# Given Hash Function Results:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Linear | | Random | |
| 50% Full | First 30 | Last 30 | First 30 | Last 30 |
| Min: 1 | Min: 17 | Min: 1 | Min: 1 |
| Max: 27 | Max: 59 | Max: 8 | Max: 14 |
| Avg: 8.83 | Avg: 18.89 | Avg: 2.93 | Avg: 3.35 |
| 90% Full | First 30 | Last 30 | First 30 | Last 30 |
| Min: 1 | Min: 34 | Min: 1 | Min: 1 |
| Max: 27 | Max: 75 | Max: 8 | Max: 18 |
| Avg: 8.83 | Avg: 27.07 | Avg: 2.93 | Avg: 3.76 |

# My Hash Function Results:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Linear | | Random | |
| 50% Full | First 30 | Last 30 | First 30 | Last 30 |
| Min: 1 | Min: 1 | Min: 1 | Min: 1 |
| Max: 3 | Max: 4 | Max: 4 | Max: 28 |
| Avg: 1.06 | Avg: 2.7 | Avg: 1.1 | Avg: 4.69 |
| 90% Full | First 30 | Last 30 | First 30 | Last 30 |
| Min: 1 | Min: 1 | Min: 1 | Min: 1 |
| Max: 3 | Max: 11 | Max: 4 | Max: 28 |
| Avg: 1.06 | Avg: 2.25 | Avg: 1.1 | Avg: 3.31 |

The most glaring flaw with the given hash was that it often contained a lot of whitespace in the given string, the second slice for characters 8-9 have a high chance to come up with whitespace in either one or both characters. This gave many of the numbers a similar seed which will give them similar output in the table. My hash addresses that by only working with the first 6 characters of the string. The next issue is multiplying by 256. 256 is an even number, this limits the amount of numbers that we could be coming up with. My hash does away with that in its entirety and I XOR the converted string by the prime number 3. This results in changing the bits, which is still quick for the compiler, and will give us a much more varied number. Finally my function will do its best to pick numbers in the middle range of decimal places, because it’s more likely that numbers will be either high or low decimals, I will divide by 100000 which will remove the lower numbers. If you look at the table, you will see that my function results in a much lower number of probes.

The expected number of linear probes when the table is at 50% is 1.5, and when it’s at the first 30 items, the expected result is 1.15, and the expected result at the last 30 items (115) is 5.42.

The given hash function ‘s average is at 8.83, and my function is at 1.06, for the first 30 items of data at 30 and 90% full. In this instance, mine is MUCH better. The last 30 items in the array have higher numbers from both of us, The given function is 18.89 when it’s 50% full and 27 when it’s 90% full, while mine is 2.7 and 2.25 respectively. The expected is 5.42, so mine is twice as fast as expected, while the given algorithm is more than 3 times slower.

The expected number of random probes expected at 50% is 1.39, and 2.56 at 90%. For the first 30, it’s expected to be 1.139, and the second it’s supposed to be 2.546. The given function averaged at 2.93 for the first 30 at both 50% and 90% full. While my function averaged at 1.1 for the first 30 at both 50% and 90% full. This isn’t bad for either functions. And it’s still okay when we get to 90% full. The given functions runs at 3.35 and 3.76 for 50% and 90% respectively. So it appears that it is much more likely to hash in the lower numbers and is uneven. My hash is slightly more even, when the table is 50% full it took 4.7 probes and at 90% full, its average probes hit 3.31.