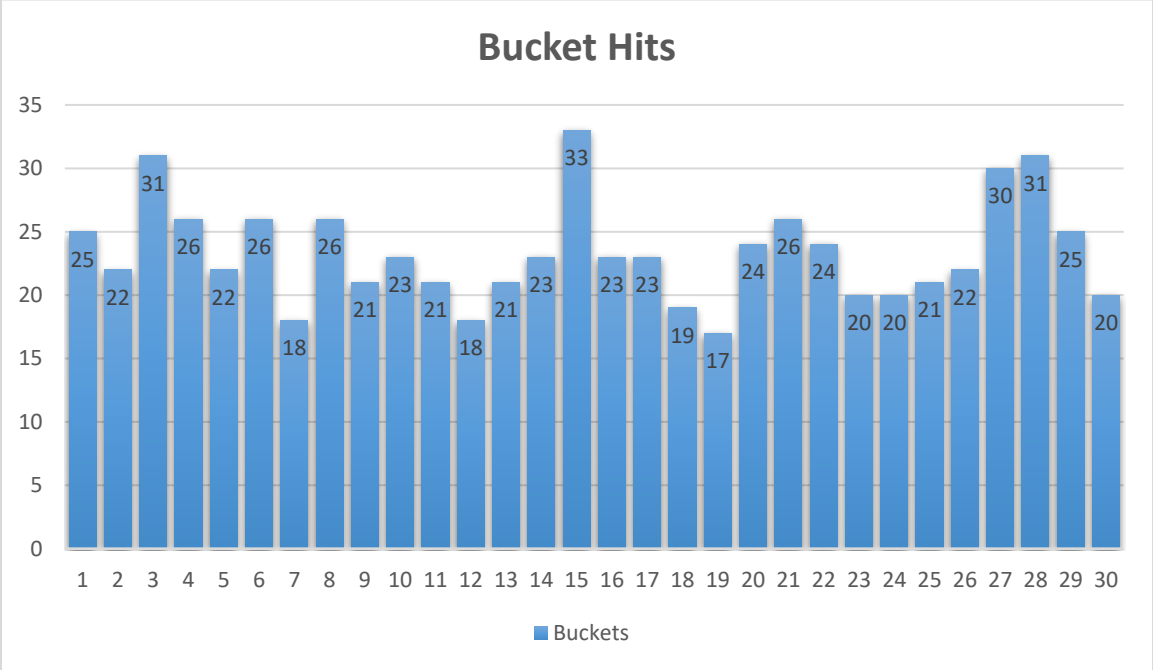
E/14/317

CO322 – Lab01

1) For sample-text2.txt file

a) Hash Code 1 for bucket size 30

```
private int hash(String key)
{
        int h = 0;
        for(int i = 0; i < key.length(); i++)
        {
           //multiplied with (31) an odd number
           h = (31* h + key.charAt(i))%table.length;
        }
        return (h%table.length);
}
```




Minimum Hits: 17

Maximum Hits: 31

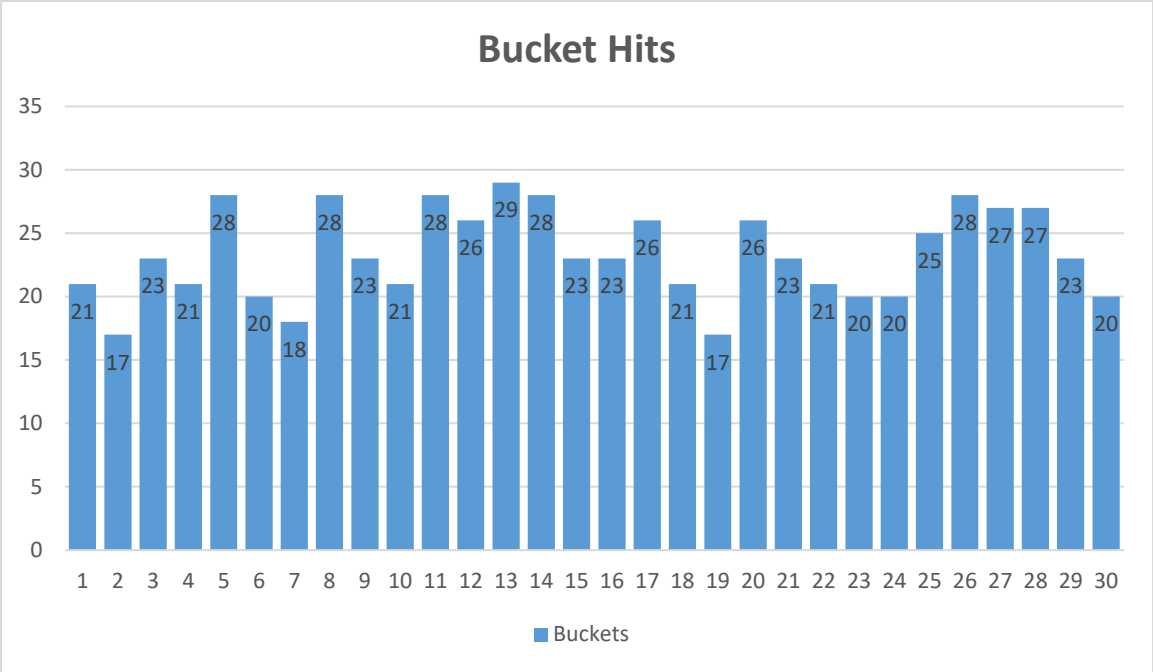
Average hits: 23.3667

Standard Deviation: 3.8728

E/14/317

b) Hash Code 2 for bucket size 30

```
private int hash(String key)
{
        int h = 0;
        for (int i = 0; i < key.length(); i++)
        {
           //multiplied with (71) an odd number
           h = (71* h + key.charAt(i))%table.length;
        }
        return (h%table.length);
}
```




Minimum Hits: 17

Maximum Hits: 29

Average hits: 23.3667

Standard Deviation: 3.4639

When we compare hash codes in part a & b we can see that average hits per bucket is same. Although both hash codes were able to distribute keys through buckets in an almost uniform manner, when consider minimum, maximum hits and Standard Deviation of hits/bucket, hash code 2 outperforms hash code 1.

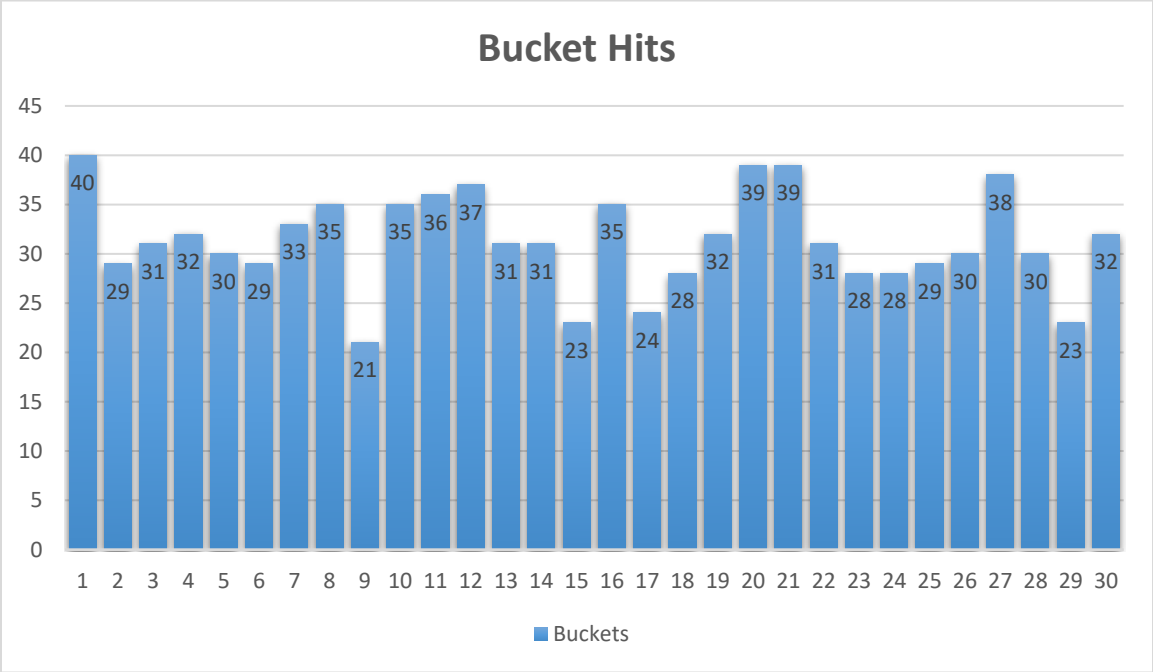
Thus we can say that in a limited memory system where number of collisions in a bucket is not a critical issue, hash code 2 will be more suitable to expect a better performance of the hash table.

E/14/317

2) For sample-text1.txt file

a) Hash Code 1 for bucket size 30

```
private int hash(String key)
{
        int h = 0;
        for(int i = 0; i < key.length(); i++)
        {
           //multiplied with (31) an odd number
           h = (31* h + key.charAt(i))%table.length;
        }
        return (h%table.length);
}
```




Minimum Hits: 21

Maximum Hits: 40

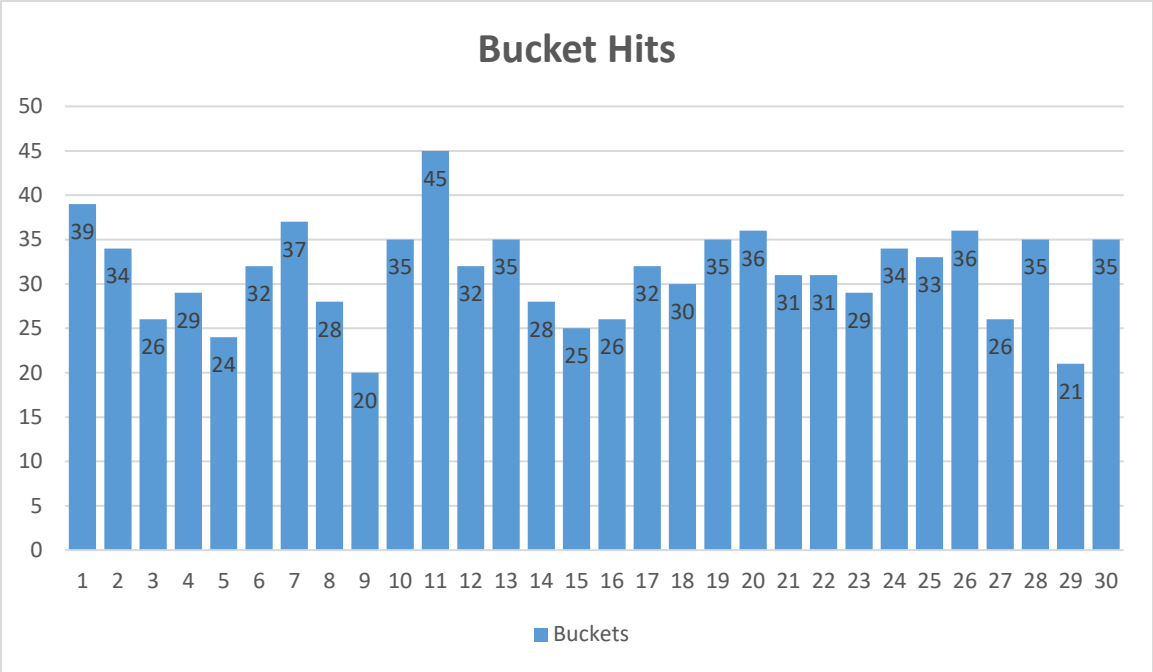
Average hits: 31.3

Standard Deviation: 4.7233

E/14/317

b) Hash Code 2 for bucket size 30

```
private int hash(String key)
{
        int h = 0;
        for (int i = 0; i < key.length(); i++)
        {
           //multiplied with (71) an odd number
           h = (71* h + key.charAt(i))%table.length;
        }
        return (h%table.length);
}
```




Minimum Hits: 20

Maximum Hits: 45

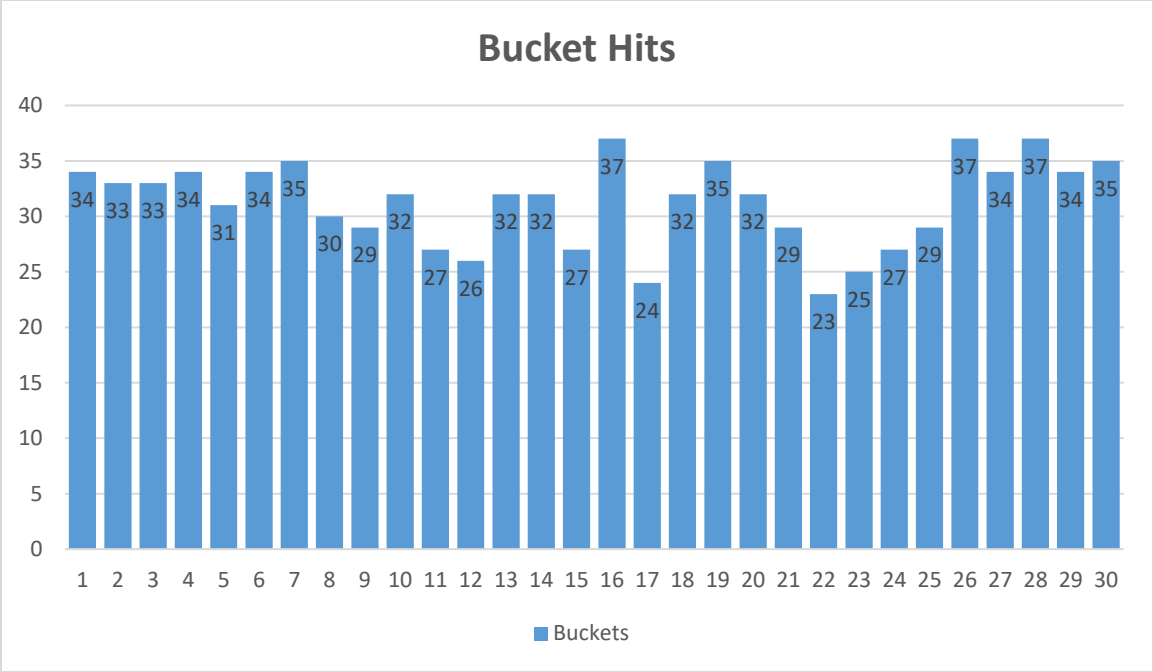
Average hits: 31.3

Standard Deviation: 5.3207

E/14/317

c) Hash Code 3 for bucket size 30

```
private int hash(String key)
{
        int h = 0;
        for (int i = 0; i < key.length(); i++)
        {
           //multiplied with (43,73,163,223,313,373,463,523,613,733,823….) an odd number
           h = (523* h + key.charAt(i))%table.length;
        }
        return (h%table.length);
}
```




Minimum Hits: 23

Maximum Hits: 37

Average hits: 31.3

Standard Deviation: 3.7828

When the same hash function is used for different files it has given different results since the complexity of vocabulary used in 2 files is almost completely different therefore the domain of generated hash codes are also completely different. However despite the complexity of 2 files when an odd number is used as the multiplier in the hash function better uniformity has been occurred than when an even number is used. Also several odd numbers has given the best uniformity above all other odd multipliers. Part C is an example for such situation.

Of course all above values changes for different bucket sizes. Here bucket size 30 is considered for all the situations for the easiness of comparing.