→ To store all 10 digits phone no...

int ar[10^{10}]

$4 \times 10^{10} = 4 \times 10 \times 10^{9}$ = 40GB (space is high)

∴ So ~~haship~~ hashing is used... to reduce the no. of indices...

* Hashing:

$$\boxed{\underbrace{hash}_{\text{hash function}}(\underbrace{x}_{\text{hash key}}) = \underbrace{x \% M}_{\text{hash value}}}$$ → size of table

ex: M = 10

data: 53, 7, 26, 11, 83, 76

hash(x)

0	1	2	3	4	5	6	7	8	9
	11		53			26	7		
			83						

53 % 10 = 3
7 % 10 = 7
26 % 10 = 6
11 % 10 = 1
83 % 10 = 3

∴ Collision occurred at 53, 83

→ Collision Techniques:

1. Separate chaining
2. Open addressing
   - linear probing
   - quadratic probing
   - Double hashing..

→ Separate chaining:

M = 10 $\boxed{hash(x) = x \% M}$

data = 15 23 87 33 25 13 6 23.

Hash(x) = [ ]

0 1 2 3 4 5 6 7 8 9

block ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓

3: 23,2 → 33,1 → 13,1

5: 15,1 → 25,1

6: 6,1

7: 87,1

(ele, count)

* Every index of the hash table acts as a head of the separate list.

(L - length of list)

insert	search	delete
L	L	L

add at starting 1

L ↓ if we maintain count

O(L) → Balanced binary search tree - O(1)

→ linear probing: hash(x) = (x + i) % M

M = 10

data = 15 27 33 41 55 63 72 81

hash = [41 72 33 63 15 55 27 81]

0 1 2 3 4 5 6 7 8 9

(15+0) % 10 = 5

(27+0) % 10 = 7

(33+0) % 10 = 3

(41+0) % 10 = 1

(55+0) % 10 = 5 ✗

(55+1) % 10 = 6

(63+0) % 10 = 3 ✗

(63+1) % 10 = 4

...

search(81) → T

delete(63)

search(81) → F

} ✗ return false when space is reached

state[] = [0 1 1 1 -1 1 1 1 1 0]

N 0 1 2 3 4 5 6 7 8 9

search(81) → T

0 - empty

1 - filled

-1 - deleted

$\boxed{P \Rightarrow \text{probing}}$

Insert: $O(p)$ | Search: $O(p)$ | delete: $O(p)$

→ **Quadratic probing:**

$$hash(x) = (x + i^2) \% M$$

QP:	0	1	2	3	4	5	1	4	→ 20
ar_N:	703	203	903	1203	103	503	704	804	
LP:	0	1	2	3	4	5	6	7	→ 28

insert(x) = $O(p)$; search(x) = $O(p)$;
delete(x) = $O(p)$

804		1203	703	203	704		903	503	103
0	1	2	3	4	5	6	7	8	9

→ L probing forms clusters
quadratic probing doesn't form clusters

→ Searching depends on hash function.

→ **Double hashing:**

$$hash(x) = [h1(x) + h2(x)] \% M \quad (M = 10)$$

hash(x):
h_1 ✓ $x \% M$
$(x + i^n) \% M$
h_2 ✓ sum of digits$(x) \% M$
$(x ^ i) \% M$
$(x !) \% M$

data: 17 25 86 ....

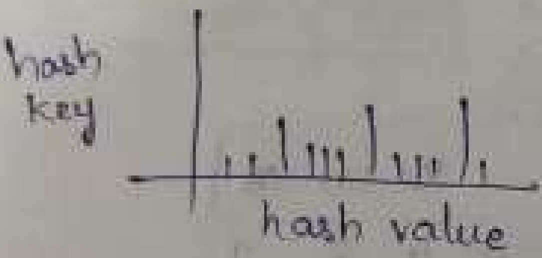
$h1(7) = 7$ $h1(25) = 5$ $h1(86) = 6$
$h2(7) = 8$ $h2(25) = 7$ $h2(86) = 14$
$15 \% 10 = \underline{5}$ $12 \% 10 = \underline{2}$ $20 \% 10 = \underline{0}$
...

Collision resolution technique

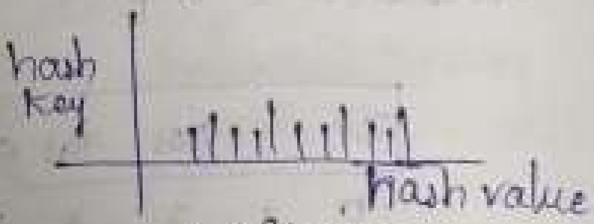
Separate chaining $O(1)$ | open addressing $O(p)$

→ ways to optimize or reduce L & P

1. use a good hash function
   - ~~Body~~ Easy to compute
   - Distribute hash value uniformly over hash keys



non-uniform



uniform

2. use decent hash tables.

29/7/24..

→ N - elements
M - size of hash table

N = 100 M = 20 ×
N = 100 M = 100 $O(P)$
N = 100 M = 200 $O(P)$ will reduce to 50%
N = 100 M = 1000 $O(P)$ will reduce to 90%

$O(P) \sim O(1)$

∴ Ideal hashtable size = 10N
↳ open addressing

*. Load factor:

Vector
Arraylist } - a dynamic datastructure is maintained
List

- Dynamic array doesn't take size while declaration.
- It resizes based on load factor.

example:

load factor (α) = 0.30

[- - - - - - - -]

70% filled
↓
it goes to resize the double of original size.

*. SubArrays: Continuous, in-order

① ex: 3 5 7 8 2 -1

∴ $\frac{n(n+1)}{2}$

3, 35, 357, 578, 82-1, 7,8,2

*. Subsequence: in-order, non-continuous

② ex: 3, 378, 32-1, 58-1

∴ $2^n - 1$

*. Subsets: non-continuous, not-in order

③ ∴ 2^n

1. length(1) = (N)
length(2) = (N-1) } add
⋮
length(5) = 1

Sum of N numbers $\boxed{= \frac{n(n+1)}{2}}$