Department of Computer Engineering University of Peradeniya

CO 322 Data Structures and Algorithms

Lab 03 - Hash Tables

Name : Jayathilaka H.A.D.T.T.

E.Number : E/16/156

Semester : 5

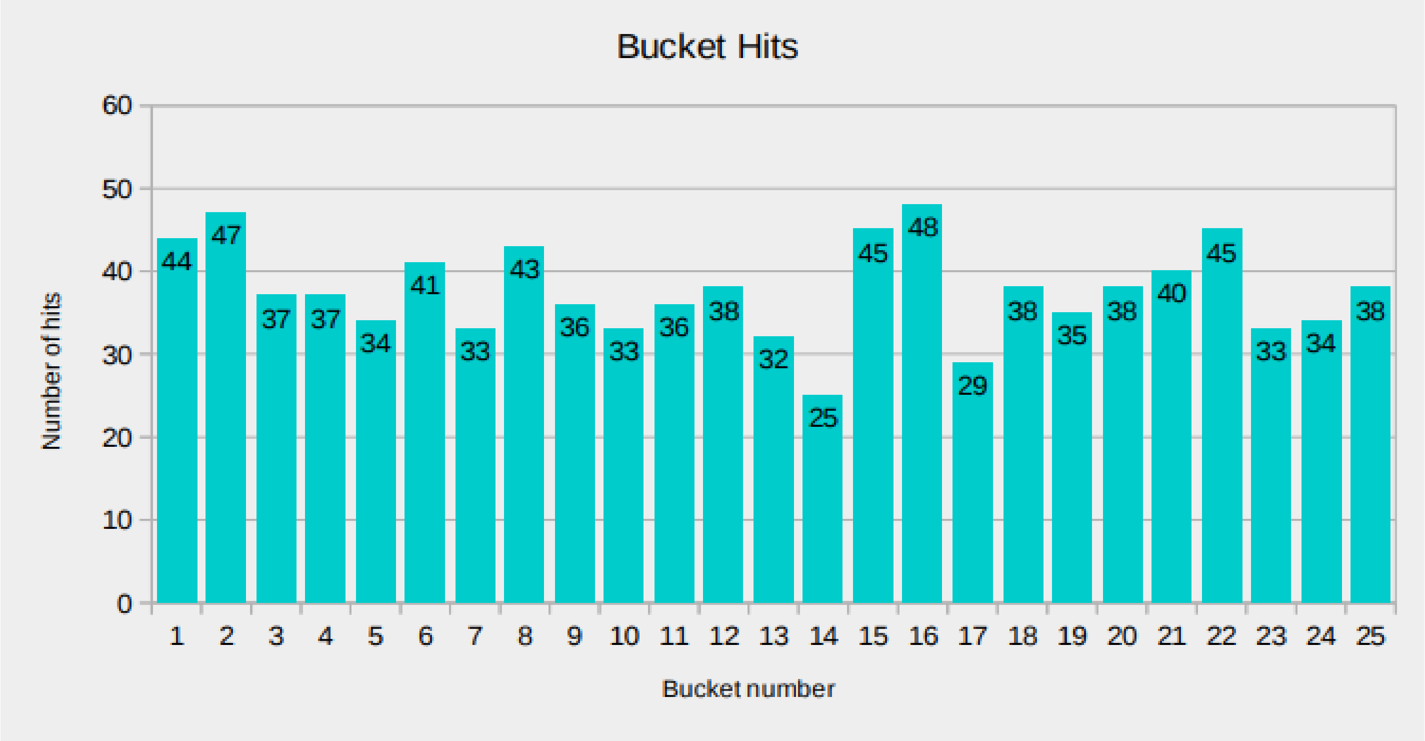
Date : 13/08/2020

1) For sample-text1.txt file

a) Modified Hash Table 1

```
// Division Method
public int valueHash(String key){
        int asciiValue = 0;
        // get the sum of ascii values
        // since there are 128 basic ascii values should get the power of 128
        for(int j=0; j<key.length(); j++){
                // for each character should get he relevant ascii value and should get the sum of them
                asciiValue = (31* asciiValue + key.charAt(j))%this.buckets;
        }
        // h(k) = k mod m
        // So the remainder should become hash value
        return asciiValue%this.buckets;
}
```

i) For Modified Hash Code 1 when bucket size 25



Modified method

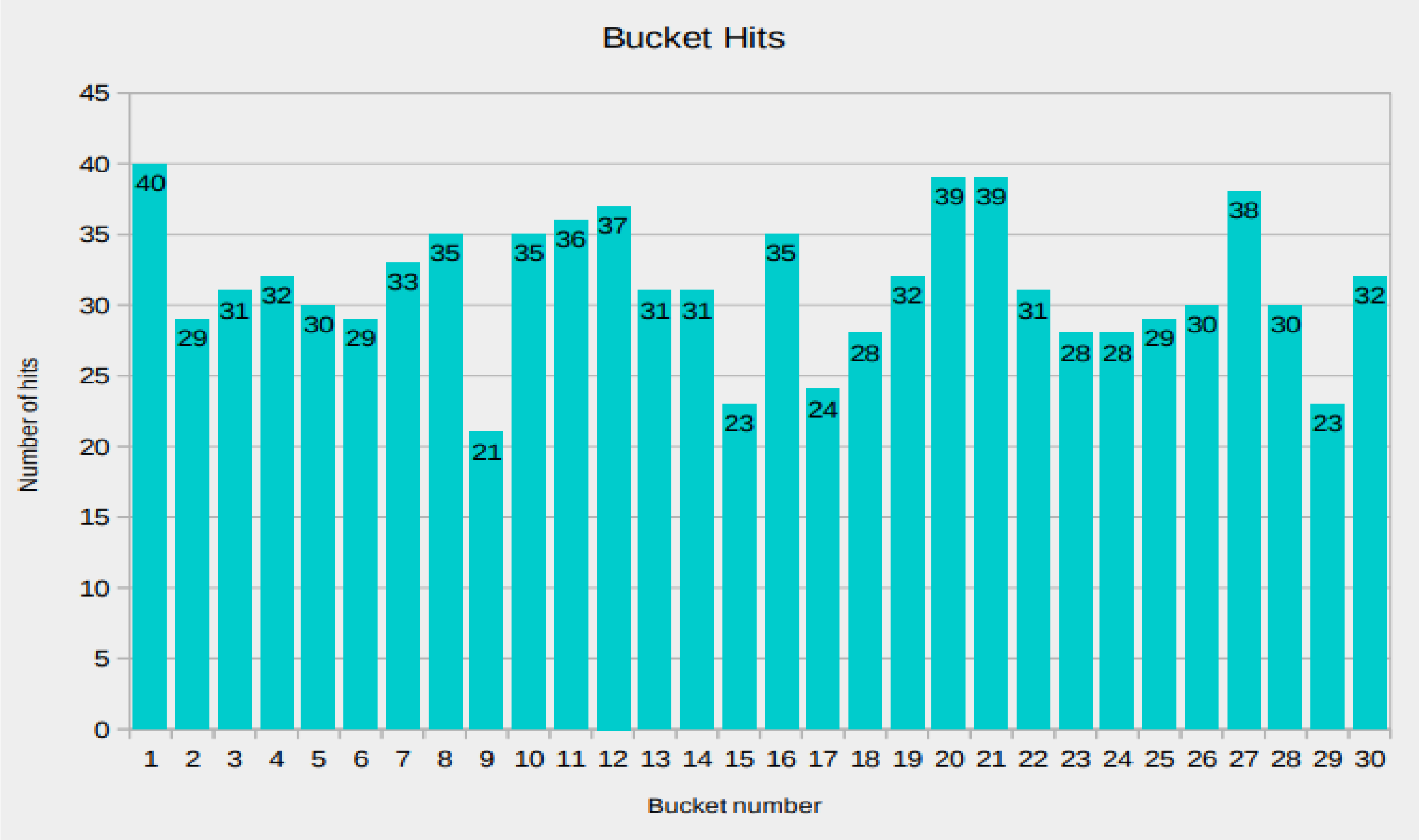
Minimum number of entries in a bucket: 25

Max number of entries in a bucket: 48

Average: 34.760000

Standard deviation: 6.194884

ii) For Modified Hash Code 1 when bucket size 30



Modified method

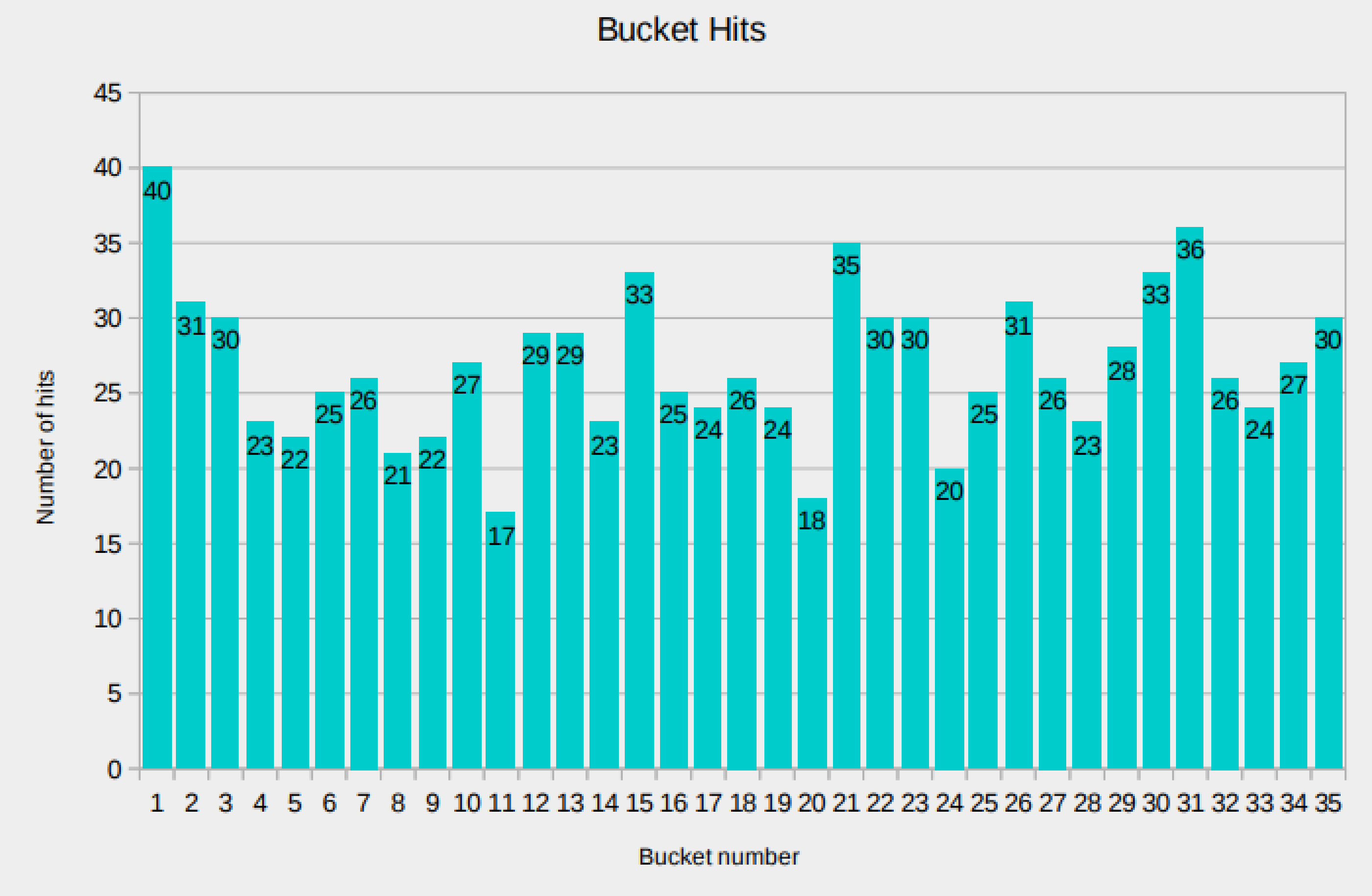
Minimum number of entries in a bucket: 21

Max number of entries in a bucket: 40

Average: 28.966667

Standard deviation: 7.258410

iii) For Modified Hash Code 1 when bucket size 35



Modified method

Minimum number of entries in a bucket: 18

Max number of entries in a bucket: 37

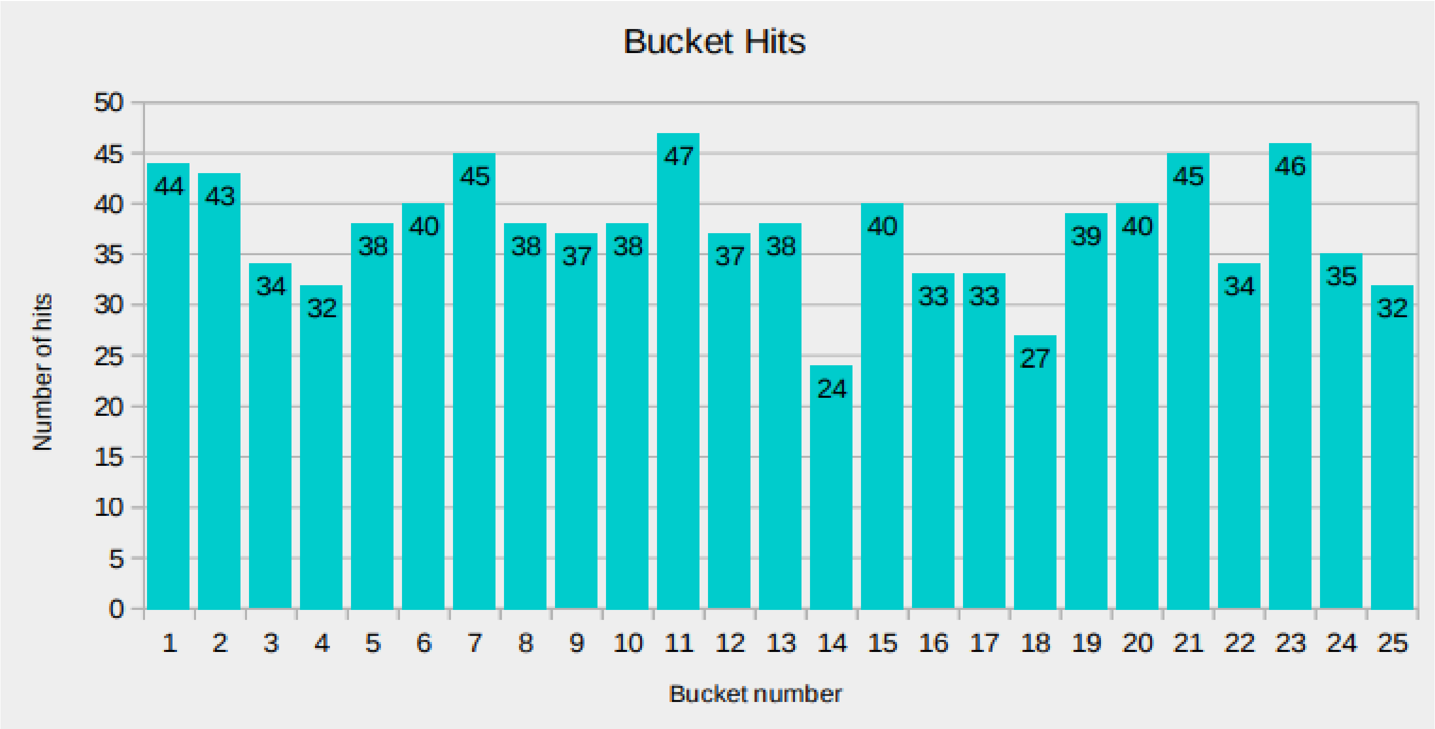
Average: 24.828571

Standard deviation: 8.750637

b) Modified Hash Table 2

```
// Division Method
public int valueHash(String key){
        int asciiValue = 0;
        // get the sum of ascii values
        // since there are 128 basic ascii values should get the power of 128
        for(int j=0; j<key.length(); j++){
                // for each character should get he relevant ascii value and should get the sum of them
                asciiValue = (71* asciiValue + key.charAt(j))%this.buckets;
        }
        // h(k) = k mod m
        // So the remainder should become hash value
        return asciiValue%this.buckets;
}
```

iv) For Modified Hash Code 2 when bucket size 25



Modified method

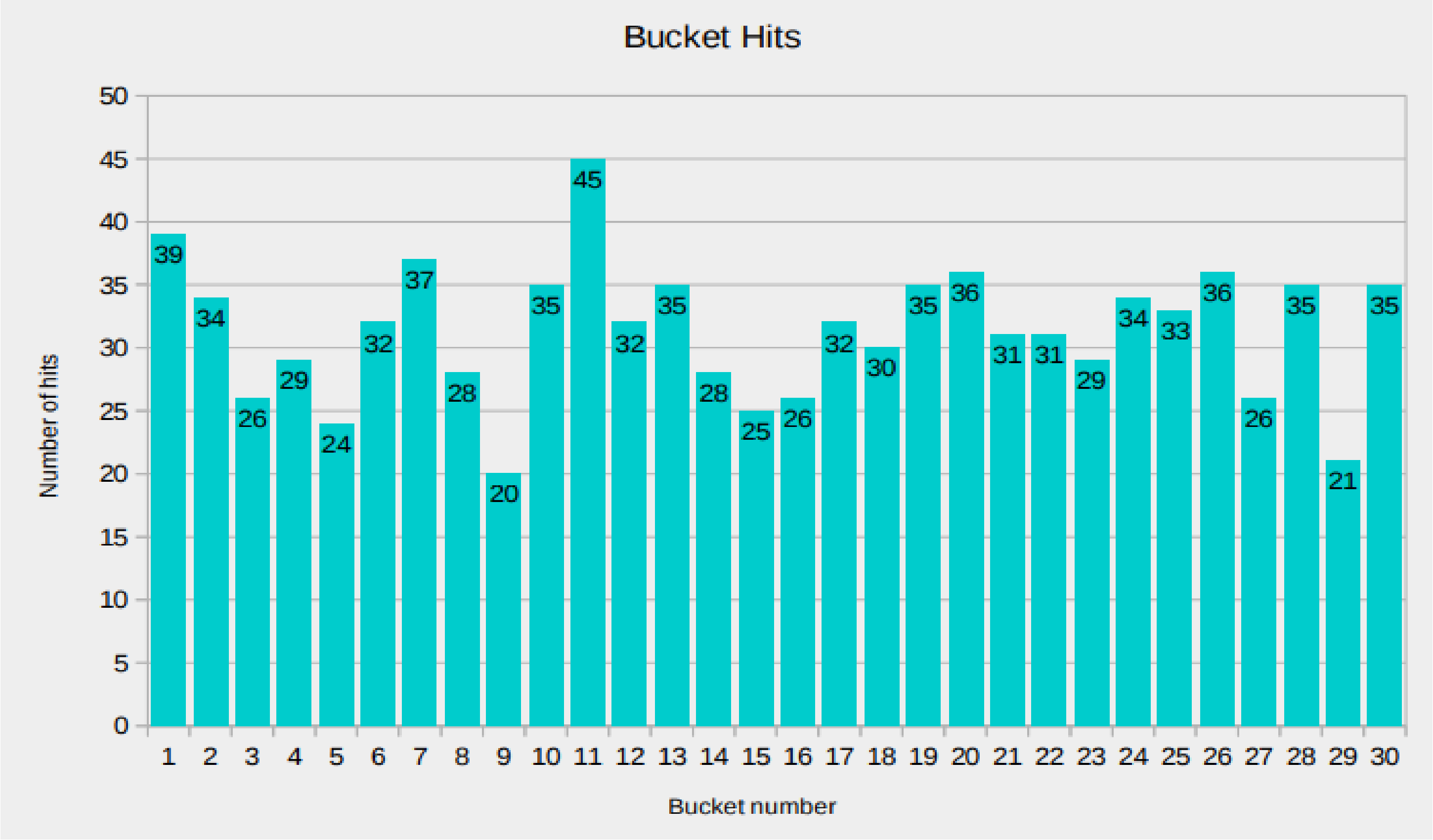
Minimum number of entries in a bucket: 24

Max number of entries in a bucket: 47

Average: 34.760000

Standard deviation: 6.139676

v) For Modified Hash Code 2 when bucket size 30



Modified method

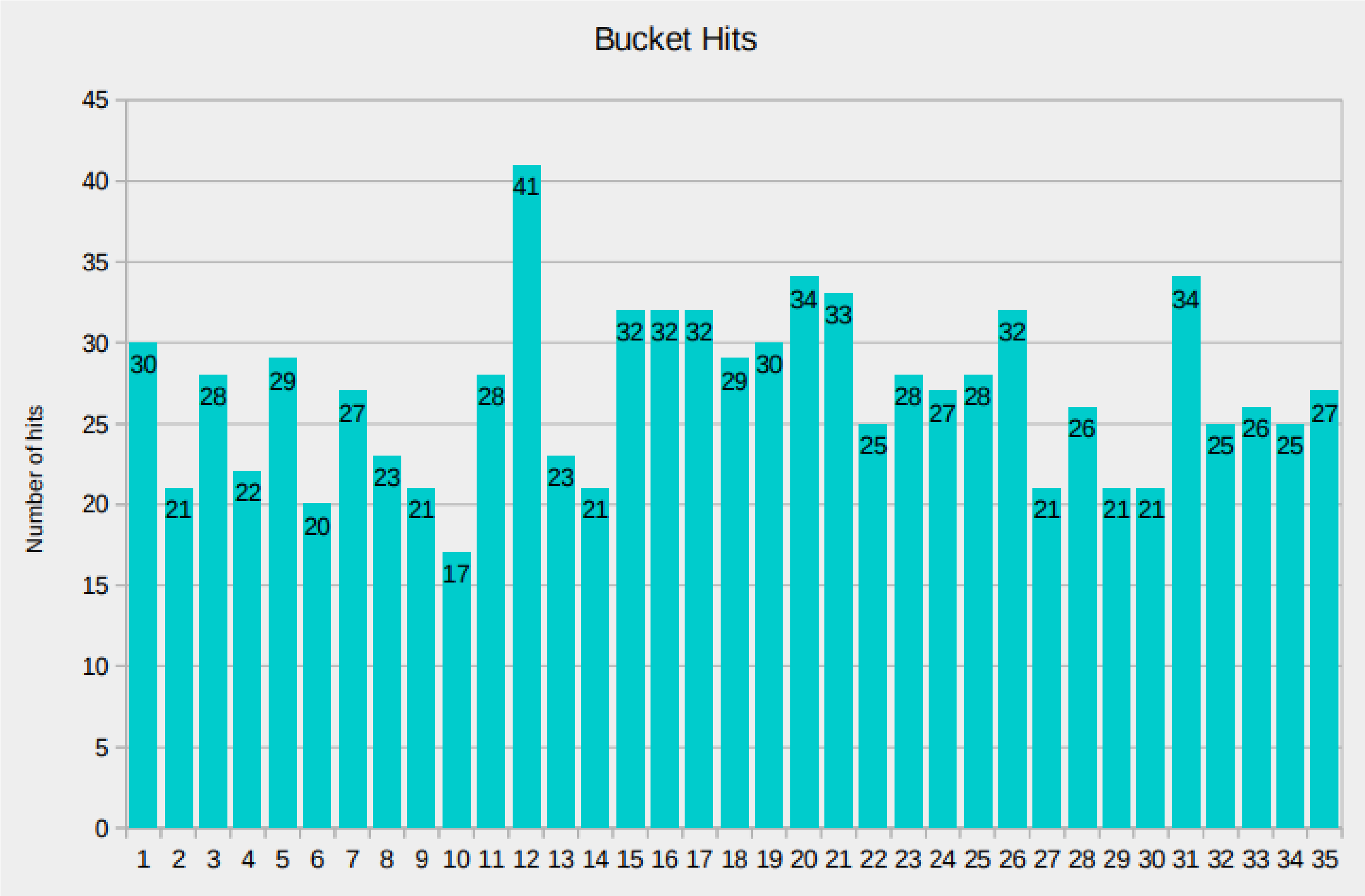
Minimum number of entries in a bucket: 20

Max number of entries in a bucket: 45

Average: 28.966667

Standard deviation: 7.389157

vi) For Modified Hash Code 2 when bucket size 35



Modified method

Minimum number of entries in a bucket: 17

Max number of entries in a bucket: 41

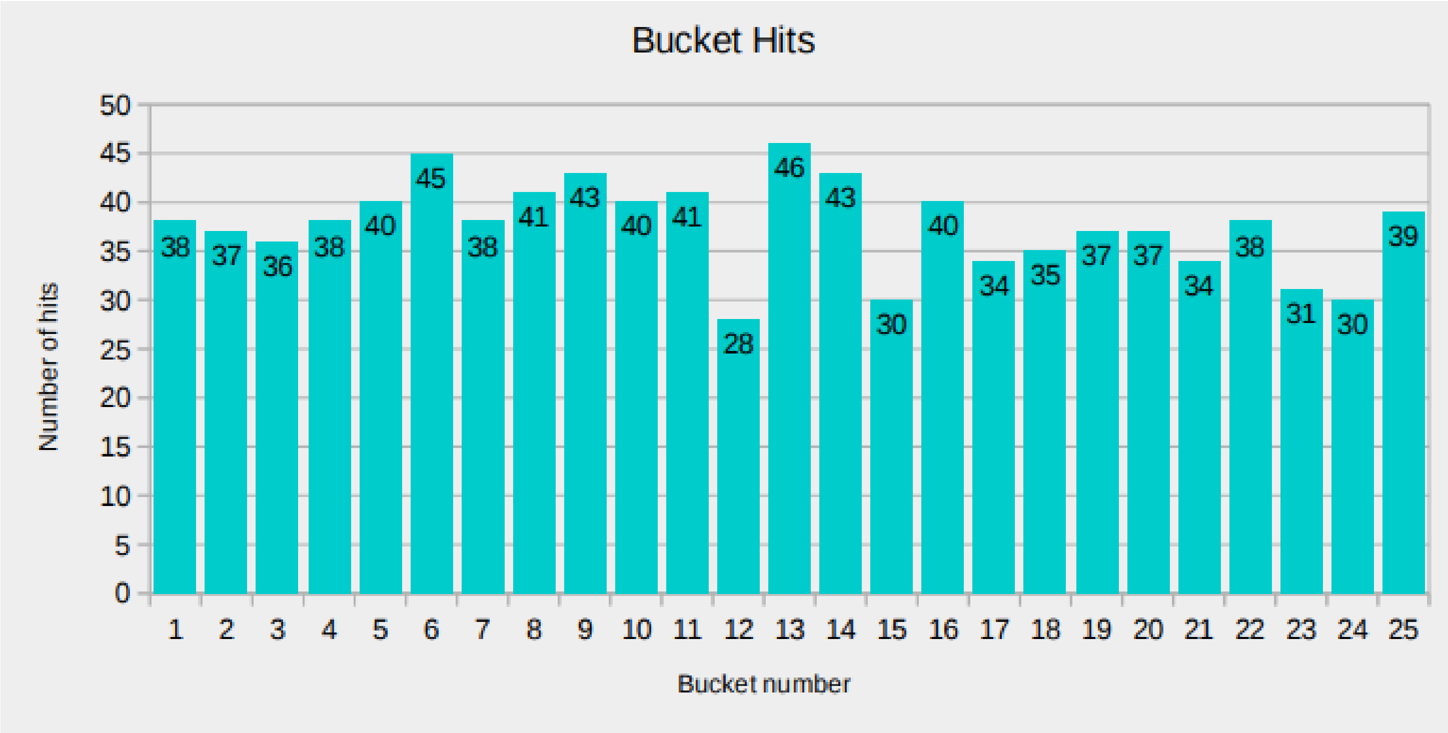
Average: 24.828571

Standard deviation: 8.615281

c) Modified Hash Table 3

```
// Division Method
public int valueHash(String key){
        int asciiValue = 0;
        // get the sum of ascii values
        // since there are 128 basic ascii values should get the power of 128
        for(int j=0; j<key.length(); j++){
                // for each character should get he relevant ascii value and should get the sum of them
                asciiValue = (523* asciiValue + key.charAt(j))%this.buckets;
        }
        // h(k) = k mod m
        // So the remainder should become hash value
        return asciiValue%this.buckets;
}
```

vii) For Modified Hash Code 3 when bucket size 25



Modified method

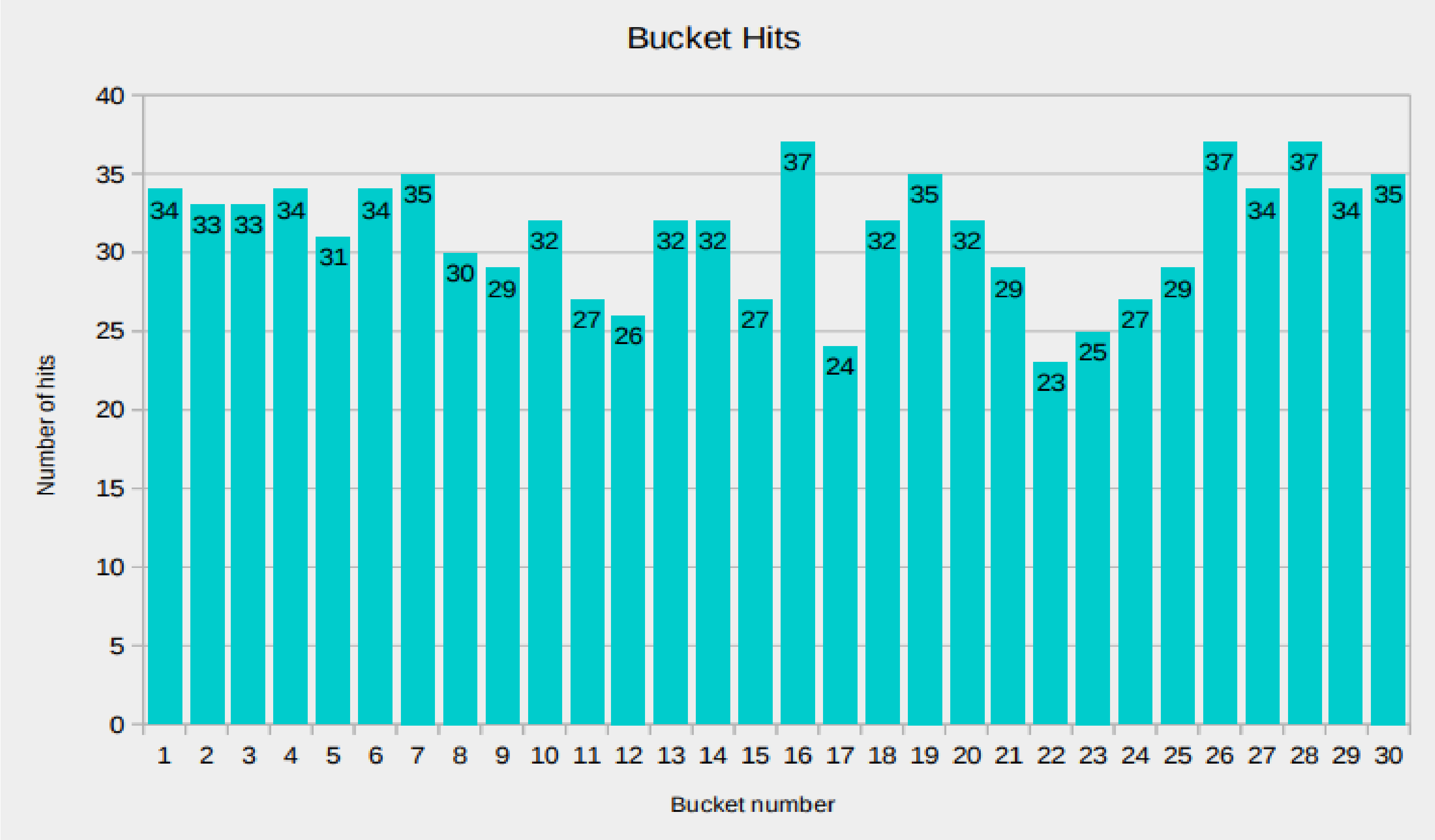
Minimum number of entries in a bucket: 28

Max number of entries in a bucket: 46

Average: 34.760000

Standard deviation: 6.014350

viii) For Modified Hash Code 3 when bucket size 30



Modified method

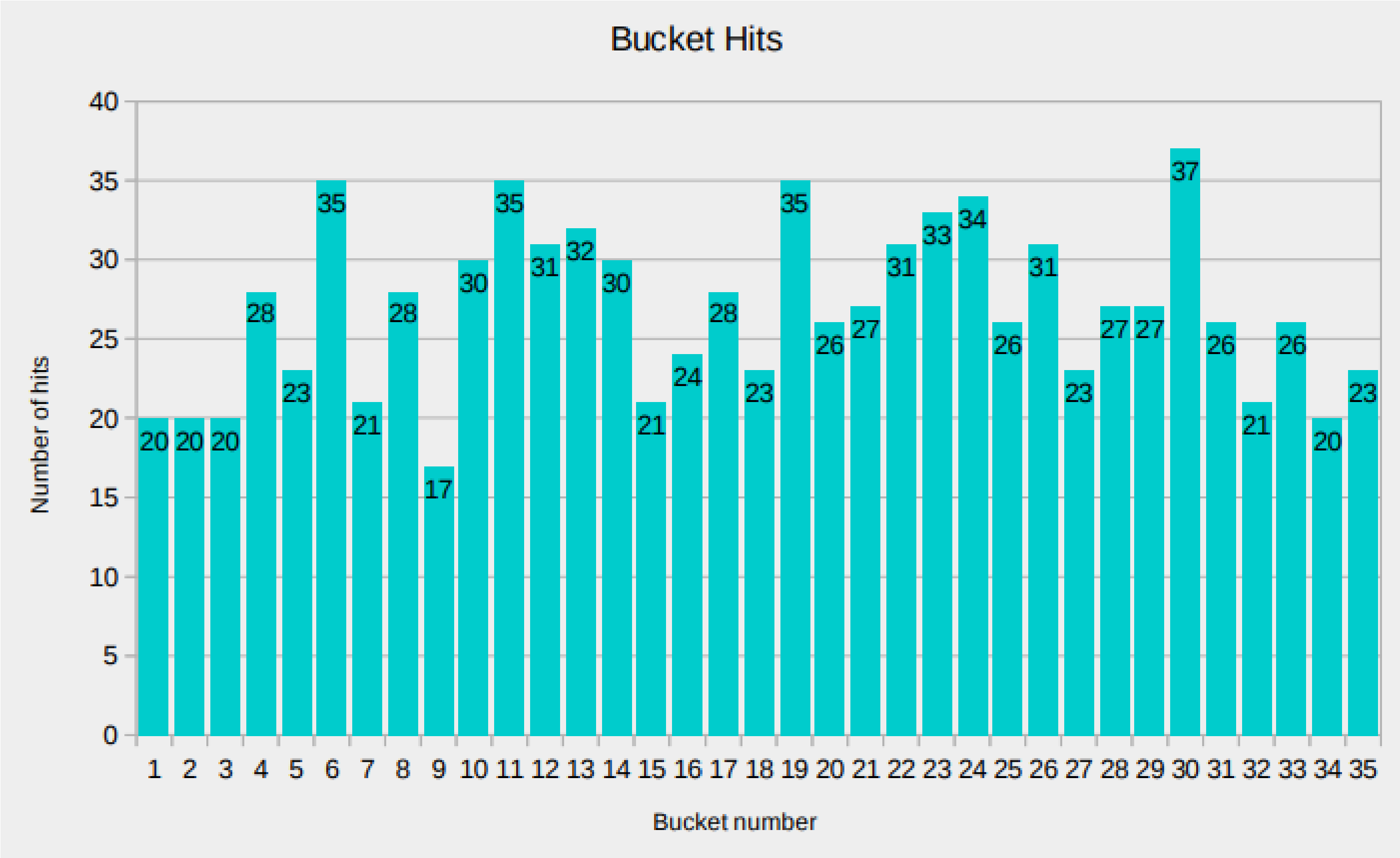
Minimum number of entries in a bucket: 23

Max number of entries in a bucket: 37

Average: 28.966667

Standard deviation: 7.396129

ix) For Modified Hash Code 3 when bucket size 35



Modified method

Minimum number of entries in a bucket: 17

Max number of entries in a bucket: 37

Average: 24.828571

Standard deviation: 8.490647

When we compare hash codes 1,2 & 3 we can see that average hits per bucket is same for all hash codes when number of buckets are fixed. Also three 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 3 has better performances when number of buckets are 25.

hash code 3 has better performances when number of buckets are 30.

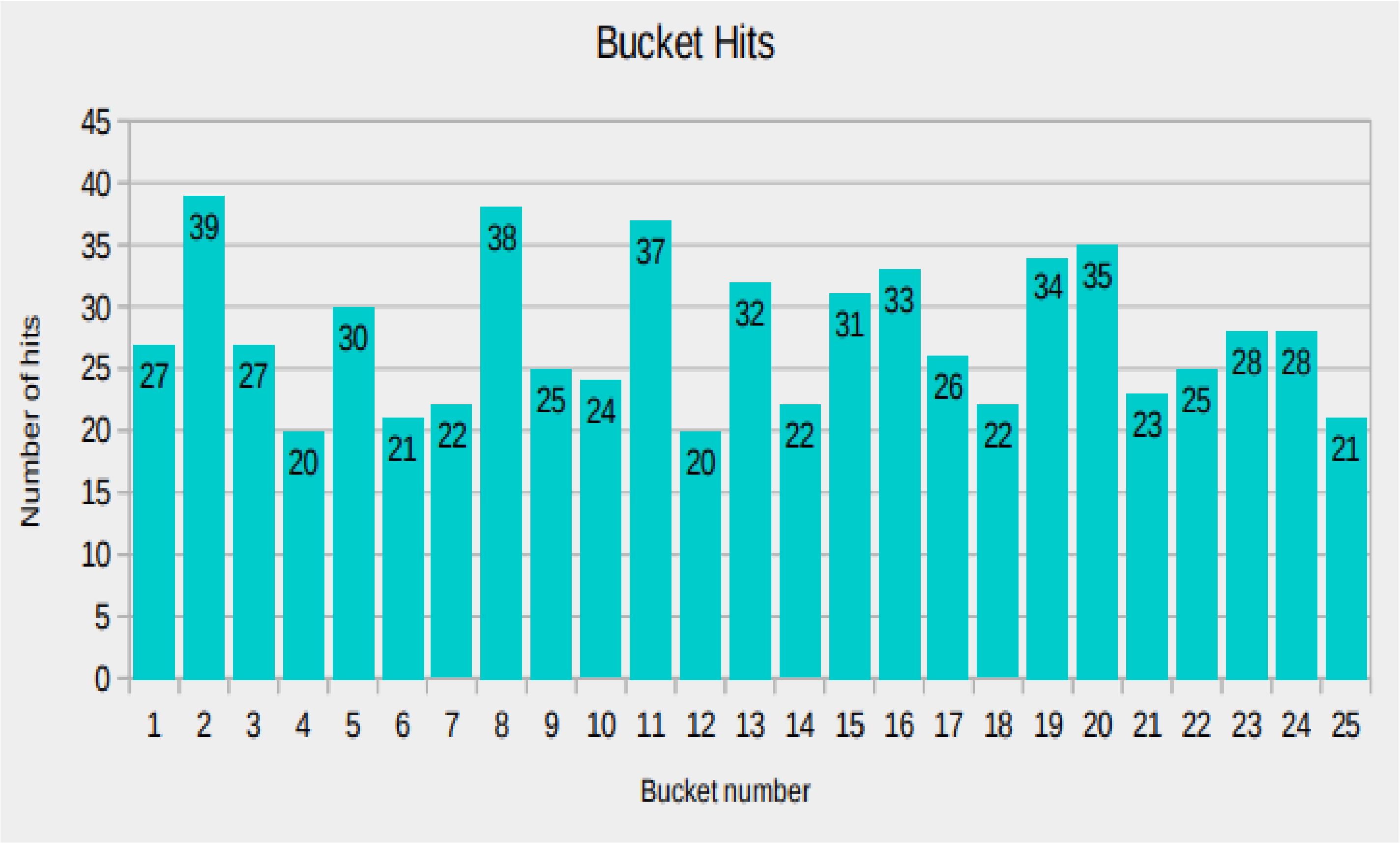
hash code 1 has better performances when number of buckets are 25. Thus we can say that hash code 3 will be more suitable to expect a better performance of the hash table.

2) For sample-text2.txt file

d) Modified Hash Table 1

```
// Division Method
public int valueHash(String key){
        int asciiValue = 0;
        // get the sum of ascii values
        // since there are 128 basic ascii values should get the power of 128
        for(int j=0; j<key.length(); j++){
                // for each character should get he relevant ascii value and should get the sum of them
                asciiValue = (31* asciiValue + key.charAt(j))%this.buckets;
        }
        // h(k) = k mod m
        // So the remainder should become hash value
        return asciiValue%this.buckets;
}
```

x) For Modified Hash Code 1 when bucket size 25



Modified method

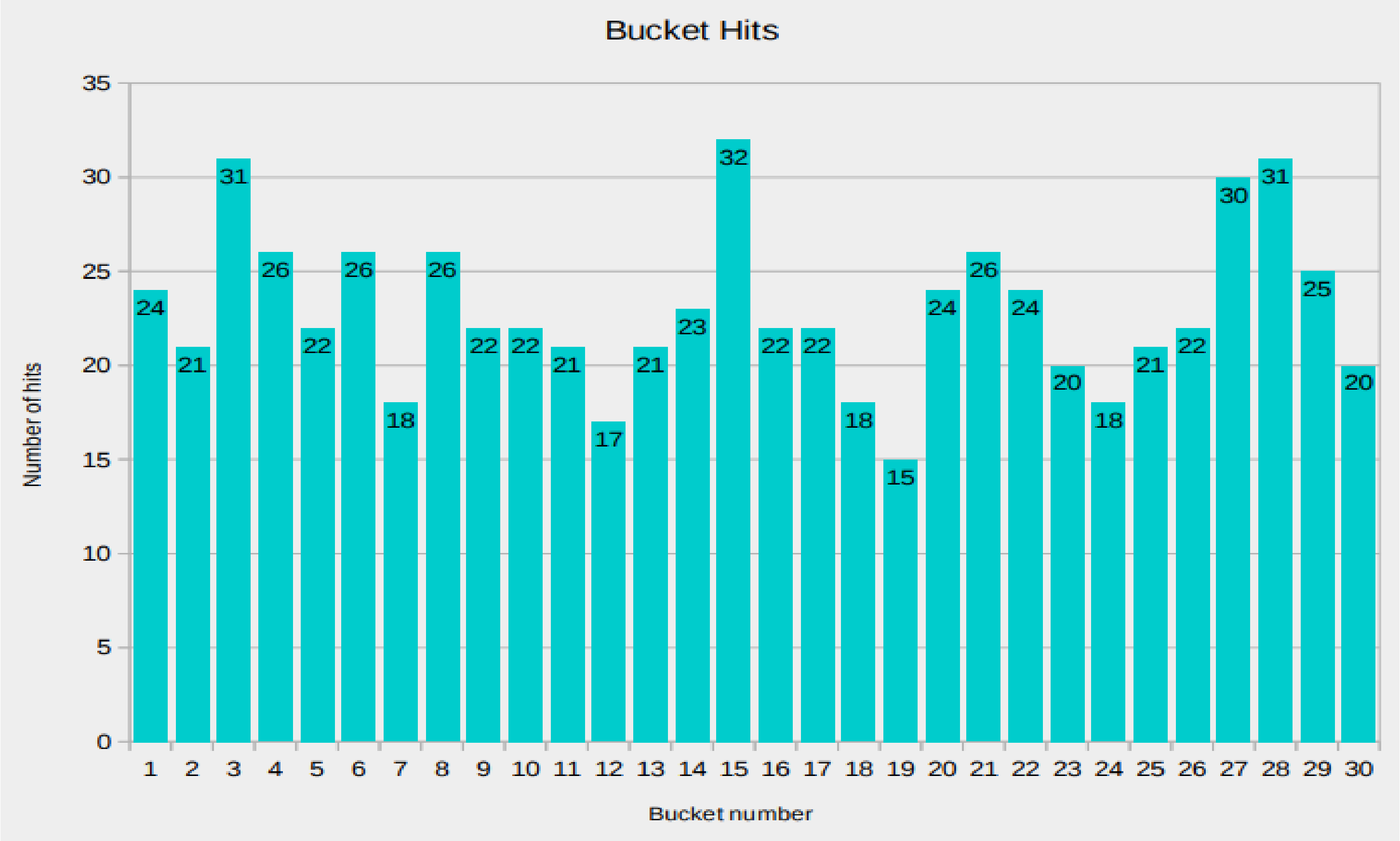
Minimum number of entries in a bucket: 20

Max number of entries in a bucket: 39

Average: 25.400000

Standard deviation: 6.900602

xi) For Modified Hash Code 1 when bucket size 30



Modified method

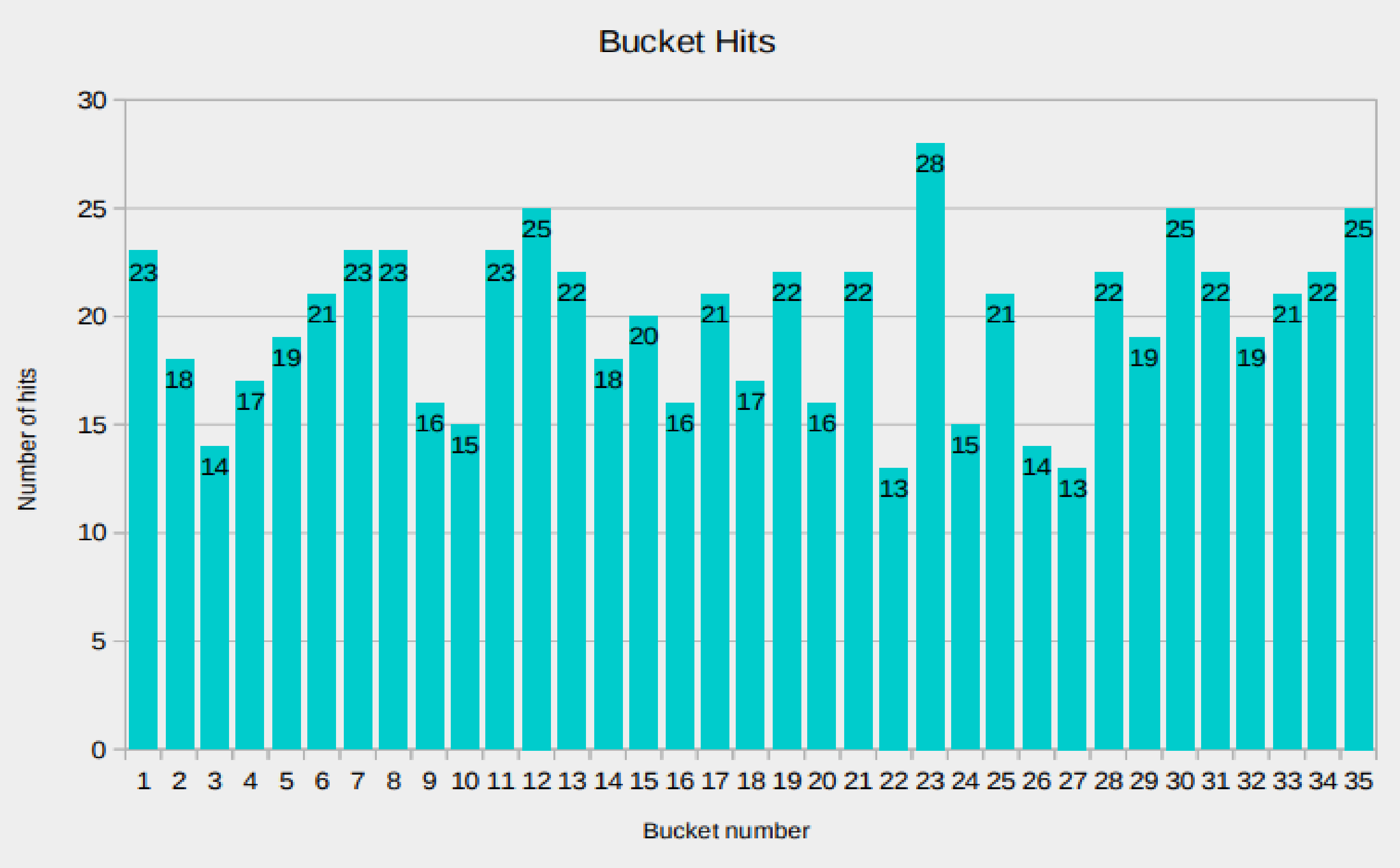
Minimum number of entries in a bucket: 15

Max number of entries in a bucket: 32

Average: 21.166667

Standard deviation: 8.413510

xii) For Modified Hash Code 1 when bucket size 35



Modified method

Minimum number of entries in a bucket: 13

Max number of entries in a bucket: 28

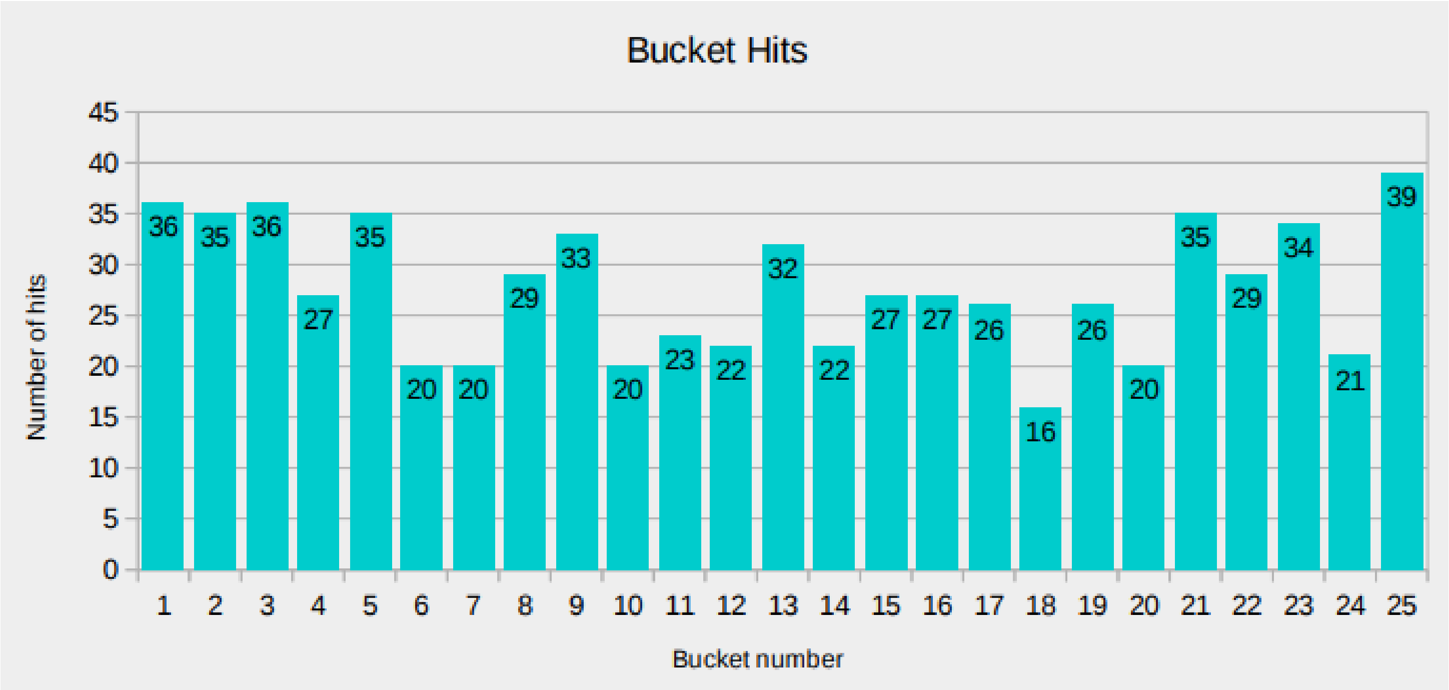
Average: 18.142857

Standard deviation: 10.044017

e) Modified Hash Table 2

```
// Division Method
public int valueHash(String key){
        int asciiValue = 0;
        // get the sum of ascii values
        // since there are 128 basic ascii values should get the power of 128
        for(int j=0; j<key.length(); j++){
                // for each character should get he relevant ascii value and should get the sum of them
                asciiValue = (71* asciiValue + key.charAt(j))%this.buckets;
        }
        // h(k) = k mod m
        // So the remainder should become hash value
        return asciiValue%this.buckets;
}
```

xiii) For Modified Hash Code 2 when bucket size 25



Modified method

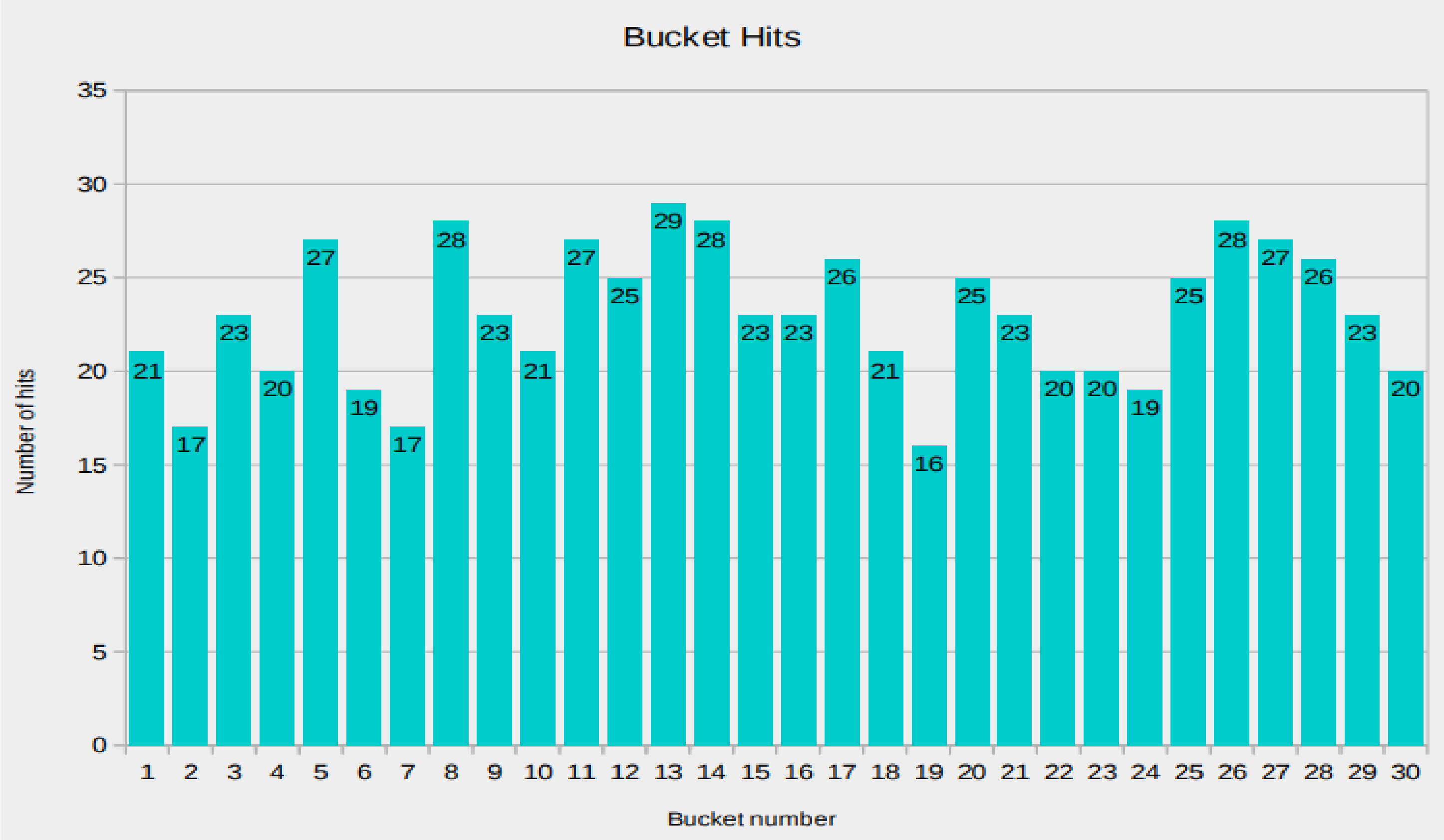
Minimum number of entries in a bucket: 16

Max number of entries in a bucket: 39

Average: 25.400000

Standard deviation: 7.098089

xiv) For Modified Hash Code 2 when bucket size 30



Modified method

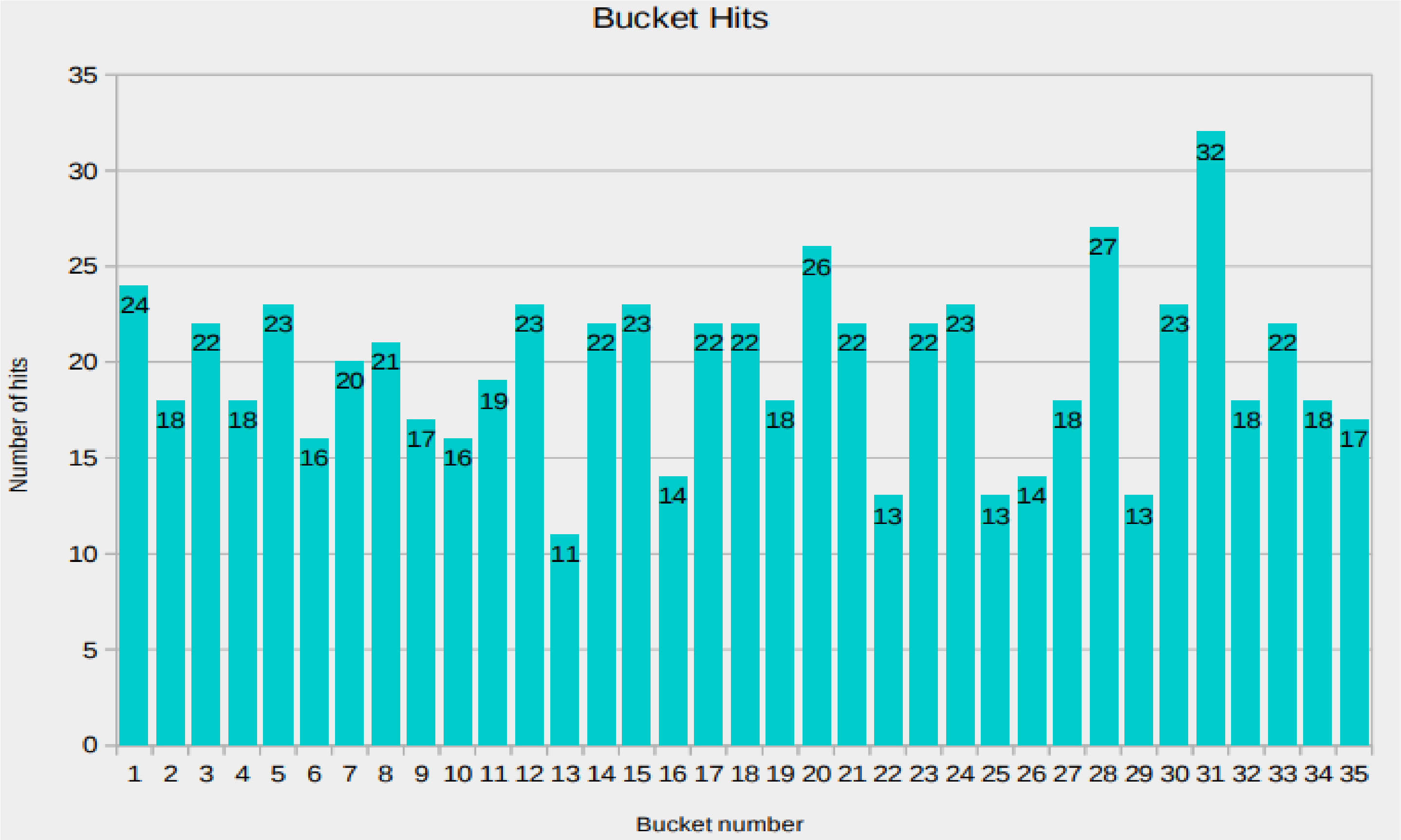
Minimum number of entries in a bucket: 16

Max number of entries in a bucket: 29

Average: 21.166667

Standard deviation: 8.485748

xv) For Modified Hash Code 2 when bucket size 35



Modified method

Minimum number of entries in a bucket: 11

Max number of entries in a bucket: 32

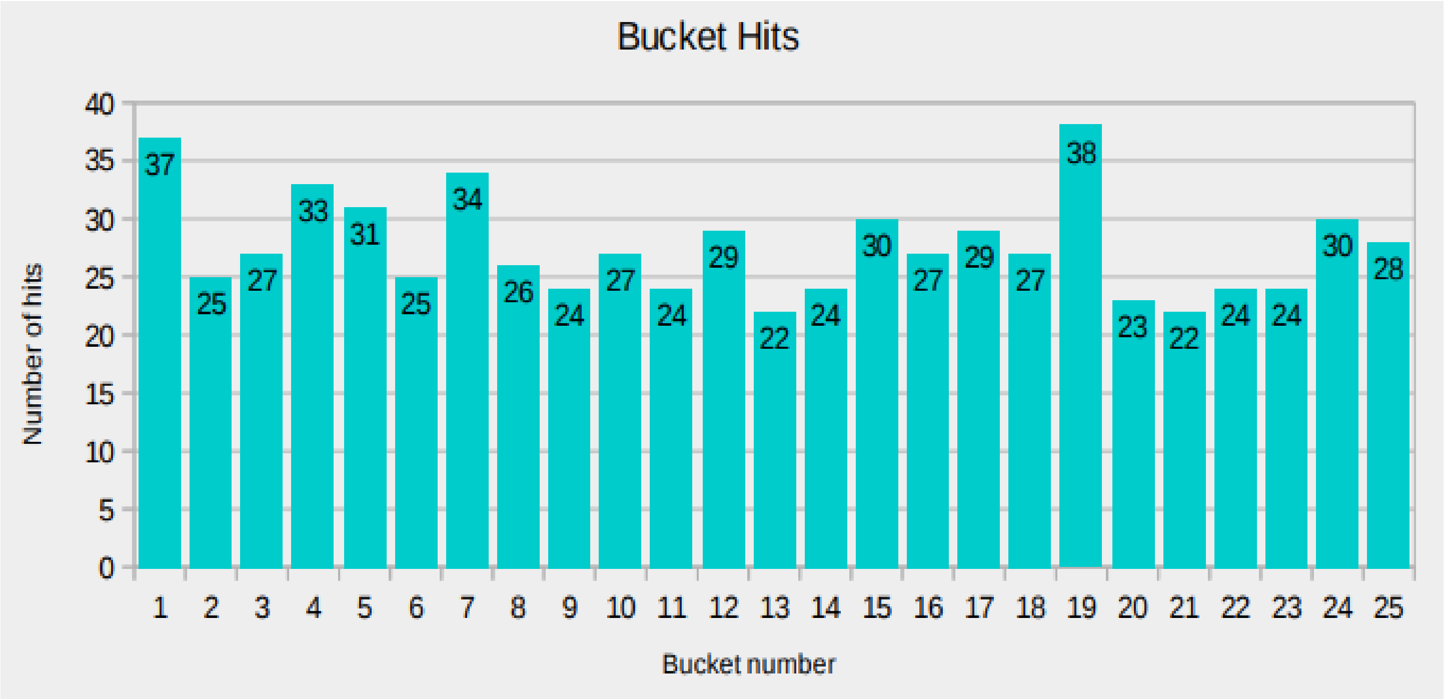
Average: 18.142857

Standard deviation: 10.031776

f) Modified Hash Table 3

```
// Division Method
public int valueHash(String key){
        int asciiValue = 0;
        // get the sum of ascii values
        // since there are 128 basic ascii values should get the power of 128
        for(int j=0; j<key.length(); j++){
                // for each character should get he relevant ascii value and should get the sum of them
                asciiValue = (523* asciiValue + key.charAt(j))%this.buckets;
        }
        // h(k) = k mod m
        // So the remainder should become hash value
        return asciiValue%this.buckets;
}
```

xvi) For Modified Hash Code 3 when bucket size 25



Modified method

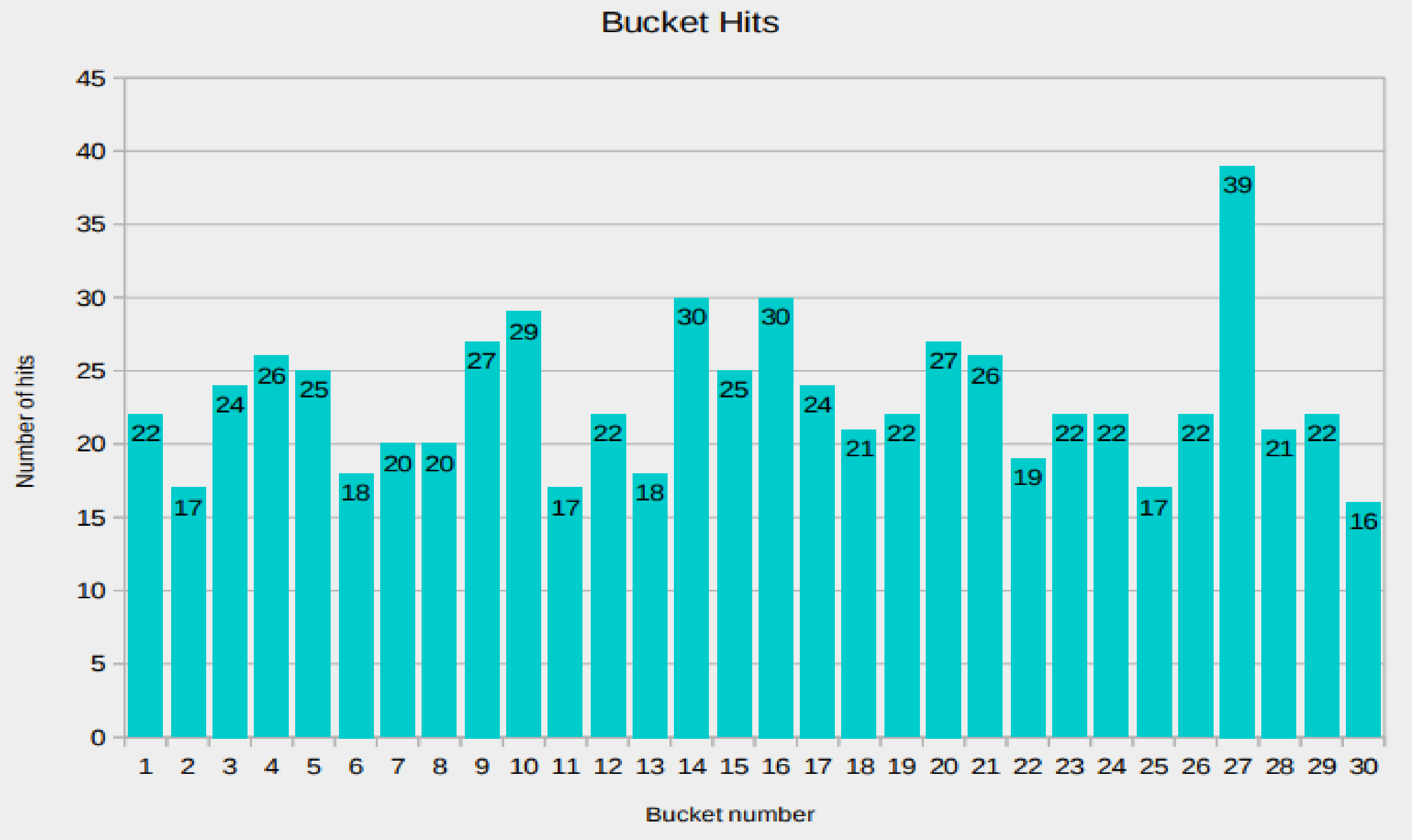
Minimum number of entries in a bucket: 22

Max number of entries in a bucket: 38

Average: 25.400000

Standard deviation: 6.934493

xvii) For Modified Hash Code 3 when bucket size 30



Modified method

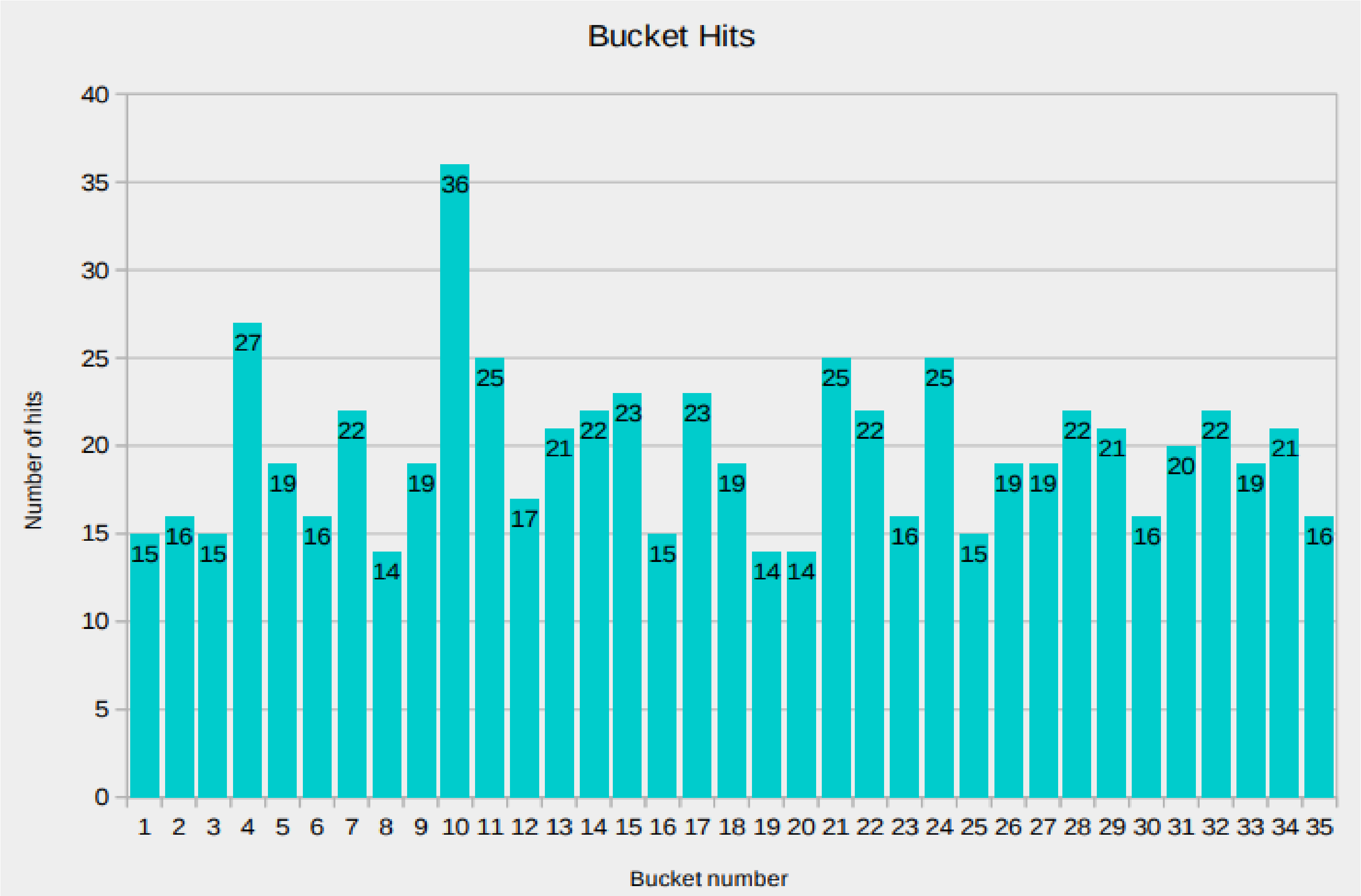
Minimum number of entries in a bucket: 16

Max number of entries in a bucket: 39

Average: 21.166667

Standard deviation: 8.459738

xviii) For Modified Hash Code 3 when bucket size 35



Modified method

Minimum number of entries in a bucket: 14

Max number of entries in a bucket: 36

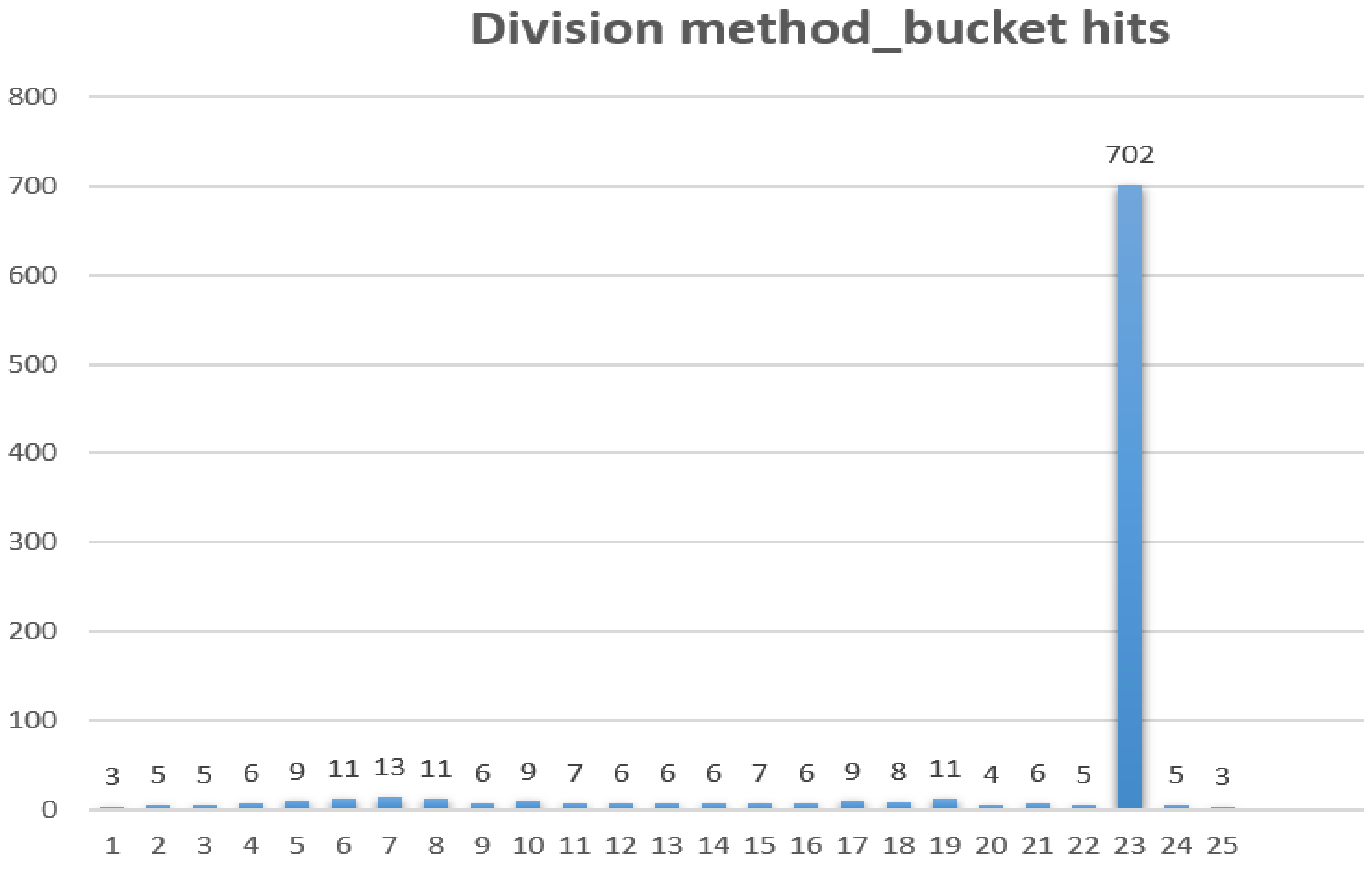
Average: 18.142857

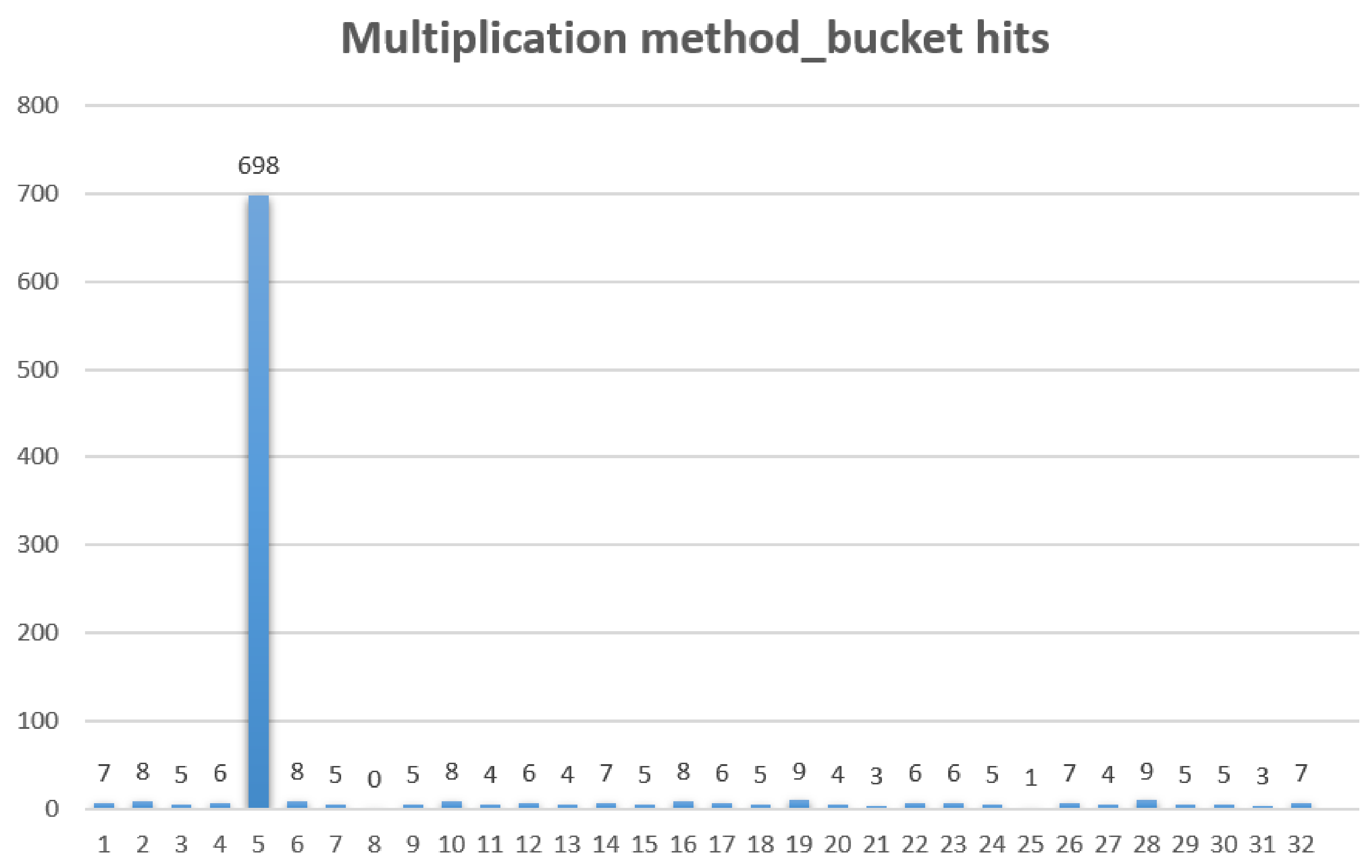
Standard deviation: 9.893387

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. Hash code 3 is an example for such situation. Of course all above values changes for different bucket sizes. Here bucket sizes 25,30 and 35 are considered for all the situations for the easiness of comparing.

Also if we see division method and Multiplication method we cannot get a proper distribution. So using Modified Method is better.

(Here is the two charts when bucket size 25 for division method and multiplication method)





If we consider above two charts do not show a uniform distribution over the all the buckets. If we consider their standard deviation, it is something like below.

Division method

Minimum number of entries in a bucket: 3

Max number of entries in a bucket: 702

Average: 0.001707

Standard deviation: 9.029980

Multiplication method

Minimum number of entries in a bucket: 0

Max number of entries in a bucket: 698

Average: 0.000536

Standard deviation: 3.851469