You may work in pairs for this assignment. If you choose to work with a partner, make sure only one of you submits a solution, and you paste a copy of the Partners Template that contains the names and PIDs of both students at the beginning of the file you submit.

You will submit your solution to this assignment to the Curator System (as `HW02`). Your solution must be either a plain text file (e.g., NotePad++) or a typed MS Word document; submissions in other formats will not be graded.

Partial credit will only be given if you show relevant work.

Part I Hash Table Basics

1. Consider a hash table consisting of $M = 11$ slots, and suppose nonnegative integer key values are hashed into the table using the hash function `h1()`:

```
public static int h1( int key ) {

   int x = (key + 7) * (key + 13);
   x = x >> 4;
   x = x + key;
   return x;
}
```

a) [27 points] Suppose that collisions are resolved by using quadratic probing, with the probe function:

$$\left(k^2 + k\right)/2$$

The integer key values listed below are to be inserted, in the order given. Show the home slot (the slot to which the key hashes, before any probing), the probe sequence (if any) for each key, and the final contents of the hash table after the following key values have been inserted in the given order:

Key Value	Home Slot	Probe Sequence
43	9	none
23	2	none
15	9	10
27	2	3
31	3	4
14	5	none
37	9	10, 1
21	3	4, 6
29	2	3, 5, 8

Final Hash Table:

Slot	0	1	2	3	4	5	6	7	8	9	10
Contents		37	23	27	31	14	21		29	43	15

b) **[27 points]** Suppose that collisions are resolved by using double hashing (see the course notes), with the secondary hash function `Reverse(key)`, which reverses the digits of the key and returns that value; for example, `G(key) = Reverse(7823) = 3287`.

The integer key values listed below are to be inserted, in the order given. Show the home slot (the slot to which the key hashes, before any probing), the probe sequence (if any) for each key, and the final contents of the hash table after the following key values have been inserted in the given order:

Key Value	Home Slot	Probe Sequence
43	9	none
23	2	none
15	9	5
27	2	8
31	3	none
14	5	2, 10
37	9	5, 1
21	3	4
29	2	6

Final Hash Table:

Slot	0	1	2	3	4	5	6	7	8	9	10
Contents		37	23	31	21	15	29		27	43	14

Probe calculations:

15: (9 + 51) % 11 == 60 % 11 == 5
27: (2 + 72) % 11 == 74 % 11 == 8
14: (5 + 41) % 11 == 46 % 11 == 2, (5 + 2*41) % 11 == 87 % 11 == 10
37: (9 + 73) % 11 == 82 % 11 == 5, (9 + 2*73) % 11 == 155 % 11 == 1
21: (3 + 12) % 11 == 15 % 11 == 4
29: (2 + 92) % 11 == 94 % 11 == 6

2. **[10 points]** The hash functions we have examined tend to use bitwise operations instead of arithmetic operations, because bitwise operations execute more quickly. You may have noticed that bitwise XOR is used frequently. Is there any reason to prefer XOR over bitwise AND or bitwise OR? Explain.

Answer:
Let X be a bit.

Then X & 0 == 0, no matter what X is. So, using AND would tend to produce lots of 0's in the hash value, so once you get a 0, it remains a 0. That would probably reduce scattering.

And, X | 1 == 1, no matter that X is. So, using OR would tend to produce lots of 1's in the hash value, again bad.

But X XOR 0 == 1 if X == 1 and 0 if X == 0, and X XOR 1 == 0 if X == 1 and 1 if X == 0, so this tends to flip bits on successive applications of XOR.

3. A hash table is being implemented, using double hashing to resolve collisions. Haskell Hoo proposes to make this more efficient by adopting the following scheme. We already have our primary hash function, `H()`, to hash the key value. Rather than use a second hash function, which would require another function evaluation, Haskell proposes that we just reuse the hash value from the primary function to compute the step length for collision resolution as follows:

```
step = 1 + H(key) % 23
```

a) **[10 points]** Why would Haskell have decided to add 1 in his proposal?

Answer:
H(key) % 23 could be anything from 0 to 22. Without the added 1, this could yield a step of 0, which would lead to simply jumping up and down on the home slot when double hashing is applied. Of course, that would make the insertion impossible.

b) **[10 points]** The goal of double hashing is that keys that collide in the same slot will (hopefully) produce different probe sequences. Will Haskell's proposal be likely to achieve that goal? Explain.

Answer:
Records with different keys can collide for two different reasons:

- **their keys yield the same hash value, and hence MOD to the same remainder**
- **their keys yield different hash values, but those values MOD to the same remainder**

In the second case, Haskell's scheme would generate the same probe sequence for both records, which is what we are trying to avoid, since that promotes clustering.

So, it seems better to base double hashing on a different hash function than the original, since that at least gives us a chance that we'll get different probe sequences for the two records.

4. Another hash table is being designed, and chaining, with singly-linked lists, will be used to resolve collisions. Haskell Hoo objects to the decision to use chaining, on the grounds that this could result in a more expensive search for some elements than if simple linear probing were used.

a) **[8 points]** Describe a scenario (perhaps unlikely) under which Haskell's claim could be correct. Note that some details of the proposed strategy have not been fully specified above, so that multiple options can be considered.

Answer:
We didn't specify how the chains are managed. Suppose that when we add a new element to a slot, we add it to the front of the chain, rather than the rear. In that case, searching for the first element that hashed to a slot will require traversing the whole chain, so if K records collide in that slot, we'll have to do K comparisons to find the one that was inserted first.

OTOH, if linear probing was used, we'd find that record in one comparison.

b) **[8 points]** Haskell says that his concern would be eliminated if the elements in the singly-linked lists were kept in sorted order. Is he correct? Explain.

Answer:
No.

Now, the first element to hash to a slot could be anywhere in the chain. In particular, if the first key that hashed to the slot was very large, that guarantees it will be near the tail of the chain.

However, there is an even more fundamental objection. Hash tables do not require that the data object implement compareTo(), just a hash function and an equals() method. So, this scheme may not even be possible.