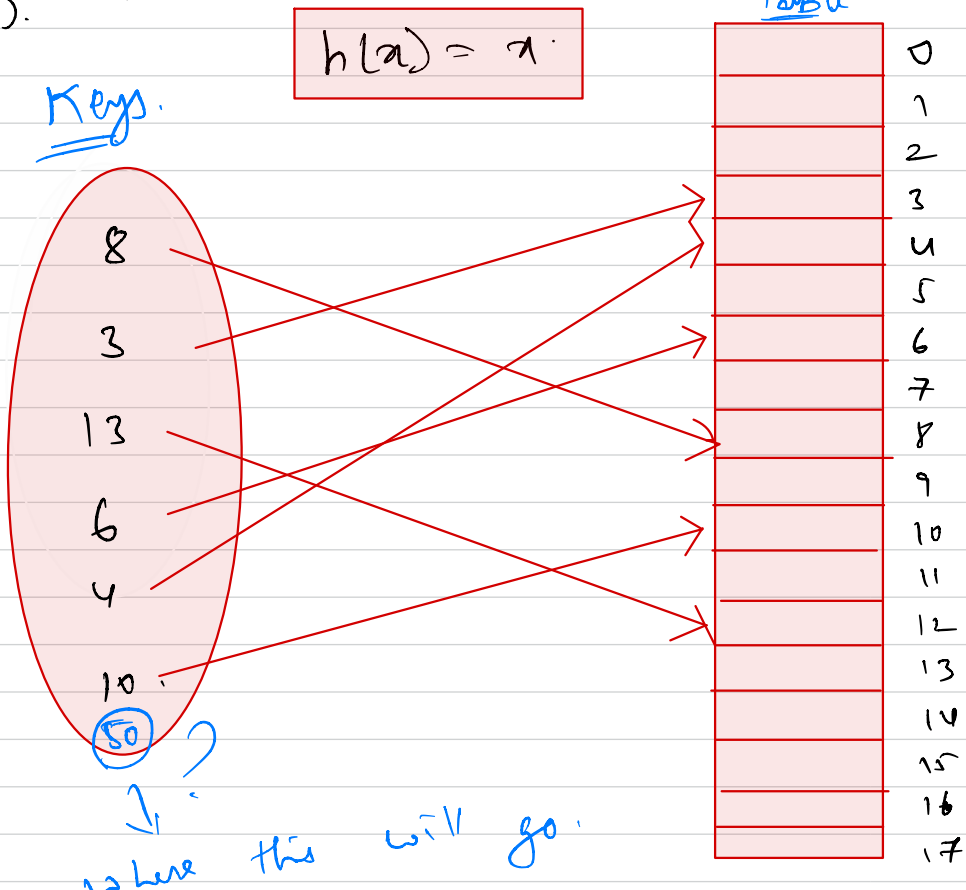
HASHING.

HASHING.

Theory:-

→ Hashing is a technique or process of mapping keys and values into hashtable by using a hash function.

→ It usually has insert and search in O(1).



↳ where this will go.

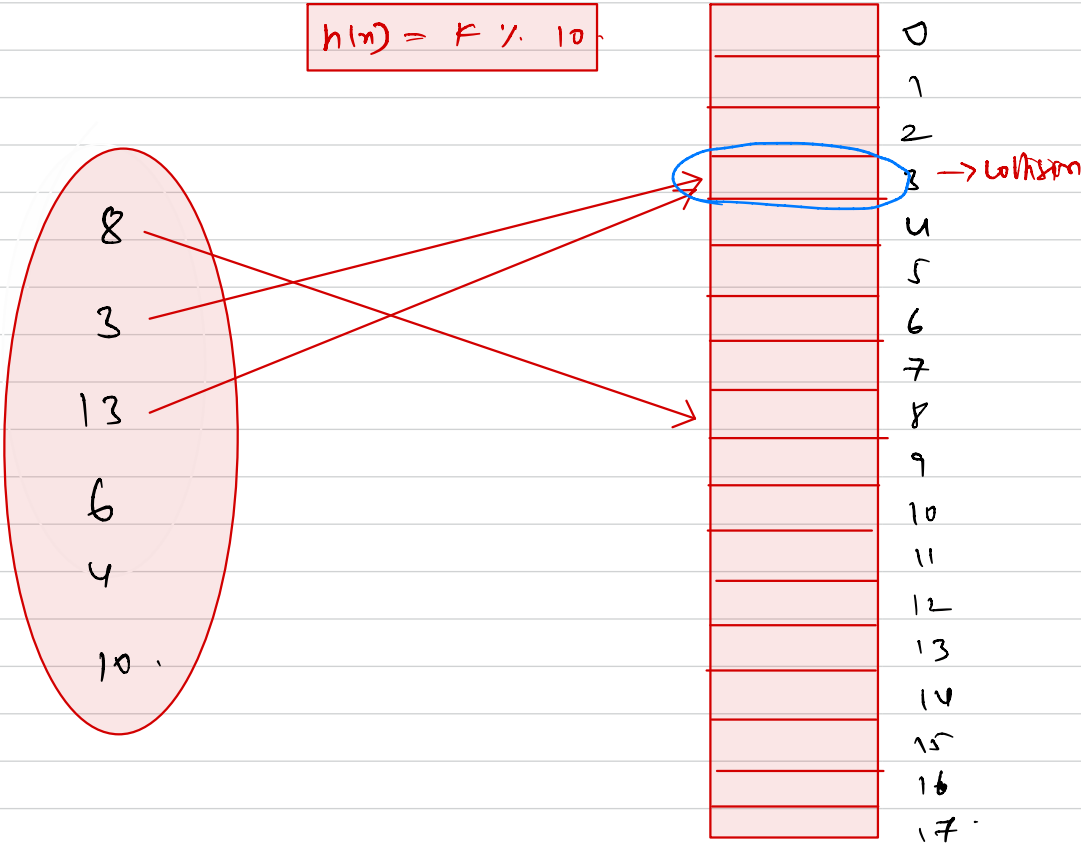
↳ This can be resolved using proper hash functions.

Hash functions:- There can be different types of Hash functions.

1. K mod 10
2. K mod n.
3. Mid Square.
4. Folding method.

→ most important.

→ Let us now consider Hash function K mod 10.



***

→ When you change the hash function there is a possibility that two keys are now mapped to same index of hash table. This is called Collision.

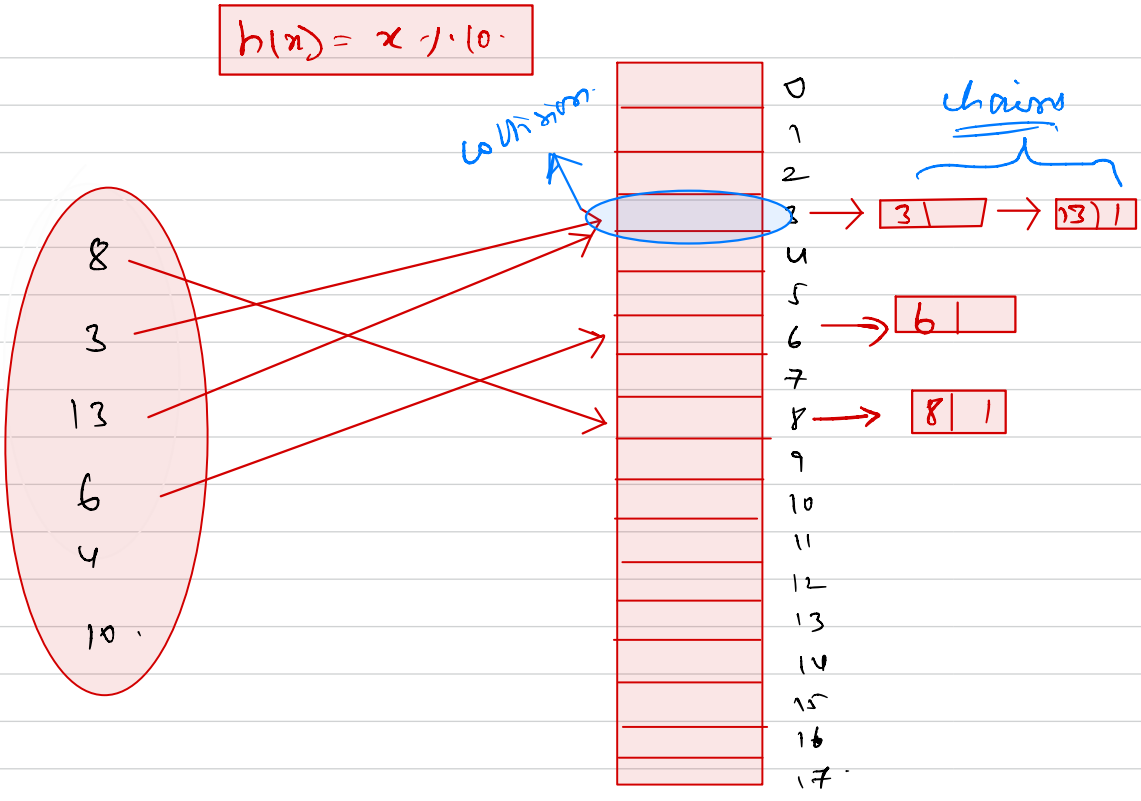
Collision resolution Techniques :-

1. Open Hashing.
   (i) — Chaining.
2. Closed Hashing.
   (i) Linear probing.
   (ii) Quadratic probing.

1. Open Hashing :- when there is a collision element will be added to a new node in a chain attached to the index. Chain is usually made of linked list.

Disadvantages :- Now the search complexity of Hash table is not O(1) because now we have to search over chains.

→



② Linear probing :- Here if there is a collision then we search for next empty slot in Hash table and store it there.

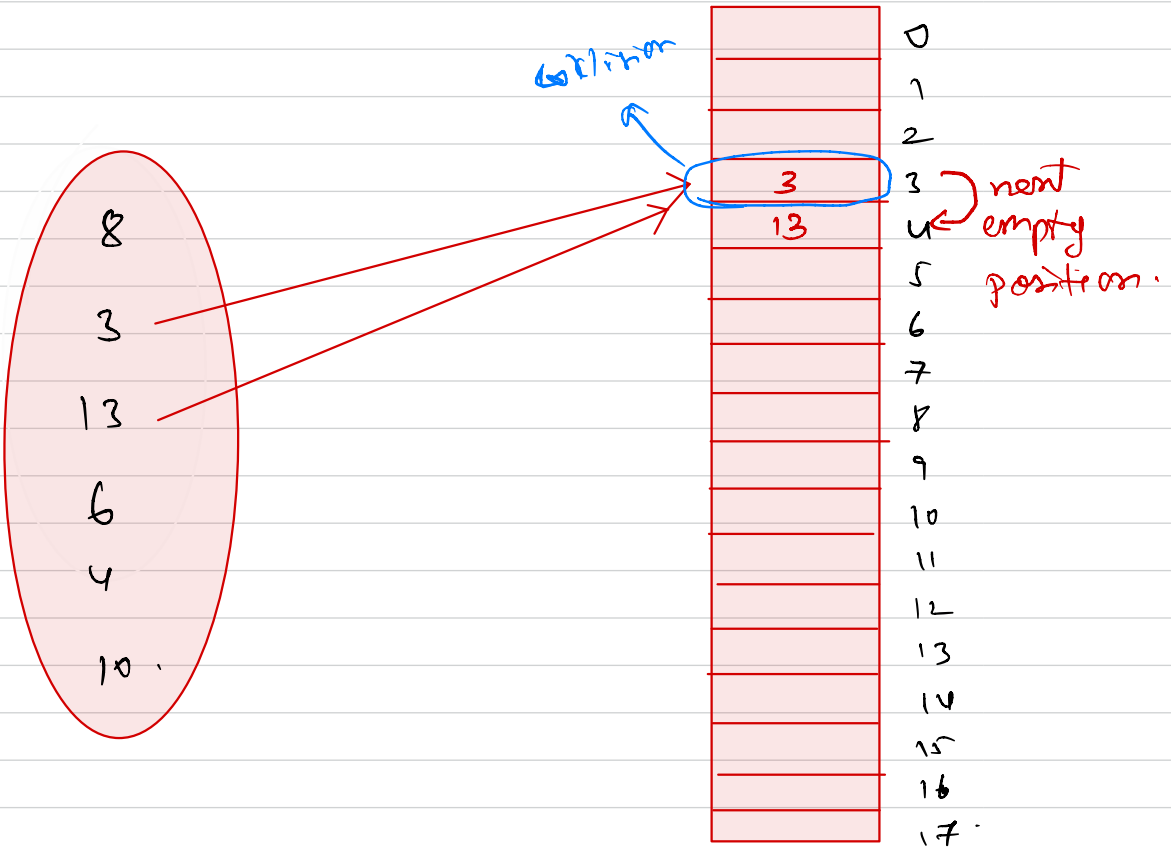
$h(x) = x \% \text{ size}$.

↓ new functions.

$$h(x) = [h(x) + f(i)] \% \text{ size}$$

$f(i) = i$

$i = 0, 1, 2, \ldots$



$h'(13) = [h(13) + f(0)] \% 10 \quad f(i) = 0, 1, 2, 3 \ldots$

$= 3.$

$h'(13) = [h(13) + f(1)] \% 10.$

$= 4. \rightarrow$ next index.

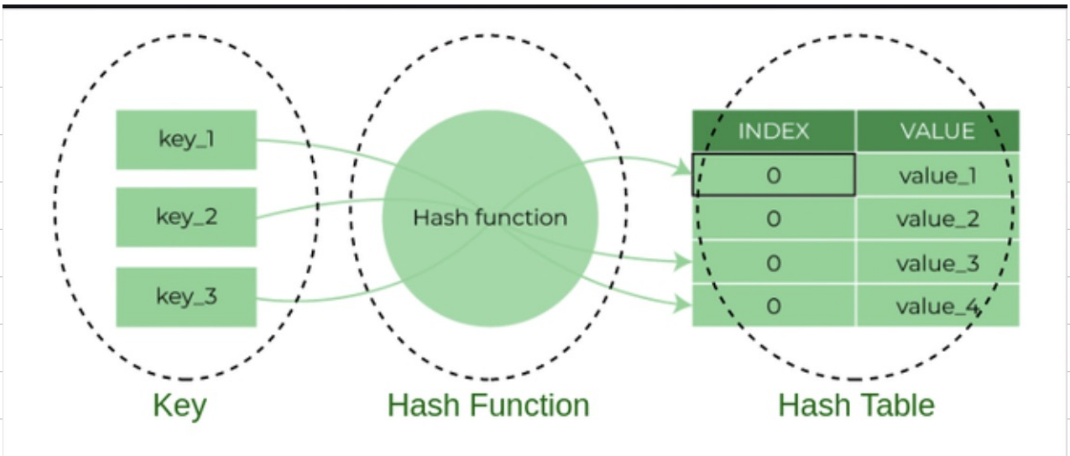
↳ keep doing until empty index is found.

③ **Quadratic Probing** :- This is also similar to linear probing but slightly different hash function.

$$h'(x) = [h(x) + f(i)] \% \text{ size}$$

$$f(i) = i^2$$

$$i = 0, 1, 2, 3 \ldots$$



Components of Hashing

Reference :- Geeks For Geeks.

Problems :-

Two Sum :-

Problem 1 :- Find indices in array such that they add upto target.

Input :- nums = [2, 7, 11, 15], target = 9.

output :- [0, 1].

nums[0] + nums[1] == 9.

Soln :-

→ The most important fact for this problem is "There is exactly one solution".

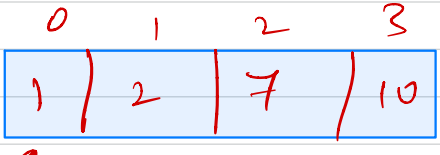
→ We can calculate complement and check if that is present.

Complement = target − current num.

↓

check if other number which adds target is present.

```
{
    Map<Integer, Integer> map = new HashMap();
    for (int i=0; i< nums.length; i++)
    {
        int complement = target - nums[i];   → most important thing.
        if( map.containsKey(complement))
        {
            return new int[] { i, map.get(complement)};
        }else {
                map.put(nums[i], i);
            }
}
```



→ → → look for complement (2).

key nums	value index
1	0
2 →	1

[2, 1]

Problem 2 :- Ransom note

Given two strings ransomNote and magazine, return true if ransomNote can be constructed by using letters from magazine.

Imp ↗ most imp part.

Input:- ransomNote = "aa".
magazine = "aab".
= true;

Each letter can be used only once.

Sol:-

→ **Main logic :-** The number of characters in ransom note should be present in magazine

→ Basically we have to calculate each and every character of magazine and see all characters of ransome note are present.

magazine = "aab".

Key	Count.
a	~~2~~ ~~1~~ 0
b	1.

← a
a

ransomnote = "aa".

↓

keep looking in Hash table and reduce number.

→ Every character should be greater than 0 in the table.