Searching: Sequential Search, Binary Search.

Hashing: Hash Functions: Truncation, Mid-square Method, Folding Method, Division Method.

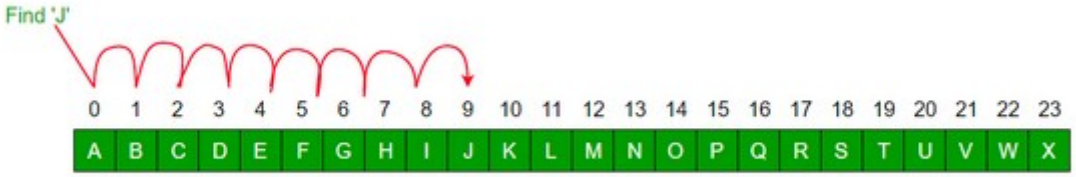
Collision Resolution: Open Addressing: Linear Probing, Quadratic Probing, Double Hashing, Separate Chaining Bucket Hashing.

Linear (Sequential) Search

A linear search scans one item at a time, without jumping to any item .

1. The worst case complexity is O(n), sometimes known an O(n) search
2. Time taken to search elements keep increasing as the number of elements are increased.

Linear Search to find the element “J” in a given sorted list from A-X

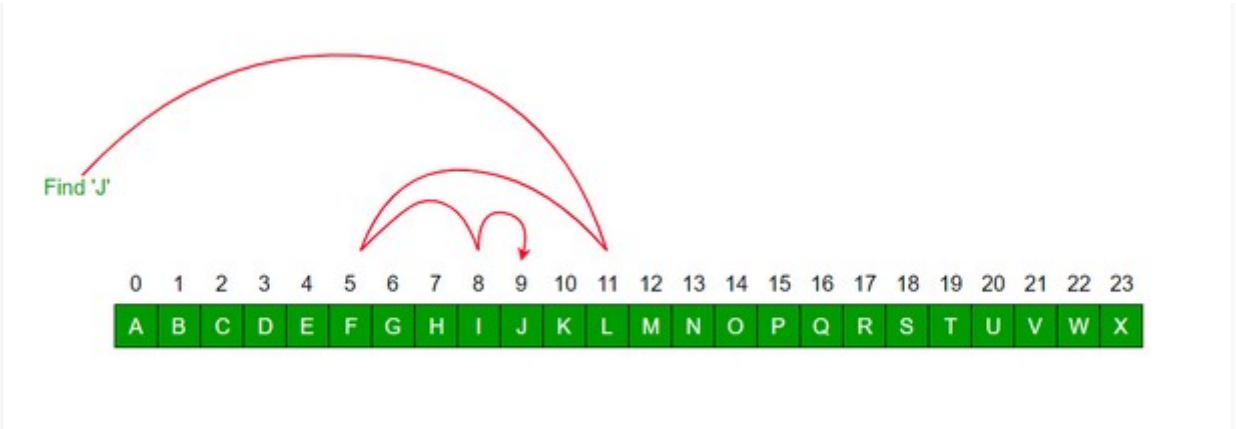


Binary Search

A binary search however, cut down your search to half as soon as you find middle of a sorted list.

1. The middle element is looked to check if it is greater than or less than the value to be searched.
2. Accordingly, search is done to either half of the given list

Binary Search to find the element “J” in a given sorted list from A-X



Important Differences

- Input data needs to be sorted in Binary Search and not in Linear Search
- Linear search does the sequential access whereas Binary search access data randomly.
- Time complexity of linear search -O(n) , Binary search has time complexity O(log n).
- Linear search performs equality comparisons and Binary search performs ordering comparisons

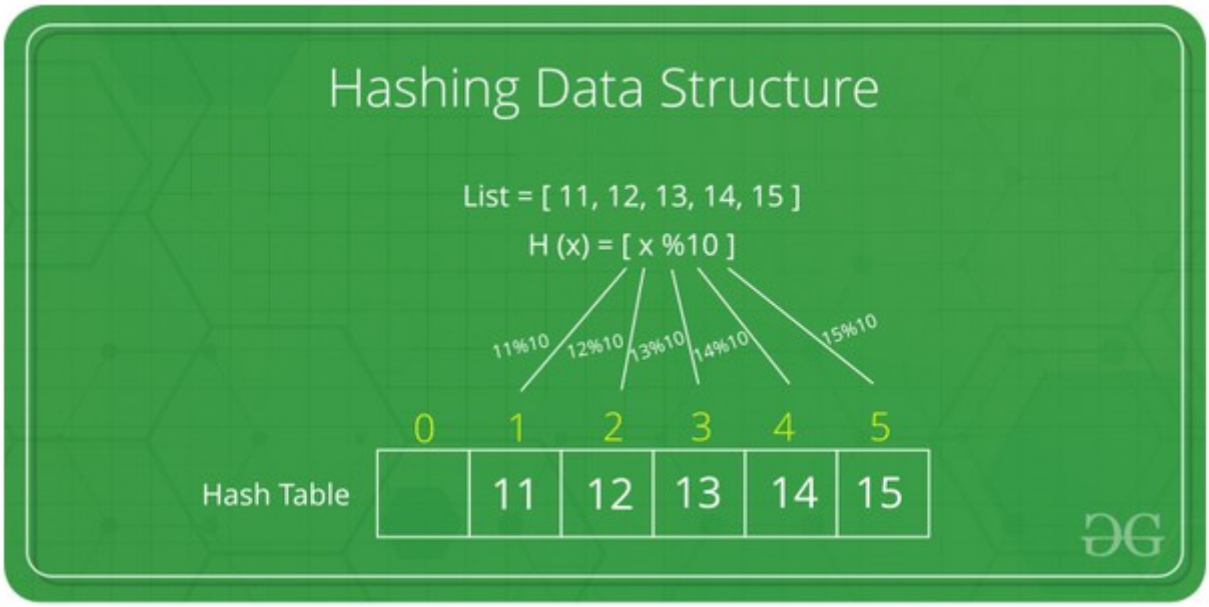
Hashing Data Structure

Hashing is a technique or process of mapping keys, values into the hash table by using a hash function.

It is done for faster access to elements.
The efficiency of mapping depends on the efficiency of the hash function used.

Let a hash function H(x) maps the value x at the index x%10 in an Array.

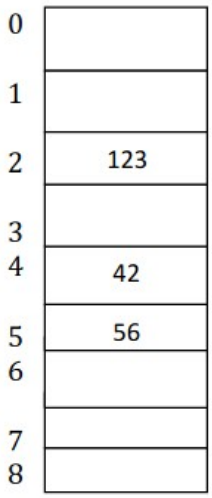
For example if the list of values is [11,12,13,14,15] it will be stored at positions {1,2,3,4,5} in the array or Hash table respectively.



Various types of Hash Functions:
Type 1: Truncation Method
Type 2: Folding Method
Type 3: Midsquare Method
Type 4: Division Method(Modulo Division)

1 Truncation Method

The Truncation Method truncates a part of the given keys, depending upon the size of the hash table.
1. Choose the hashtable size.
2. Then the respective right most or left most digits are truncated and used as hash code| value.
Ex: 123,42,56 Table size = 9



H(123)=1

H(42)=4

H(56)=5

2 Midsquare Method :

It is a Hash Function method.

1. Square the given keys.
2. Take the respective middle digits from each squared value and use that as the hash value | address | index | code, for the respective keys.

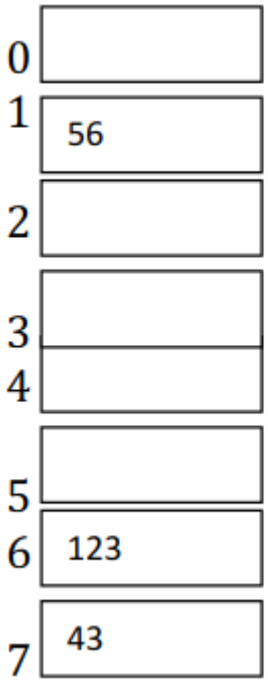
0	
1	123
2	
3	56
4	
5	
6	
7	42
8	
9	

$H(123)=1\ [\ 123^2 = 15\underline{1}29]$

$H(42)=7\ [\ 42^2 = 1\underline{7}\ \underline{6}4\ ]$

$H(56)=3\ [\ 56^2 = 31\underline{3}6\ ]$

Folding Method :
Partition the key K into number of parts, like K1,K2,.....Kn, then add the parts together and ignore the carry and use it as the hash value



H(123)= [1+2+3 =6]

H(43)= [4+3 = 7]

H(56)= [5+6 = 11]

4 Division Method :

Choose a number m, larger than the number of keys. The number m is usually chosen to be a **prime number.**

The formula for the division method :

Hash(key)= key % tablesize

Tablesize : 10
keys: 20, 21,24,26,32,34

0	20
1	21
2	32
3	
4	24
5	
6	26
7	
8	
9	

34 collision

H(20)= 20 % 10 = 0

H(21)= 21 % 10 = 1

H(24)=24 % 10 = 4

H(26)= 26 % 10 = 6

H(32) = 32 % 10 = 2

What is Collision?
Since a hash function gets us a small number for a key which is a big integer or string, there is a possibility that two keys result in the same value. The situation where a newly inserted key maps to an already occupied slot in the hash table is called collision and must be handled using some collision handling technique.

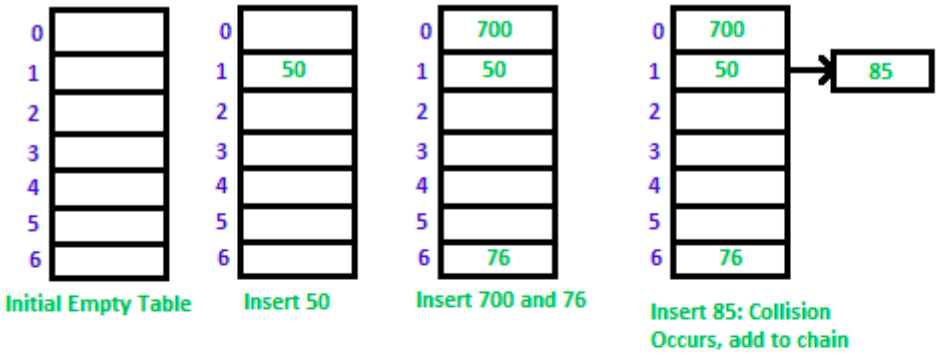
How to handle Collisions?

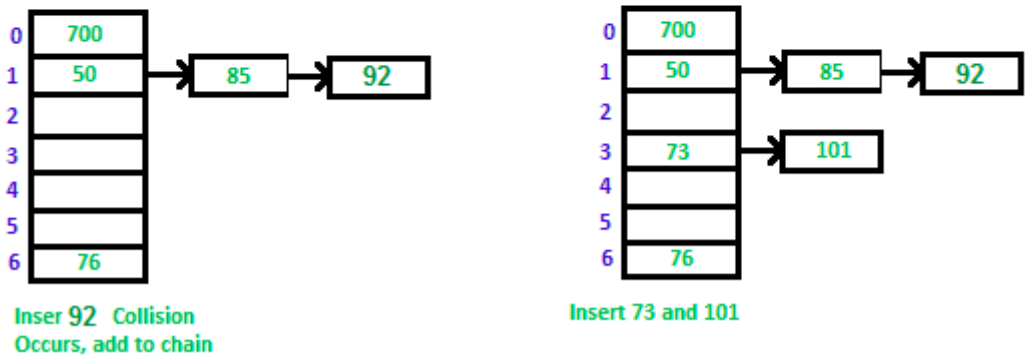
COLLISION RESOLUTION TECHNIQUES

It is the process of finding another position for the collided record.
The techniques are:

Separate Chaining:
The idea is to make each cell of hash table point to a linked list of records that have same hash function value.

Let us consider a simple hash function as "**key mod 7**"
and sequence of keys as 50, 700, 76, 85, 92, 73, 101.





Open Addressing

Like separate chaining, open addressing is a method for handling collisions.
In Open Addressing, all elements are stored in the hash table itself.

So at any point, the size of the table must be greater than or equal to the total number of keys

Open Addressing is done in the following ways:

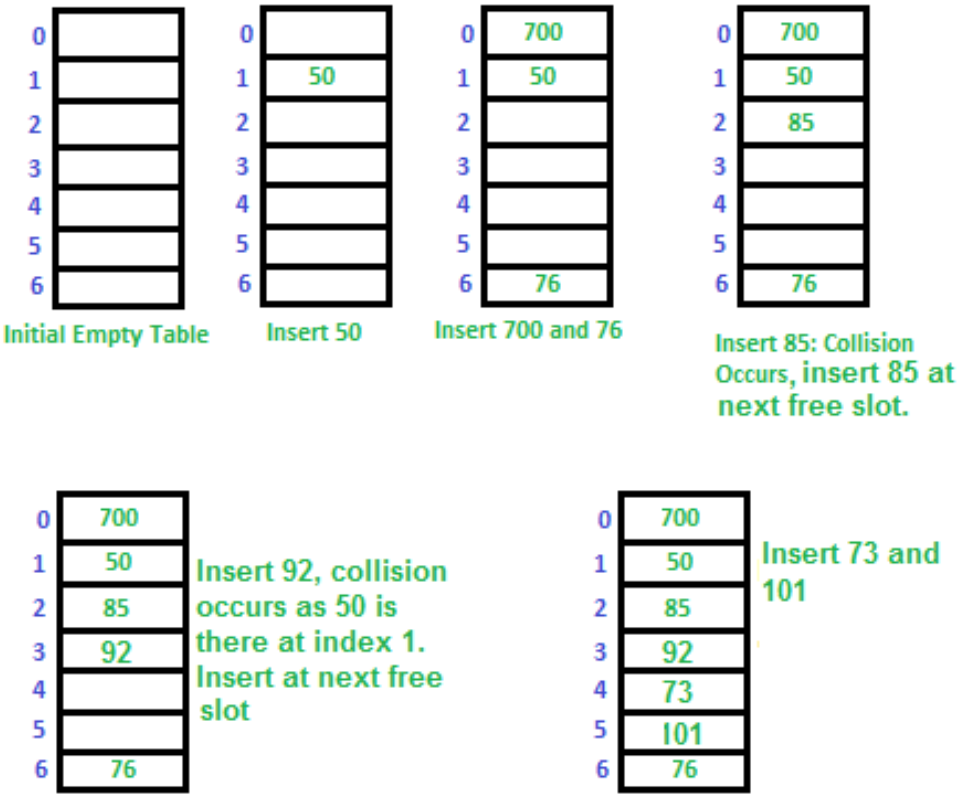
a) Linear Probing: In linear probing, we linearly probe for next slot.

For example, the typical gap between two probes is 1 as seen in the example below.

Let **hash(x)** be the slot index computed using a hash function and **S** be the table size

```
If slot hash(x) % S is full, then we try (hash(x) + 1) % S
If (hash(x) + 1) % S is also full, then we try (hash(x) + 2) % S
If (hash(x) + 2) % S is also full, then we try (hash(x) + 3) % S
................................................
................................................
```

Let us consider a simple hash function as “key mod 7” and
a sequence of keys as 50, 700, 76, 85, 92, 73, 101.



b) Quadratic Probing

We look for i^2'th slot in i'th iteration.

```
let hash(x) be the slot index computed using hash function.
If slot hash(x) % S is full, then we try (hash(x) + 1*2) % S
If (hash(x) + 1*2) % S is also full, then we try (hash(x) + 2*2) % S
If (hash(x) + 2*2) % S is also full, then we try (hash(x) + 3*2) % S
.................................................
```

c) Double Hashing

We use another hash function hash2(x) and look for i*hash2(x) slot in i'th rotation.

```
let hash(x) be the slot index computed using hash function.
If slot hash(x) % S is full, then we try (hash(x) + 1*hash2(x)) % S
If (hash(x) + 1*hash2(x)) % S is also full, then we try (hash(x) + 2*hash2(x)) %
S
If (hash(x) + 2*hash2(x)) % S is also full, then we try (hash(x) + 3*hash2(x)) %
S
.................................................
.................................................
```

Insert(k): Keep probing until an empty slot is found. Once an empty slot is found, insert k.
Search(k): Keep probing until slot's key doesn't become equal to k or an empty slot is reached.

Delete(k): ***Delete operation is interesting***. If we simply delete a key, then the search may fail. So slots of deleted keys are marked specially as "deleted".
The insert can insert an item in a deleted slot, but the search doesn't stop at a deleted slot.