Work with your neighbor. (This will be graded for participation only.)

1. You are given the following hash function:

```
hash(key)= key % 7
```

Insert the keys `15, 30, 28, 48` into the hash table below. Use **linear probing** to resolve collisions.

0	1	2	3	4	5	6

a) Now insert (put) 23 into the hash table. What is the probe sequence for this insertion?

b) If you lookup (get) 23 in the hash table, what is the probe sequence?

*****Wait for lecture to continue before doing the next problems.**

2. The function `char_count(text)` below returns a dictionary containing a mapping of each character in `text` and the count of the number of occurrences in `text` for the character:

```
def char_count(text):
    cdict = {}
    for char in text:
        if char in cdict:
            cdict[char] = cdict[char] + 1
        else:
            cdict[char] = 1
       return cdict
```

Usage:

```
>>> char_count("to be or not to be")

{'t': 3, 'o': 4, ' ': 5, 'b': 2, 'e': 2, 'r': 1, 'n': 1}
```

a. Write a new version of the function, `char_countv2(text)`, that returns a built-in Python dictionary that maps each character in `text` to a list of integers, where the length of the list is the number of occurrences of the character in `text`.

Usage:

```
>>> char_countv2("to be or not to be")

{'t': [1, 1, 1], 'o': [1, 1, 1, 1], ' ': [1, 1, 1, 1, 1], 'b': [1, 1],
'e': [1, 1], 'r': [1], 'n': [1]}
```

b. Assume that you have a Dictionary 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)
- defines __contains__() so that
  - `in` returns `True` if a key is in the `Dictionary` and `False` otherwise.
- Note: a `Dictionary` is of fixed sized and is set to its capacity when created
- Usage:

```
>>> d = Dictionary(7)
>>> d.put('t', [1])
>>> 't' in d
True
>>> d.put('o', [1])
>>> val_list = d.get('t')
>>> val_list
[1]
```

Modify your function `char_countv2(text)` to use the Dictionary ADT defined above instead of a Python built-in dictionary. Call this version `char_countv3(text)`.

```
'''
This version uses the Dictionary ADT.
'''
def char_countv3(text):
```

3. We have 60,000 items to store in a hash table using open addressing with linear probing and we want a load factor of .75.

   How big should the hash table be?

4. Use the hash function `hash(key) = key % 7` and **separate chaining** to insert the keys below into the hash table. On a collision, add to the **front** of the linked list.

   `15, 30, 16, 8`

0	1	2	3	4	5	6