Madhushanee G.G.R.D.

E/15/208

CO322- Lab03

- Bucket size = 25 (All below readings will differ with bucket size. Lets use 25 bucket size to compare)

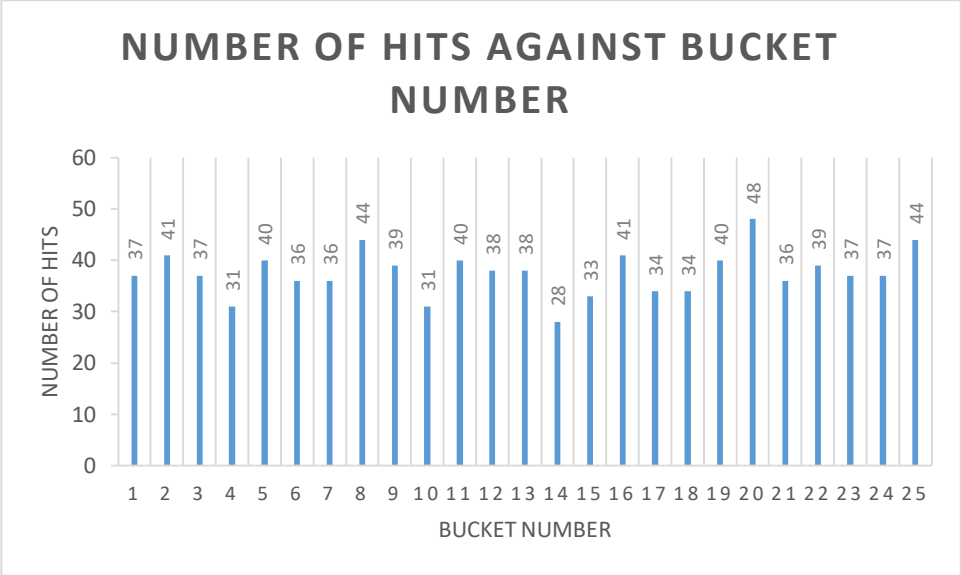
- Hash function:

```
private int hash(String key)
  {
        int hashcode = 0;
        for (int i = 0; i < key.length(); i++)
        {

              hashcode = (changing_value* hashcode +
key.charAt(i))%table.length;
        }
        return (hashcode%table.length);
  }
```

1. For sample-text1

a)

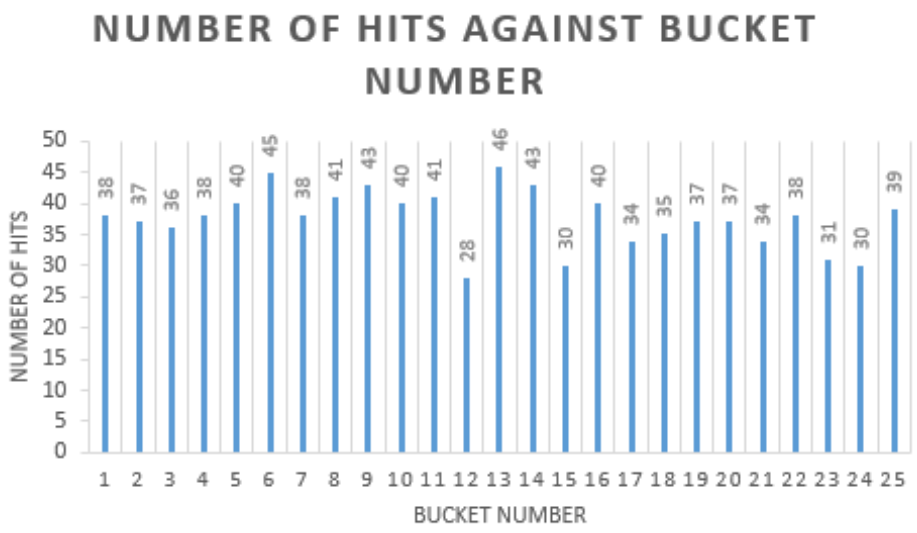
```
private int hash(String key)
  {
        int hashcode = 0;
        for (int i = 0; i < key.length(); i++)
        {

              hashcode = (57* hashcode + key.charAt(i))%table.length;
        }
        return (hashcode%table.length);
  }
```



Maximum : 48
Minimum : 28
Average:37.56
Deviation:4.336634234561983

b)

```
private int hash(String key)
  {
            int hashcode = 0;
            for (int i = 0; i < key.length(); i++)
            {

                        hashcode = (23* hashcode + key.charAt(i))%table.length;
            }
            return (hashcode%table.length);
  }
```



Maximum : 46
Minimum : 28
Average:37.56
Deviation:4.535006524767908

Comparison between a) and b)

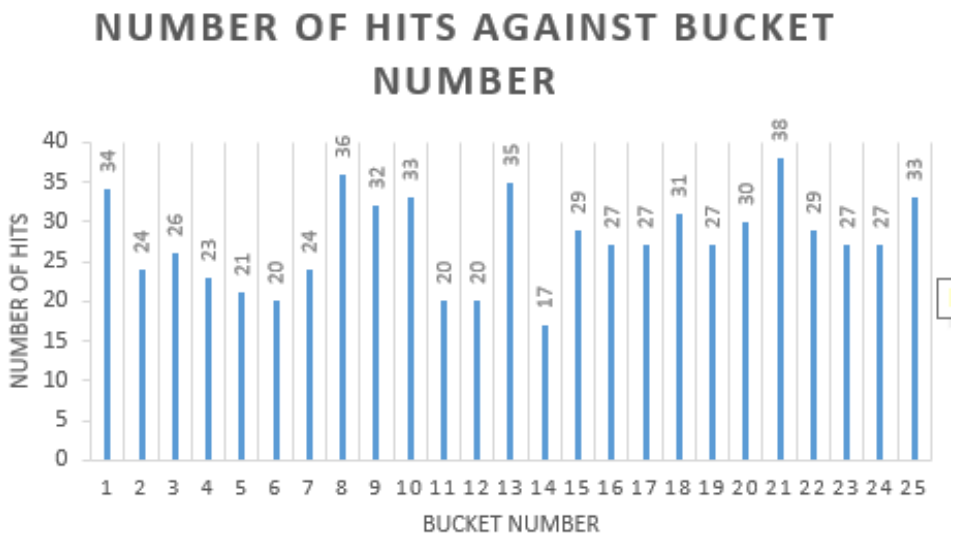
	a	b
Maximum value	48	46
Minimum Value	28	28
Average	37.56	37.56
Deviation	4.3366	4.5350

Here there are slight changes in a) and b) when comparing them with each other. Both has the same average of 37.56. But when considering the deviation each of them has got different values. a)'s deviation is lesser than b)'s deviation. So for a) , the chance of collision is minimized to lowest possible for any possible input than b).
So the uniformity of the hash function of a) in higher than b).
Hence, in this case a) is better.

2. <u>For sample-text2</u>

a)

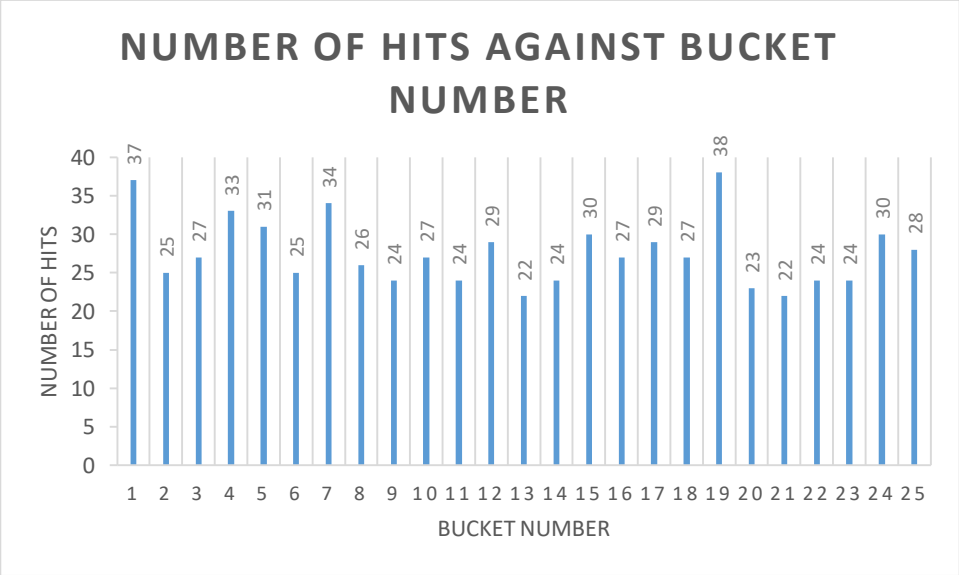
```
        private int hash(String key)
{
          int hashcode = 0;
          for (int i = 0; i < key.length(); i++)
          {

                    hashcode = (57* hashcode + key.charAt(i))%table.length;
          }
          return (hashcode%table.length);
}
```



Maximum : 38
Minimum : 17
Average:27.6
Deviation:5.49180964205163

b)

```
private int hash(String key)
 {
        int hashcode = 0;
        for (int i = 0; i < key.length(); i++)
        {

                hashcode = (23* hashcode + key.charAt(i))%table.length;
        }

        return (hashcode%table.length);
 }
```



Maximum : 38
Minimum : 22
Average:27.6
Deviation:4.289517849235069

Comparison between a) and b)

	a	b
Maximum value	38	38
Minimum Value	17	22
Average	27.6	27.6
Deviation	5.4918	4.2895

Here there are some changes in a) and b) when comparing them with each other. Both has the same average of 27.6. They have the same maximum value but have different minimum values with a bigger difference. When considering the deviation each of them has got different values. a)'s deviation is higher than b)'s deviation. So for b) , the chance of collision is minimized to lowest possible for any possible input than a). So the uniformity of the hash function of b) in higher than a). Hence, in this case b) is better.

The hash function for hash tables should have these two properties

- **Uniformity** all outputs of *H()* should be evenly distributed as much as possible. In other words the for 32-bit hash function the probability for every output should be equal to 1/2^32. (for n-bit it should be 1/2^n). With uniform hash function the chance of collision is minimized to lowest possible for any possible input.
- **Low computational cost** Hash functions for tables are expected to be *FAST*, compared to cryptographic hash functions where speed is traded for *preimage resistance* (eg it is hard to find the message from given hash value) and *collision resistance*.

One should not choose a random hash function to generate a good hash table. As with any hashing task, there are the three classical issues to consider:

- The size of the hash in terms of the number of bits of output needed to hit your collision (two distinct keys hashing to the same value) goals and remain within your storage constraints
- The distributions of hashes on **your** input data, and the related problem of collisions
- Computation time