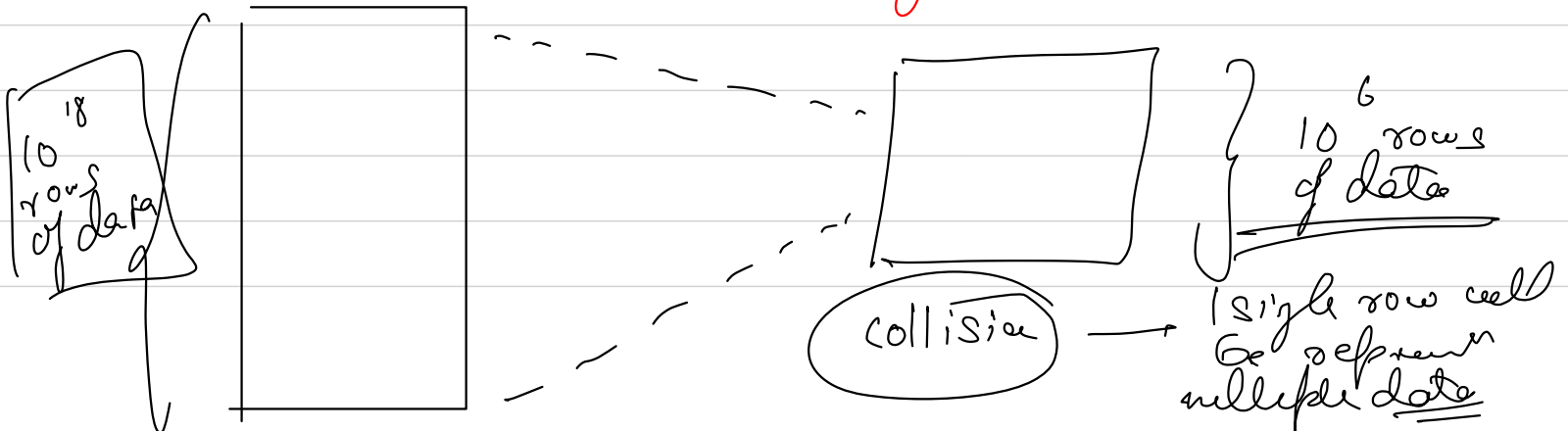
Hashing → Basic Concept of hashing

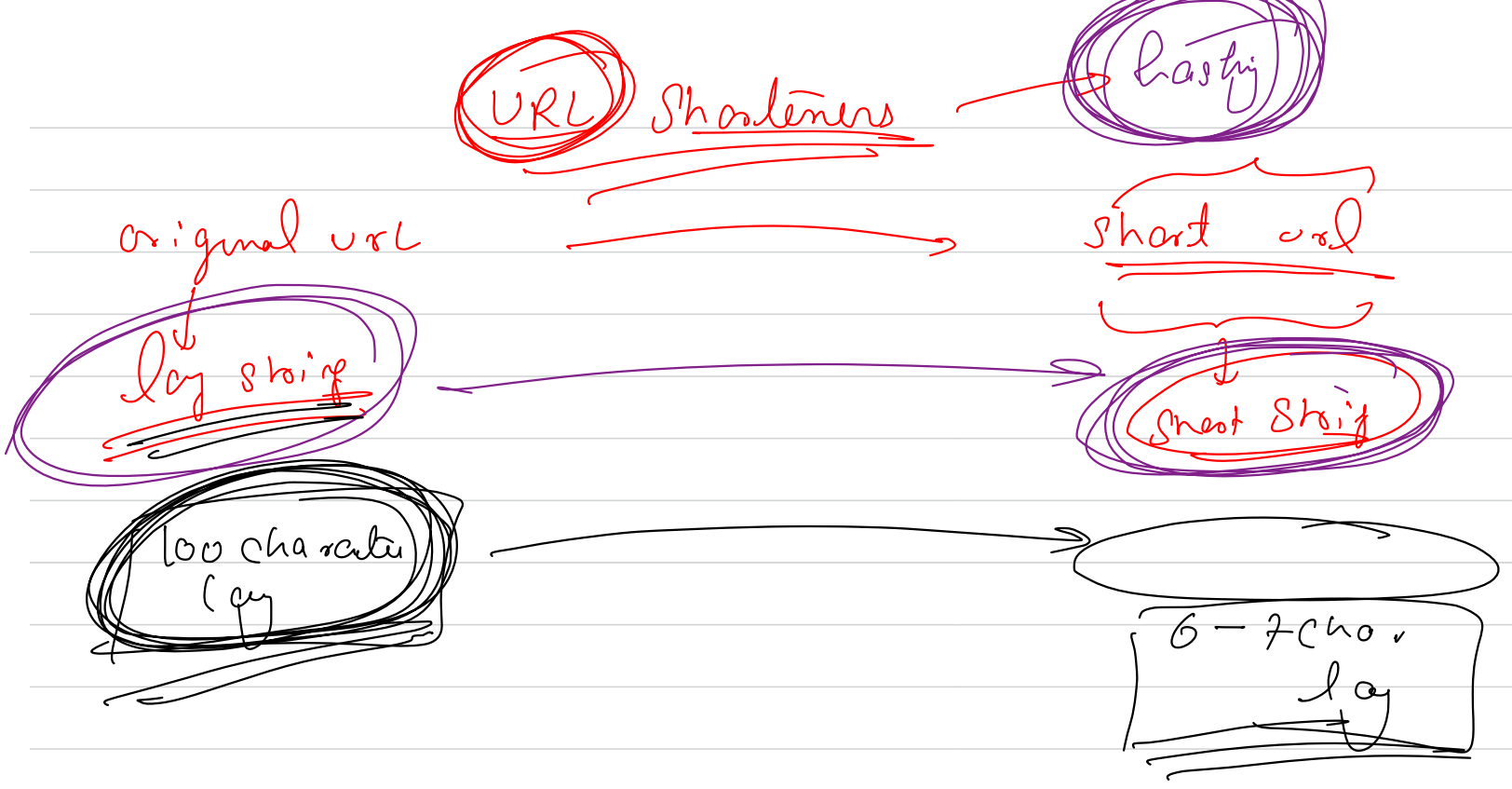
Applications of hashing

How to implement unordered_map / Dict / HashMap

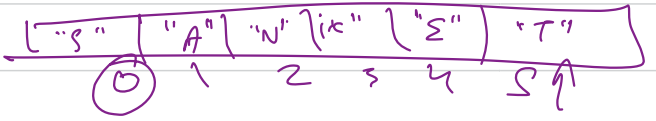
Data structures like unordered_map / dict / hashmap under the hood use the concept of hashing.

Hashing means, mapping of a large set of data to a small set of data.



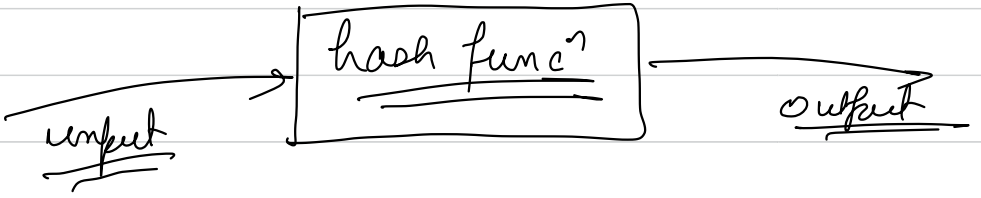

URL Shorteners
Bitly
original url
long string
short url
short string
100 characters
6-7 char

Arrays → <index, value>



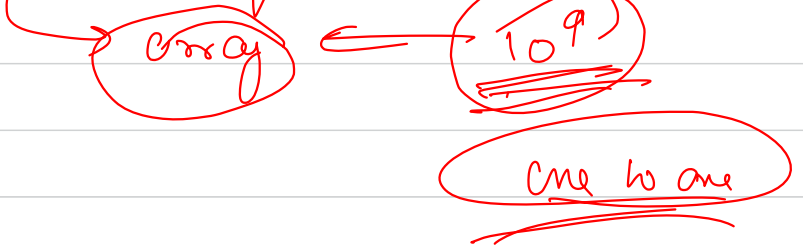
How to do hashing ??

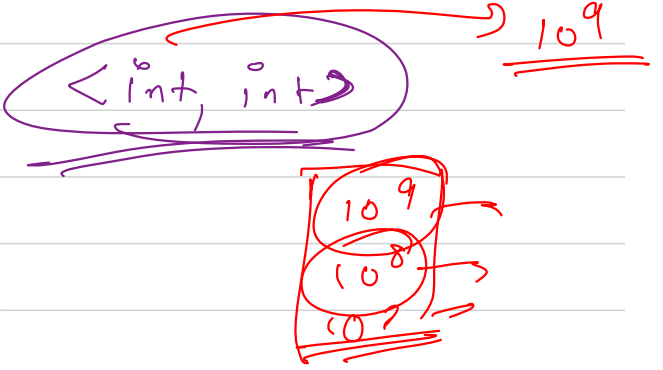
→ There are multiple algorithms available which are pretty complex.



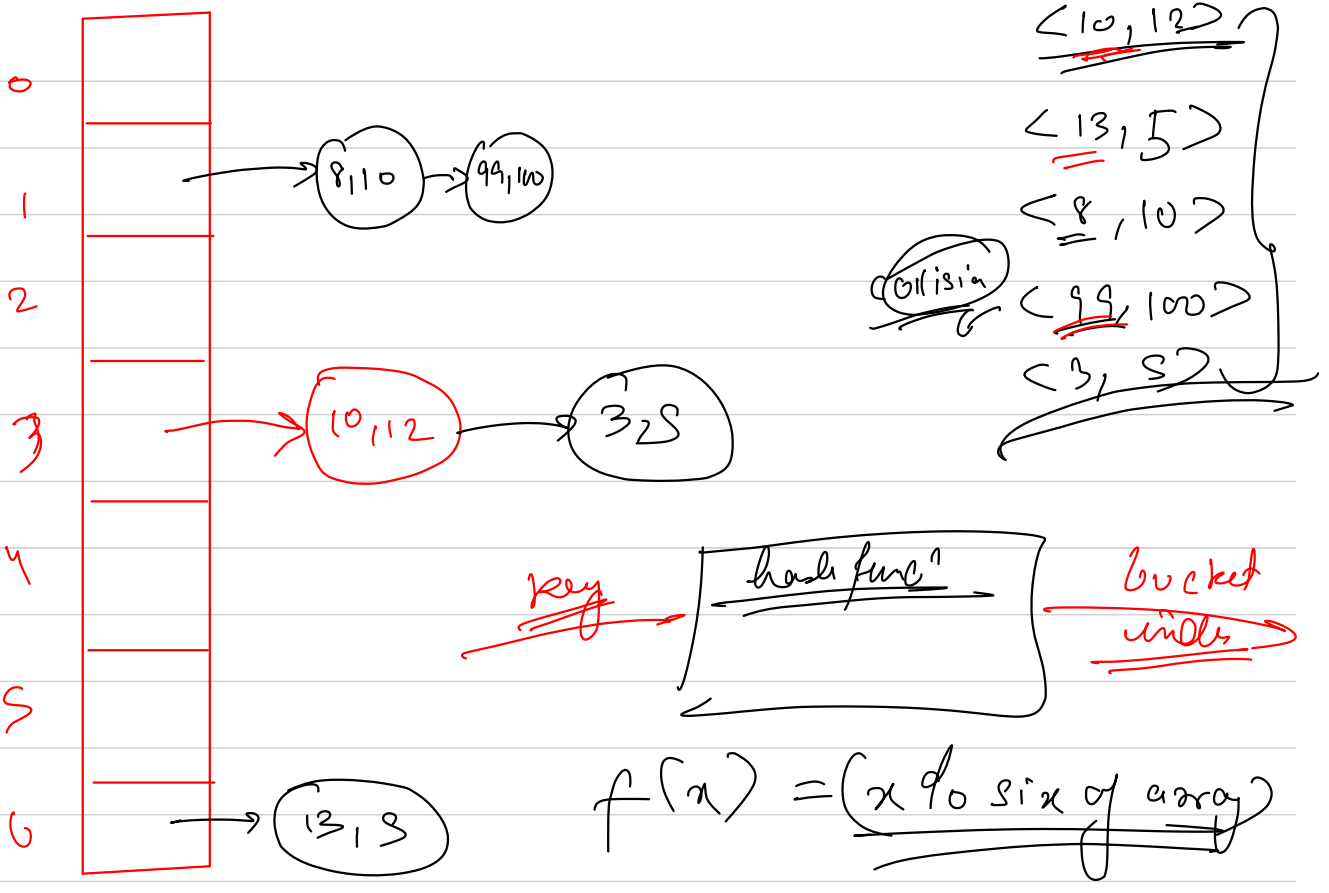
→ HashMaps / Unordered_Maps / Dict ← How are these implemented ??

↳ array of linked list





one index of array is called bucket of linked list


key
value
<10, 12>
<13, 5>
<8, 10>
collision
<99, 100>
<3, 5>
0
1
2
3
4
5
6
8,10
99,100
10,12
3,5
13,5
key
hash func^n
bucket index
f(x) = (x % size of array)

How to avoid collision?? → make good hash func^n

What size of array we should take??

What if we get very long linked list chains

How the TC → O(1)

How to manage collision ??

- Probing
  - Linear probing
  - Quadratic probing
- Separate chaining

Linear probing → if the bucket index is occupied we search for first empty location.

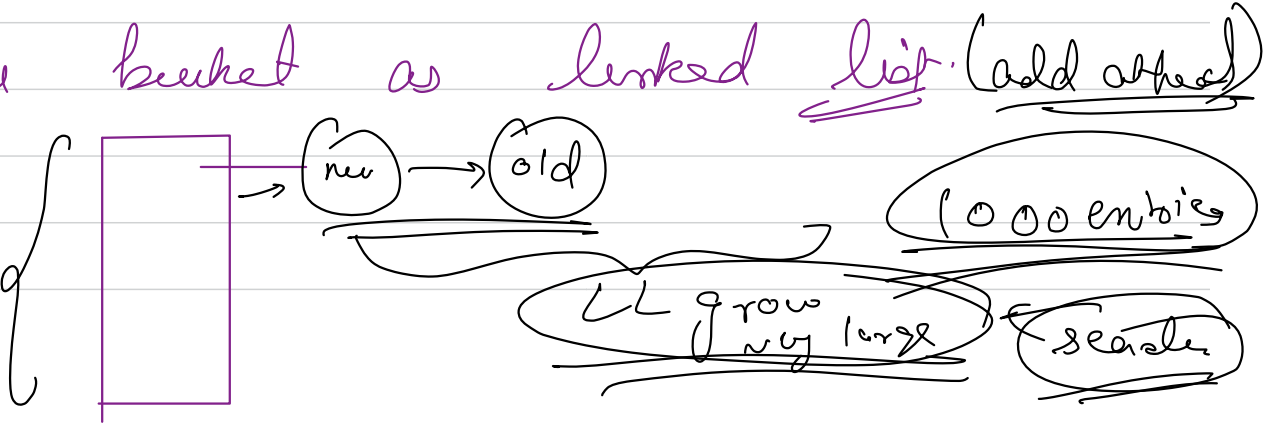
Quadratic probing → b_i ← already filled

b_i+1^2, b_i+2^2, b_i+3^2, b_i+4^2 .. ....

Double hashing → $h_1(key)$ → bi

(key, value) → $bi + h_2(key)$ → bi'

Separate Chaining → if we get multiple entries for same bucket, we store them in the same bucket as linked list. (add at head)



* Load factor (λ) = $\frac{\text{amount of data in the HT}}{\text{Size of hashtable}}$

how much your hashatable is filled.

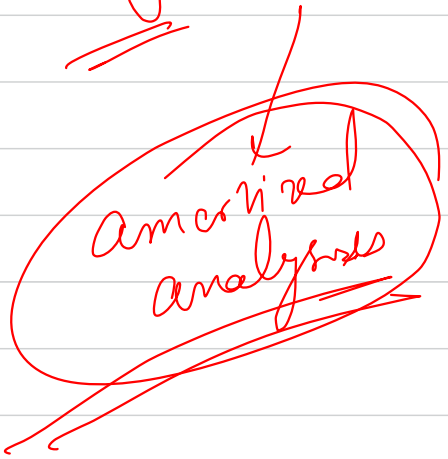
$\lambda >$ delta → rehashing

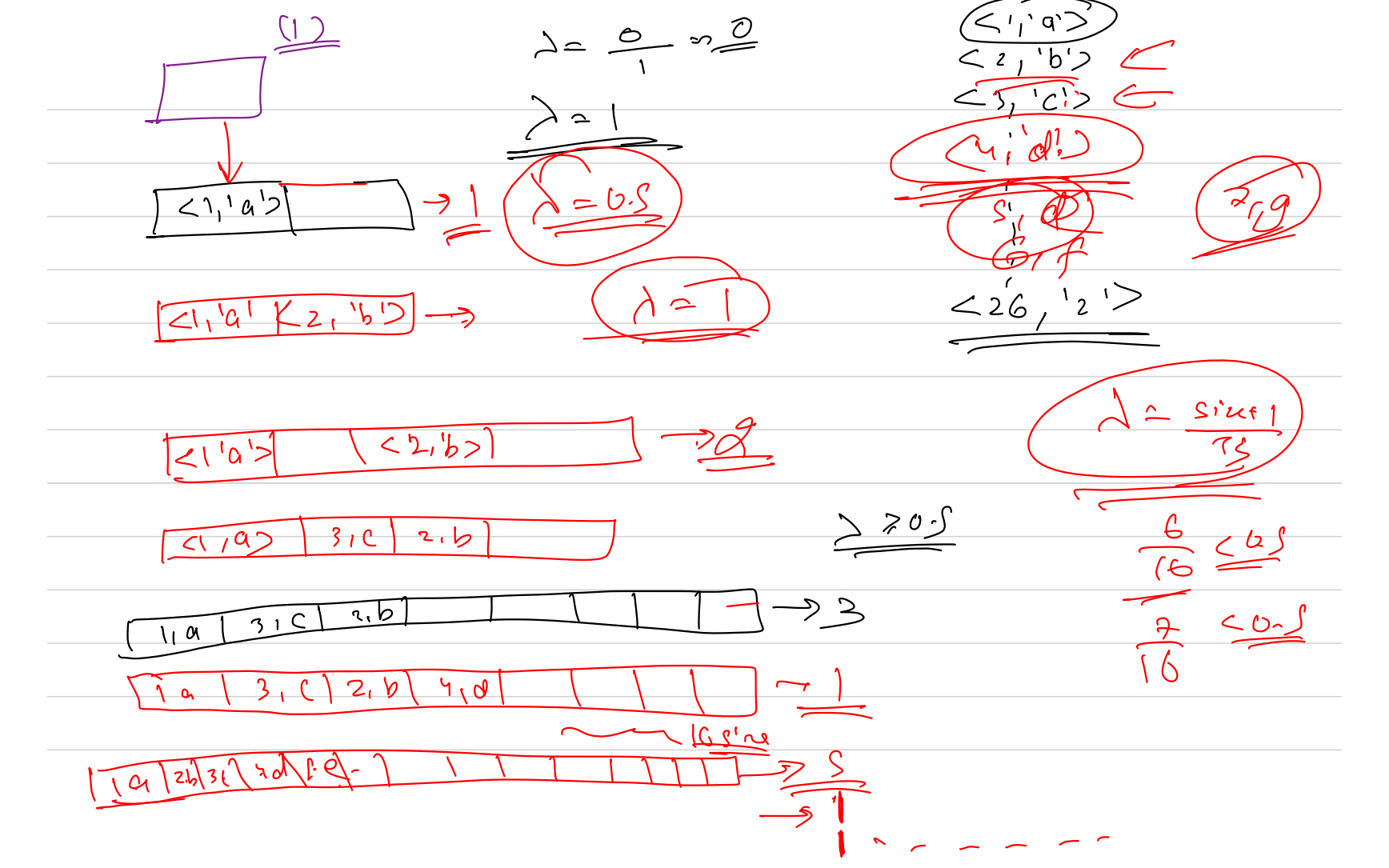
$\lambda \geq 0.5$ — cpp

$\lambda > 0.6$ — Jav

rehashing → double the size of array &

again add all the elements based on Hfn




(1)
λ = 0/1 ≈ 0
λ = 1
λ = 0.5
λ = 1
λ = size of TS
λ ≥ 0.5
6/16 < 0.5
7/16 < 0.5

as soon as

$\lambda > 0.5$

↓

rehashing

(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)

operation

1 → 1

2 → 2 → $2^0 + 1$

3 → 3 → $2^1 + 1$

4 → 1

5 → 5 → $2^2 + 1$

6 → 1

7 → 1

8 → 1

9 → 9 → $2^3 + 1$

⋮ ⋮

n

TC → total no of operation / total no of inserts

$TC \rightarrow 1+(2^0+1)+(2^1+1)+1+(2^2+1)+1+1+1+(2^3+1)\ldots$

$TC \rightarrow \underbrace{(1+1+1+1\ldots\ldots)}_{n} + \underbrace{(2^0+2^1+2^2\ldots\ldots)}_{\log n \text{ terms}}$ → GP

n

$n + 1 \times 2^{\log n} - 1$

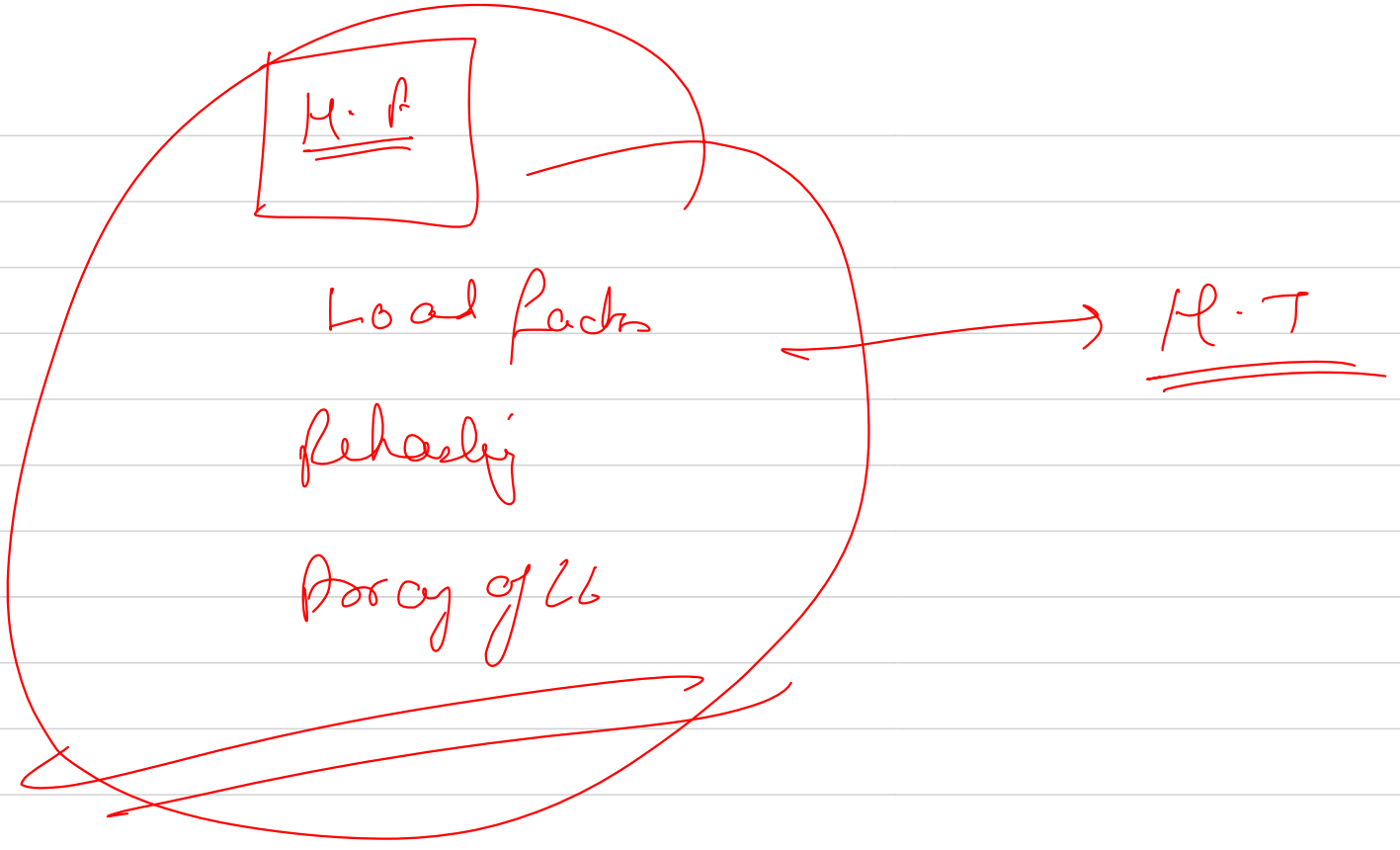
n

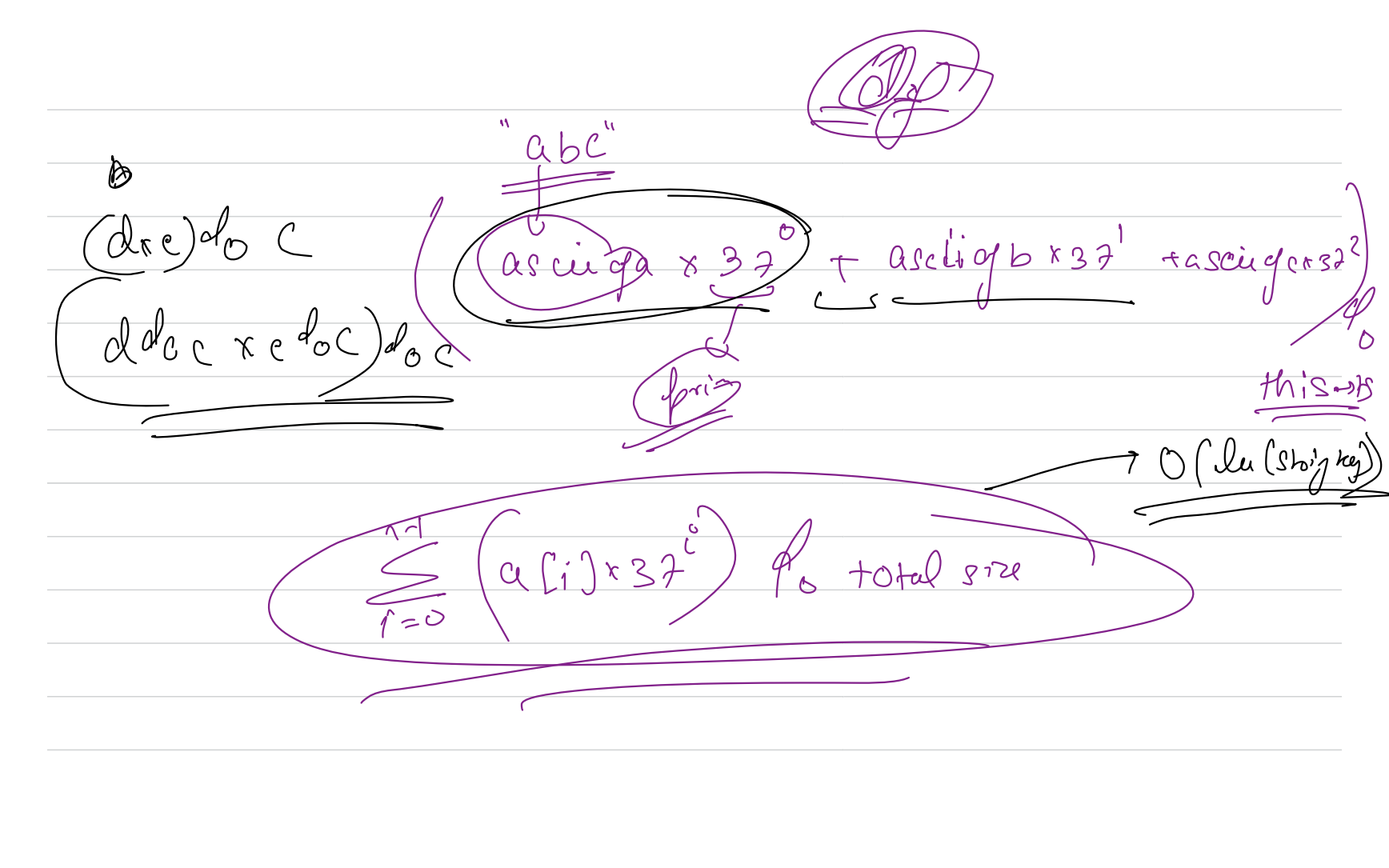
$\Rightarrow \frac{n + n - 1}{n} \rightarrow \frac{2n}{n} \rightarrow$ const

avg complexity

$\Rightarrow (1)$

$M.T \rightarrow O(1)$


M. F
Load Factor
Reliability
Decay of Cc
M. T


"abc"
ascii(a) x 37^0 + ascii(b) x 37^1 + ascii(c) x 37^2
prime
this->hs
O(len(string))
sum_{i=0}^{n-1} (a[i] x 37^i) % total size

$(a+b) \circ c \longrightarrow (a \circ c + b \circ c) \circ$