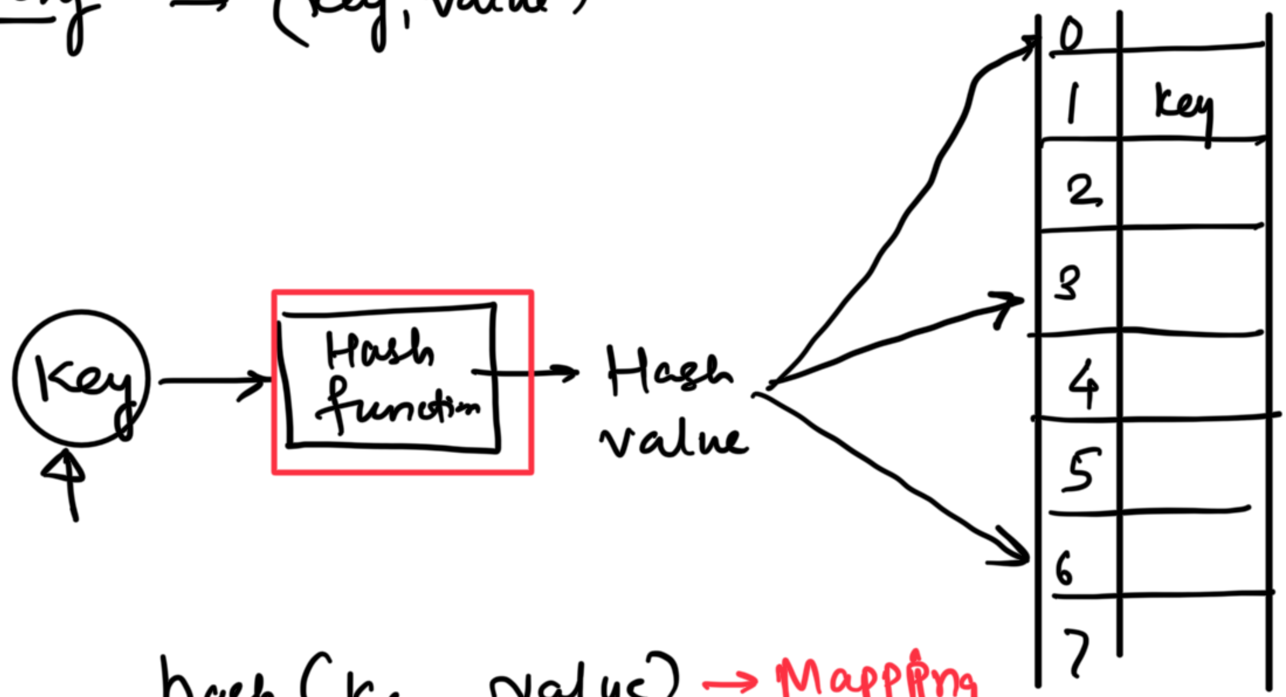
Hashing → (key, value)



hash (key, value) → Mapping

key → Value

Hash Table

key ↓ Number

value ↓ index of HashTable

Defn – Hashing is a technique used to map data (keys) to a unique index in a fixed-size table called as hash table

– primarily used to optimize search, insertion & deletion operation

– Insertion, Deletion, Search — O(1)

– A hashtable is a data structure that stores elements & allows insertions, lookups, and deletions to be performed in O(1) time

– In a hash table, a hash function is used to map keys into positions in a table. This is called as hashing

– Operations

Search – Compute f(k) & see if a pair exist

Insert - Compute f(k) & place it at that position

Delete - Compute f(k) & delete it at that position

Example: keys ⇒ 8, 13, 3, 6, 4, 10, 50

hash funct ⇒ key % size of hash table

key ↓ Input

size of hash table ↓ bucket size

Basic Hash Function

$\boxed{h(x) = key \% size}$

10 → 0-9

size = 10

0	1	2	3	4	5	6	7	8	9
10	50		13	3	4	6		8	

8 % 10 = 8 → index

13 % 10 = 3 }
3 % 10 = 3 } → index is same

↓ Collision

handle → sol^n

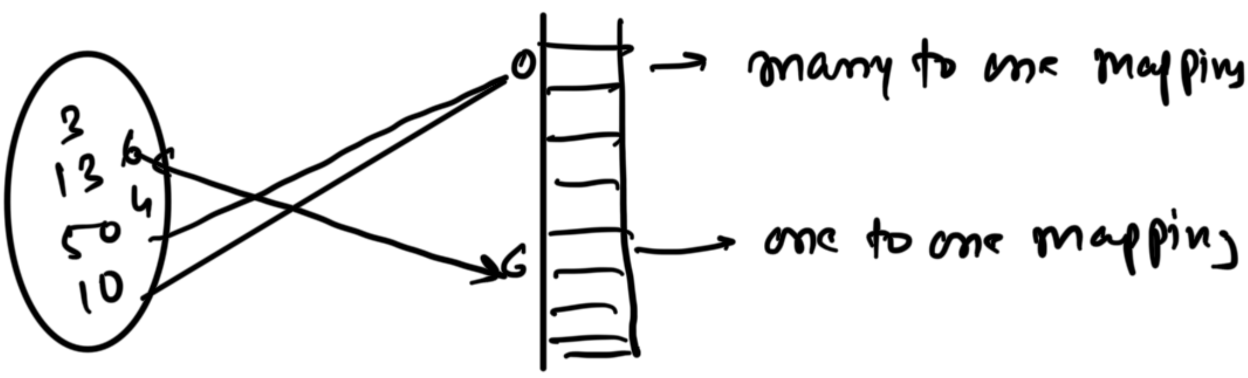
Linear probing

6 % 10 = 6

4 % 10 = 4

10 % 10 = 0 }
50 % 10 = 0 } Same index → Collision

Sol^n → Linear probing



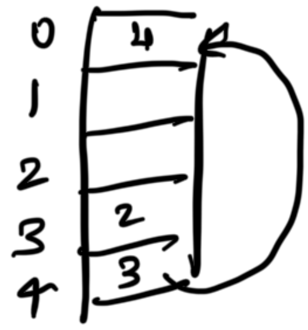
Common Hashing Techniques -

1. Direct Hashing -

- Each key is mapped directly to a specific index in the hash table

$h(x) = 42 \% 10 = 2 \rightarrow$ index 2 $\boxed{42}$

2. Division Method (Modulo Method)



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

Drawback - Collision

3. Multiplicative Method

$h(x) = \text{floor}(\text{table size} * (\text{key} * A \% 1))$

A = constant (0 or 1)

- A key is multiplied by a constant & the fractional part of the result is multiplied by the table size to get the index

eg: key 42

$h(x) = \text{floor}(10(42 * 0.610 \% 1))$

$= \boxed{} \rightarrow$ index for element 42

$h(x) = 4 \rightarrow$

$h(x)\ 4 \% 10 = 4$

$h(x) = 10(4 * 0.61 \%$

$= 401.36$

$= 401$

4. Folding Method

eg. 987654

987 654

$h(x) = \boxed{987 + 654 = 1641} \rightarrow$

- The key is divided into equal parts, and the parts are added to get the hash index.

$h(x)1 = 1641 \% \text{tablesize} = \underline{\text{index}}$

↑ unique

5. Mid Square Method

eg. 4567

$(4567)^2 = 2085\,1489$

$h(x) = 57 \% \text{tablesize}$

$= \underline{\text{index}}$

2085745

857%

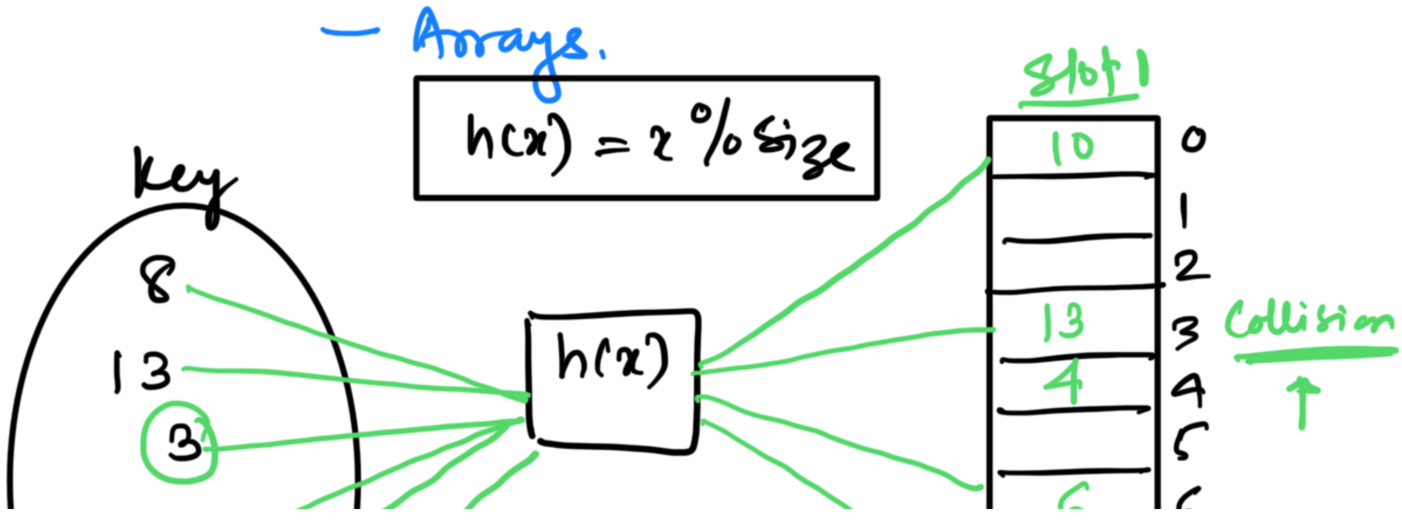
ta

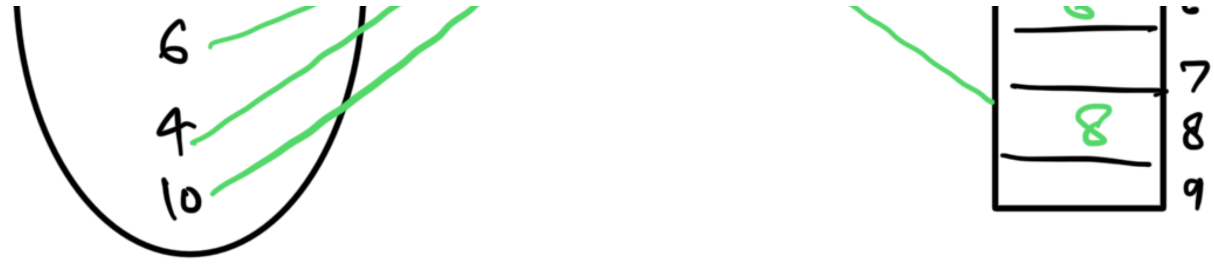
The key value is squared and the middle digits are extracted to form the index

Collision Handling Techniques

1. Separate chaining (Open hashing)
   - Linked list
2. Open Addressing (Closed hashing)
   a) Linear probing
   b) Quadratic probing
   c) Double hashing
   - Arrays.

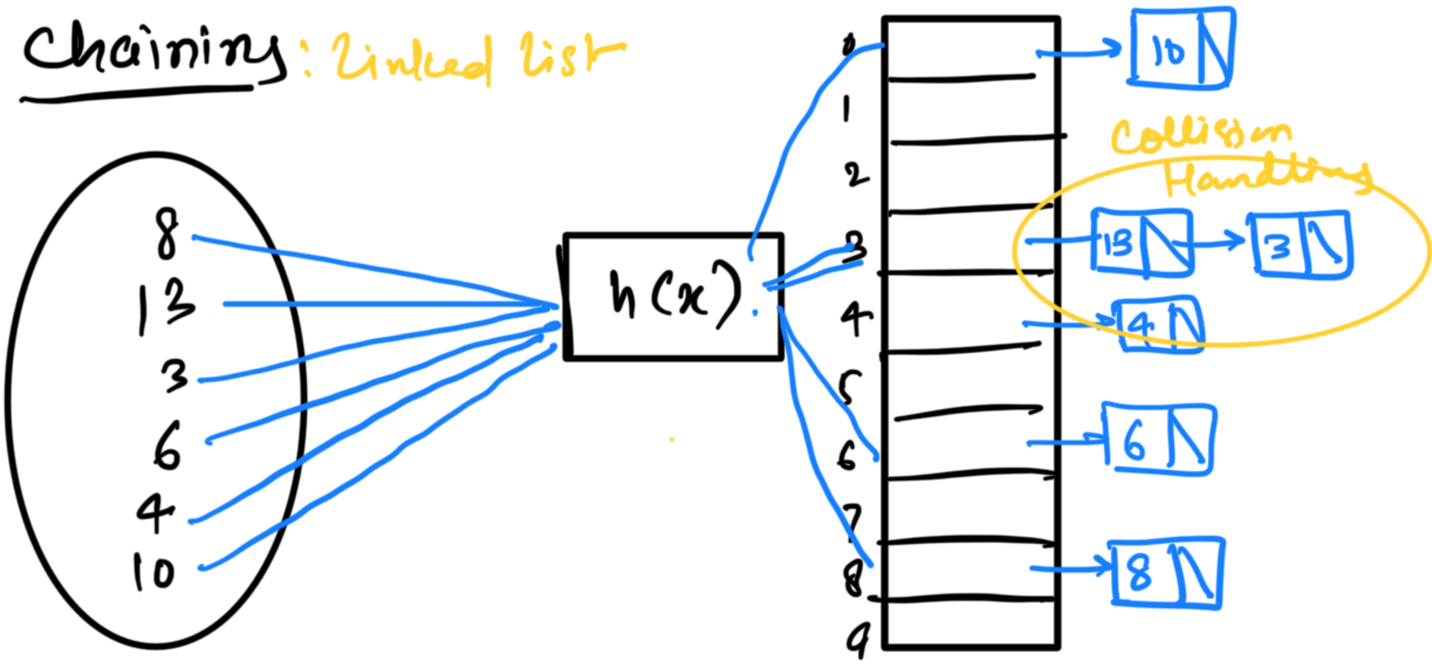
$\boxed{h(x) = x \% \text{size}}$





Standard Method with Array

Chaining: Linked list



Ex: 0, 1, 4, 9, 16, 25, 36, 49, 64, 81 → **Chaining**

hash(key) = key % 10

Open Adding –

– Collisions are resolved by finding another empty slot with the hash table

1. Linear Probing:

– Increment the index sequentially until an empty slot is found.

$$\boxed{h(key) = (h(key) + i)\ \%\ tablesize}$$

eg: 25, 35, size 10

25 % 10 = 5

35 % 10 = 5

(35 + 1) % 10 = 6

2. Quadratic Probing

2. Quadratic Probing

- The next index is found by incrementing the square of the attempted number

$$h(x) = (h(key) + i^2) \ \% \ table\_size.$$

key: 20, 30, 2, 13, 25, 24, 10, 9

$h(x) = x \ \% \ 11$

$20 \% 11 = 9$

$30 \% 11 = 8$

$2 \% 11 = 2$

$13 \% 11 = 2 \rightarrow 2 + 1^2 = 3$

$25 \% 11 = 3 \rightarrow 3 + 1^2 = 4$

$24 \% 11 = 2 \rightarrow 2 + 1^2 = 3$

$\qquad 2 + 2^2 = 6$

$10 \% 11 = 10$

$9 \% 11 = 9 \rightarrow 9 + 1^2 = 10 \times$

$\qquad 9 + 2^2 = 13 \% 11 = 2$

$\qquad 9 + 3^2 = 18 \% 11 = 7$

	0
	1
2	2
13	3
25	4
	5
24	6
9	7
30	8
20	9
10	10

3. Double Hashing

$$\boxed{h(x) = (h(key) + i * h(key)) \ \% \ tablesize}$$

- uses a second hash function to determine the step size after a collision

$h1 = key \ \% \ tablesize$

$h2 = h1 \ \% \ table size$

$(h1 + i * h2) \ \% \ table size$

Load Factor in Hashing -

Load factor (α) — measures that indicates how full a hash table is.

$$\boxed{\text{Load factor } (\alpha) = \frac{n}{m}}$$

n = Number of elements in the hash table

m = Total number of available slots (buckets)

$\alpha \approx 1$

$\alpha << 1 \longrightarrow$ low Hashing

$\alpha >> 1 \longrightarrow$ almost full, High Hashing

Open Hashing	Closed Hashing
Linked list	Arrays
Sperate chaining	Linear probing quadratic, double
dynamic size	fixed size
Access Time $O(n)$	Access Time $O(n)$

Time Complexity

	Average case	Worst case
Search	$O(1)$	$O(n)$

Insert	$O(1)$	$O(n)$
Delete	$O(1)$	$O(n)$
<u>Space Complexity</u>	$O(n)$	$O(n)$

→ Separate chaining → $O(n+m)$

n = keys, m = buckets

→ Linear probing → $O(m)$

m = table size