Check if given elem exists

A = { 2 4 11 15 6 8 14 9 }

Q = { 4 10 17 14 }

Sol^n ⇒ Iterate for each query element
TC: $O(NQ)$

Sol^n 2 ⇒ Direct Access Table (DAT)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	~~0~~	0	~~0~~	0	~~0~~	0	~~0~~	~~0~~	0	~~0~~	0	0	~~0~~	~~0~~
		1		1		1		1	1		1			1	1

Advantage → Insert, Search ⇒ $O(1)$

Disadvantage → 1) Space wastage
2) Cannot create big arrays

How to solve this issue

A = {	21	42	37	45	99	30 }
	↓	↓	↓	↓	↓	↓
	1	2	7	5	9	0

Lets say I can only creat array of size = 10

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

Can I take a[i] % 10 ? yes

Issue ⇒ Collision 21 & 41

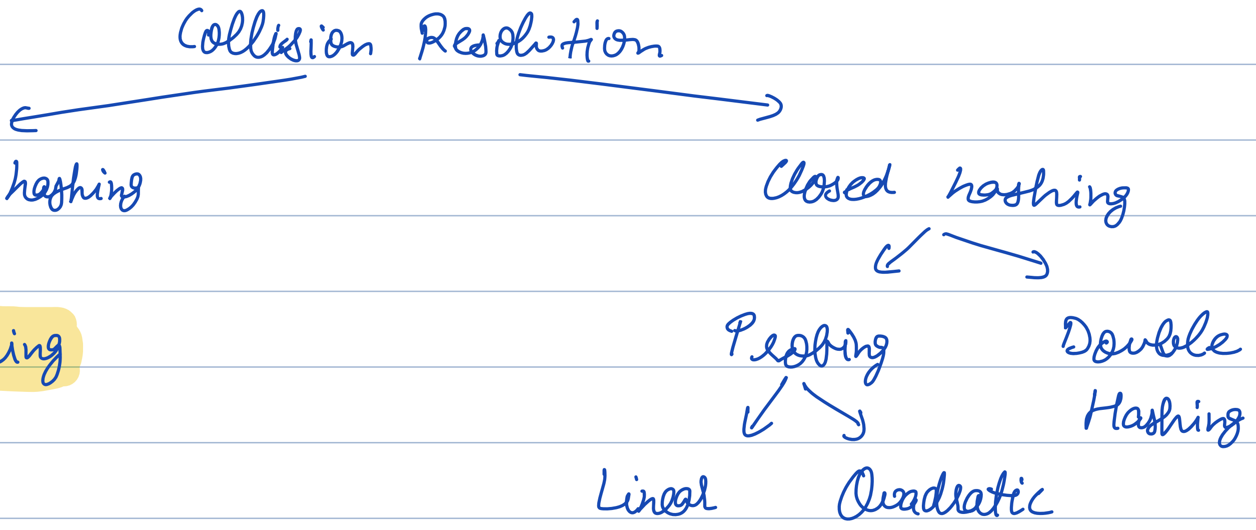
↓ ↓

1 1

value A → hash func → same value

value B → hash func ↗

Collision Resolution



- Chaining

$A = \{ 21 \quad 42 \quad 37 \quad 45 \quad 77 \quad 99 \quad 31 \}$

0	1	2	3	4	5	6	7	8	9
	21	42			45		37		99
	↓						↓		
	31						77		

Each hash value is a linked list

- TC of insert

  If at tail ⇒ $O(n)$
  If at head ⇒ $O(1)$

- TC of search /delete

  Average ⇒ less than λ (Lambda)
  worst case ⇒ $O(n)$

- What is λ (Lambda / Load factor)

Ratio of num of elements / size of DAT

0 16
1
2
3 11 → 27 → 19
4
5
6 22 → 6
7

num of elems = 6 DAT size = 8

$\lambda = 6/8 = 0.75$

Lets say predefined threshold = 0.7

We need to do rehashing using diff hash function
Create DAT with double size of original DAT

Now $\lambda = \frac{6}{16} = 0.375 <$ threshold
(within threshold)

0 6
1 11
2
3 27
⋮
16
⋮
15 22

Code

```
class HashMapNode {
    int key
    int value
}
Arraylist < HashMapNode> buckets [ 4 ]    // 4 is size
                                          of DAT
```

0

1   30 → 200 → 20

2

3

```
void insert ( key, value) {
    idx = hash ( key )
    if   value   in   buckets [idx]
            // already present
    else {
        buckets (idx) . add ( new HashMapNode (key , value))
        size ++                     // total number of elements
        lamda =     size / buckets. length
        if ( lamda > threshod )
              rehash ()
    }
}
```

```
int hash ( int key ) {
     int ans = key % buckets . size ()
     return ans
}
```

```
void rehash () {
    int original_size = buckets.size ()
    Arraylist < HashMapNode> old_buckets = buckets
    int new_size = 2 * original_size
    buckets.resize (new_size)
    for (i=0; i< old_buckets.size() ; i++) {
        for ( HashMapNode hmnode : old_buckets [i] ){
            insert (node.key, node.val)
        }
    }
}

bool search ( int key) {
    int idx = hash (key)
    bool ans = false
    for ( HashMapNode hmNode : buckets [idx] ) {
        if ( hmNode.key == key )
            ans = true
    }
    return ans
}
```

```
void delete (int key) {
    int idx = hash (key)

    for ( HashMapNode hmNode : buckets [idx] ) {
        if ( hmNode.key == key ) {
                    // remove this entry from
                         Arraylist
        }
    }
    size --
}
```

{done}