Problem Solving 1

Some important characteristics of good hash function includes

- ***Speed - the time complexity of the function.***
- ***Consistency- given the same input, does the function always produce the same output?***
- ***Frequency of Collisions - how likely is it that two different values will have the same hash code?***

In the space to the right, explain why each of these characteristics is important, and how a *bad* hash function would perform in each area.

Speed: Speed is important because that means the hash function can run in constant time or close to it relative to N. The faster the hash function, the less time and money it takes. A bad hash function with a high speed amount would lead to many problems and result in more time and money. You want the hash function to perform the quickest in the least amount of time as possible.
Consistency: Consistency is important because the hash function has the values stored in it and retrieves it so as a result, it always returns the same hash code for the same input no matter how many tries. The output will always be the same. A bad hash function that is not consistent would result in many errors and slow down everything. If something is not consist enough, then what is the point of it? It will be useless.
Frequency of Collisions: Frequency of collisions is important because it minimizes collisions by ensuring that as much as possible, two different input values will result in two different hash codes. A bad hash function would be one that has a lot of collisions (maximum amount) which would be really bad. The more the collisions the function has to interact with, the more time it will take which means much higher speed time and some inconsistency too.

```
def hash_sum(string):

   (sum) = (0)
   for (char) in (string):
      (sum) += ord(char)
   return (sum)
```

Speed: Less than 1 seconds…
Consistency: Perfect about all the time…
Frequency of collisions: Less than 10 collisions, 50% total collision rate…

Problem Solving 2

One possible hash function would use the sum the value of the ASCII characters in a string as the hash code for the string.

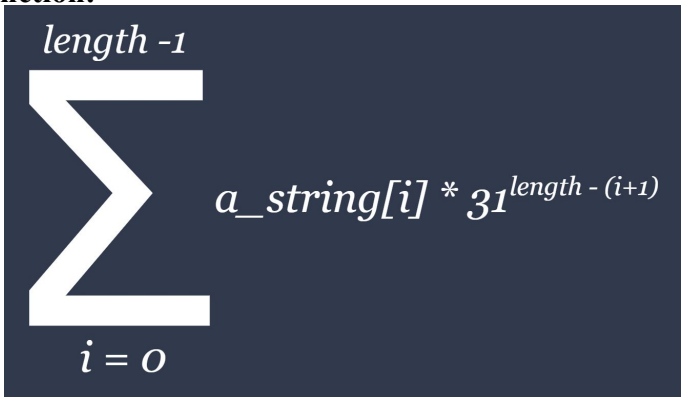
Implement such a function in the space to the top left. Remember, you can use the *built-in* ord() function to convert a character into its ASCII value (an integer).

In the space to the bottom left, discuss how you think this hash function would perform relative to the three characteristics (*speed*, *consistency*, and *frequency of collisions*).

2

Problem Solving 3

Another hashing function modifies the ASCII values of the characters in a string in some way rather than simply adding them together. Consider the following function:



Implement a hash function that uses this formula in the space on the top right. Then, in the space to the bottom right, discuss how you think this hash function would perform relative to the three characteristics (*speed*, *consistency*, and *frequency of collisions*).

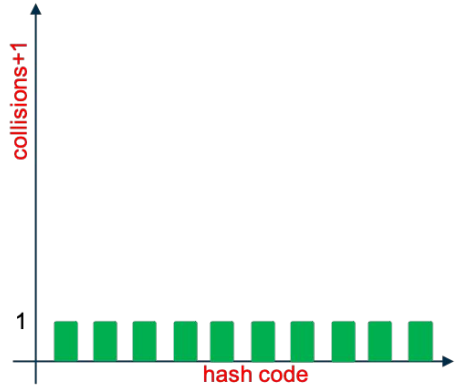
```
def hash_positional_sum(string):

    (sum) = (0)
    (power) = (1)
    for (i) in range(len(string)):
        (sum) += ord(string[i]) * (power)
        (power) *= (31)
    return (sum)
```

Speed: Less than 2 seconds…
Consistency: Perfect about all the time…
Frequency of collisions: Less than 0 collisions, 100% total collision rate…

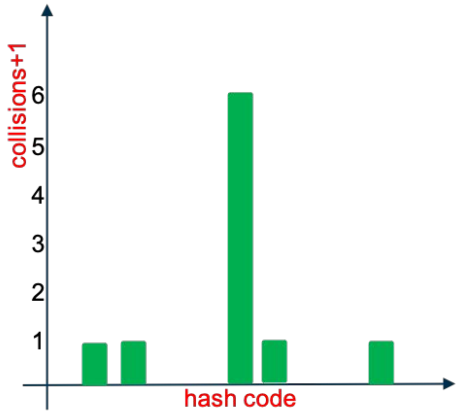
Problem Solving 4

- We'll consider how to test the *quality* of hash functions.
- For *speed*, we can simply measure the time taken by hashing input data. Consider the two hash functions in the previous problems. If the input strings are long, which hash function will take more time to evaluate? (hash_positional_sum function)
- For the *collision* test, we will hash *n distinct* strings and count collisions.
- An ideal hash function hashes the messages to different outputs as shown in graph 1 at right, where *n = 10*. How many collisions occurred? (10)
- Consider the two other graphs at right and fill the following table:

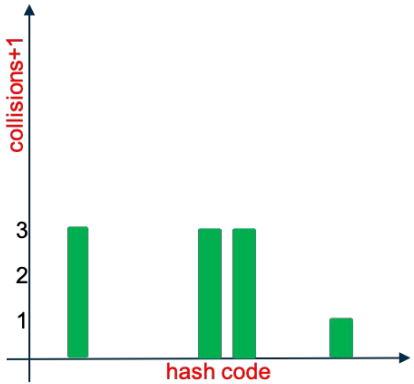
	graph 2	graph 3
Total collisions	10	10
Max collisions (in any hash code)	11	11
Spread (= num of distinct hash code / n)	1.1%	1.1%



<graph 1>



<graph 2>



<graph 3>

Problem Solving 5

- **Which data structure would you use to represent the graphs in the previous problem? You can think of the graph as a map from hash code to collisions. (dictionary)**
- **Assume there is a text file named 'data.txt' that you will use to hash each line and count collisions. Write a function named build_collision_counter that declares a hash function as a parameter, constructs a collision counting data structure by hashing the lines using the given hash function, and returns it.**

```
def build_collision_counter(hash_function):

    (collision_count) = ({})
    with open("data/data.txt") as (file):
        for (line) in (file):
            (hashed_line) = hash_function(line.strip())
            if (hashed_line) in (collision_count):
                (collision_count[hashed_line]) += (1)
            else:
                (collision_count[hashed_line]) = (1)
    return (collision_count)
```