NAME _____ **ICA-16**

NetID _____@email.arizona.edu

Problem 1. Suppose you are given the hash and probe decrement functions below:

hash(key) = key % 11
probe(key) = max(1, key / / 11)

a) Fill out the columns in the table below for the **Hash value** and **Probe decrement** based on the functions defined above.

Key	Hash value	Probe decrement
19		
22		
25		
33		
36		
42		

b) Give the final configuration of a hash table below that results from inserting the keys listed above into an empty table using double hashing. The order of insertion of the keys is: 33, 25, 36, 19, 22, 42.

0	1	2	3	4	5	6	7	8	9	10

Problem 2. We have 42,000 student names to enter as keys into a hash table. If we want a load factor of .70, what should the table size be?

Problem 3. Recall the following definition of a dictionary from lecture.

- A dictionary is an ADT that holds key/value pairs and provides the following operations:
  - put(key, value)
    - makes an entry for a key/value pair
    - assumes key is not already in the dictionary
  - get(key) looks up key in the dictionary
    - returns the value associated with key (and None if not found)
  - Note: a dictionary is of fixed sized and is set to its capacity when created
  - Usage:

```
>>> d = Dictionary(7)
>>> d.put('five', 5)
>>> d.put('three', 3)
>>> d._pairs
[['five', 5], ['three', 3], None, None, None, None, None]
```

The code below implements the dictionary ADT:

```
class Dictionary:
    def __init__(self, capacity):
        # each element will be a key/value
        self._pairs = [None] * capacity

    def _hash(self, k):
        return len(k) % len(self._pairs)

    def put(self, k, v):
        self._pairs[self._hash(k)] = [k,v]

    def get(self, k):
        return self._pairs[self._hash(k)][1]
```

a) What is the complexity of the `put()` and `get()` in this implementation?

b) Modify the implementation to use open addressing with double hashing.The hash function is given in the code above. Use the following for the probe decrement function:

probe(key) = max(1, len(key) / / 7)

Assume that there is always room the in hash table for a new entry and that the value being stored is always a new value. That is, you do not have to handle the following:

```
>>> d.put('three', 3)
>>> d.put('three', 48)
```

Also, assume that `get()` will always find a value in the dictionary.

Problem 4. Modify the dictionary class in Problem 3 with *chaining* to resolve collision.