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

1. You are given the following functions for `hash()` and `probe()`:

```
hash(key)= key % 7
prob(key)= max(1, key // 7)
```

Complete the table for the following key values: `10, 2, 19, 14, 24, 23`

ANS:

key	hash value	probe decrement
10	3	**1**
2	2	**1**
19	5	**2**
14	0	**2**
24	3	**3**
23	2	**3**

2. You are given the following functions for `hash()` and `probe()`:

```
hash(key)= key % 7
prob(key)= max(1, key // 7)
```

Complete the table for the following key values: `15, 30, 28, 48, 23`

ANS:

key	hash value	probe decrement
15	**1**	**2**
30	**2**	**4**
28	**0**	**4**
48	**6**	**6**
23	**2**	**3**

Insert the keys `15, 30, 28, 48, 23` into the hash table below. Use **double hashing** to resolve collisions.

0	1	2	3	4	5	6
28	**15**	**30**	**23**			**48**

3. Consider the information in the table below, which shows the hash value and probe decrement (for double hashing) for the given keys from problem 1:

key	hash value	probe decrement
10	3	1
2	2	1
19	5	2
14	0	2
24	3	3
23	2	3

Walk through the probe sequence for inserting 24 into the hash table below. Verify that the probe sequence covers all slots exactly once and would find *any* available slot (i.e., if only slot 0 were open, the probe sequence would find it; if only slot 1 were open, the probe sequence would find it, and so on.)

For this problem you need to work through several examples by hand. For key 24, we assume that the slot that it hashes to is taken (specifically, by 10 in the example below). Now mark all of the slots as taken except for one by using x's to indicate that the slot is occupied. In the example below, we see that the slot at index 0 is available. Walk through the table by the probe decrement (which is 3 in this case) and make sure that you get to the open slot. Then write out the probe sequence that was needed to find the available slot.

Below, index 0 is the only slot available. To insert 24, we first hash and get 3. That slot is taken, so we decrement by the probe decrement which is also 3. That takes us to index 0, which is available and we put 24 in that slot. The probe sequence for inserting 24 is: **3, 0**

0	1	2	3	4	5	6
	x	x	10	x	x	x

Now do this for every possible scenario where only 1 slot is unoccupied. Notice that for some cases, you may have to traverse the table more than once. Write out the probe sequence for each case.

For example, now index 1 is the only slot available. Now insert 24 and give the probe sequence:

Probe sequence: 3, 0

0	1	2	3	4	5	6
24	x	x	10	x	x	x

Now do this for every possible scenario where only 1 slot is unoccupied. Notice that for some cases, you may have to traverse the table more than once. Write out the probe sequence for each case.

ANS:

Probe sequence: 3, 0, 4, 1

0	1	2	3	4	5	6
x	24	x	10	x	x	x

Probe sequence: 3, 0, 4, 1, 5, 2

0	1	2	3	4	5	6
x	x	24	10	x	x	x

Probe sequence: 3, 0, 4

0	1	2	3	4	5	6
x	x	x	10	24	x	x

Probe sequence: 3, 0, 4, 1, 5

0	1	2	3	4	5	6
x	x	x	10	x	24	x

Probe sequence: 3, 0, 4, 1, 5, 2, 6

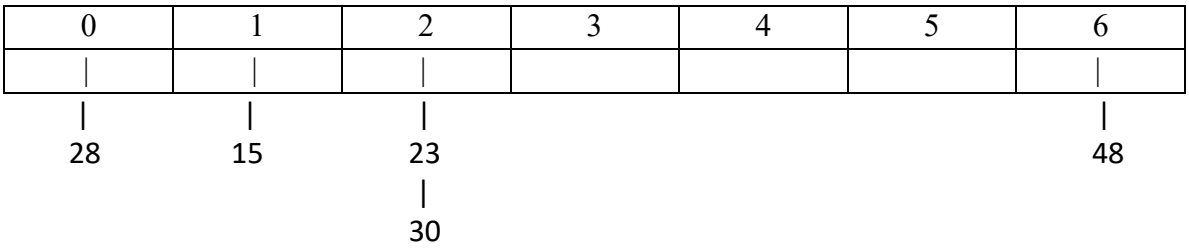
0	1	2	3	4	5	6
x	x	x	10	x	x	24

4. Use the hash function `hash(key) = key % 7` and **separate chaining** to insert the following keys into the hash table:

```
15, 30, 28, 48, 23
```

Remember that in separate chaining, a value in the table at a given slot (or index) is a node. Use an arrow to refer to the first node in the chain.

ANS:



Notes:

1) The straight lines should be drawn as arrows.
2) Assume the number represents node values in a `Node`.
3) When inserting 23 in the linked list at slot 2, whether 23 goes to the end of the linked list or the beginning depends on the implementation of the linked list. Here we are inserting at the beginning of the linked list.