1. **Problem 3.1** (3 pts) Consider inserting keys 10, 22, 17, 28, 15, 4, 31, 88, 59 into a hash table of length $m = 11$ using open addressing with the hash function $h_1(k) = k \bmod m$. Ilustrate the result of inserting these keys using either linear probing, quadratic probing with $c_1 = 1$ and $c_2 = 3$, or double hashing with $h_2(k) = 1 + k \bmod (m - 1)$

   ## Answer

   (a) **Linear Probing** hash function : $h_1(k) = k \bmod m$
   collision resolution : increment 1 or in mathematical if there exist collision then

   $$h(k, i) = h(k) + i \bmod m$$

   at first we have empty hash table like this

key	0	1	2	3	4	5	6	7	8	9	10
value	-	-	-	-	-	-	-	-	-	-	-

   to insert 10 to the hash table $h(10) = 10 \bmod 11$ then we insert 10 to the 10^{th} key

key	0	1	2	3	4	5	6	7	8	9	10
value	-	-	-	-	-	-	-	-	-	-	10

   next insert 22 to hash table $k(22) = 22 \bmod 11 = 0$ then we insert 22 to the 0 key

key	0	1	2	3	4	5	6	7	8	9	10
value	22	-	-	-	-	-	-	-	-	-	10

   next insert 17 to hash table $k(17) = 17 \bmod 11 = 6$

key	0	1	2	3	4	5	6	7	8	9	10
value	22	-	-	-	-	-	17	-	-	-	10

   next for $k(28) = 28\,11 = 6$ as we see there is no slot for 6^{th} key then we just choose at $6 + 1^{th}$ key

key	0	1	2	3	4	5	6	7	8	9	10
value	22	-	-	-	-	-	17	28	-	-	10

   then we skip the process for simplicity now we have the hash table are given like this

   $k(15) = 15 \bmod 11 = 4$
   $k(4) = 4 \bmod 11 = 4$ (collision at key 4 to fifth key)
   $k(31) = 31 \bmod 11 = 9$
   $k(88) = 88 \bmod 11 = 0$ (collision at key 0 to first key)
   $k(59) = 59 \bmod 11 = 4$ (collision again at key 4,5,6,7 to 8^{th} key)

key	0	1	2	3	4	5	6	7	8	9	10
value	22	88	-	-	15	4	17	28	59	31	10

   (b) Quadratic Probing
   hash function :

   $$h_1(k) = k \bmod m$$

   collision resolution :

   $$h_2(k, i) = h_1(k, i) + c_1 i + c_2 i^2 = h_1(k) + i + 3i^2$$

   to simplify process the computation are given by

k	10	22	17	28	15	4	31	88	59
$h_1(k)$	10	0	6	6	4	4	9	0	4
$h_2(k,1) = h(k) + 1 + 3.1^2 \bmod 11$				10 collision		8		4 collision	8 collision
$h_2(k,2) = h(k) + 2 + 3.2^2 \bmod 11$				9 collsion				3 collision	7
$h_2(k,3) = h(k) + 3 + 3.3^2 \bmod 11$				3				8 collision	
$h_2(k,4) = h(k) + 4 + 3.4^2 \bmod 11$								8 collision	
$h_2(k,5) = h(k) + 5 + 3.5^2 \bmod 11$								1	

then the final hash table are given by

key	0	1	2	3	4	5	6	7	8	9	10
value	22	88	-	28	15	-	17	59	4	31	10

(c) double hashing hash function :

$$h_1(k) = k \bmod m$$

collision resolution : double hashing

$$h(k,i) = [h_1(k) + ih_2(k)] \bmod 11 \qquad h_2(k) = 1 + k \bmod (m-1)$$

the answer is given by

k	10	22	17	28	15	4	31	88	59
$h_1(k)$	10	0	6	6 collsion	4	4 collision	9	0 collision	4 collision
$h_2(k)$	1	3	8	9	6	5	2	9	10
$h(k,1)$				4 collision		9 collision		9 collision	8
$h(k,2)$				2		3		7	

then the final hash table using double hashing are given by

key	0	1	2	3	4	5	6	7	8	9	10
value	22	-	28	4	15	-	15	88	59	31	10

2. **Problem 3.2** (1 pt) suppose that we are storing a set of n key into a hash table of size m, what is the best-case searching time, and what is the worst-case searching time ? Write down your answer in the form of $T(n) = \Theta(g(n))$. Briefly discuss your answer

   - **Best case searching time** the best case occur when the each key, has a unique index, implies there is no collision so the best searching time is
   $$T(n) = O(1)$$
   - **Worst case searching time** the worst case occur when all the given key, has same index in hash table, it implies single key contain all n elements. the worst searching time is
   $$T(n) = n$$

3. **Problem 3.3** (2 pts) Discuss when (1) we have a very small set of possible keys $|U|$; (2) we have very large number of inserted keys n, relative to the number of all possible key $|U|$ will hashing still be useful ? Explain your answer.

   ## Answer

   when we have very small set of possible keys $|U|$ hashing still can be usefull, it more likely each key will hash into unique index, it reduce collision.

   when we have very large number inserted keys n relative to number of all possible keys $|U|$, hash will be useless since it becomes less efficient, hashing will separate the data into some cluster. in this scenario, if n too big larger than $|U|$ the hash table will be dense, that will so many collision since we only have small unique keys. as the result the searching time will increase, and less efficient to perform hashing. it's recommended to have hash table that runs in $O(1)$.

4. **Problem 3.4** (1 pt) If the order of the key insertion sequence sequence change, will be obtain the same hash result? Briefly discuss your answer.

   ## Answer

   NO!, changing order key insertion on sequence will lead different hash result. especially if there has a collision, but whereas the hashing result will same, in other words only unique key will produce unique index implies no collision, so it doesn't matter to change the sequence change.

   But there is no guarantee if we change the sequence will obtain same hash result, the guarantes happen when there is no collision.

5. **Problem 3.5** (3 pts) Write pseudocode for HASH-DELETE (using the pseudo-codes similar to slides no. 24 & 25), and modify HASH-lNSERT accordingly to handle this situation. Let us assume the link to the element that we want to delete has been given

```
HASH-DELETE(T, k)
i = 0
repeat
    j = h(k, i)
    if T[j] == k
        T[j] = 'Hello Kitty' // Delete the key by setting it to 'Hello Kitty'
        return j
    else
    i = i + 1
until i == m
error "Key not found in the hash table"


MODIFIED-HASH-INSERT(T, k)
i = 0
repeat
    j = h(k, i)
    if T[j] == NIL or T[j] == 'Hello Kitty'
        T[j] = k
        return j
    else
    i = i + 1
until i == m
error "hash table overflow"
```

there is some issue if we not tag the deleted items, with some tag for example at here is hello kitty. so we must give 'hello kitty' for the deleted key