Hashing

- a technique used for searching purposes. → searching TC : O(1)

① Linear Search

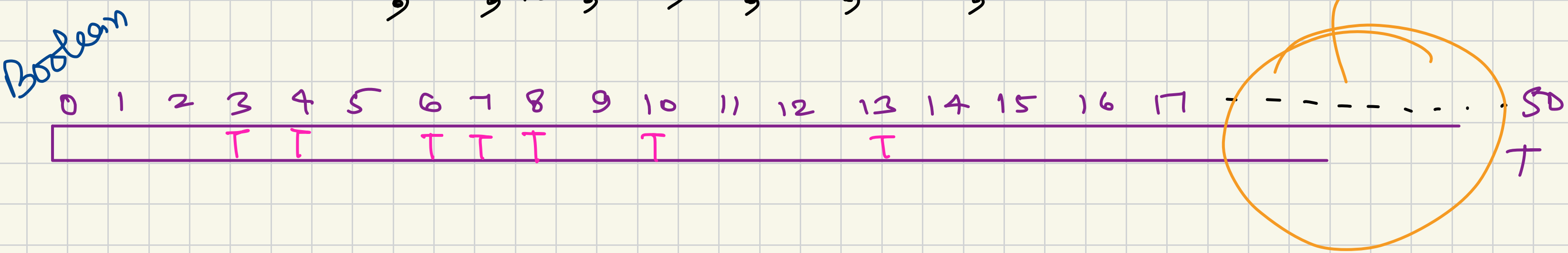
└→ TC' O(N)

② Binary Search

└→ TC : O($\log_2 N$)

i/p by users

8, 3, 13, 6, 7, 4, 10, 50

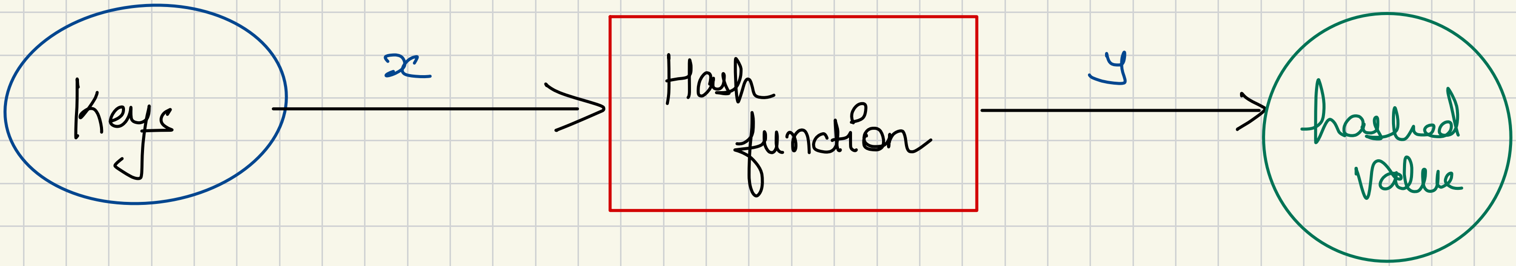


memory wasted

```
search(int tar)
{
    if (arr[tar] == true)
        return true;
    else
        return false;
}
```

→ TC: O(1)
SC: O(1)

high memory usage!
hashing was introduced



hash f^n

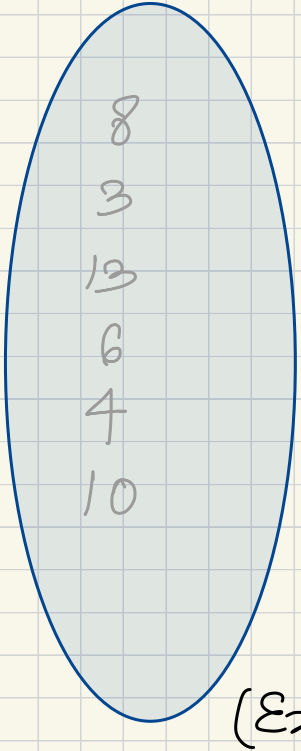
step 1 $\quad g(\underset{\text{key}}{x}) = \underset{\text{integer value}}{K}$

step 2 $\quad h(k) = y$

"apple"

↳ 09986

Key Space



hash funⁿ

$f(K) = K$

$f(8) = 8$

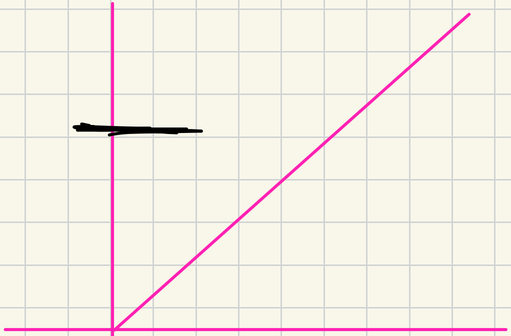
$f(3) = 3$

$f(13) = 13$

$f(6) = 6$

$f(4) = 4$

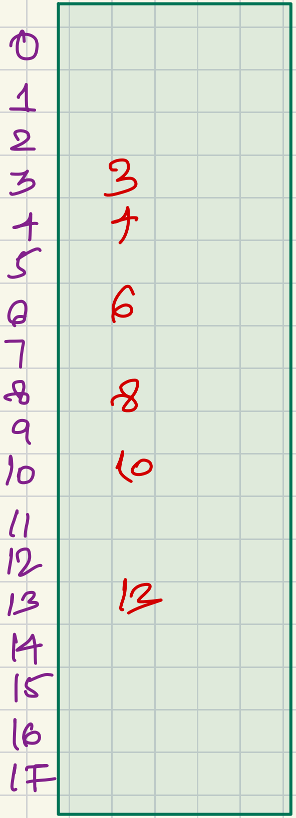
$f(10) = 10$



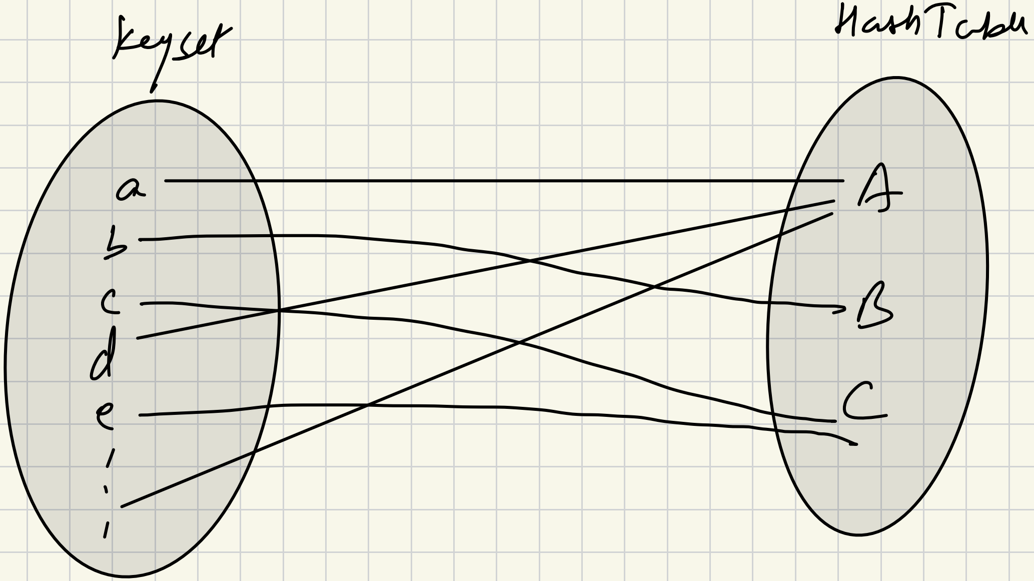
One - One relation

(Extra Memory is used)

hash Table

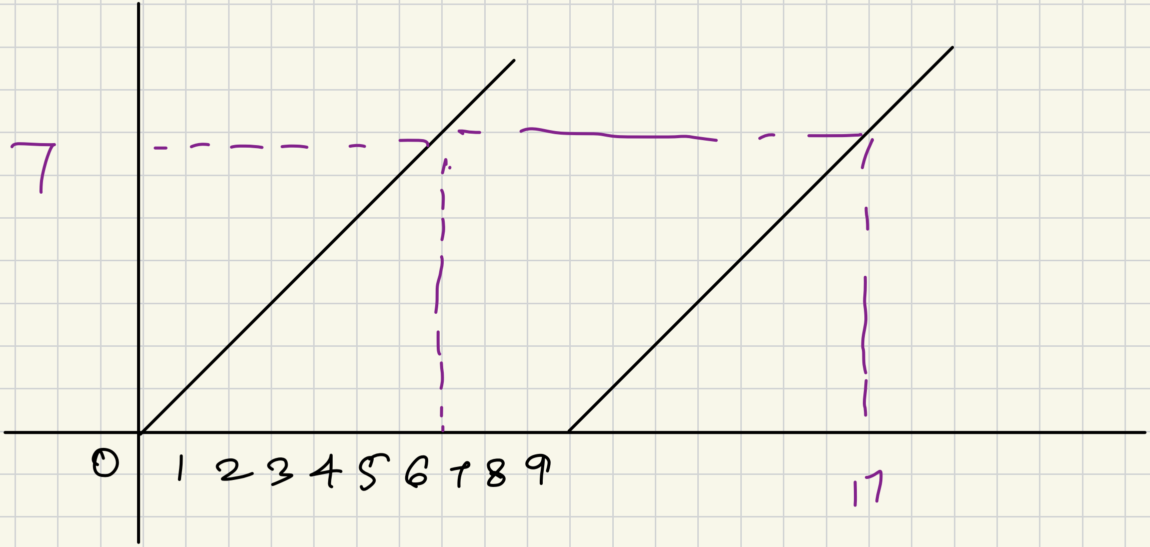


Many to One Relation

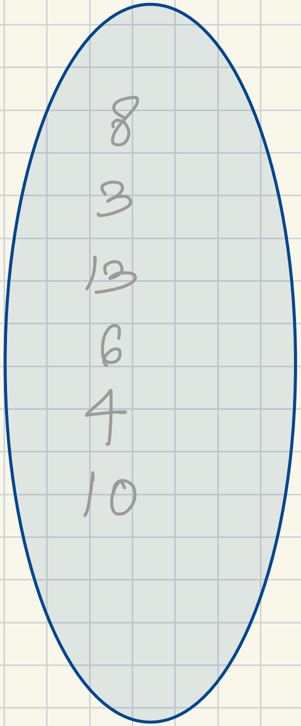


hash funⁿ

$$f(K) = K \% 10$$



Key Space



hash fun

$f(k) = k \% 10$

$f(8) = 8 \% 10 = 8$

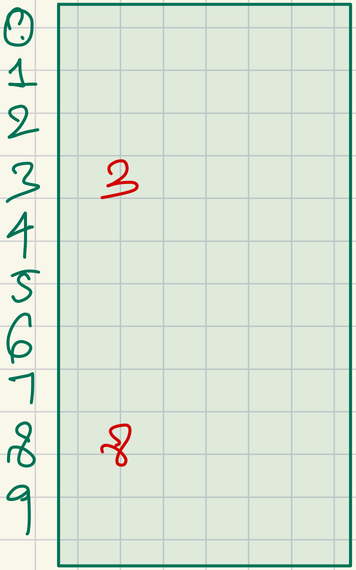
$f(3) = 3 \% 10 = 3$

$f(13) = 13 \% 10 = 3$

DP

Collision

hash Table



Methods to remove collision

Open hashing

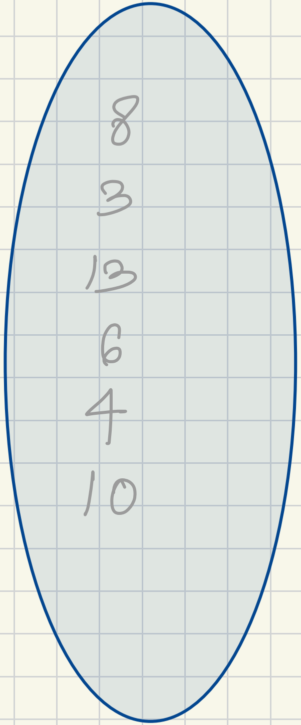
1. Chaining

Closed hashing

1. Linear Probing
2. Quadratic Probing

Chaining.

Key Space



hash fun

$f(K) = K \% 10$

$f(8) = 8 \% 10 = 8$

$f(3) = 3 \% 10 = 3$

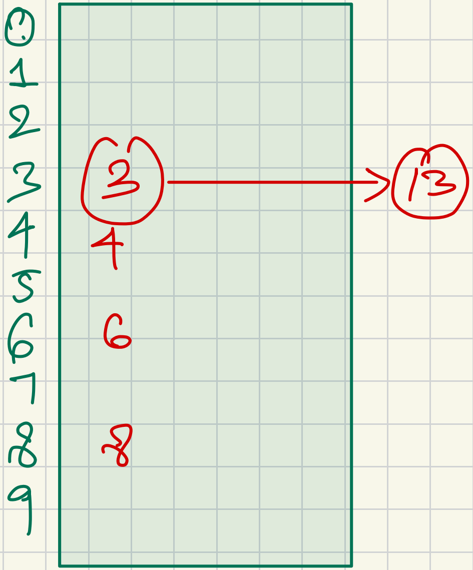
$f(13) = 13 \% 10 = 3$

$f(6) = 6 \% 10 = 6$

$f(4) = 4 \% 10 = 4$

search(13)
{
$13 \% 10 = 3$

hash Table



Searching.

TC: $O(1)$

$$\boxed{f(k) = \cancel{k} \% 10}$$

$\underline{\underline{f(k) = k \% \text{size}}}$

size = range × LF

0 - 10000

2

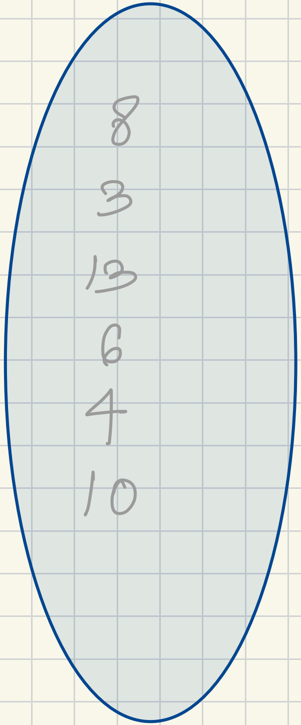
75%

7500

f(k) , k % 7,250

Quadratic Probing.

Key Space



hash fu^{n}

$$f'(k) = \{f(k) + g(i)\} \% 10$$

$f(k) = k \% 10$ $g(i) = 0, 1, 2 . --$

$f'(8) = (8+0) \% 10 = 8$

$f'(3) = (3+0) \% 10 = 3$

$f'(13) = (3+0) \% 10 = 3$

$f'(13) = (3+1) \% 10 = 4$

$f'(6) = (6+0) \% 10 = 6$

$f'(4) = (4+0) \% 10 = 4$

$f'(4) = (4+1) \% 10 = 5$

$f'(10) = (0+0) \% 10 = 0$

hash Table

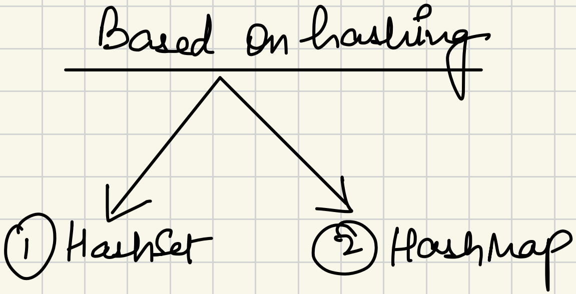
0	10
1	
2	
3	3
4	13
5	4
6	6
7	
8	8
9	

Quadratic Probing

$$f'(k) = \{ f(k) + g(i) \} \% size$$

$$f(k) = k \% size$$

$$g(i) = i^2 \quad 0, 1, 4, 9, 16, 25, 36 \ldots$$

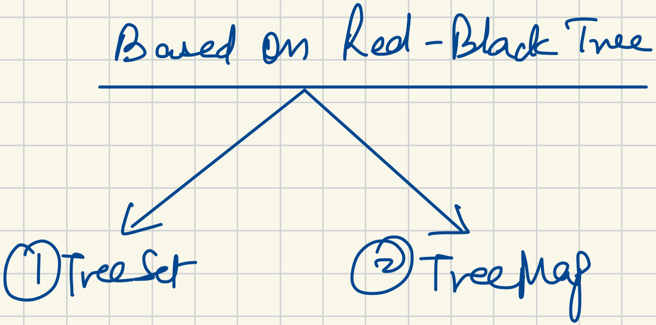

Based on hashing

① HashSet

② HashMap

searching TC: O(1)

insertion TC: O(1)


Based on Red-Black Tree

① TreeSet

② TreeMap

searching TC : $O(\log_2 N)$

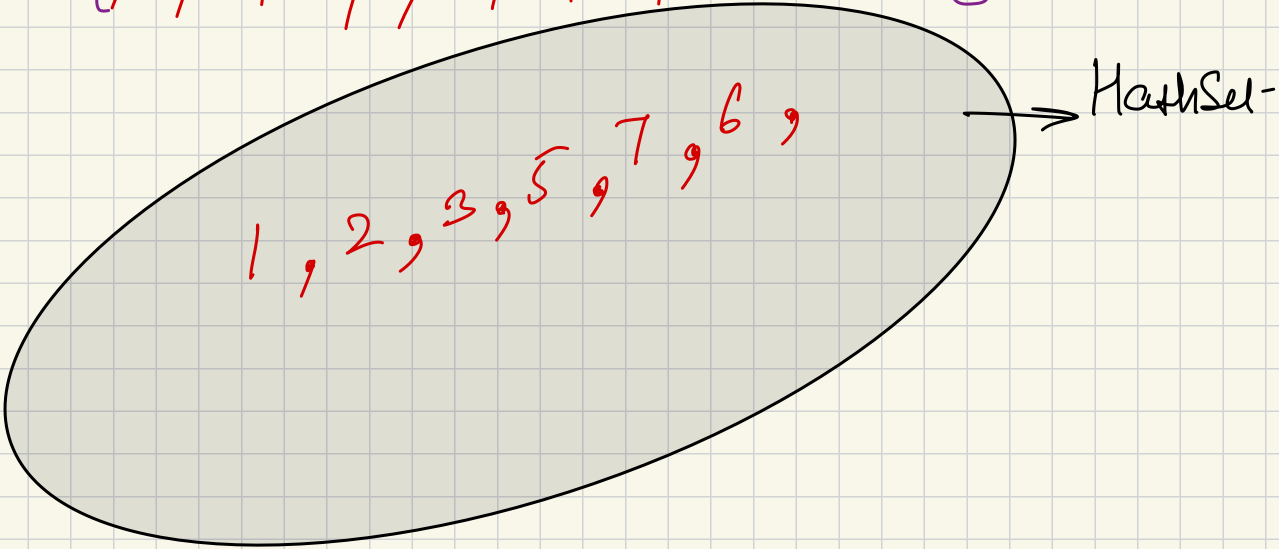
insertion TC : $O(\log_2 N)$

HashSet.

→ set of unique entities.

✓ o Holds unique entities
✓ o searching of entities O(1)

KeySet = {~~1~~, ~~2~~, ~~3~~, ~~2~~, ~~5~~, ~~7~~, ~~6~~, ~~5~~, ~~2~~, ~~3~~, ~~1~~}



HashMap

ds to store (key, value) pairs

key ① unique entity
② Hashing is applied over keys

implement a shopping cart

key	value
item	qty
lays	3
eggs	6
oranges	3