Class 18
25 February 2023 10:58

CLASS-18

Searching and Sorting

Sorting Algos
(Time Complexities)
Stability
inplace

Binary Search
- Peak element
- Rotated array
- Binary Search on Ans

Hashing

Data → small Data
Processed Data

Hi My Name is Naman

Data → function → output (Hashed value)

Data != Hashed Value

abcdef (Hashed Data)

{key, value}

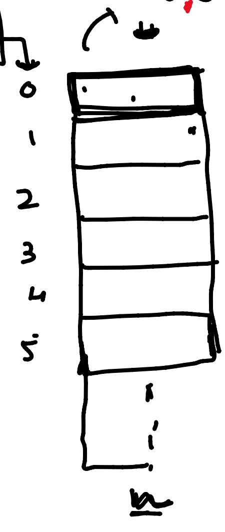
Hash function

C++

Hash Table

unordered_map

string,
int,
custom/user Defined
Datatype



Data
↓
Hash function
↓
output

Hash Functions (form)

String

Data
int, char,
string, user-Def.

Pair ⇒ C++

pair<int, int> P

Price

"mango" 100
"orange" 50

{string, int}

{

{string, int} → {"orange", 50}

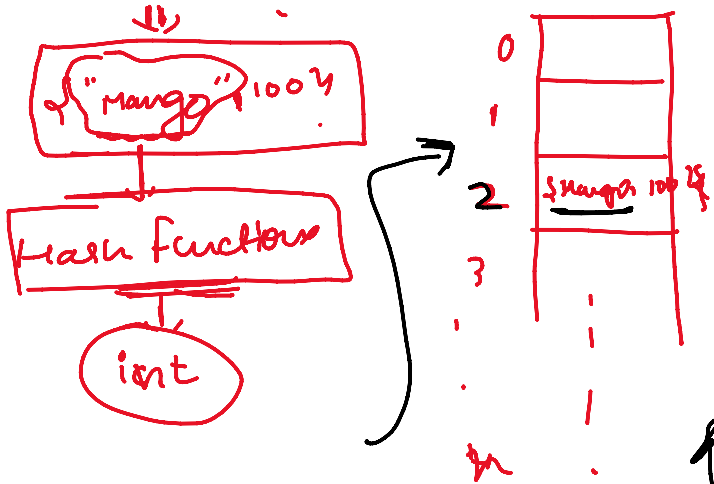
{int, int} → {5, 5}

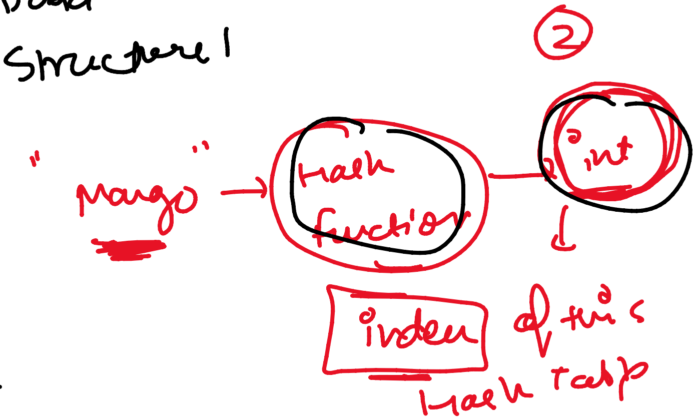
pair< pair<int, int>, int>

pair< datatype1, datatype2> p

vector, set, int, string, char, user-defined data

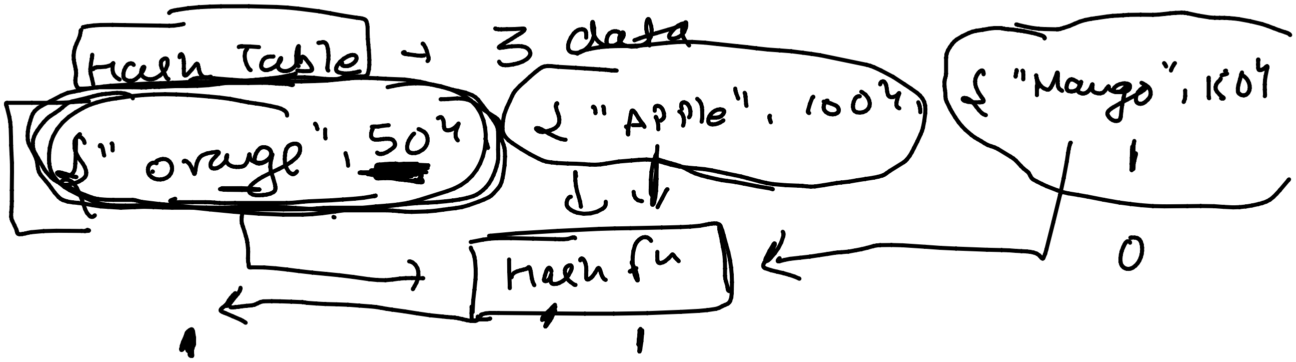
(Data structure)





Hashing

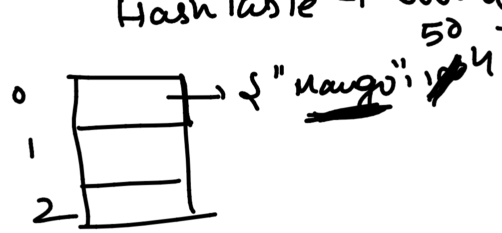
Random



Random

HashTable → unique value

Key → {value}



{"mango", 100}

{"Mango" 50}

Hash fn → 0

{ unique Key Value Pairs

Hashing → Array

class {

~~unord~~ Hashing → Array

class {
Array
HashFunction
}

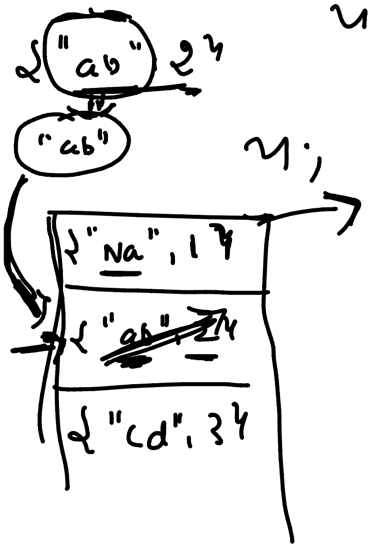
Hash Table
class {
Pair<int,int> ar [100];
void Hashfn (int k)
{
...
}

insert (Pair<int,int> p)
{ int key = p.first;
int idx = HashFn (key)
ar[idx].first = p.first;
ar[idx].second = p.second;
}

HashTable
insertion

DS advanced array + Hashing

⇒ Hash Table operations
→ Insert {key, value}
→ Search by Key
→ erase using key



query → "ab"

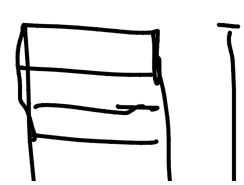
HashTable
Data store
represented
by Key

Duplicate value ×

{Key, value} Pair

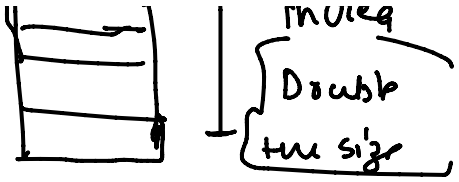
Hash Table → Fixed Size But it is Dynamic

Hash function



usage
increase
Double

Hash Function

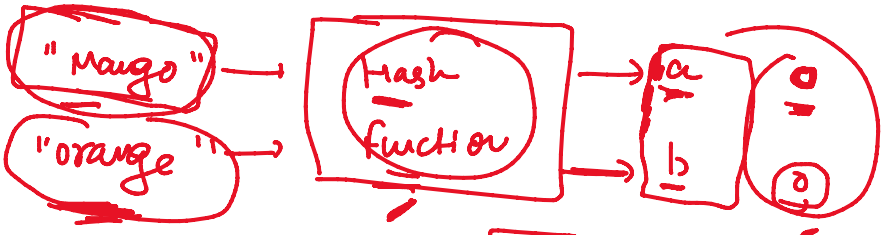


Hash Functions → overview

→

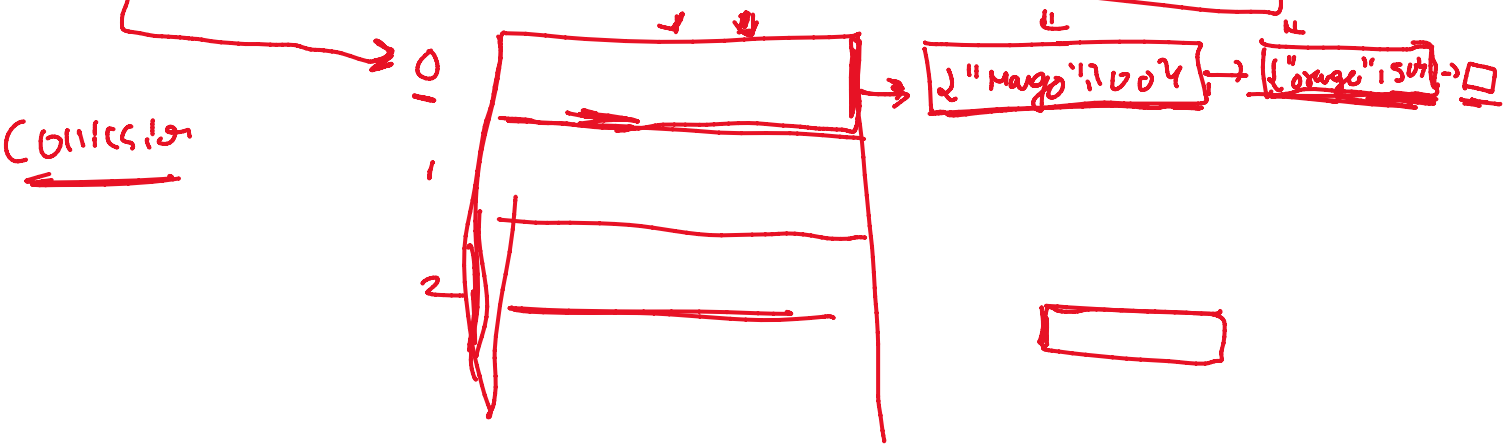
↳ "Mango", 100₹

↳ "orange", 50₹



a == b = 0

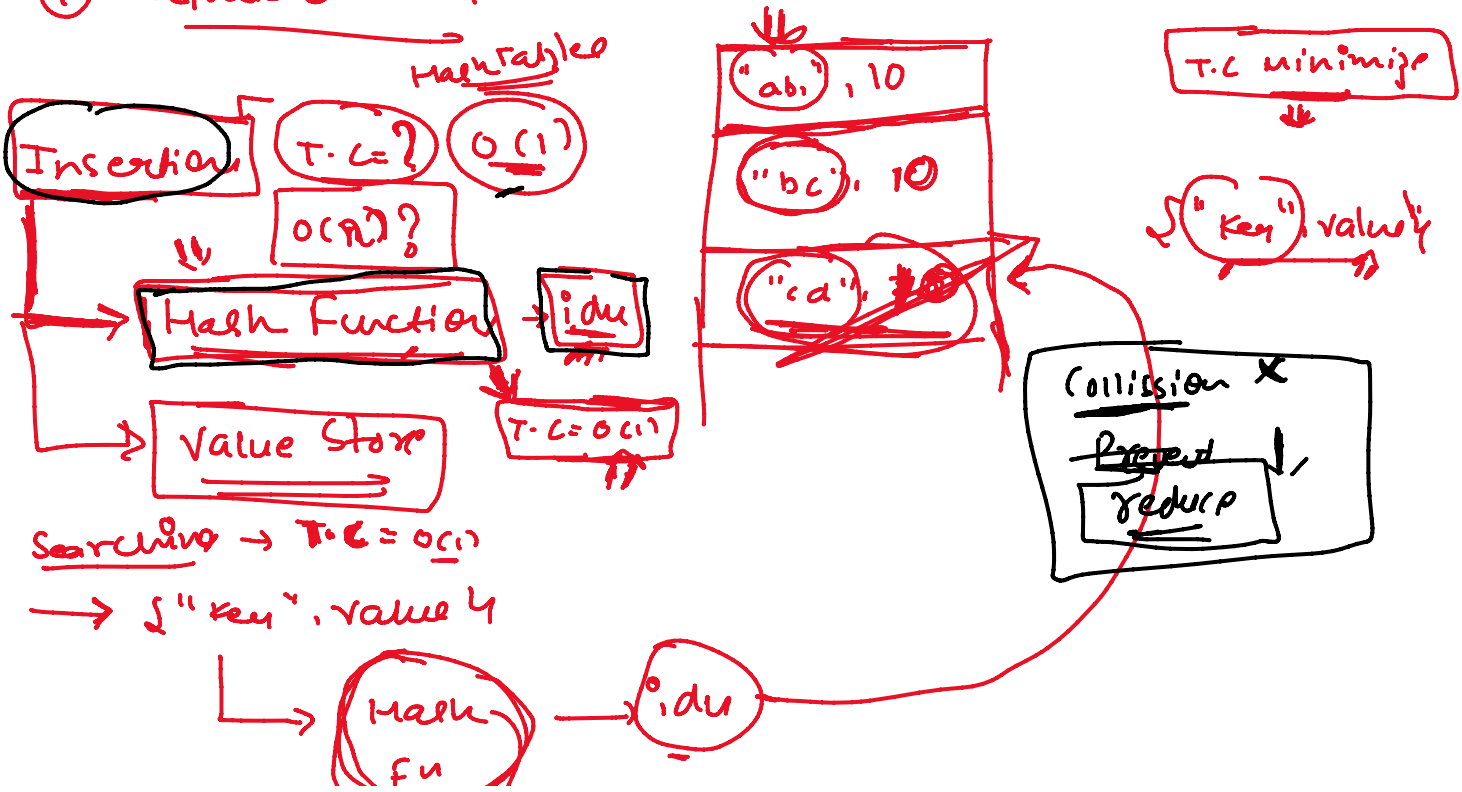
Collision in Hash Table



Collission

Collission Handling Schemes

① Separate Chaining

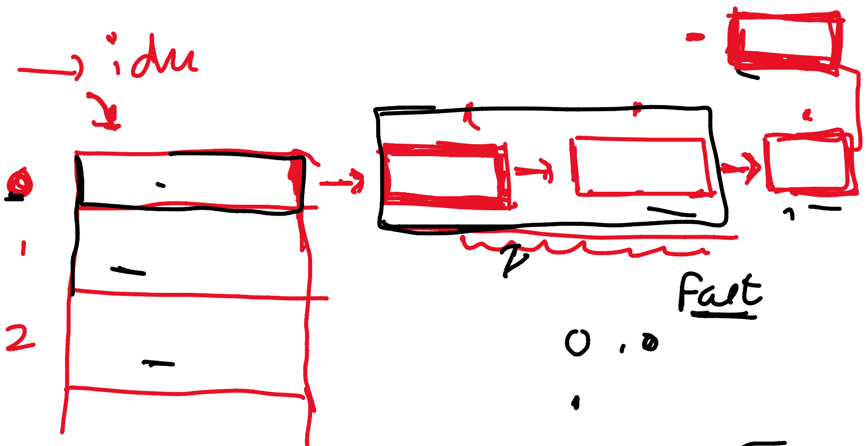


→ Hash fn → idx

T.C: O(1)

→ erase {"key", value}

→ Hash fn → idx



Insertion → O(1)

{key, value} → Hash fn → O(n) worst case

idx

Searching → O(n)

Erase / delete → O(n)

Good Hash function

→ uniformly Distribute (chance of collision ↓)

→ Reduce chance of collision

→ Fast to compute

"abc" → int

Eg → 97 + 98 + 99 = 296 % table size → idx → 0 - T.S - 1

chance of collision ↑

"bac" → 98 + 97 + 99 = 296

"cab" → 99 + 97 + 98 = 296

"bca"

"cba"

Collision chance ↑

"abc" → Base 26

97 × 26^0 + 98 × 26^1 + 98 × 26^2 = idx] collision

$97 * 26^0 + 98 * 26^1 + 98 * 26^2 = idx_1$

"bac"

$98 * 26^0 + 97 * 26^1 + 99 * 26^2 = idx_2$

] Collission Prevent

⇒ Prime No and Table size = Prime No

Collission ↓ data uniformly distribute in Table

Prime No

n = 37

"abc"

$97 * 37^0 + 98 * 37^1 + 99 * 37^2 = (idx) \% T.S = idx_1$

↓ Distributed

Collission reduce too much

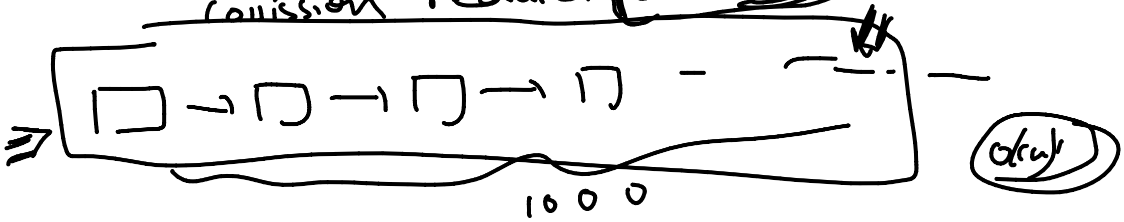


Table size → Prime No

Rehashing

Load Factor

T.S = Total size = No of element can be stored

C.S = No of element stored

C.S = No of elements stored currently

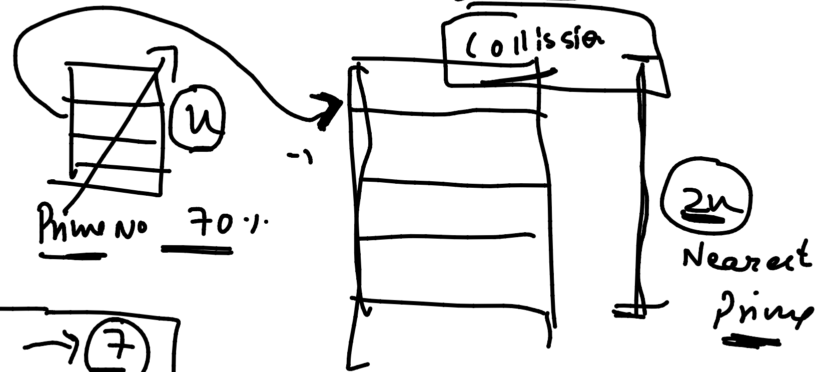
L.F = C.S / T.S

50% — 0.5 - 0.6 — 60% — Threshold

Jyada 80% — 90%

Collission

Hash Table doubled



Rehashing

3 → 7

Set → value

- "abc" → 0 + 1 + 2 = 3

- HashTable
- Hash function
- Rehashing
- Collission Handle → linked list

class { }

Table, Hashfunction, Hashfunc, Rehashing, Collission Handling

STL

unordered_map<int, int> m;

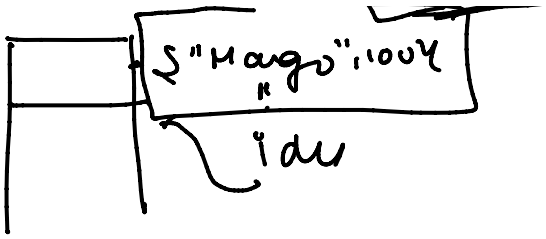
→ HashTable → { } {key:value} Pairs

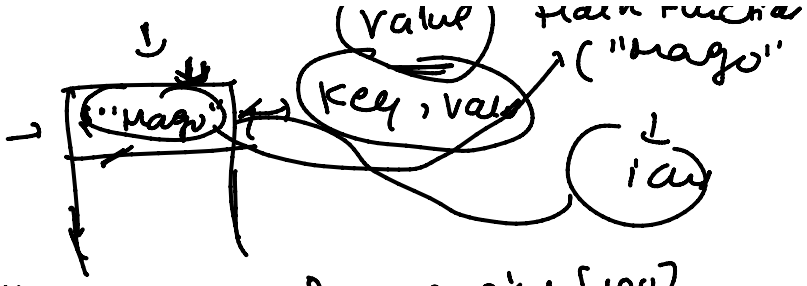
vs map

unordered_set → Hash-Table

value — Hash function ("Mango"

{"Mango", 100}

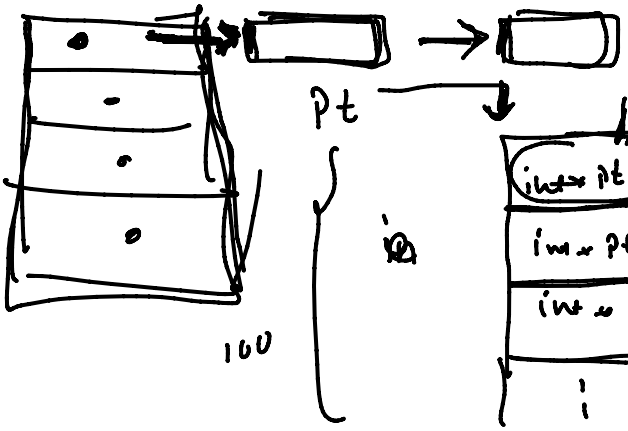


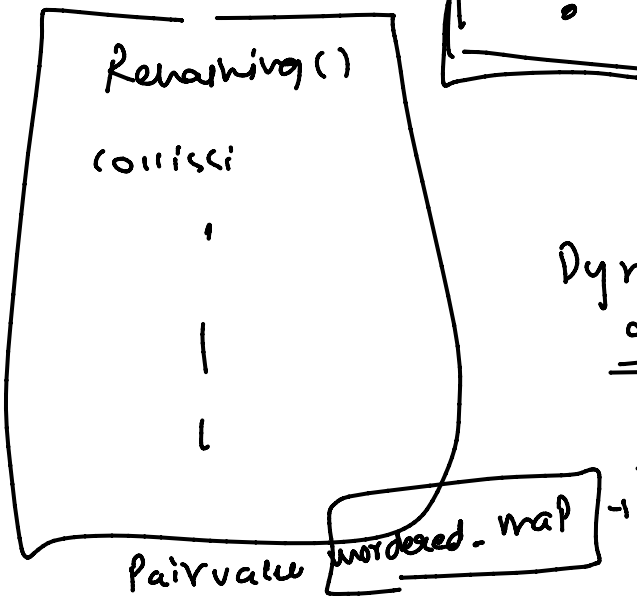


int ar[]

Dynamic Array

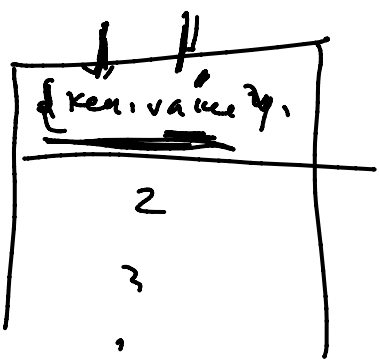
int* pt = new int[100]





Dynamic 2D Array

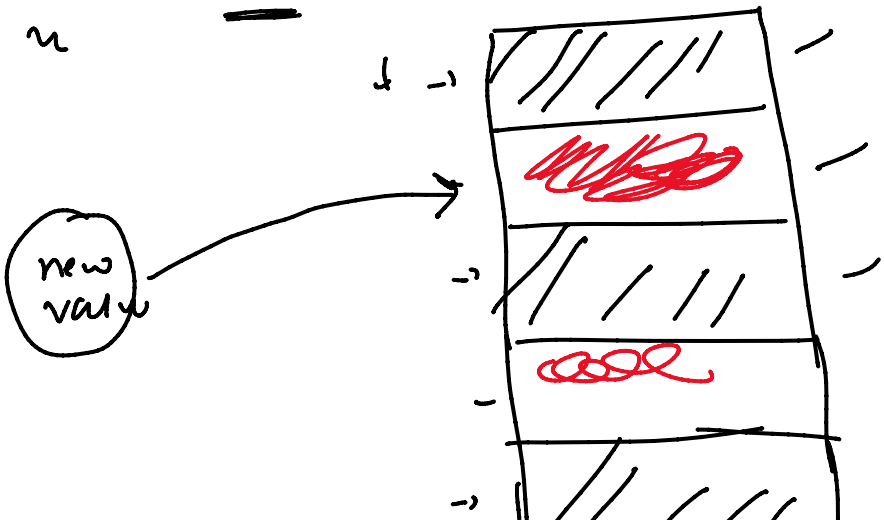
int** pt = new int*[10];



For(auto i: map)

i.first

i.second

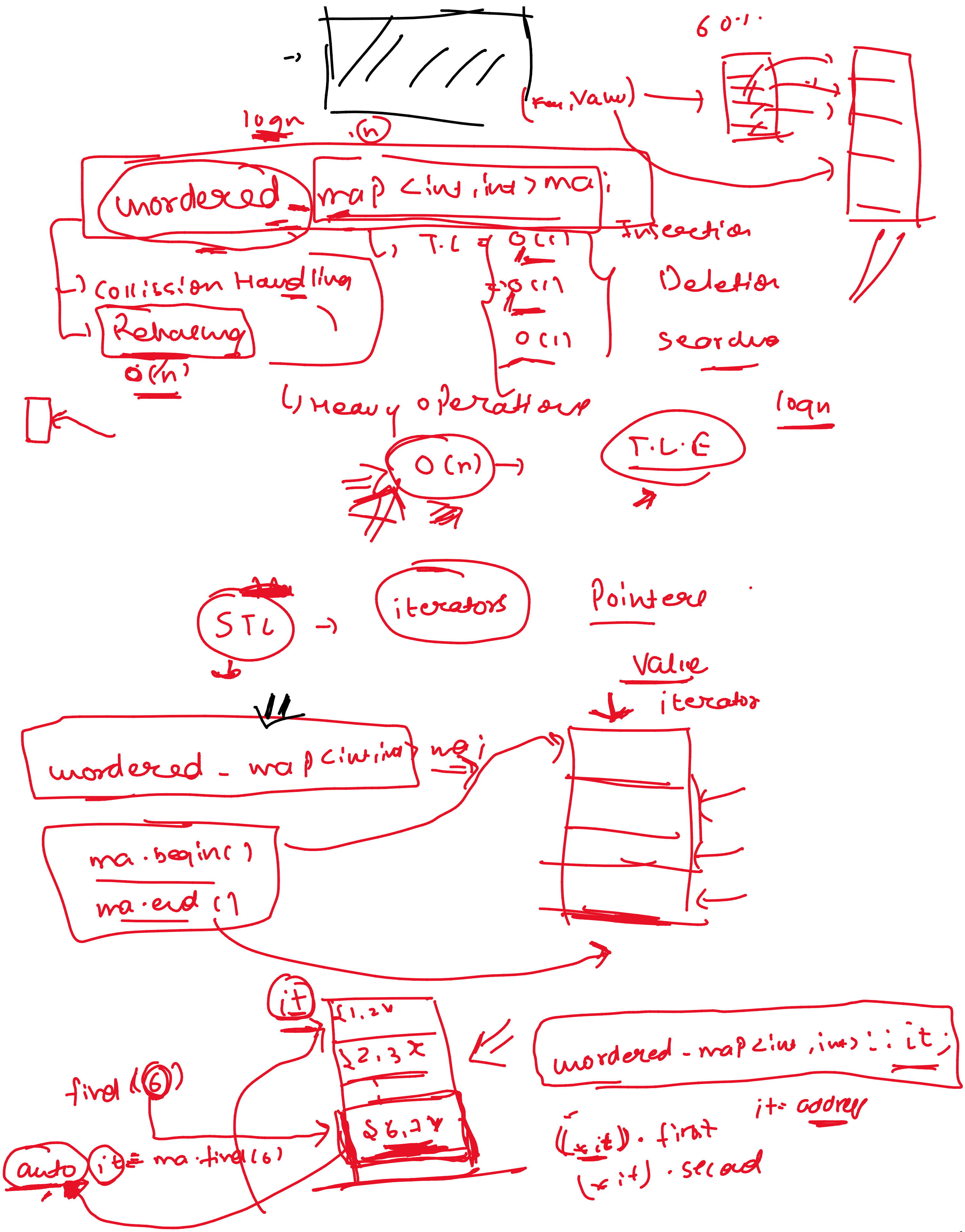


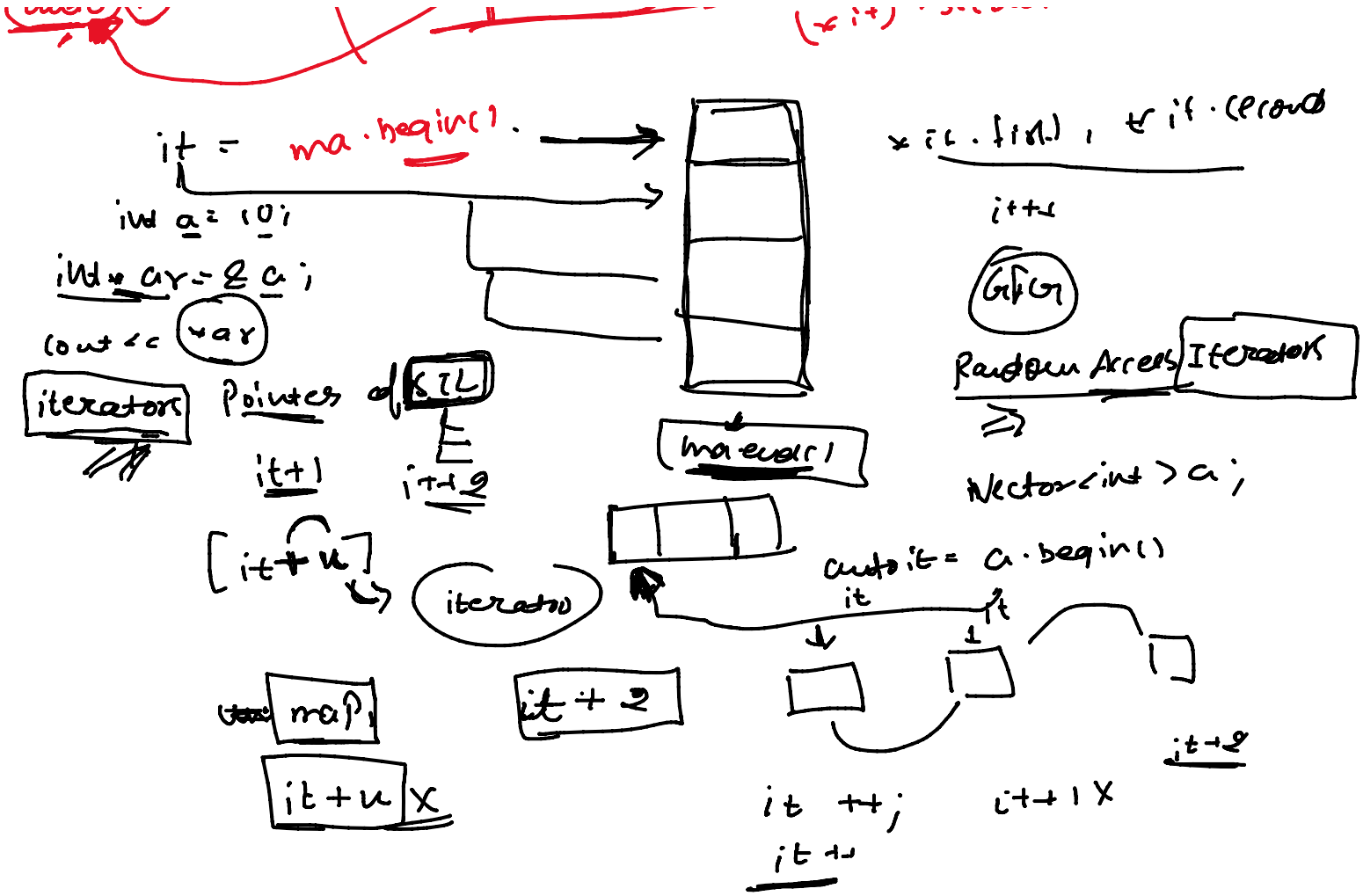
hashfn → idx

0 , 9, 13

(5)

60.1.


60%
(key, value)
log n
O(n)
unordered_map<int, int> ma;
T.C = O(1)
Insertion
O(1)
Deletion
O(1)
Searching
Collision Handling
Rehashing
O(n)
Heavy operations
O(n)
T.L.E
log n
STL
iterators
Pointers
Value
iterator
unordered_map<int,int> ma;
ma.begin()
ma.end()
it
{1,2}
{2,3}
{6,7}
find(6)
auto it = ma.find(6)
unordered_map<int, int>::it;
it = address
(*it).first
(*it).second


it = ma.begin().
int a = 10;
int &ar = &a;
cout << ar
iterator
Pointer of STL
it+1
it+2
[it+n]
iterator
map
it+n X
it + 2
ma.end()
auto it = a.begin()
it
it
1
it ++;
it+1
it+1 X
it+2
Vector<int> a;
Random Access Iterator