Prepare your answers to the following questions in a plain text file. Submit your file to the Curator system by the posted deadline for this assignment. No late submissions will be accepted.

One way to compare the performance of several hash functions is to simulate the addition of a fixed collection of records to a hash table, using each hash function, and counting the number of hits each slot in the table receives. From that information, we can construct a histogram showing how many slots received one hit, how many received two hits, and so forth. We can also calculate the average number of steps that a search would take.

For instance, given a collection of 1000 words and a hash function that computes a nonnegative integer from a string, we might use a table size of 2000 and discover that:

# of hits	# of slots receiving that # of hits
0	1042
1	932
2	15
3	7
4	3
5	1

Assume that each slot in the hash table uses a linked list to hold the elements that collide in that slot. From that information, we would then calculate that the average number of steps in a search would be about 1.064, since the histogram above implies that:

- 958 words would require 1 step (932 + 15 + 7 + 3 + 1)
- 26 words would require 2 steps
- 11 words would require 3 steps
- 4 words would require 4 steps
- 1 words would require 5 steps

Three files of strings are provided on the course website:

`WordData.txt`	111,444 English words (somewhat loosely)
`NMFeatures.txt`	45,686 medium-length strings naming NM geographic features
`SQ.txt`	26,381 (mostly) long strings representing amino acid sequences

Your task is to compare the performance of four different hash functions on the latter two files:

`elfhash()`	classic elfhash function employing complex folding
`sbdm()`	string hasher employing more complex folding
`BPHash()`	another string hasher
`APHash()`	Arash Partow's general-purpose hash function

Java code for each of the four functions is provided in the course notes or on the course website. You will implement driver code to apply each of the hash functions to all of the strings in each of the latter two files and compute the results needed to complete the table given below. The results for the first file are included below to help you validate your logic.

In accordance with common guidelines for measuring performance, you will use consider table sizes that are multiples of the number of strings being hashed.

For each of the supplied data files, you'll complete a table like the one shown below. The values I obtained for the `WordData.txt` file and the `elfhash()` function are shown.

```
Keys:  WordData.txt   111444 strings

Hash fn   load     avg     worst    % < log N
---------------------------------------------
elf       1.0      1.543    8        100%
          1.2      1.452    7        100%
          1.4      1.361    7        100%
          1.6      1.323    7        100%
          1.8      1.280    6        100%
          2.0      1.323    7        100%

sbdm      1.0
          1.2
          1.4
          1.6
          1.8
          2.0

BPHash    1.0
          1.2
          1.4
          1.6
          1.8
          2.0

APHash    1.0
          1.2
          1.4
          1.6
          1.8
          2.0
```

The average number of steps is computed in the manner described in the example given on the previous page.

The worst case is the maximum number of steps that would be needed to find a word.

The last value (% < log N) is the percentage of the given strings that would be found in fewer than log N steps, where N is the number of strings that were hashed.

You should not that it is NOT necessary to implement a fully-functional hash table in order to solve this assignment. In fact, it may be counter-productive to do so. You may, and should, take advantage of the standard library.