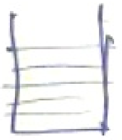
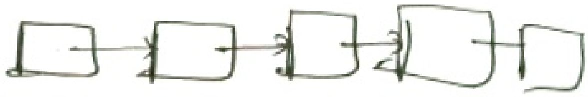
Contacts

x. Phone numbers

	Insert (x),	Search (x)	delete (x)
unsorted array	1	N	N
Sorted array	Searching $\log_2 N + N$	$\log N$	$N + N \log N$
FIFO Stack/Queue lifo	1	N,	N
unsorted linked list	1	N	N
Sorted SLL	N	N	N
		H	H





BST => $\log_2 N \rightarrow H$
$\searrow N$ (Skewed)

Skewed

BBst log^N_2 Log^N_2 log^N_2

Count array   C[300] ⇒ T,   If (C[x]) Thue else false   , C[x]=c

boolean ct array   1   1   1

↳ 1

Direct Access table:

Mobile numbe → 2 digits ⇒ C[90]

⇒ 10 digits ⇒

1000, 000000

999 9999999

R ⇒ 6000, 00000]   C[b-a+1]

1 = 999 999, 9999

#

Hashing

$h(x) = x \% 5$

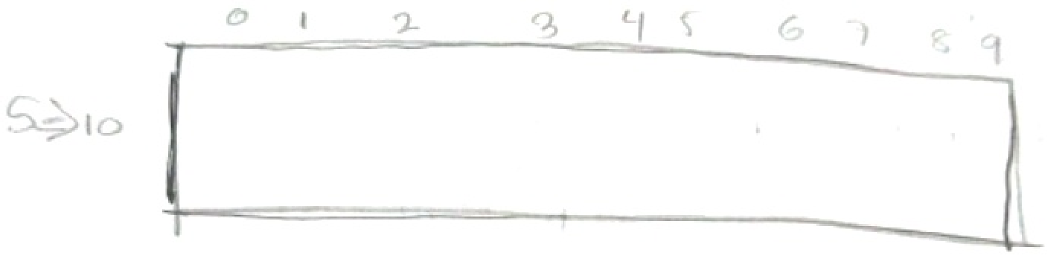
↓

hash function

Phonepe: → Yes bank

< 6-8 Year's hours

→ ICICI

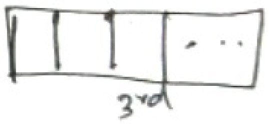


$\underbrace{203, 500, 707, 24, 84, 60}_{\text{mobile numbers} \Rightarrow 6}$

no is 203

⇒ Size of table is 10

⇒ $203 \% 10$ → 3 at Index



Size of table is 10

$= 500 \% 10 \Rightarrow$

Collision ⇒ when two different number are having Same hash value

0 203	204	707

⇒ hashing Solve Space related Problem, but collision resolution is Pending

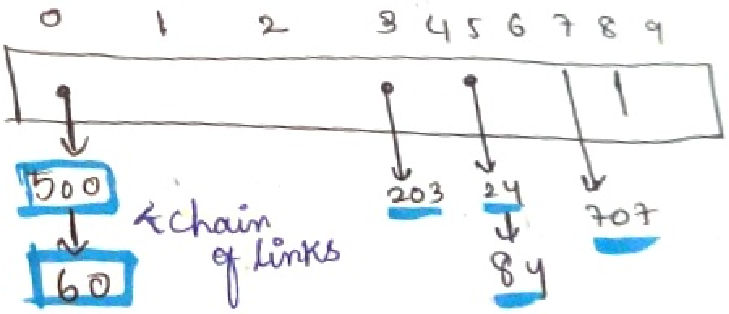
Collision resolution techniques

1) Seperate chaining → Linked List, → BBST
2) open Addressing → Linear Probing, → Quadratic Prob, → Double hashing

⇒ 6 → 203, 500, 707, 24, 84, 60
collision

0	1	2	3	4	5	6	7	8	9
500			203	24			707		

S ⇒ 10



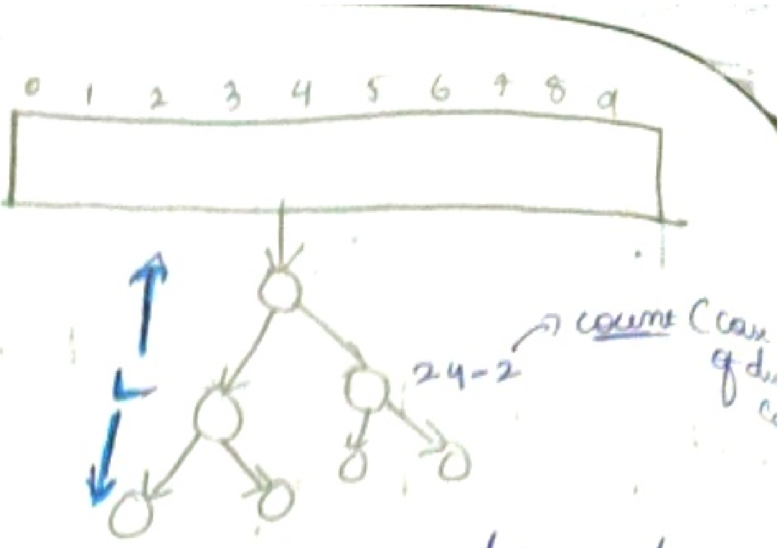
① Linked List   Insert(x)   Search(x)   delete(x) ↳

Ψ 1   L   L
at the head

(2) B.B.S.T (balanced binary search)

$H = \log_2 N$

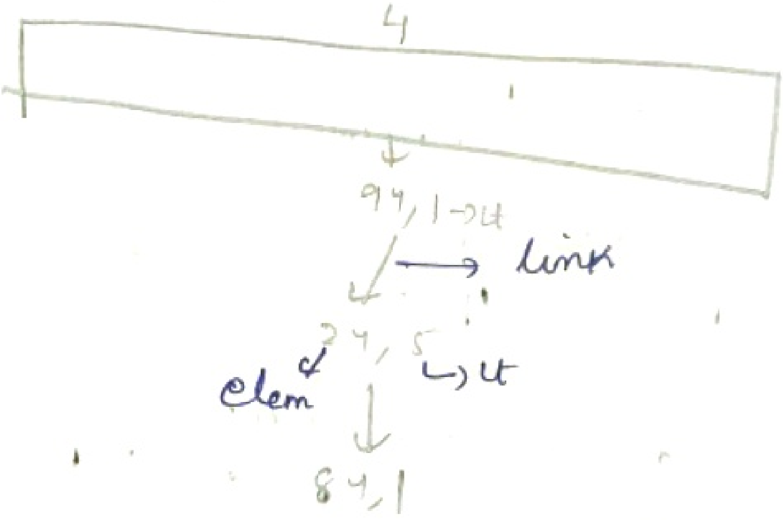
L → is height



Insert => $\log_2 L$, Search = $\log_2 L$, delete = $\log_2 L$

→ 6 → 203, 500, 707, 24, 84, 60, 94, 24, 24, 24

for ebultiple elements we can take frea and increment
or (duplicates),



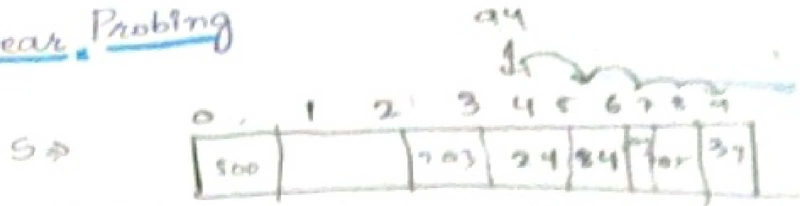
=)

Insert, delete Search

L, L, L

(Length of link)

Linear Probing



Lonmm→ 6, 203, 500, 707, 24, 84, 60

$h(x) = (x+i) \% 9$ $i \to 0, 1, 2, \ldots$

$\Rightarrow (203+0) \% 10$

$\Rightarrow (84+0) \% 10 \to$ occupied

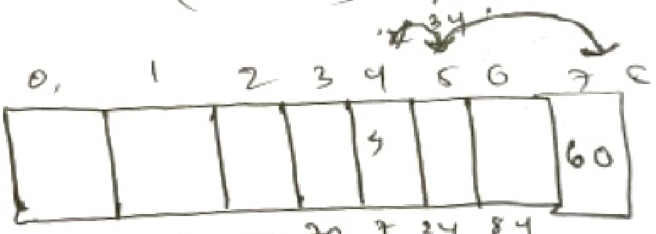
$\Rightarrow (84+1) \% 10 \Rightarrow$ Empty

If occupied it will check for the remaining indexes

If we empty we can ~~Put~~ Store the value

Quadratic Probing

$h(x) = (x + i^2) \% 5$



6, 203, 500 70 7 24 84

$\Rightarrow$ for $(34+0^2) \% 10$

arleady exist

$(34+1^2) \% 10 \Rightarrow$

already exist

$\Rightarrow (34+2^2) \% 10$

$\Rightarrow$ 8th index

$\Rightarrow$

Lp Vs QP probes 1, 2 3 603+0 60+1² 603+2²

LP: 3 4 5 6,

S=) 100 → 403, 503, 603, 703, 803, 4..

↓

Size of table

QP: 403 4 7 2 3 4 7 4

Problem P: 8 1 2 3 803

Probes

100) 703 (70
700
3

= 704
705
706

100) 703) 70
700
3

703 + (1)² => 704
703+(2)²=707
703+(3)²=712

100) 803 (80
800
3

= 803+1² =
803+2²=807
803+3²=
803+4²=

10) 819 (80
800
~~19~~

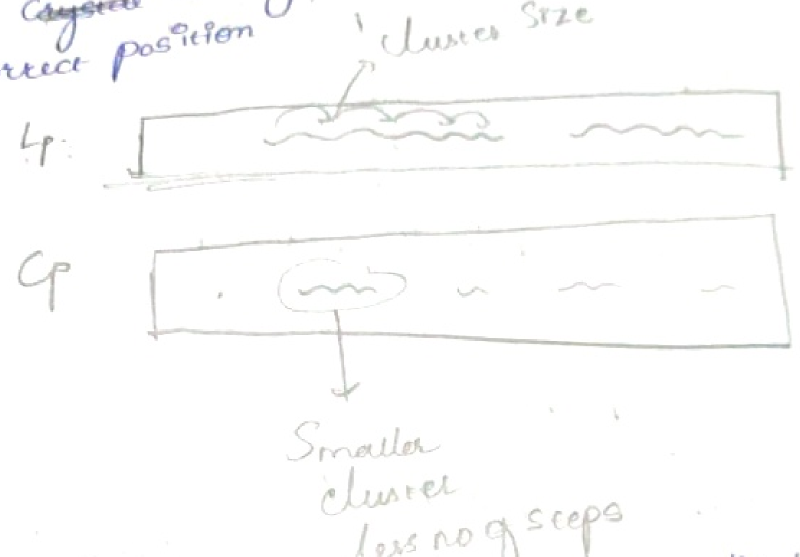
40, 50, ...

LP Probes → 0, 1, 2, 3, 4, 4, 5, 6 → 28

4 ... 6

QP Probes → 0, 1, 2, 3, 4, 1, 2, 3 → 16
(jumps)

Note:
Quadratic Probing is better when compared to lp
If ~~crystal~~ cluster is huge it will take time to go in to
correct position



70④
7
1②

Cluster Size ∝ no of Probes

time complexity

Insert   Search   Delete
P   P

P = Probes

+

6 -> 10

0	1	2	3	4	5	6	7	8	9

nums -> 6, 203, 500, 707, 24, 84

Availability

0	1	2	3	4	5	6	7	8
T	T	T	~~T~~	~~T~~	~~T~~	~~T~~	~~T~~	T
			F	F	F	F	F	

True -> Empty

F ↓ already Pre

delet -> 84 means get the Index and make It is True in Availaple array

Search -> 34 => ~~4~~ 5 Probe

You -> will think 34 is not Present

Problem is we dont differentiate btwn Empty and delete

→ If deleted keep on Searching

→ If Empty stop Searching & found!

3 things
- → Empty
- → occupied
- → Deleted

⇒ taking integer array

1) Empty → 0
2) Occupied → 1
3) Deleted → -1

-) S=10.

800				203	24	84	34	707	

Avalibility int

0	1	2	3	4	5	6	7	8	9
1		0	0	...	-1				

deleted

delete → 84 ⇒

Search → 34

3) Double hasing (far cluster)

$$h(x) = h_1(x) + h_2(x)$$

or

$$h_1(x) * h_2(x)$$ → Some func

$h_1(x) \Rightarrow$ linear $(i^2+x)\%s$

$h_2(x) \Rightarrow (x+i^2)\%s,\ (x+x+i^2)\%s$

Note:- $\{(x^i)\%s,\ (x!)\%s,\ (x^x)\%s\}$ bad

↳ Pow(x, i) → log {more time Complexity)

Good Hash functions

1) uniformly distribute the keys over a ta
2) Easy to compute
3)

Final time Complexity

SC → numbers   table Size

OA → 100   10 → yes

OA → 100   10 → NO

100 → yes 0%

OA → 100 ⟶ 1000 → yes (90% Empty)

OA → 100 ⟶ 10000 → yes (99% empty)

=> Higher Size more Search optimization

Internal   resizing

double

100~~0~~ → 200

101 → 200

```
Class Hashmap {

    Int ht [100000];

    Int availability[10000] = {0};
      # Void Insert
    int Val = (x+i) % 10000;
while loop
i=>0,
                                        idx
            If (availability [value] != 1)
                                      idx
                        ht [value] = x;
                        availabiity[value] = 1

        }
```

```
bool
Void deletion (int x)
    int y = Search (x);
        If (y! = = -1)
            avail [y] = -1;
    else
        Print("element found)
```

```
int Search (int x) {
=> loop i=>0, ...
        int value = (x+i)^2 / 1000
            idx
Empty
    If (availability[index]==
     ret false
    found :-
     If (availability[ind
          ==1 && ht[index
        Return index
```

Avaliable array :)

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

Empty in

Dempty → 0
2) Occupied → 1
3) Deleted → -1

arr :)
S = 10 :)

0	1	2	3	4	5	6	7	8	9
500		,	203	24	84	34	707		

Elements =) 6, Elements → 203, 500, 707, 24, 84, 3,
707.

i → 0, 1, 2, 3, 4, ..

h(x) => (x+i) % 5

=> (203+0) % 5

= 10) 203 (20
200
3

=) arr [h(x)] => Element

=)

=)

index :

Search (x)

```
=) for i → 0, .. n
    int idx = (x+i] % 100
    =) Emery
    if (avai[idx] != 0)
        return false
    If (avai[idx] == 1 And
        arr[idx] == x)
        return true
```

i → 0, 1, 2, 2, 0, 1

=) 500

=) (500+0) % 5

= (500) % 10

=) 0 → index   delete

= h [0] => 50   If found.
avai index [ ]