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.

---

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 + 11) * (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	5	
23	5	5+1 -> 6
15	3	
27	4	
31	9	
14	1	
37	3	3+1 -> 4, 3+3 -> 6, 3+6 -> 9, 3+10 -> 2
21	5	5+1 -> 6, 5+3 -> 8
29	3	3+1 -> 4, 3+3 -> 6,3+6 -> 9, 3+10 -> 2, 3+15 -> 7

Final Hash Table:

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

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(7823) = 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	5	
23	5	5+32->4
15	3	
27	4	4+72->10
31	9	
14	1	
37	3	3+73->10, 3+146->6
21	5	5+12->6, 5+24->7
29	3	3+92->7, 3+184->0

Final Hash Table:

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

2. A hash table is being implemented, using double hashing to resolve collisions. We already have our primary hash function, `H()`, to hash the key value, and we have decided on a table size of 997, expecting to store no more than about 500 elements in the table.

Haskell Hoo proposes to make double hashing more efficient by adopting the following scheme. Rather than use a second hash function, which would require another function evaluation, Haskell proposes that we just calculate the step distance for probing this way:

```
step_distance = 997 - home_slot_index
```

a) **[12 points]** One potential problem with double hashing is that it might only access a few table slots before repeating itself. Would Haskell's scheme avoid **that difficulty**? Explain. (There may be other difficulties with this idea, but we are not considering them here.)

The only way double hashing will repeat itself in this manner is if the step size is a divisor of the table size. In this case, the table size is 997, which is prime. So the only divisors are 1 and 997.

So, if the home slot index is 996 we get a step size of 1, which will result in simple linear probing and not repeat until we have cycled through all the slots. But, if the home slot is 0, we get a step size of 997, which is 0 when modded by 997, and in that case double hashing will just repeatedly access the home slot.

So, no, this does not entirely avoid the problem described here.

b) **[12 points]** In any case, no matter what conclusions you reached in part a), there is a fundamental, **fatal** flaw in Haskell's scheme for double hashing that has nothing to do with the issue considered in part a). Explain.

With this scheme, every element that collides in a slot will get the same probe step size, and therefore follow exactly the same probe sequence; that's exactly what double hashing is intended to avoid.

As a second alternative, Haskell proposes computing the probe step distance for double hashing by reusing the result from the original hash function, instead of having to evaluate a second function. Specifically, he proposes that we compute the step distance for probing this way:

```
step_distance = 17 + H(key)
```

c) **[12 points]** Would Haskell's second scheme avoid **the difficulty considered in part a)**? Explain. (There may be other difficulties with this idea, but we are not considering them here.)

Again, no. For example, if H(key) == 980, we get a step distance of 997, and we will simply bounce around on the home slot.

d) **[10 points]** Would Haskell's second scheme avoid **the difficulty you identified in part b)**?

No. If two elements collide in the same slot, it is entirely possible that their keys hashed to the same integer value. In that case, both elements will have the same step size. (That's why double hashing uses a second hash function.)