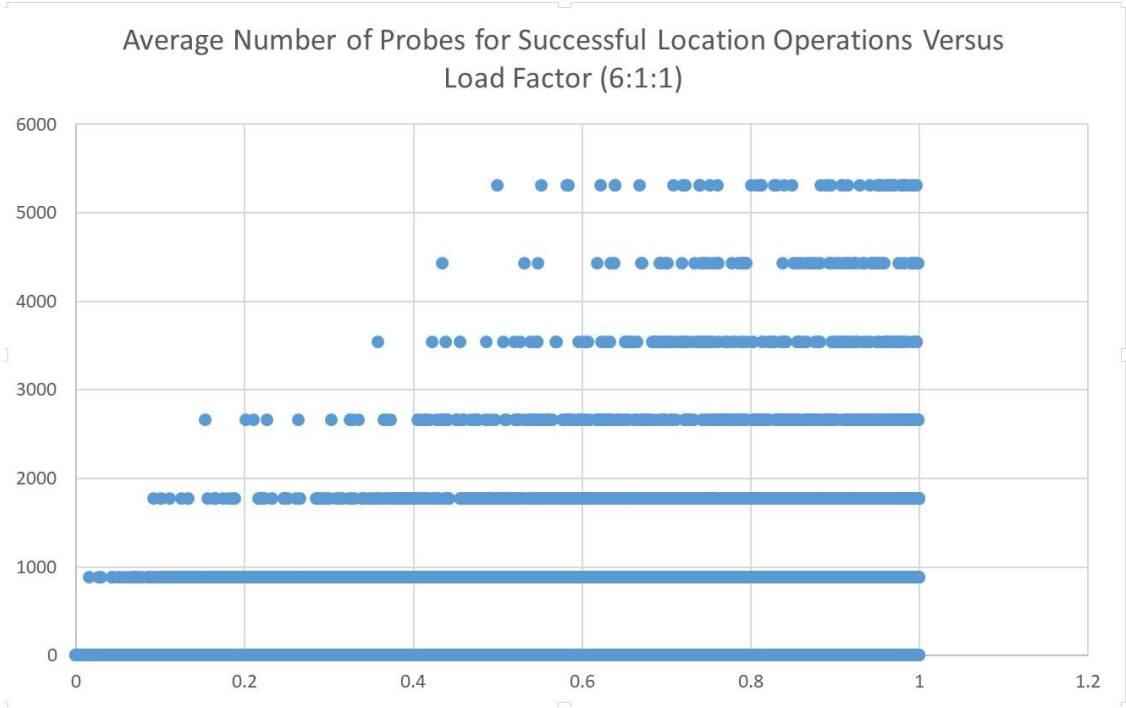


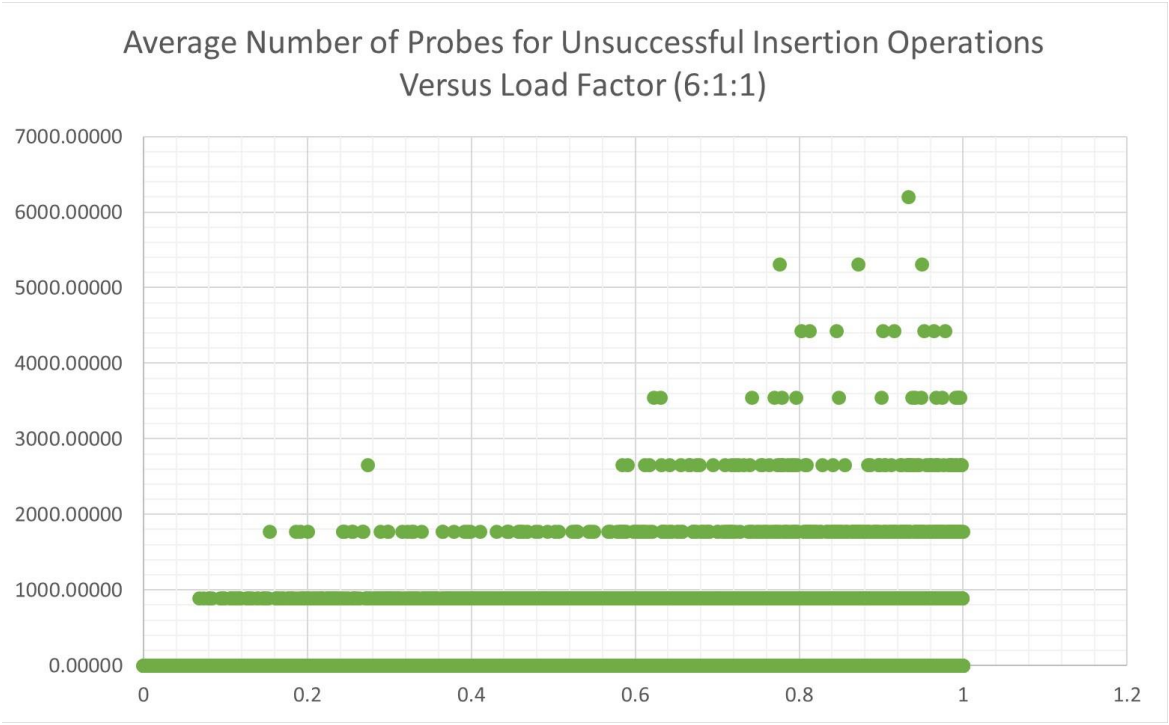
---

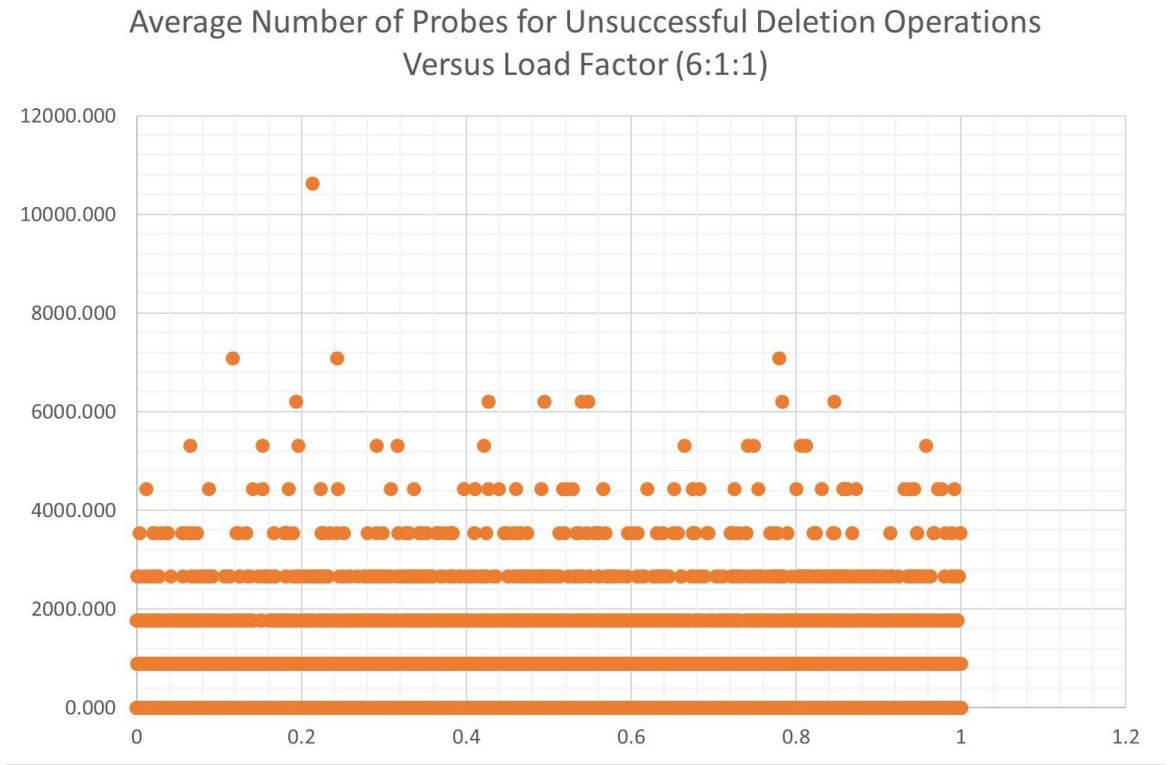
c. Probability (6:1:1)

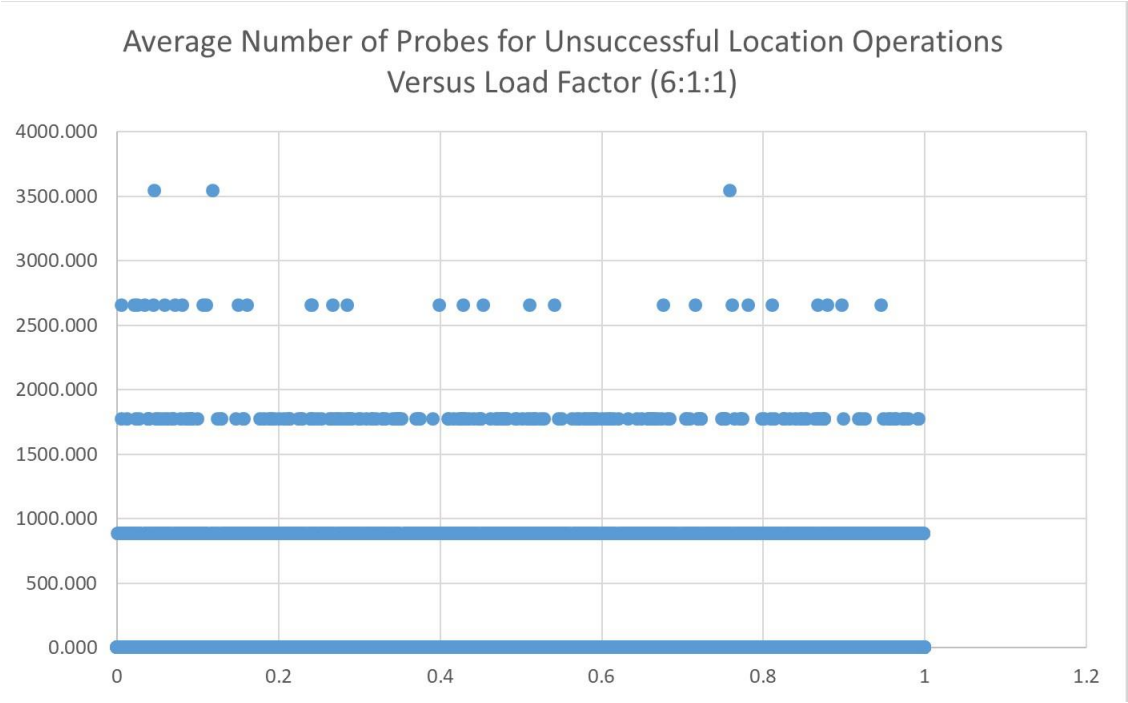



Average Number of Probes for Successful Deletion Operations Versus Load Factor (6:1:1)
4000.000
3500.000
3000.000
2500.000
2000.000
1500.000
1000.000
500.000
0.000
0
0.2
0.4
0.6
0.8
1
1.2


Average Number of Probes for Successful Location Operations Versus Load Factor (6:1:1)
6000
5000
4000
3000
2000
1000
0
0
0.2
0.4
0.6
0.8
1
1.2


Average Number of Probes for Unsuccessful Insertion Operations Versus Load Factor (6:1:1)
7000.00000
6000.00000
5000.00000
4000.00000
3000.00000
2000.00000
1000.00000
0.00000
0
0.2
0.4
0.6
0.8
1
1.2


Average Number of Probes for Unsuccessful Deletion Operations Versus Load Factor (6:1:1)
12000.000
10000.000
8000.000
6000.000
4000.000
2000.000
0.000
0
0.2
0.4
0.6
0.8
1
1.2



Part III: Observations & Analysis:

From the resulting graphs we can state that the average number of probes increase with respect to load factor, meaning that the hashing operation travels through more and more indexes as the table gets more and more full, as expected. Also, number of probes increase drastically for insertion and location operations for probability 2:1:5 and 4:2:2 whereas they do not take too much probes for probability 6:1:1 and the unsuccessful occurances of location operation for probability (2:1:5) increases drastically as the table gets more and more full

In addition to above information, it is noteworthy that it takes more and more number of probes for succesful insertions as the load factor increases whereas it takes more and more number of probes for unsuccesful deletions and find operations, yet as expected again.