CS300 HW3 – REPORT

1) CHANGES IN THE HASH TABLE CODE

- I've changed the findPos function for linear probing.
- I've changed the checking part so that it recognizes the deleted indexes.
- In new function I also return the probe number as reference variable.

```
int HashTable::findPos(const int & x, int & i) const
{
        int currentPos = myhash(x);

        while(array[currentPos].info == ACTIVE && array[currentPos].element != x)
        {
                currentPos++;
                i++;

                if (currentPos >= array.size())
                        currentPos -= array.size();
        }

        return currentPos;
}
```

- I've changed the contains, insert and remove function such that it returns an integer instead of a boolean value.
- The return value is negative if the operation has failed, and positive otherwise.
- The magnitude of the return vaue is the probe number I've obtained during findPos.

```
int HashTable::contains (const int & x) const
{
        int probe = 1;

        int dummy = findPos(x, probe);

        if (isActive(dummy))
                return probe;
        else
                return -1*probe;
}

int HashTable::insert (const int & x)
{
        int probe = 1;
        int currentPos = findPos(x, probe);

        if(isActive(currentPos))
                return -1*probe;

        array[currentPos].element = x;
        array[currentPos].info = ACTIVE;
        currentSize++;

        return probe;
}

int HashTable::remove(const int & x)
{
        int probe = 1;
        int currentPos = findPos(x, probe);

        if(!isActive(currentPos))
                return -1*probe;

        array[currentPos].info = DELETED;
        currentSize--;

        return probe;
}
```

- Also I've added 2 accessors for the current size and the total size.

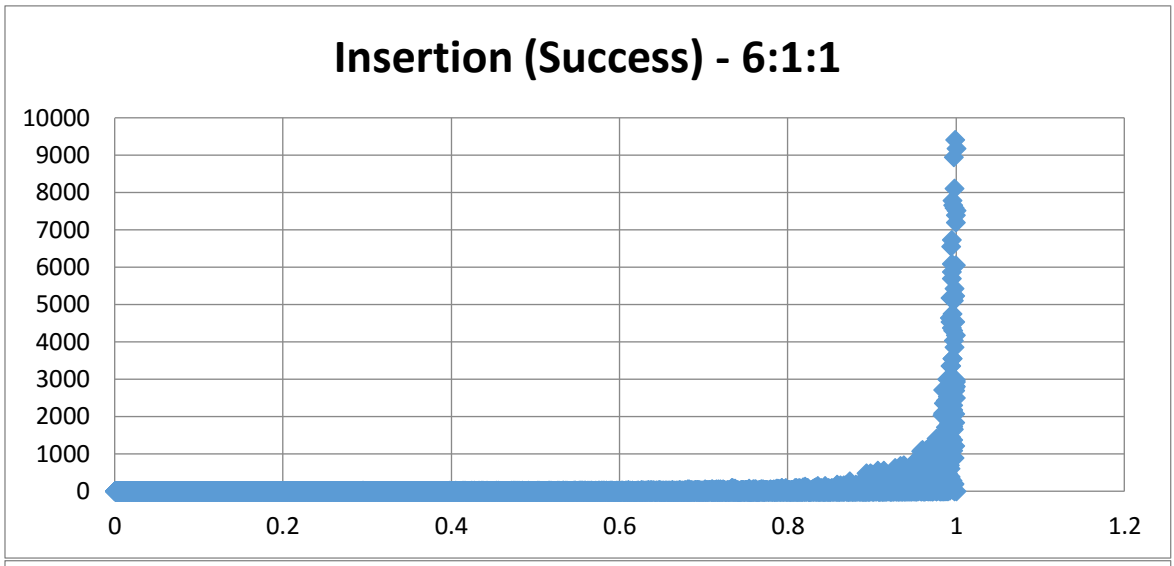
- Finally, I've write a function to get the next minimum prime number.

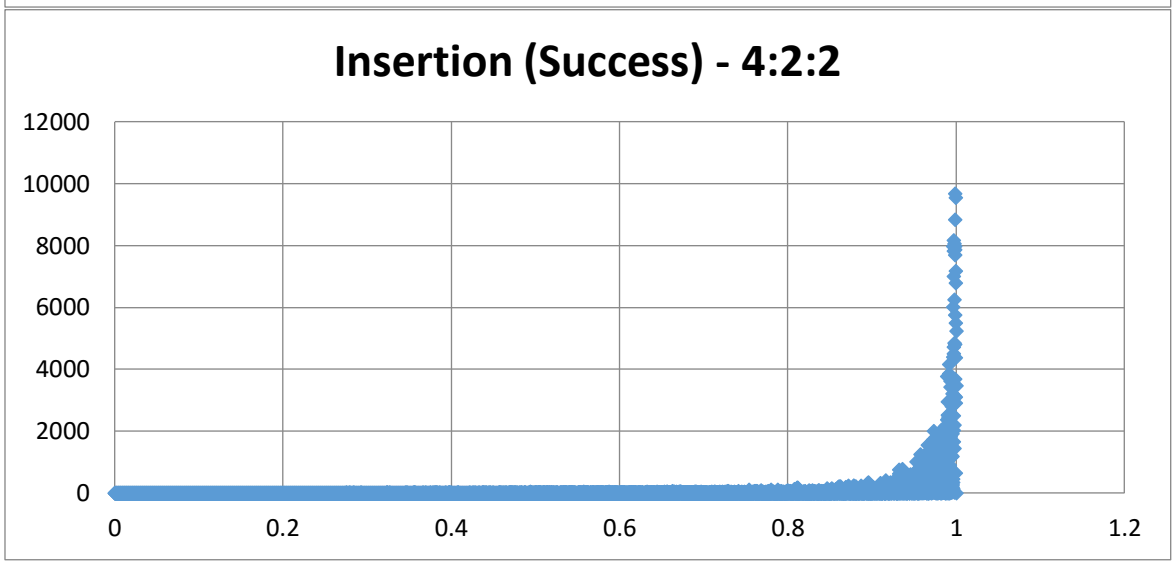
- My hash function is number%size.

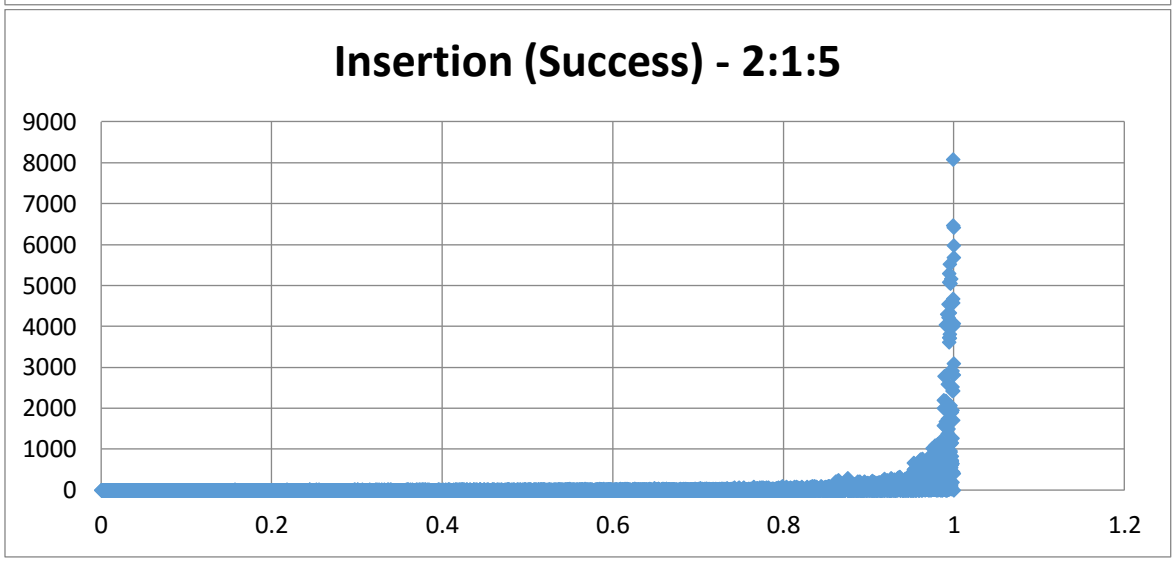
- For statistics I take 6 array for ins_suc, ins_fail, del_suc, del_fail, find_suc, find_fail.
- Each array has a size of (hashtable size + 1): 1 entry for every possible load factor
- Each entry of the array has 2 values: total probes and total transactions.
- In the end, I divide them and find the average number of probes for each load factors.

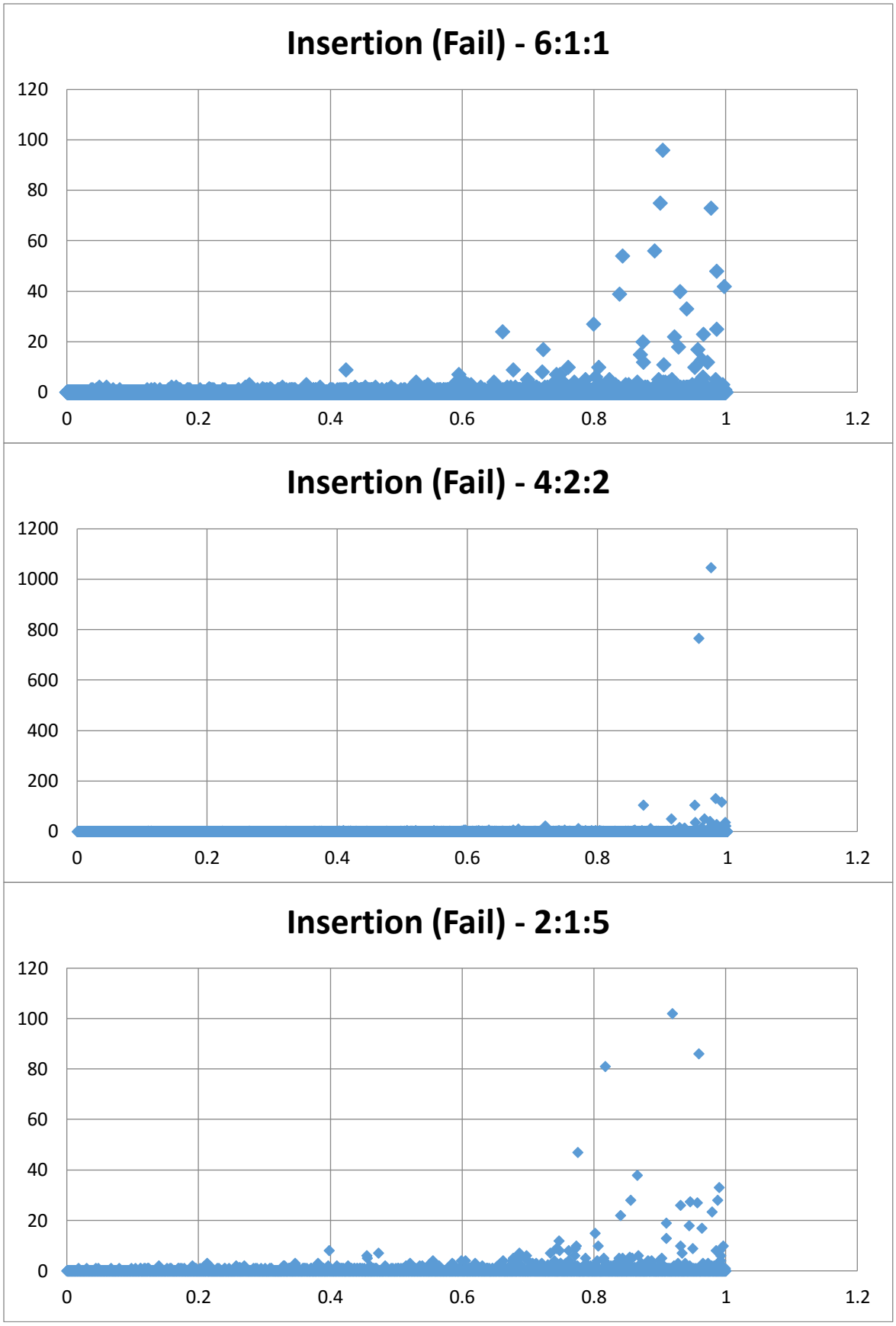
2) GRAPHS OBTAINED FROM THE STATISTICS

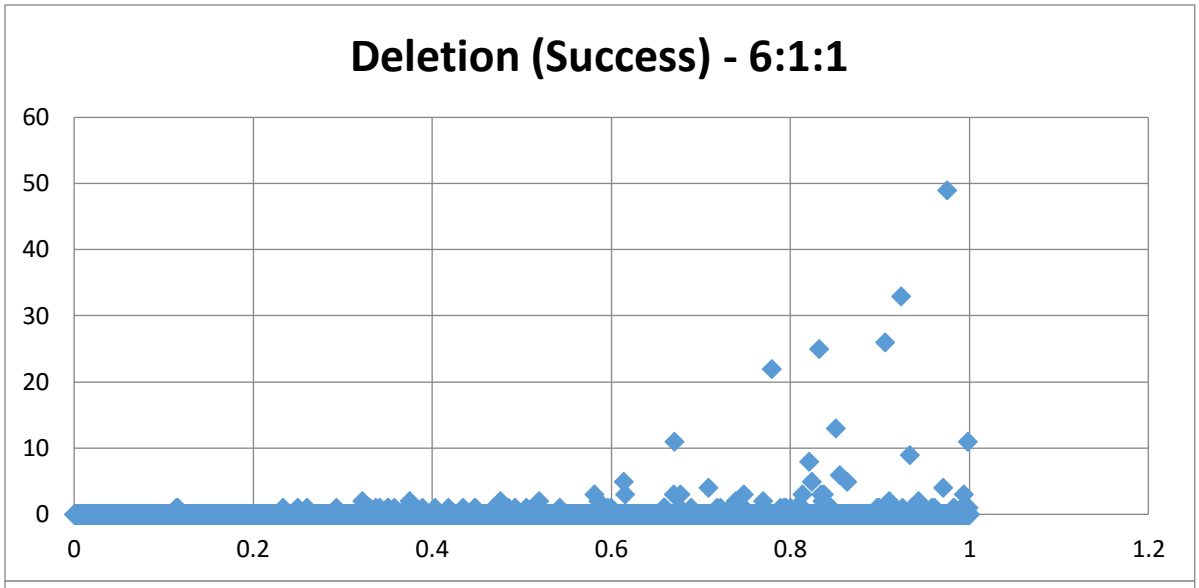
(Note: in all graphs: x-axis: load factor, y-axis: avg # of probes)

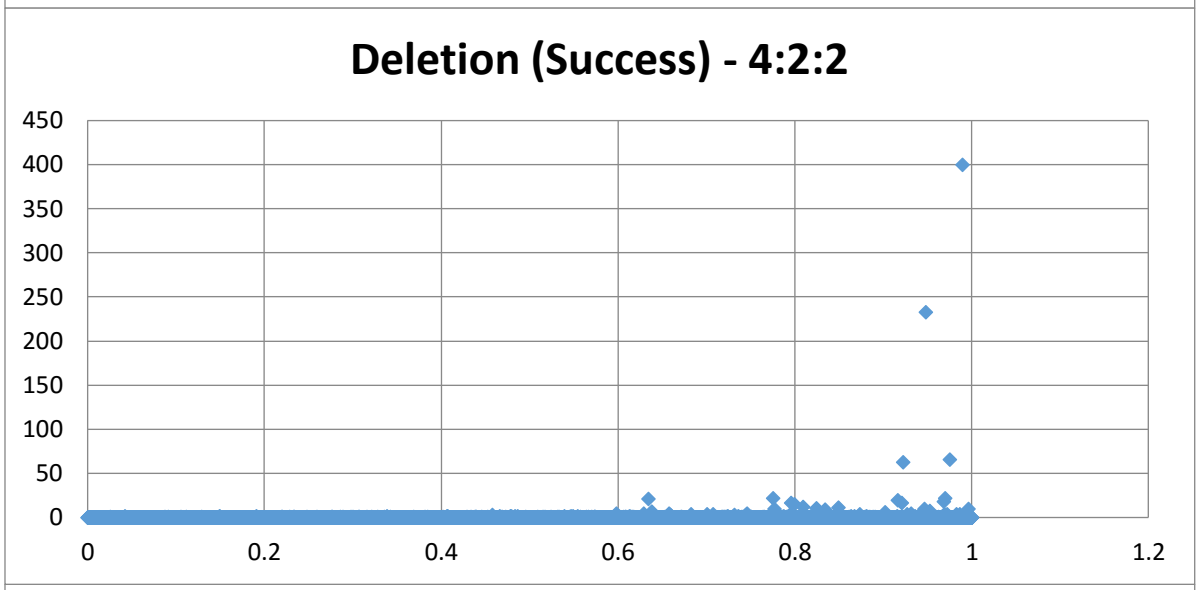


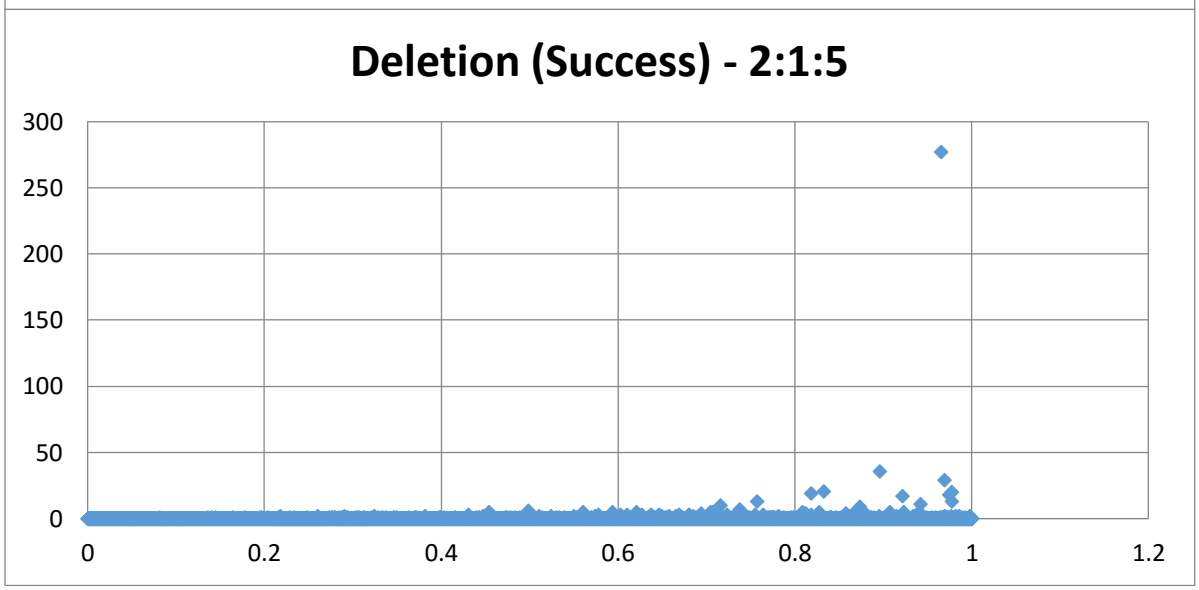


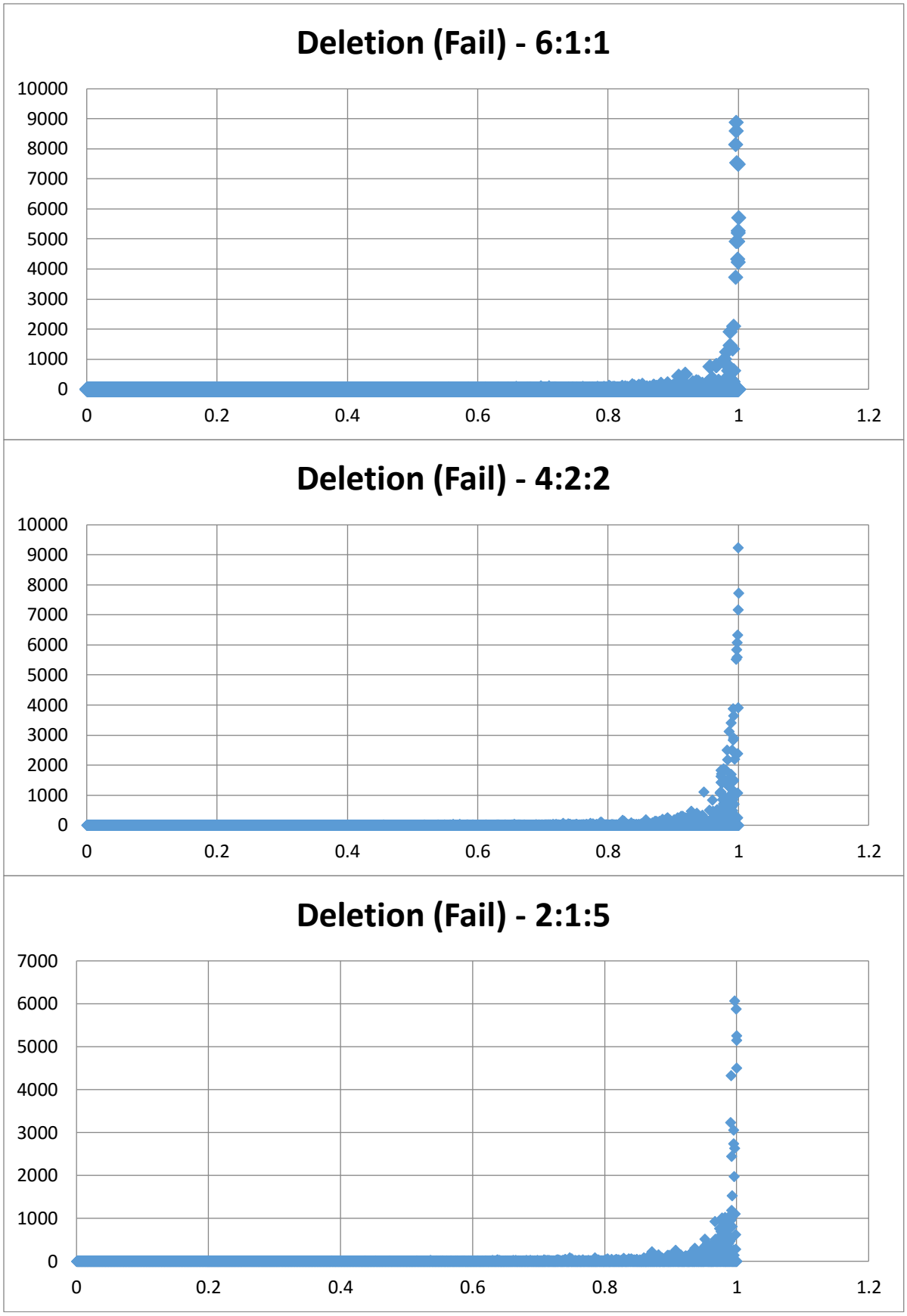


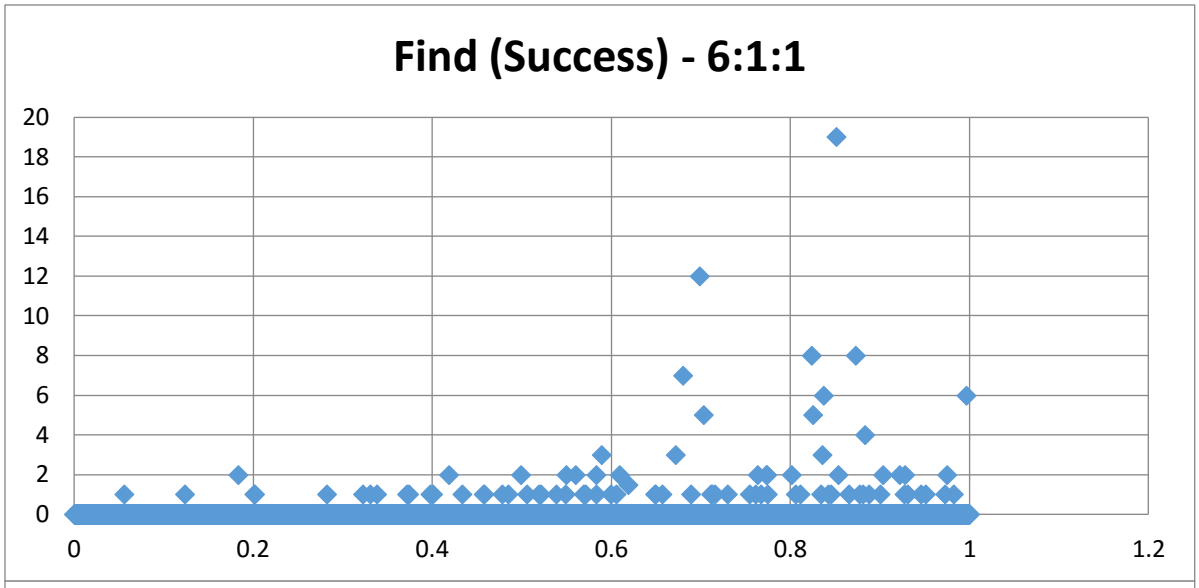

Insertion (Fail) - 6:1:1
120
100
80
60
40
20
0
0
0.2
0.4
0.6
0.8
1
1.2
Insertion (Fail) - 4:2:2
1200
1000
800
600
400
200
0
0
0.2
0.4
0.6
0.8
1
1.2
Insertion (Fail) - 2:1:5
120
100
80
60
40
20
0
0
0.2
0.4
0.6
0.8
1
1.2

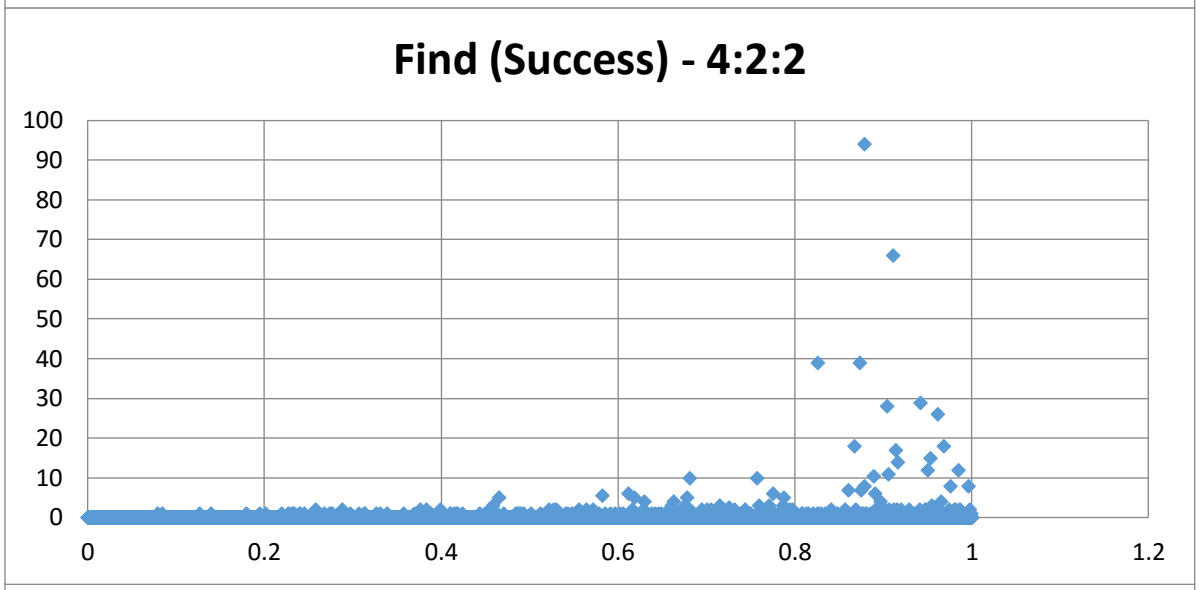

Deletion (Success) - 6:1:1
60
50
40
30
20
10
0
0
0.2
0.4
0.6
0.8
1
1.2

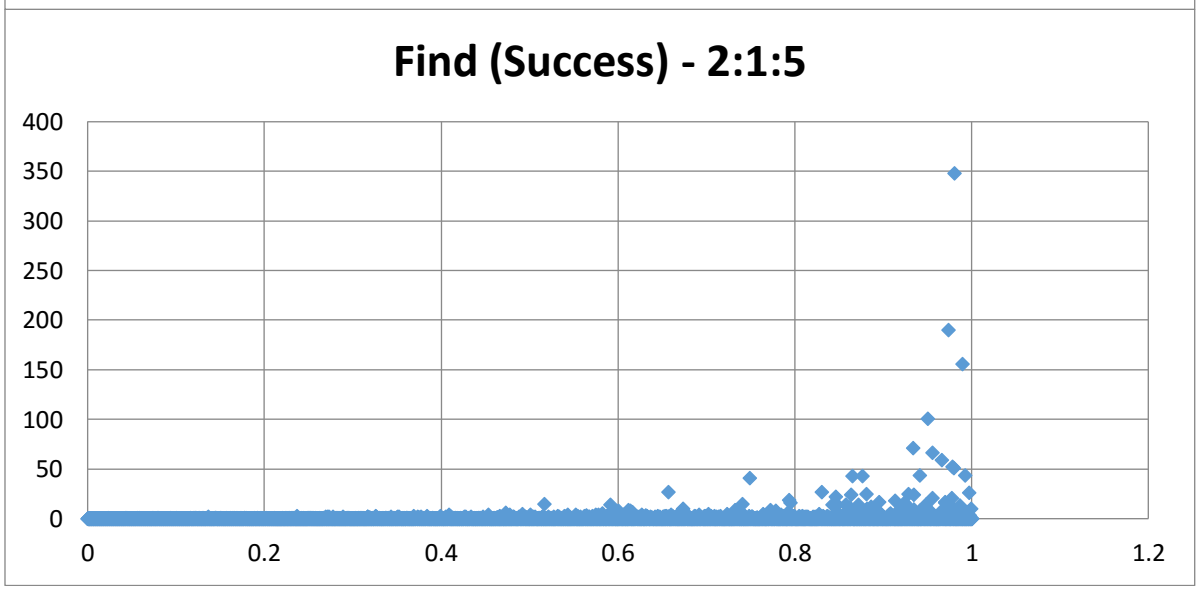

Deletion (Success) - 4:2:2
450
400
350
300
250
200
150
100
50
0
0
0.2
0.4
0.6
0.8
1
1.2

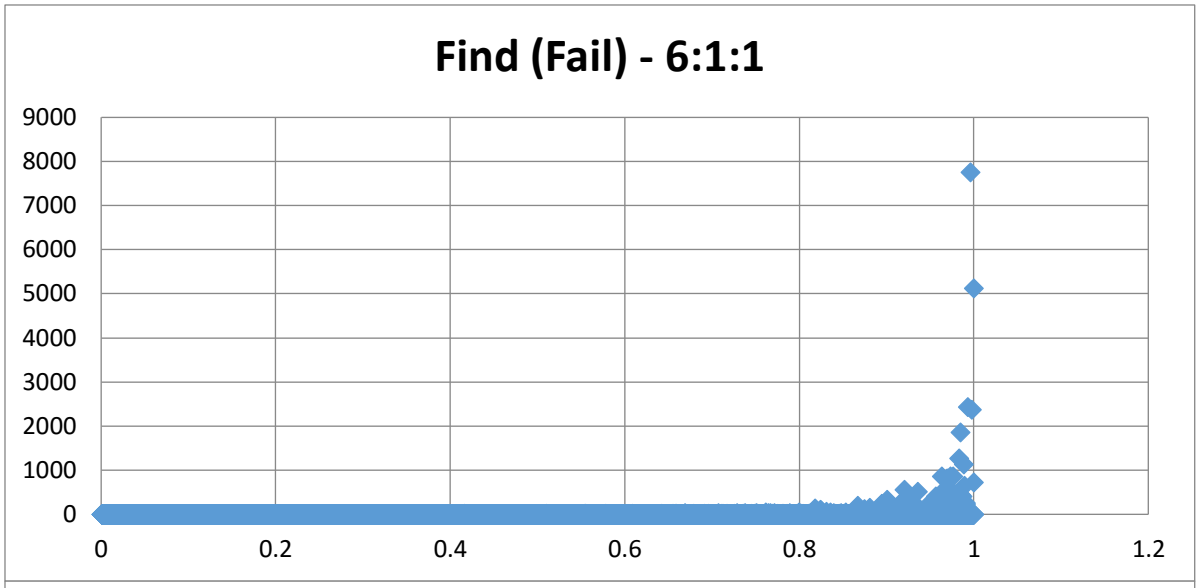

Deletion (Success) - 2:1:5
300
250
200
150
100
50
0
0
0.2
0.4
0.6
0.8
1
1.2

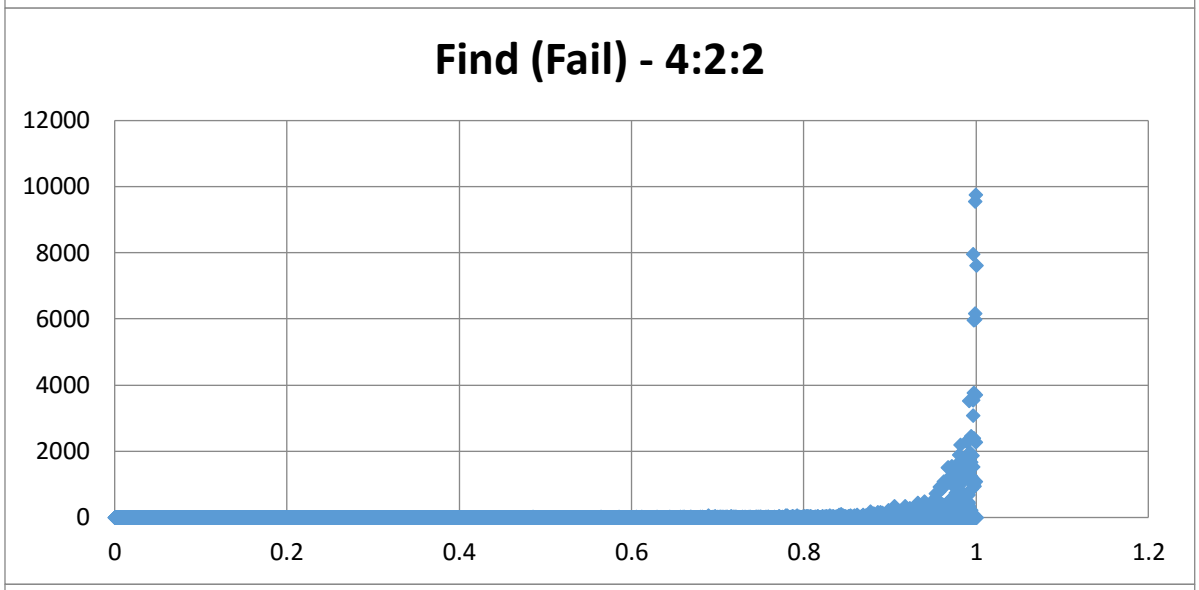

Deletion (Fail) - 6:1:1
10000
9000
8000
7000
6000
5000
4000
3000
2000
1000
0
0
0.2
0.4
0.6
0.8
1
1.2
Deletion (Fail) - 4:2:2
10000
9000
8000
7000
6000
5000
4000
3000
2000
1000
0
0
0.2
0.4
0.6
0.8
1
1.2
Deletion (Fail) - 2:1:5
7000
6000
5000
4000
3000
2000
1000
0
0
0.2
0.4
0.6
0.8
1
1.2

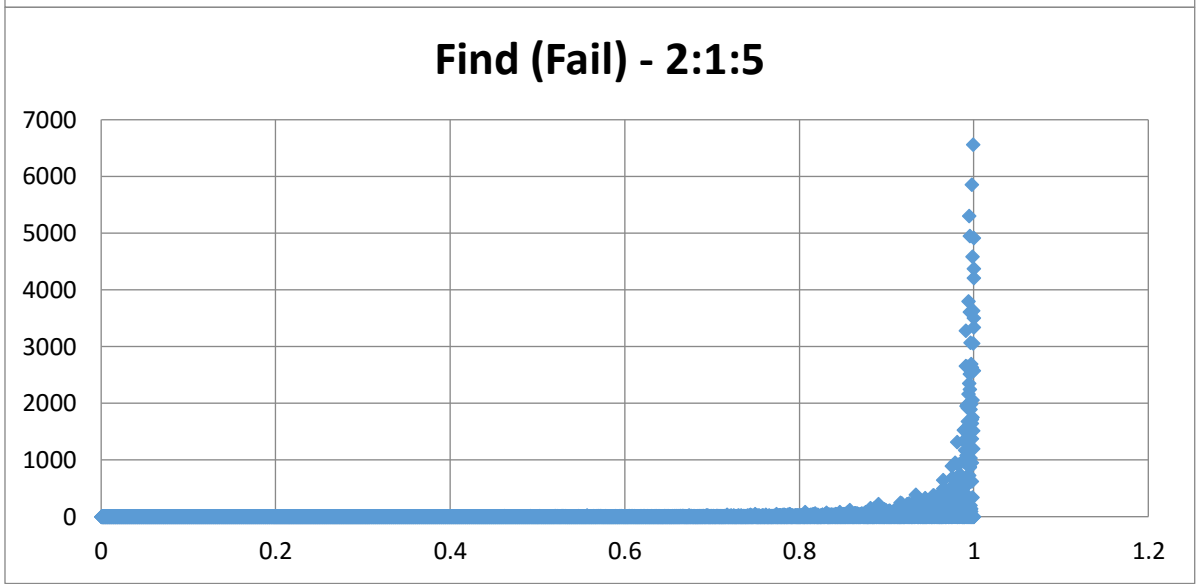

Find (Success) - 6:1:1
20
18
16
14
12
10
8
6
4
2
0
0
0.2
0.4
0.6
0.8
1
1.2


Find (Success) - 4:2:2
100
90
80
70
60
50
40
30
20
10
0
0
0.2
0.4
0.6
0.8
1
1.2


Find (Success) - 2:1:5
400
350
300
250
200
150
100
50
0
0
0.2
0.4
0.6
0.8
1
1.2


Find (Fail) - 6:1:1
9000
8000
7000
6000
5000
4000
3000
2000
1000
0
0
0.2
0.4
0.6
0.8
1
1.2


Find (Fail) - 4:2:2
12000
10000
8000
6000
4000
2000
0
0
0.2
0.4
0.6
0.8
1
1.2


Find (Fail) - 2:1:5
7000
6000
5000
4000
3000
2000
1000
0
0
0.2
0.4
0.6
0.8
1
1.2

3) CONCLUSIONS

When we look at the graphs, we see that in all graphs, avg number of probes increases as the load factor increases.
This indicates that when the load factor increases (get closer to 1), we go over many indexes in each operation instead of 1.
Therefore, we need to increase the capacity and rehash the table as the load factor increases an passes a certain level.

Another conclusion that we can derive from the graphs is that in find and delete operations, it takes much number of probes when fail, however in insertion the opposite is true.

Also, find and insert opertaions take much more probes in successful 4:2:2 and 2:1:5 than 6:1:1.

The main reason for last 2 results is that when the insertion/deletion ratio decreases, hash table becomes much more randomized, therefore number of probes increase in find and delete operations especially when there is a success.

BERKAN TEBER 19080