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 `HW02P1` and `HW02P2`). 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()`:

```
int h1 (int key) {
  int x = (key + 7) * (key + 7);
  x = x / 16;
  x = x + key;
  x = x % 11;
  return x;
}
```

a) [5 points] Suppose that collisions are resolved by using linear probing. 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	1	
23	2	
1	5	
0	3	
15	1	2, 3, 4
31	0	
4	0	1, 2, 3, 4, 5, 6
7	8	
11	9	
3	9	10

Final Hash Table:

Slot	0	1	2	3	4	5	6	7	8	9	10
Contents	31	43	23	0	15	1	4		7	11	3

The home slot computations look like:

```
key   hash  home
43:   199   1
23:   79    2
 1:   5     5
 0:   3     3
15:   45    1
31:   121   0
 4:   11    0
 7:   19    8
11:   31    9
 3:   9     9
```

b) [5 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	1	
23	2	
1	5	
0	3	
15	1	2, 4
31	0	
4	0	1, 3, 6
7	8	
11	9	
3	9	10

Final Hash Table:

Slot	0	1	2	3	4	5	6	7	8	9	10
Contents	31	43	23	0	15	1	4		7	11	3

c) [5 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, `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	1	
23	2	
1	5	
0	3	
15	1	8
31	0	
4	0	4
7	8	8, 0, 7
11	9	
3	9	1, 4, 7, 10

Final Hash Table:

Slot	0	1	2	3	4	5	6	7	8	9	10
Contents	31	43	23	0	4	1		7	15	11	3

The probing goes as follows:

```
15:  G(15) == 51 → 1 + 51 % 11 == 8
4:   G(4) == 4 → 0 + 4 % 11 == 4
7:   G(7) == 7 → 8 + 7 % 11 == 4, 4 + 7 % 11 == 0, 0 + 7 % 11 == 7
3:   G(3) == 3 → 9 + 3 % 11 == 1, 1 + 3 % 11 == 4, 4 + 3 % 11 == 7,
                  7 + 3 % 11 == 10
```

2. Consider implementing a hash table for an application in which we will build an initial hash table by inserting a substantial collection of records. After this, we expect that the number of insertions and the number of deletions performed to be roughly the same, although there may be long runs of consecutive insertions or consecutive deletions. Furthermore, the table will use a probe strategy to resolve any collisions that occur during insertion, and therefore we will "tombstone" cells from which a record has been deleted.

a) [10 points] If we implement the hash table described above, then when we search for a record, we cannot conclude the record is not in the table until we have found an empty cell in the table, not just a tombstone. (We will ensure that the table never reaches the state that there are no empty cells.) Explain carefully why the search cannot stop when a tombstone is encountered.

Suppose that when some element, say X, is inserted to the table, it collides with a previously-inserted element Y; that will imply X is inserted after one or more additional probe steps beyond the location of Y.

Now, suppose Y is deleted from the table, its location is marked as a tombstone, and nothing else is inserted that fills that slot in the table. Then, when we search for X, we will encounter the tombstone before we find X. If we used the tombstone as an indication that X is not in the table, we would be wrong.

But, as more and more deletions occur, the number of tombstones will affect the performance of the hash table, since those cells are essentially locked into an unusable state. Therefore, we could adopt the following strategy on insertion. If we pass a tombstoned cell during an insertion search, we will remember the first such cell we have seen, and once the insertion search concludes at an empty cell, we will place the new record into that tombstoned cell.

b) [10 points] If we implement insertion as described above, would searches for absent records (as described in part a) still need to proceed until an empty cell is found? Justify your conclusion carefully.

Yes, nothing would change regarding searches for elements that are NOT in the table. Even if we "recycle" tombstones, a tombstone will still be present for some time before it is recycled, if it ever is. So the situation described in the previous part is still possible.

3. Suppose you are implementing a hash table and are trying to choose between using a probing strategy and using chaining (each slot is actually a linked list that can hold as many records as needed). You will use the same hash function and hash table size no matter which strategy you select.

a) [10 points] A *primary collision* occurs when two records map to the same home slot in the table. Will the number of primary collisions be lower if you choose probing or if you choose chaining? Justify your conclusion carefully.

The home slot of a record depends only on the key, the hash function, and the size of the table. The collision resolution strategy does not come into play in selecting the home slot.

Therefore the number of primary collisions will be same no matter what collision resolution strategy is used.

b) [10 points] One objection to using chaining is that you'll have to perform a linear traversal of the linked list to find a record, and linear traversals are slow. Considering that, and assuming that a small but significant number of primary collisions will occur, would searching be more efficient if you chose linear probing instead of chaining? Justify your conclusion carefully.

In short, no. Using linear probing cannot require fewer steps than chaining, under the terms of the question*.

When chaining is used, the only records that need to be traversed are those that actually collided in the slot in question.

When linear probing is used, elements that hash to different home slots can collide as probing is performed. If that occurs, then searching for those elements will require looking at more elements than if chaining were used. In the best case, the number of elements we encounter will be the same as when chaining were used.

***There is one quibble. The question statement did not actually say that these were the ONLY keys that collided in this slot. If other keys reached this slot before the ones listed, then the number of comparisons would be greater.**

c) [10 points] Reconsider part b under the assumption that you'd choose quadratic probing as an alternative to chaining.

Quadratic probing will decrease the probability that "secondary" collisions will occur during probing, since elements that hash to adjacent slots will (for the most part) probe to a different sequence of slots.

But there is still a chance of "secondary" collisions with quadratic probing. So, the conclusion is essentially the same.