Distinct Element Using Hashing

Exercise 1

The hash function $h(x)$ with b (the number of bits) and k (the number of ints) both equal to 32. We can think of A as being the matrix with 32 rows, k, and 32 columns, b.

```
public static int hash_h(int x){
  int hash=0;
  int[] A = matrix.getMatrix();
  for(int i = 0; i<32;i++){
   hash <<= 1;
   hash |= Integer.bitCount(x & A[i]) % 2;
  }
  return hash;
}
```

The code takes an integer x and outputs a hash value of 32-bits.

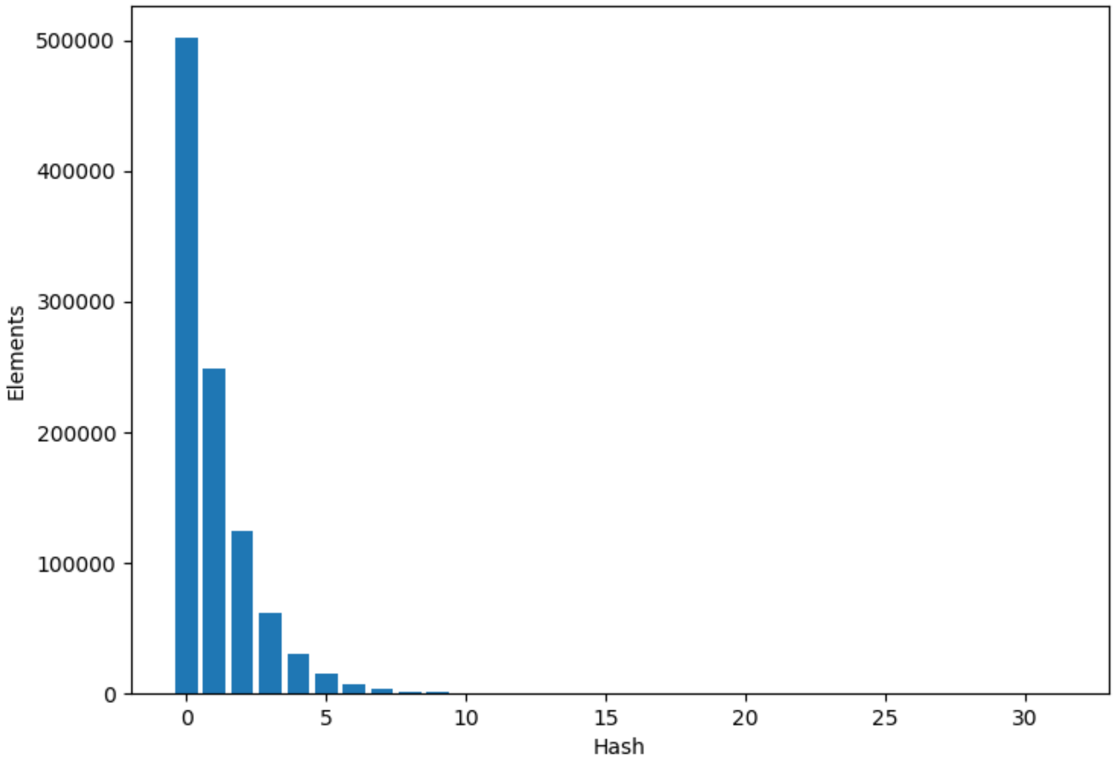
Exercise 2

We created a million values $x \in \{1, \ldots, 10^6\}$ and plotted the hash values $h(x)$ in the following graph.

The x-axis shows the number of leading zeros to the hash-value - 1. We took off one as all our numbers are positive and the first bit in an int defines whether a number is positive or negative. The y-axis is the total amount of hash-values that have that number of leading zeros.

As expected the distribution of hash values seems to satisfy the relation $Pr(phi(y) = i) = 2^{-i}$ for i from 1 to k for random y in $\{0, 1\}^k$. ie. half of all the hash ints have just one leading zero (which we ignore), a quarter has two, an eighth three etc.

This makes logical sense as there are twice as many possible numbers to be constructed from say 4 bits as there are from 3. Therefore, if our random hash function is working correctly, we could and should and did expect our distribution to look as it does.



Exercise 3

We created a text file with all of the numbers from 10^6 to $2 * 10^6 - 1$ named *input.txt* and implemented the algorithm with $m = 1024$ and $k = 32$.

Upon running the input through the HyperLogLog algorithm we returned the double: 973089.2722159306.

This number is, in layman's terms, pretty close to a million.

Exercise 4

The HyperLogLog algorithm does not react as we expected.

We created an input file with a million distinct random numbers in the range $\{1, 10n\}$ where $n = 10^6$. We set $seed = 30$ and using *rand_input_gen.py* created a text file named *rand_1mil_seed30.txt* .

We expected our estimate to get more precise as m gets larger however as can be seen in the table below our implementation has peak precision when $m = 2048$.

However, as can by seen below the estimation, of all our values of m, is always within $(1 \pm 2\rho) * 10^6$.

m	E **calculated by HyperLogLog**	ρ	**within** $(1 \pm \rho) * 10^6$	**within** $(1 \pm 2\rho) * 10^6$
512	940747444536	0.045962	FALSE	TRUE
1024	985207537650	0.0325	TRUE	TRUE
2048	999646365089	0.022981	TRUE	TRUE
8192	988574761804	0.01149	TRUE	TRUE
16384	988676715205	0.008125	FALSE	TRUE