Zak Scholl
CS 367
M. Hidayath Ansari

P4

The hash function I chose to implement takes the 1st, last, 1st quartile and 3rd quartile characters in the list, then multiplies their respective integer ASCII values to obtain an integer hash value. My thinking behind this function was that a single ASCII value takes up 8 bits, so if I multiplied 4 of these bytes together I would get a number between 0 and 2^{32}. This strategy turns out to be pretty ineffective for "tough" samples. Any value sequence that varies only between the values I chose to hash will be very ineffective. And I didn't implement any type of rolling hash function, so it is computed every time the window moves.

I also added a bit of code to the main program file that ouputs the running time in seconds, up to three decimal places each time the program is run.

Theoretical running time:
For the brute force method it is O(n*m) because you go over every item in the CharacterList m times. For the hashing method it is O(n*m) because it will go over n/m intervals, for the hash and if succesful, it will go over m characters in each interval , but in the worst case there is a match at every window, so it is again O(n*m).

I ran each case 5 times and averaged the results, there was a significant amount of variance between running times.

Test Case	**Brute-Force Running Time (sec)**	**Hash Running Time (sec)**
Search string 1/haystack 1	.0014	.0018
Search string3/mobydick	8.937	.202
Search string2/haystack 2	178.723	4.781
Modified challenge input	177.925	208.684
Short String in Moby Dick	1.894	.238

This could be used to detect plaigarism by evaluating if a phrase or block of text is in a particular assignment submission. It could also be used to quickly "find" a particular string in a document, by hashing each String that's the size of the search String and displaying the number of results.