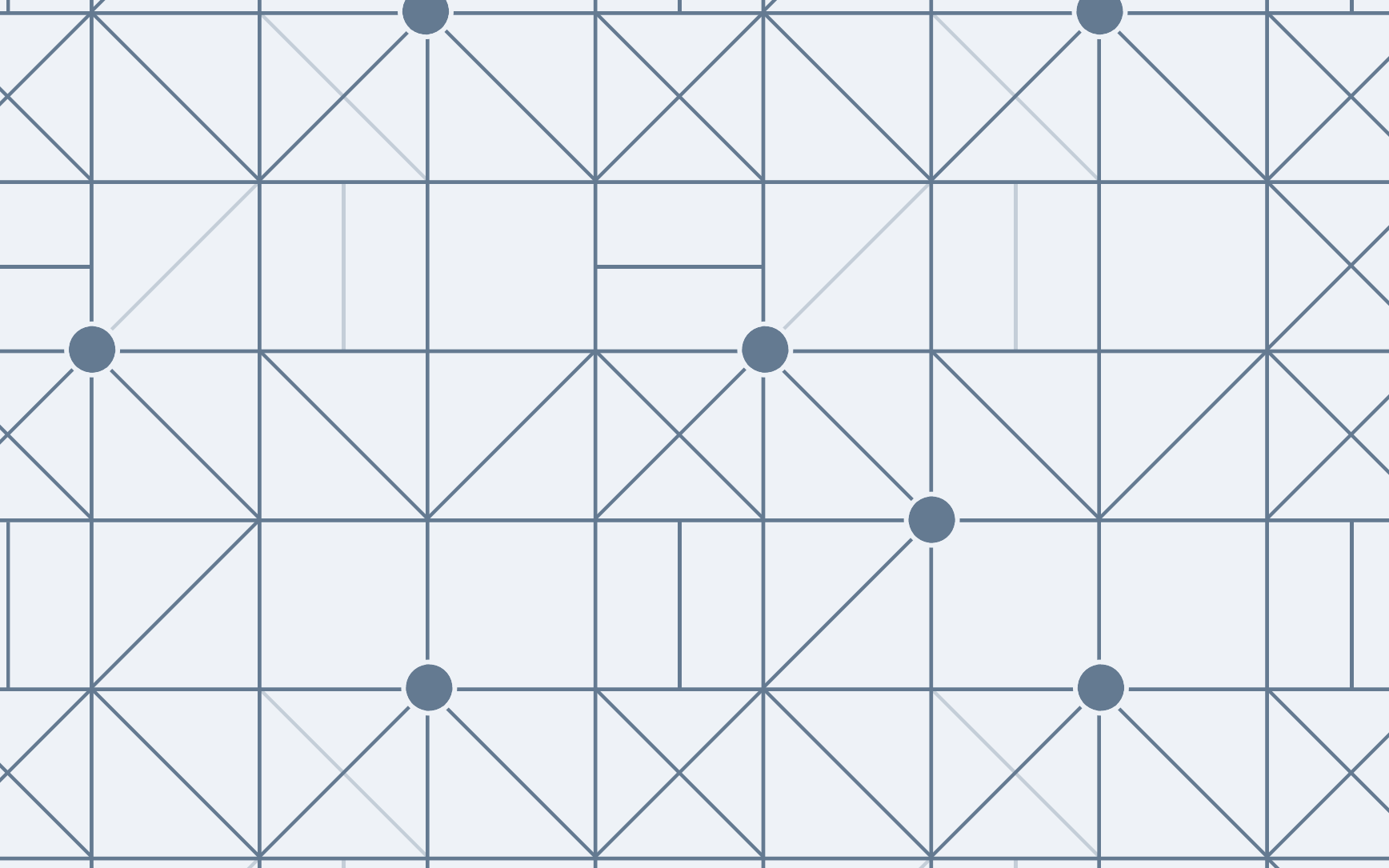


Minimum Limit of Balls in a Bag.

int[] A = { 3 (0), 2 (1), 2 (2), 4 (3), 1 (4), 4 (5) } maxOpt = 4

(1, 2)

opt = 1 { 1, 2, 2, 2, 4, 1, 4 } penalty = 4

(1, 3) (2, 2)

opt = 2 { 1, 2, 2, 2, 2, 2, 1, 4 } penalty = 4

(1, 3) (2, 2)

opt = 3 { 1, 2, 2, 2, 2, 2, 1, 1, 3 } penalty = 3

(1, 2)

opt = 4

{ 1, 2, 2, 2, 2, 2, 1, 1, 1, 2 }

penalty = 2

int[] A = { 3 (0), 2 (1), 2 (2), 4 (3), 1 (4), 4 (5) }

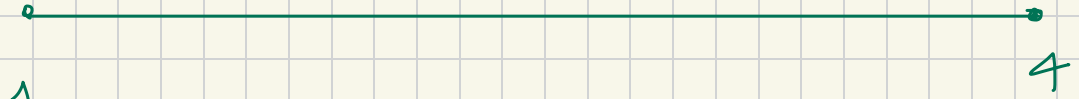
Case 1 maxOpt = ∞

{1, 1, 1 . . . . . }

penalty = 1

Case 2 maxOpt = 0

penalty = 4


1
4

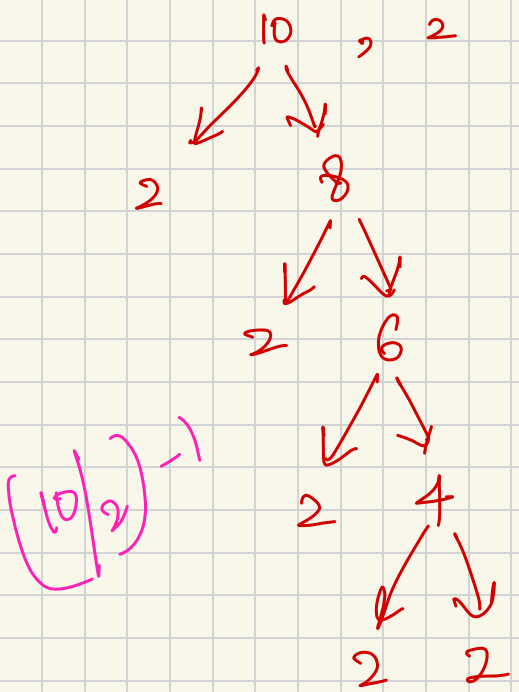
int[] A = { 3, 2, 2, 1, 1, 1 } (indices 0 1 2 3 4 5) maxLen = 2

(1,2)

numOpt - 0 ~~1~~ ~~2~~ 3

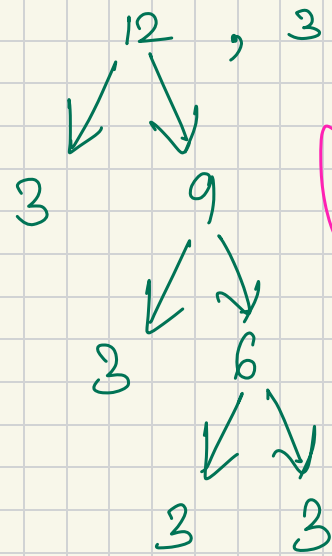
num = x pen = y

min NumOfOpt =



$(10|2)-1$

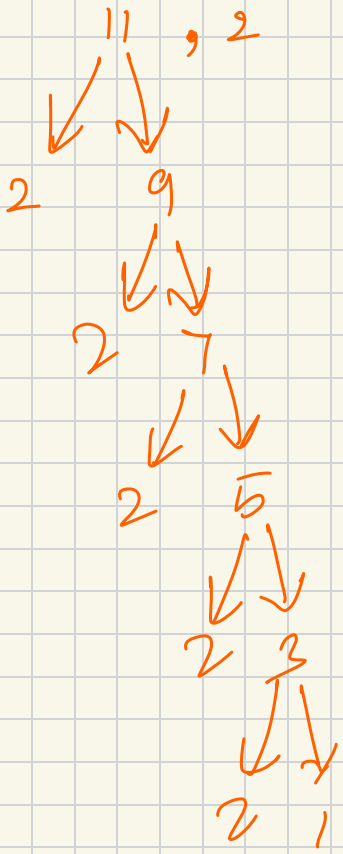
4 opt



$(12|3)-1$

3 opt

$maxOpt = (x|y) - 1$ ✓

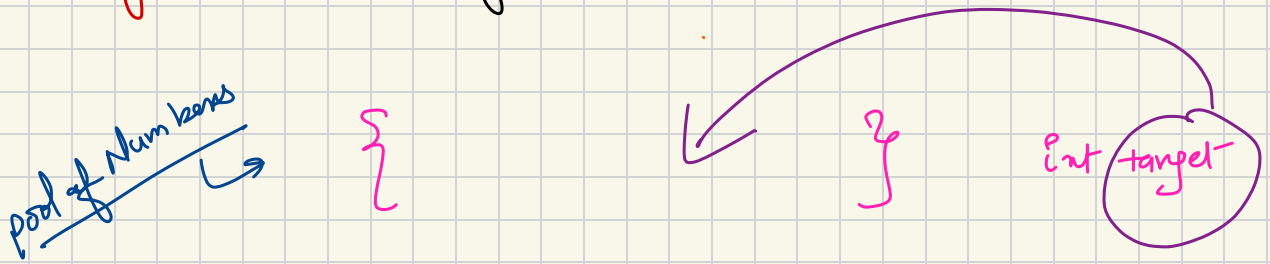


x is divisible by y

$(x/y - 1)$ }

(x/y) }

Hashing : (searching technique)



(1) Linear search TC : O(N) SC : O(1)

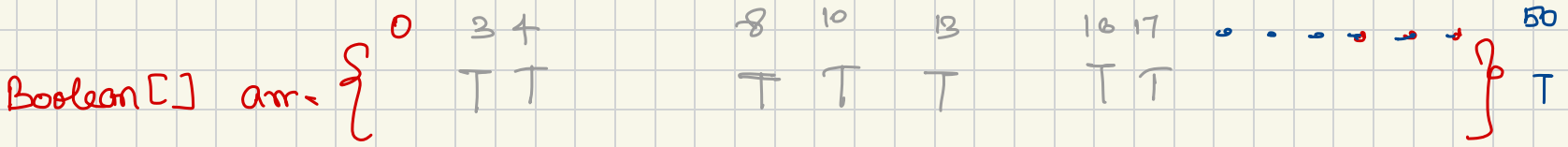
(2) Binary search TC : O($\log_2 N$) SC : O(1)

} stores i/p array

(3) Hashing TC : O(1), SC : O(1)

} store i/p hashTable

i/p 8, 3, 13, 16, 17, 4, 10 50



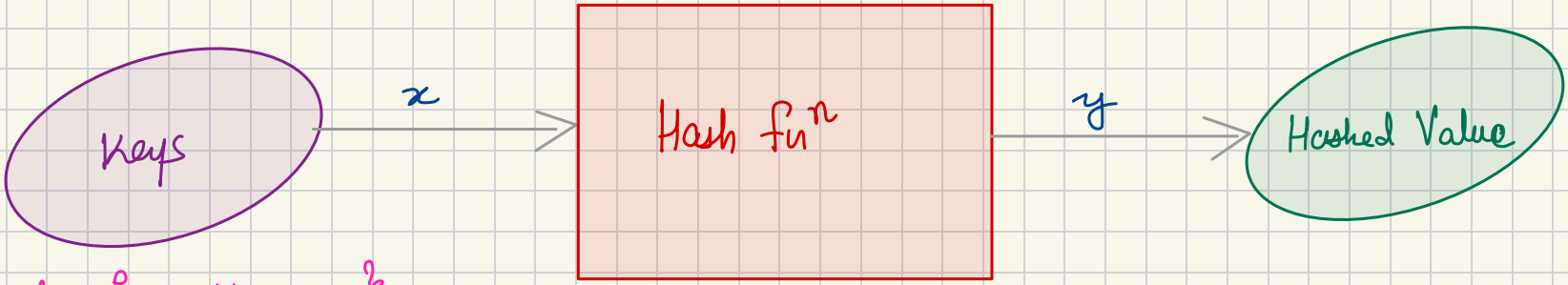

```
search(int target)
{
    if(arr[target] == true)
        return "found";
    else
        return "not found";
}
```

8

TC: O(1)
SC: O(1)

- high Memory consumption
  - hashing was introduced

Basic Mechanism of Hashing.



{values inserted by user}

Hash fun^n. {2-step fun^n}

Step 1 $g(x) = K$

- x → Key
- K → integer

Step 2 $f(K) = y$

- K → integer
- y → Hashed

Key Space

8 ✓
3 ✓
13 ✓
6 ✓
4
10

Hash fun

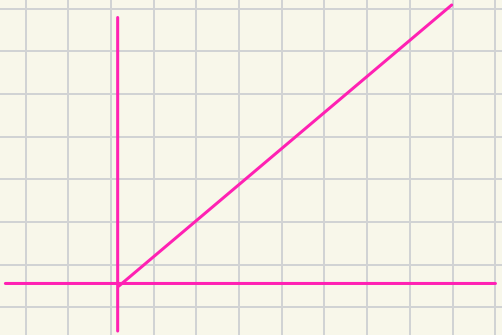
$f(K) = K$ (one to one fun)

$f(8) = 8$

$f(3) = 3$

$f(13) = 13$

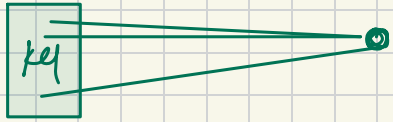
$f(6) = 6$



Hash Table

0	
1	
2	
3	3
4	
5	
6	6
7	
8	8
9	
10	
11	
12	
⋮	13

many to one relation

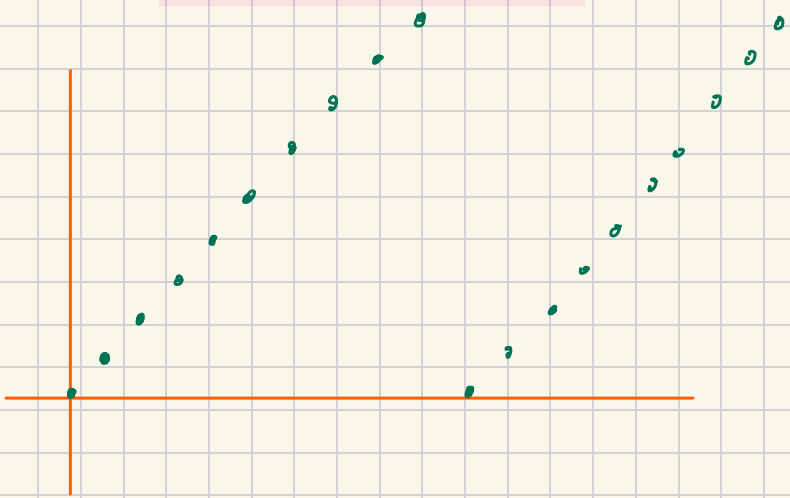


hashing fun^n :

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

Range : [0, 9]

↳ fixed hashed Table Size



Key = 12

hashedValue = 2

Key = 2

hashed Value = 2

Key Space

8 ✓
3 ✓
13
6
4
10

Hash fun

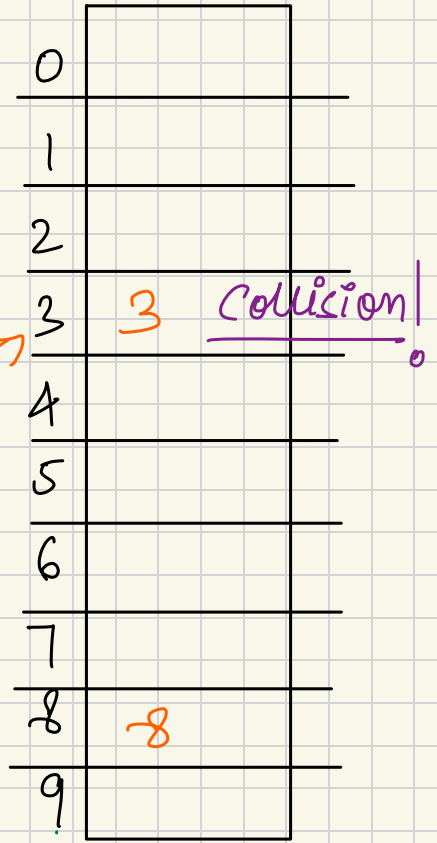
$f(K) = K \% 10$

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

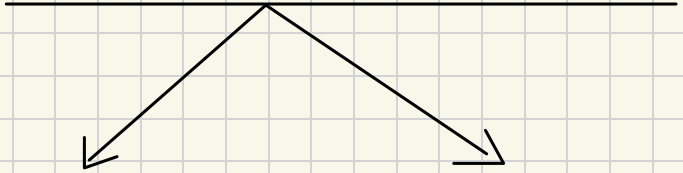
$f(3) = 3 \% 10 = 3$

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

Hash Table



Methods to remove collision



Open Hashing

- chaining

Closed Hashing

- Linear Probing
- Quadratic Probing

chaining

Key Space

8
3
13
6
4
10
103

Hash fun

$f(k) = k \% 10$

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

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

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

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

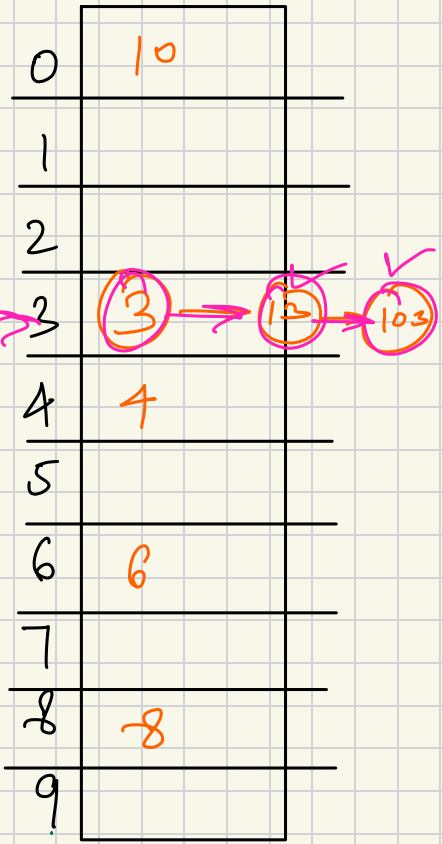
$f(4) = 4 \% 10 = 4$

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

$f(103) = 103 \% 10 = 3$

search(103) ⟹ hashed value

Hash Table



load factor

↳ 75% of Range

Hash fun = key % Load factor

Linear Probing

Key Space

8 ✓
3 ✓
13
6
4
10

Hash fun

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

$f(k) = k \% 10$

$g(i) = 0, 1, 2 \ldots$

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

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

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

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

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

$f'(13) = \{f(13) + g(1)\} \% 10 = 4$

Hash Table

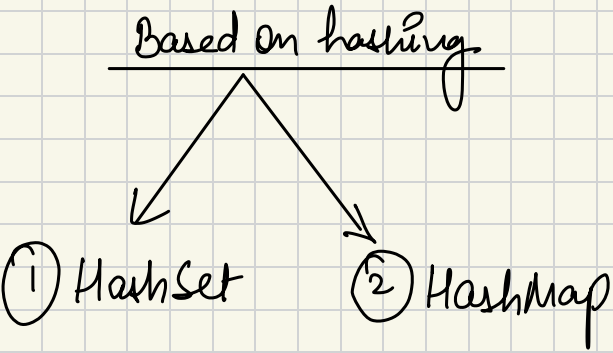
0	
1	
2	
3	3
4	13
5	
6	
7	
8	8
9	

Quadratic Probing

$$f'(k) = \{ f(k) + g(i) \} \% \text{ L.F.}$$

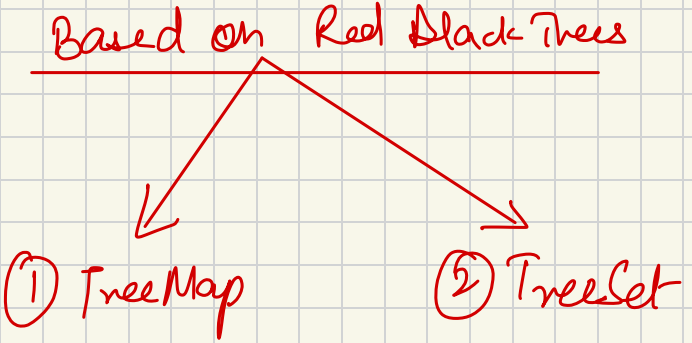
$f(k) = k \% \text{ L.F.}$

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



TC: O(1) Searching

TC: O(1) insertion

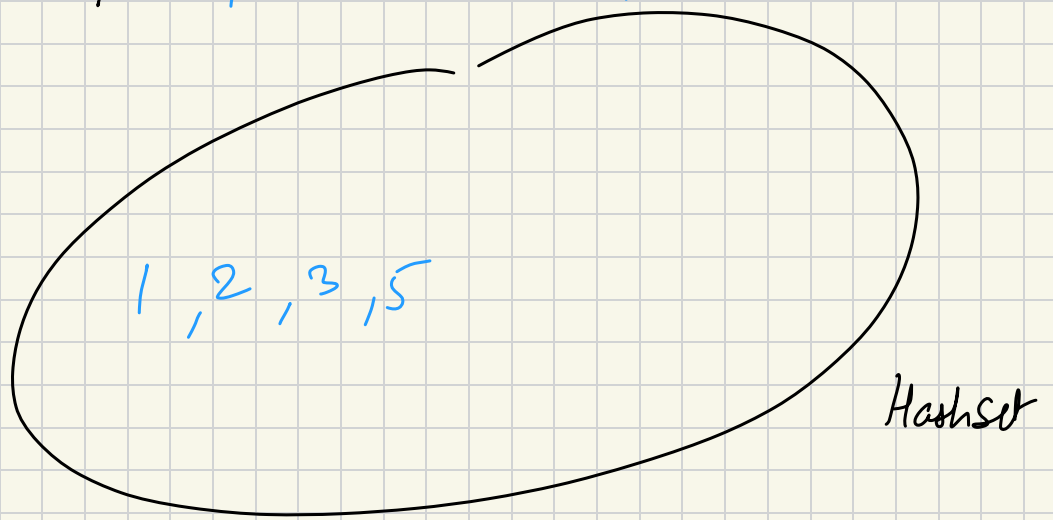


TC: $O(\log_2 N)$ Searching, Insertion.

HashSet

→ set of unique entities

keyset = {1, 2, 3, 3, 5, 2, 1, 3, 5}



Hash Map → stores (key, value) pair

HashSet → unique entities

Implete a Shopping Cart

item → unique entity

qty

Hashing

- lays → 2
- eggs → 12
- oranges → 2
- yogurt → 4

```
HashSet<G> name_hs = new HashSet<>();

HashMap<G> name_hm = new HashMap<>();
```