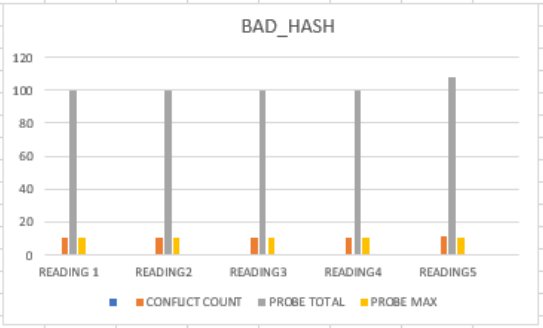
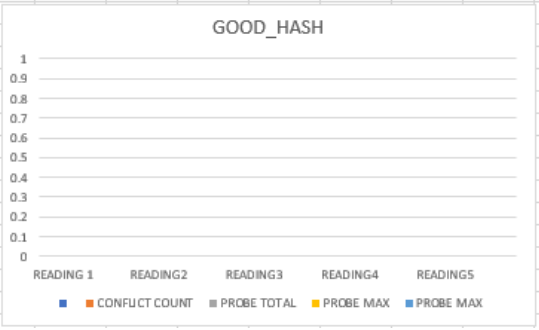
Group T07G01
Assignment 3

1. **Hash Table Analysis**

	READING 1	READING2	READING3	READING4	READING5
CONFLICT COUNT	10	10	10	10	11
PROBE TOTAL	100	100	100	100	108
PROBE MAX	10	10	10	10	10



	READING 1	READING2	READING3	READING4	READING5
CONFLICT COUNT	0	0	0	0	0
PROBE TOTAL	0	0	0	0	0
PROBE MAX	0	0	0	0	0



Given that there are 2 hash functions (good_hash) and (bad_hash), they are expected to output different statistics. In terms of statistics, we look at **conflict count** (which occurs when we attempt to insert a key into our hash table,but the location for that key is already occupied by a different key), **probe max** (the length of the longest probe chain throughout the execution of the code) and **probe total** (the total distance probed throughout the execution of the code).In terms of conflict_count, (good_hash) is expected to produce a very much lower conflict count as evident by the result,the conflict count produced is 0 for all 5 readings . The (bad_hash) is expected to produce a larger conflict count. This stems from the fact the ASCII values of the first 2 characters are taken. If 2 names have the same first 2 characters, it would lead to conflict as evident by the result, the(bad_hash) produces 10,10,10,10 and 11 on 5 readings.In terms of probe total, (bad_hash) will have a longer probe chain as evident per the result , probe total for bad_hash produces 100,100,100,100 and 108 on 5 readings while good_hash produces 0 on all 5 readings. When there is conflict, the array is searched for an empty spot. This coincides with the fact the larger the conflict count, the longer the probe chain. The same goes for probe max as per the

result. Bad_hash produces 10,10,10,10,10 on all 5 readings while good_hash produces 0 on all 5 readings.

Below is the function used for good_hash:

```
def good_hash(cls, potion_name: str, tablesize: int) -> int:
    """
    A method that hashes the potion name to a specific index position
    :param cls:
    :param potion_name:
    :param tablesize:
    :return: Returns the hash value of potion_name
    complexity: O(1)
    """
    value = 0
    a = 31415
    b = 27183
    for char in potion_name:
        value = (ord(char) + a * value) % tablesize
        a = a * b % (tablesize - 1)
    return value
```

Below is the function used for bad_hash: (Located in hash_analysis.py)

```
def bad_hash(cls, potion_name: str, tablesize: int) -> int:
    """
    A method that hashes the potion name to a specific index position
    :param cls:
    :param potion_name:
    :param tablesize:
    :return: Returns the hash value of potion_name
    complexity: O(1)
    """
    val =0
    a = 35415
    b = 29183
    for char in potion_name:
        for i in range(0,1):
            val = (ord(char[i]) + a * val) % tablesize
            a = a * b % (tablesize - 1)

    return val
```

Below is the code used to generate strings as well as utilise the hash functions

```
from hash_table import LinearProbePotionTable


# # creating a hash table
# potion_hash = LinearProbePotionTable(100)
#
# # insert in the hash table
 for i in range(101):
     potion_hash._setitem_(str(i), str(i))
     print(potion_hash.statistics())
```