C++ STL revision

- → some intro to CP
- → STL data structures
- * → STL on binary search
- → some tricks on STL

10^8 operations in C++ (CF)

$n \to 2 \times 10^5 \longrightarrow O(n) \to 2 \times 10^5 < 10^8$ ✓ I see

↳ $O(n \log n) \to 2 \times 10^5 \times 20$

$O(n^2)$ → $(2 \times 10^5)^2 \rightarrow 4 \times 10^{10}$ → $4 \times 10^6 < 10^8$

→ 400 sec ✓ 1 sec

>>> 1 sec

tle

t

$$\begin{bmatrix} n_1 \\ n_2 \\ 1 \\ n_t \end{bmatrix}$$

$n_1 + n_2 + \cdots n_t \leq 2 \times 10^5$

$\underbrace{n + n + \cdots n}_{t \text{ times}}$

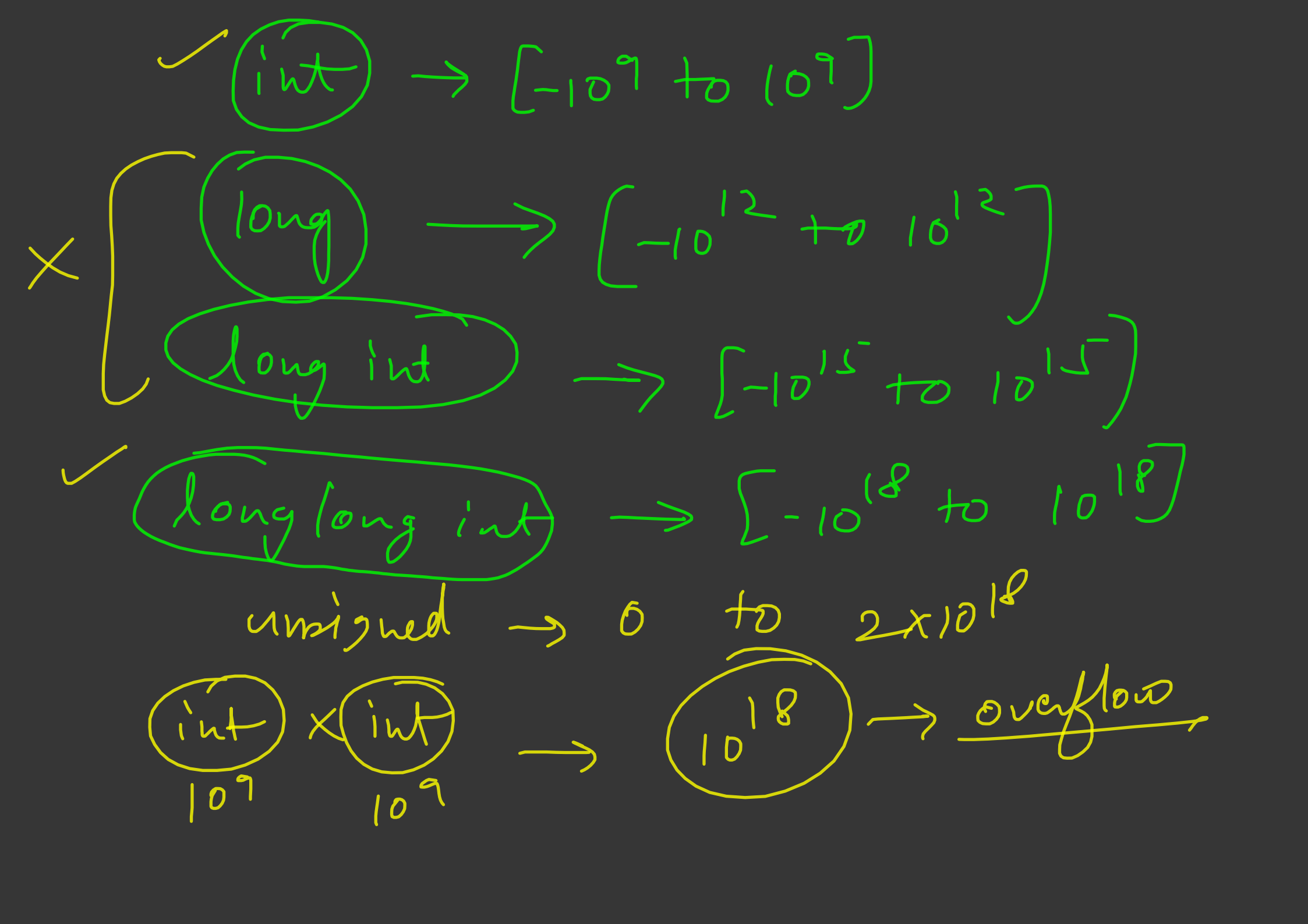
$n t \leq 2 \times 10^5$ ✓

$n \times t \rightarrow 2 \times 10^5 \times 10^4 \rightarrow 2 \times 10^9$ ✗

✓ (tle)

$t \times n \rightarrow 10^4 \times 2 \times 10^5$ ✗

2×10^5 ✓


int → [-10^9 to 10^9]
long → [-10^12 to 10^12]
long int → [-10^15 to 10^15]
long long int → [-10^18 to 10^18]
unsigned → 0 to 2x10^18
int x int → 10^18 → overflow
10^9
10^9

(long long) x (long long) → 10^{36}

↳ modulo arithmatic comes to the picture

Containers

↳ vectors → dynamic array.

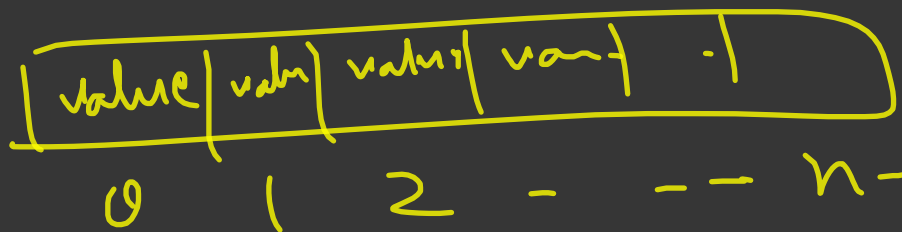
① → vector<int> vec,

emplace_back()

vec.push_back(1),

② → vector<int> vec(n);

(3) → vector<int> vec(n, value);



initial value

vec.pop_back()

vec.back()

vec[vec.size()-1];

RTF

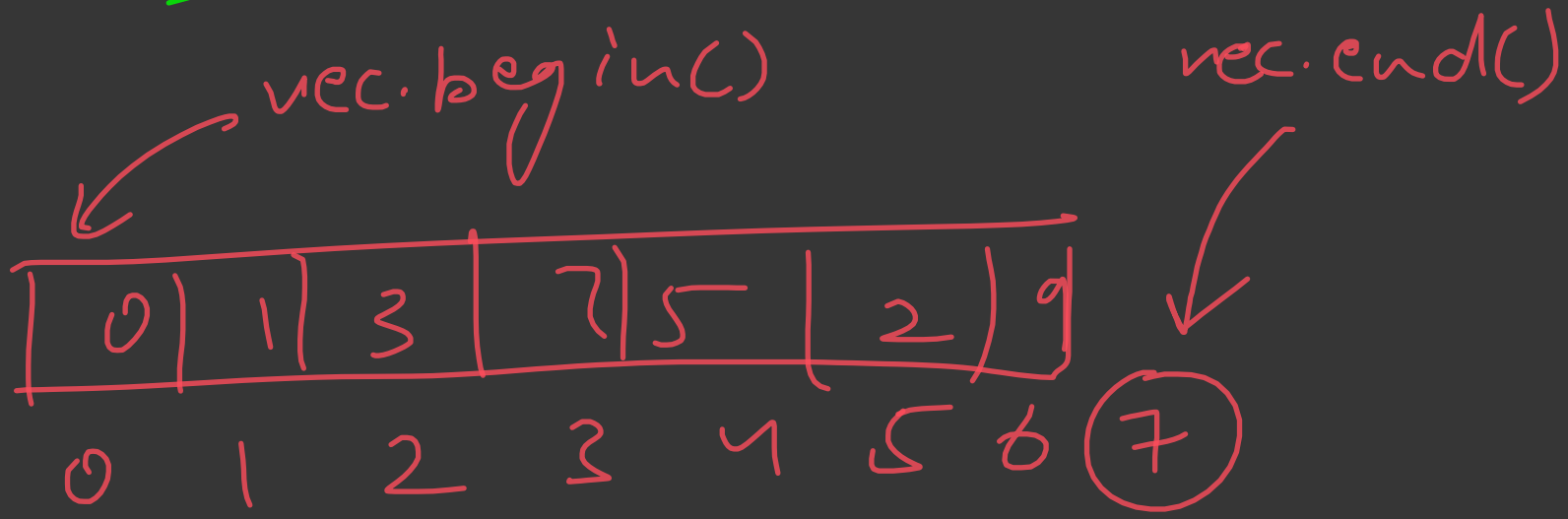
vec[index]

↳ n-1

vec.erase(vec.begin() + index)

↳ O(n) → very bad

Eg

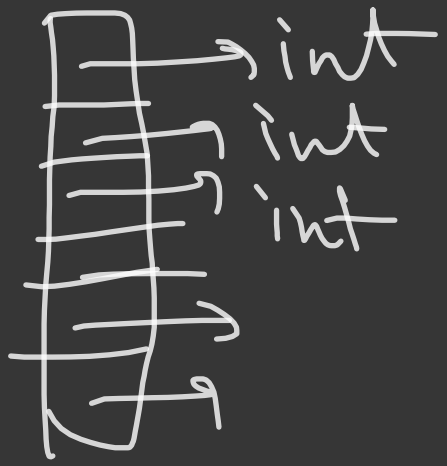


(vec.begin() + 2)

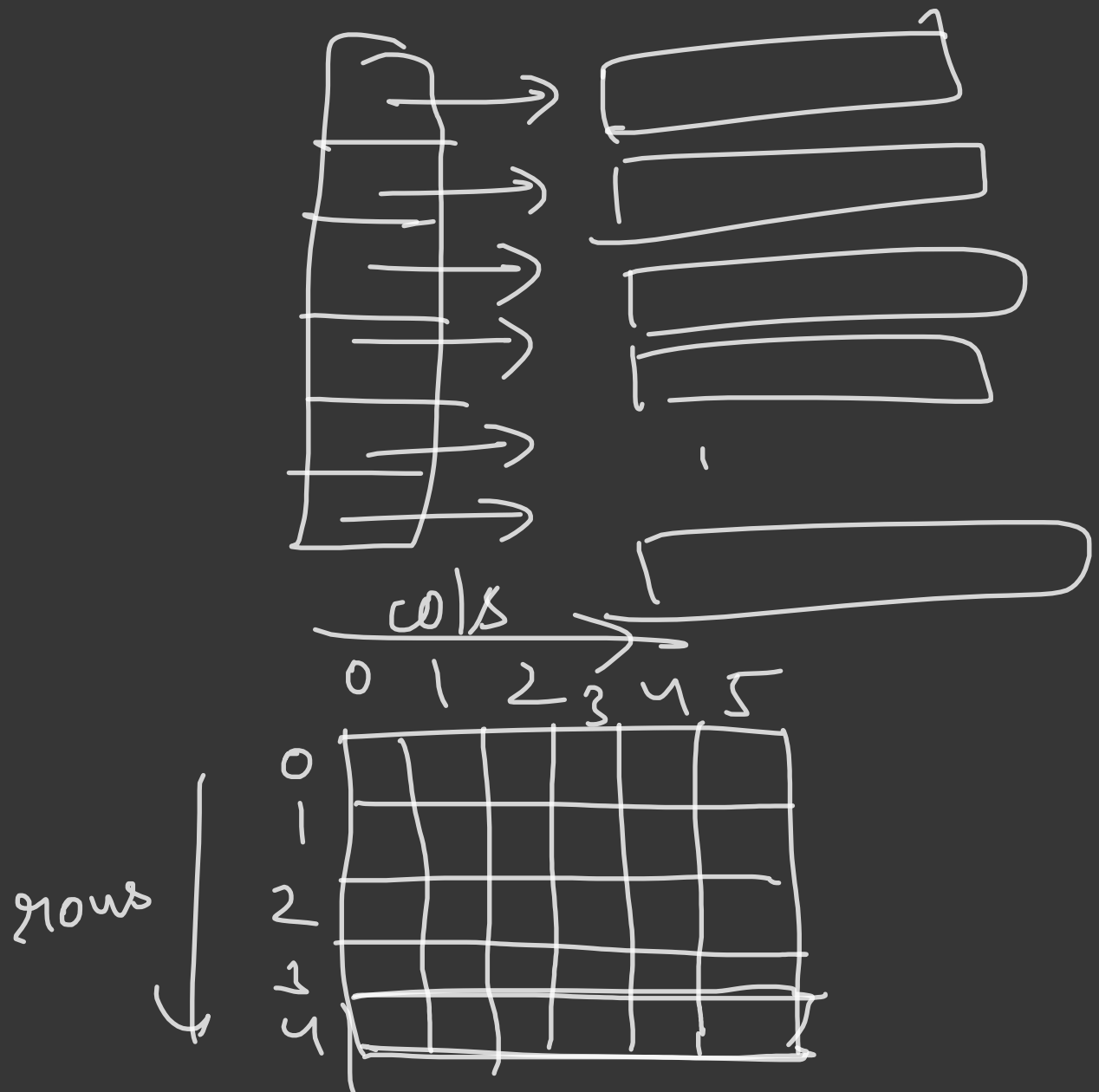
→ (vec.begin() + index)

2D vectors →

vector <int>



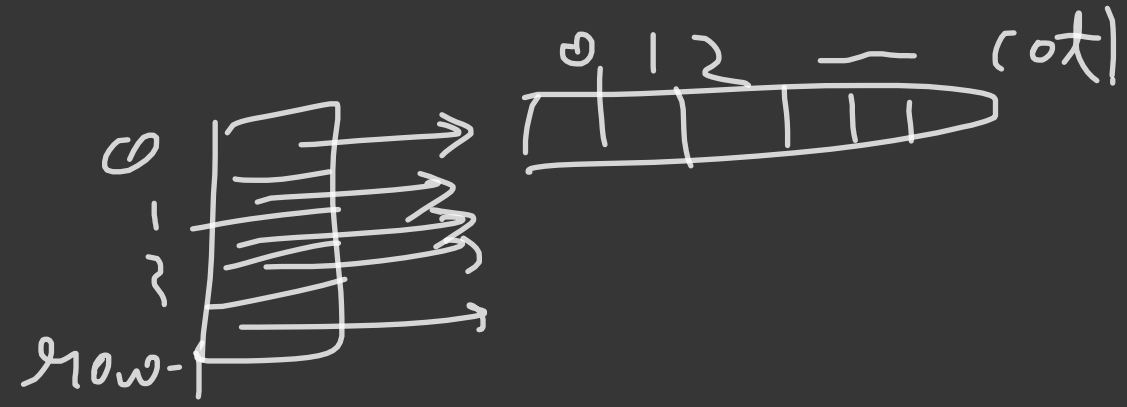
vector<vector<int>>



vector<vector<int>> mat;

mat.pb( )

rows → 5
cols → 10

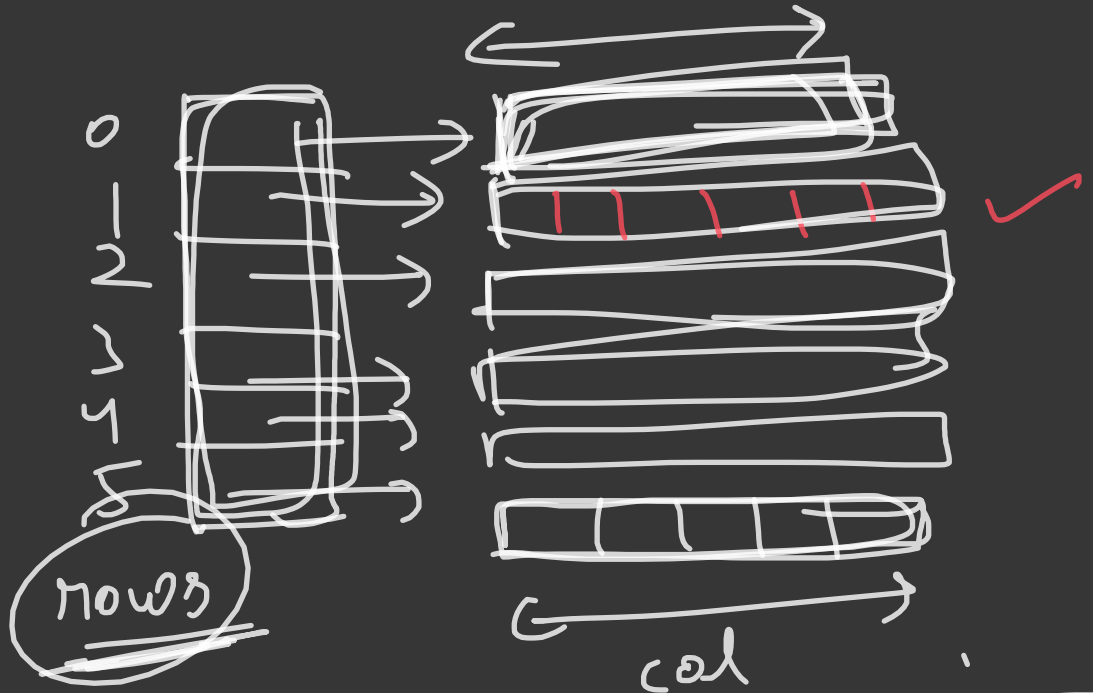


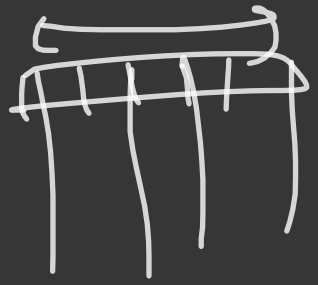
vector<vector<int>> mat(rows,

vector<int>(cols));

cols.

cols





vector<int> vec(n, 0);

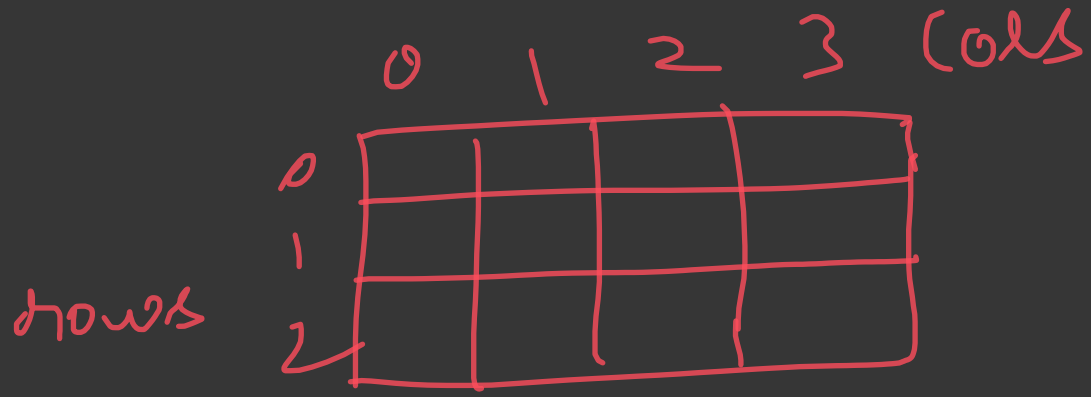
vector<vector<int>> mat(rows, vector<int>(col));

vector<int>

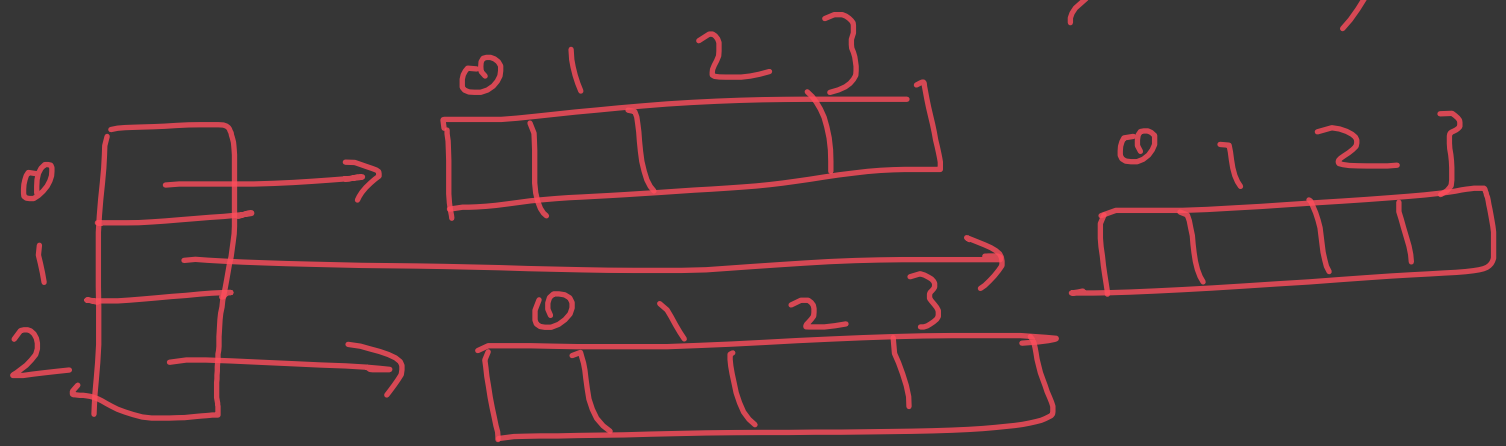
vector<int> vec(n,) → vector<int>(n, 0)

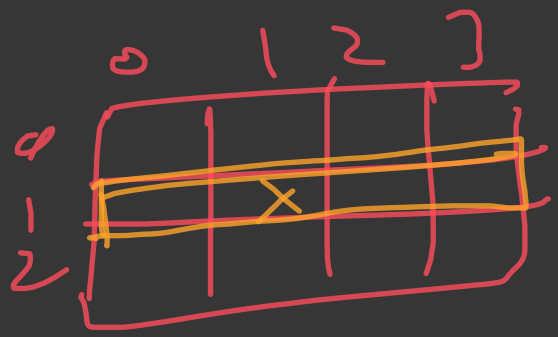
row → 3

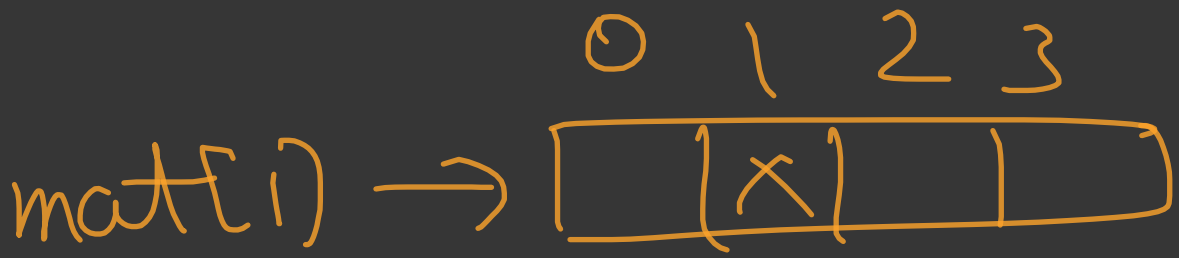
cols → 4

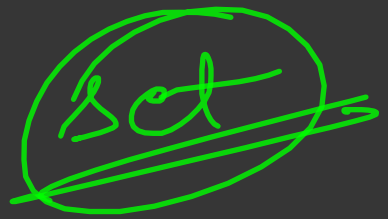


vector<vector<int>> mat(3, vector<int>(4));



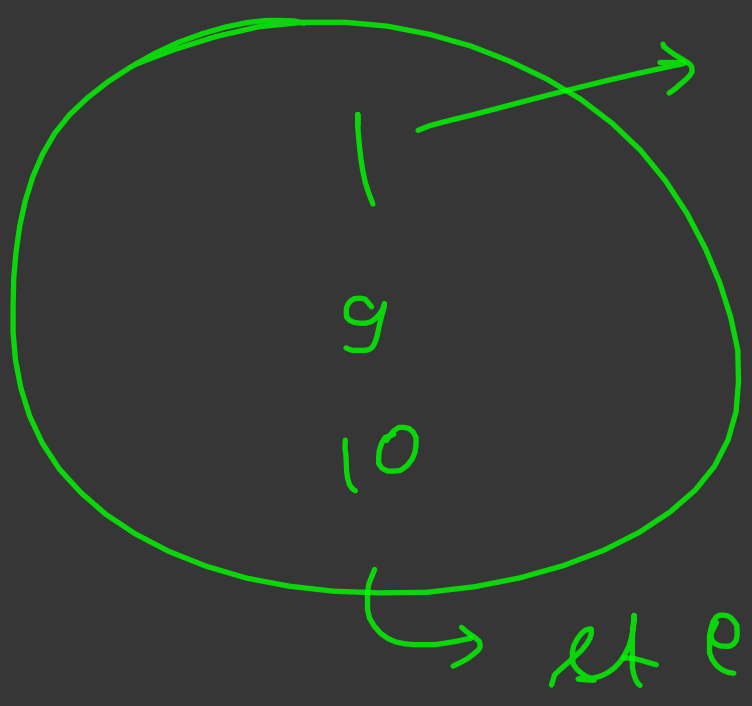






→ unique values ✓
→ sorted order ✓
→ O(log N)

set<int) st,




```
st.insert(1);
st.insert(1);
st.insert(10);
st.insert(9)
```

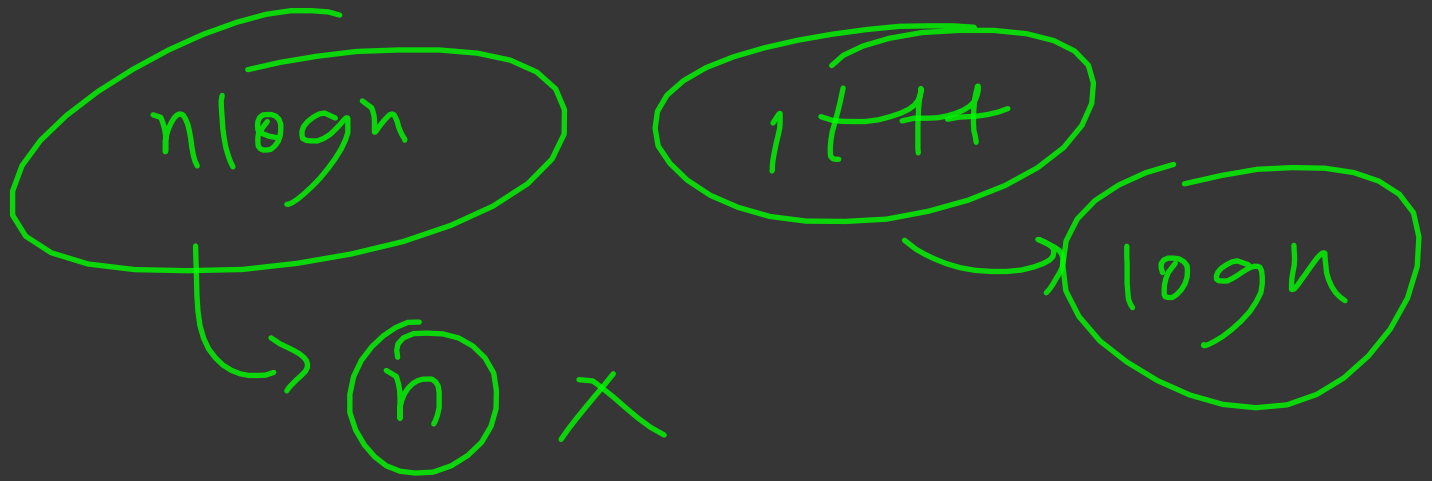
```
for (auto it = st.begin(); it != st.end(); it++){
```

cout << *it << " ";

}

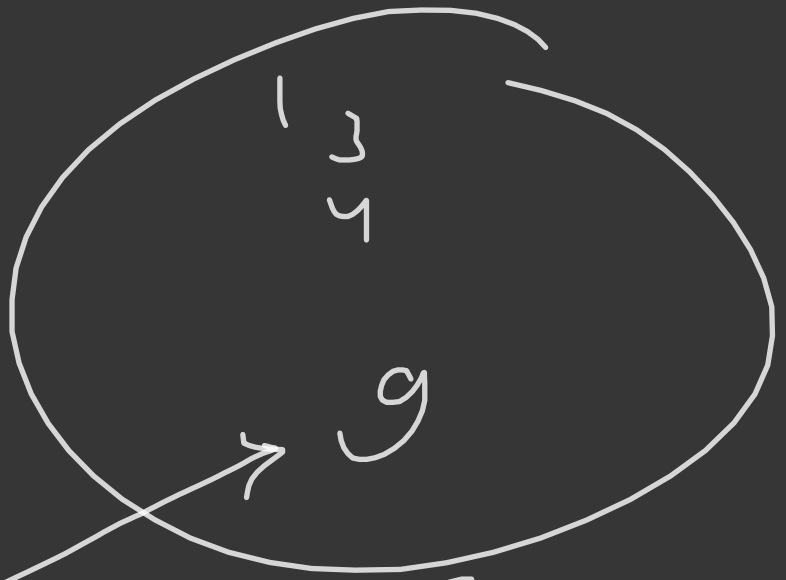
→ for each loop

```
for (auto it : st){
    cout << it << " ";
}
```



st.erase(7),

↳ O(logn)



it=st.find(9)

st.end()

*it

st.find(-7)

st erase (st.find (9))

unordered_set

→ unique values

→ Random order

→ O(1) → insert, find, delete

unordered_set<int> st;

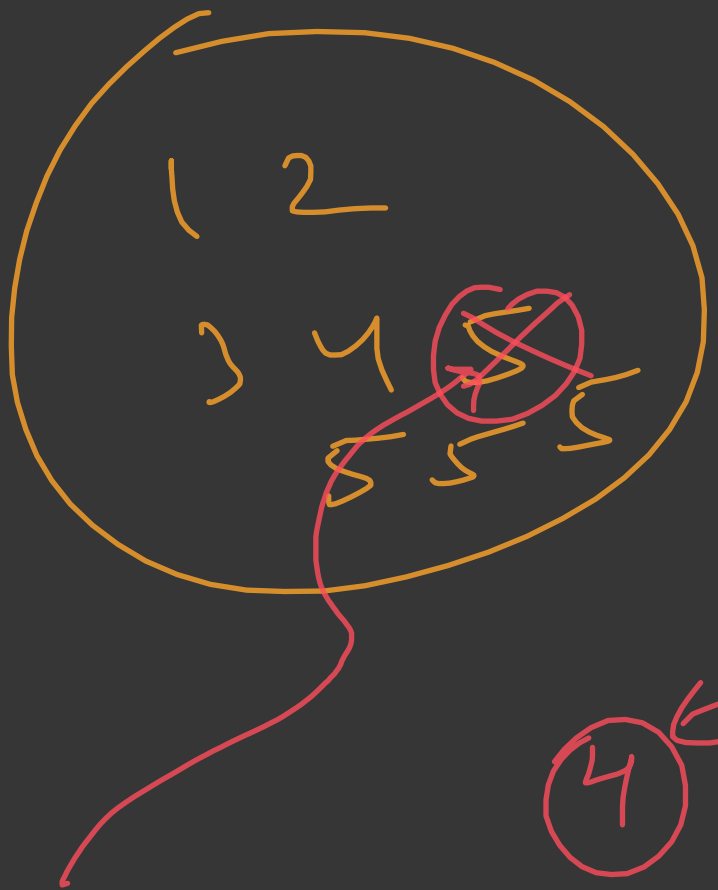
→ TC on unordered_set → set.

n → 10^5 log n → 20

multiset

multiset<int> st;

- sorted order
- O(log N)
- duplicate values



→ 1 2 3 4 5 5 5 5

st.count(5);

4

delete all 5 → st.erase(5),

delete only 1 (5) → st.erase (st.find(5))

st.clear() → O(n × log n) → multiset

→ O(n × O(1)) → no set

pair<int, int> p → { first , second }

p.first p.second

map → {key, value}

→ unique values of key

O(logN) → sorted order of key

map<int, int> mp;

mp.insert({1,2})

2

mp[1]=2;

1 → 2

mp

mp[1]

auto it = mp.find(1);

it → first

(*it).first

unordered_map <int, int> mp; → random order → unique key → O(1)

multimap <int, int> mp; → sorted → duplicate keys → O(log n)

queue → FIFO

queue<int> q;

q.push(3)
(1)
(2)
(6)

O(1)

q.top() = 3

O(1)

q.pop()

O(1)

stack

↳ LIFO

stack<int> st;

st.push(1)

(3)

(5)

(7)

O(1)

st top() } O(1)

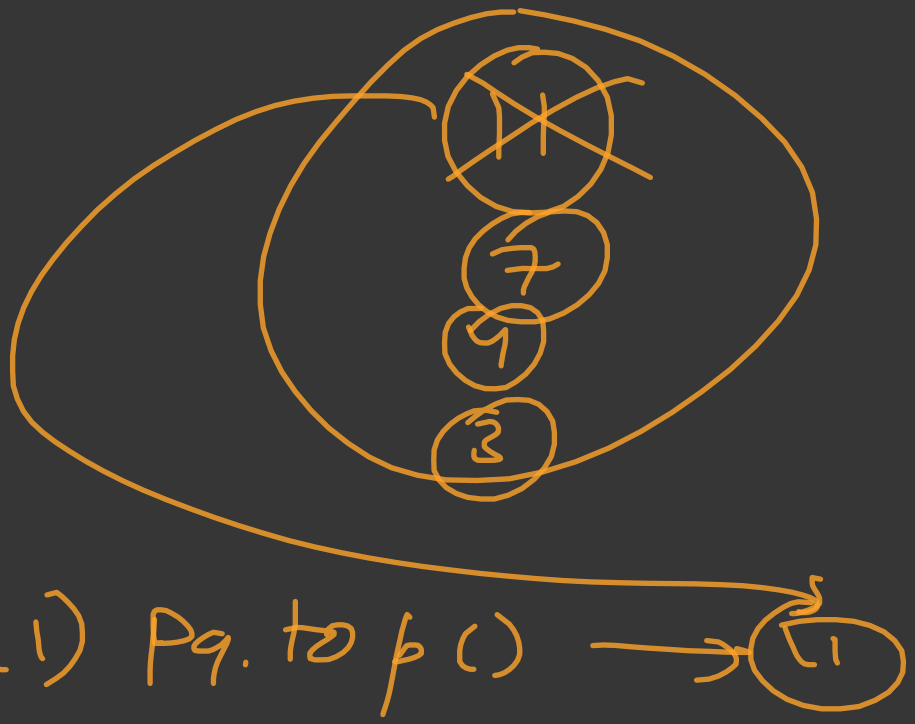
st.pop()

Priority - queue → by default → max given ↑ priority max pq

priority_queue<int> pq;

pq.push(3)
(4)
(7)
(11)

O(log n)



O(1) pq.top() → 11

O(log n) pq.pop()

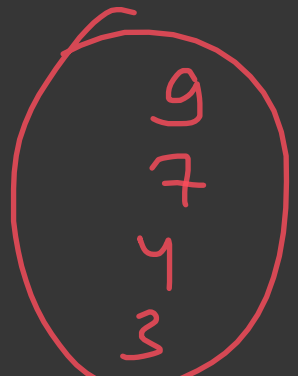
11	7	4	3

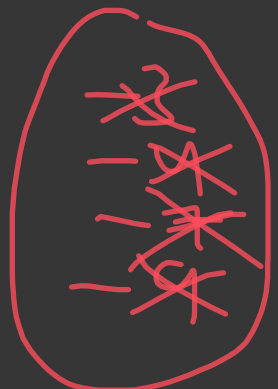
2 method

① priority_queue<int, vector<int>, greater<int>> pq;

② priority_queue<int> pq;

7
3
9
4





3 4 7 9

min heap

pq.push(-x);

int x = -pq.top();

pq.pop();

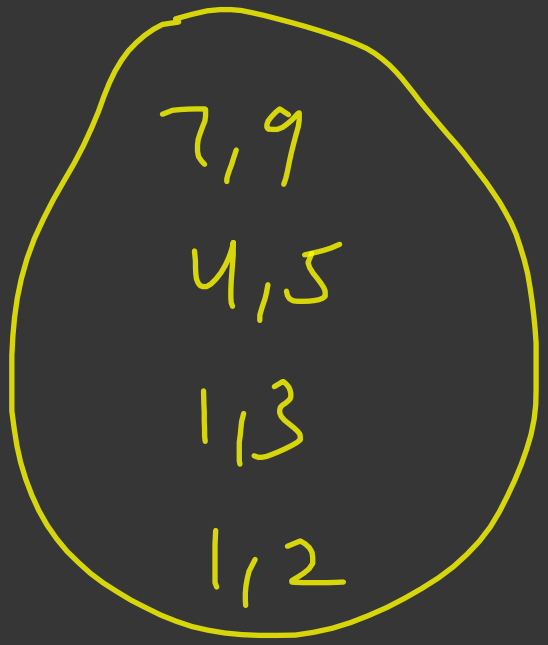
priority_queue< pair<int, int>> pq;

→ max pq

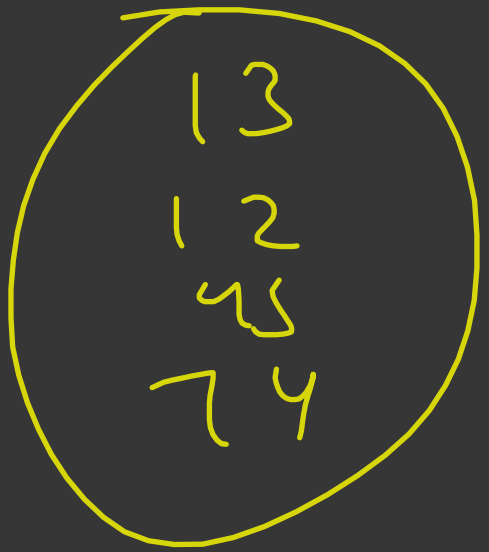
{1, 3}

{1, 2}

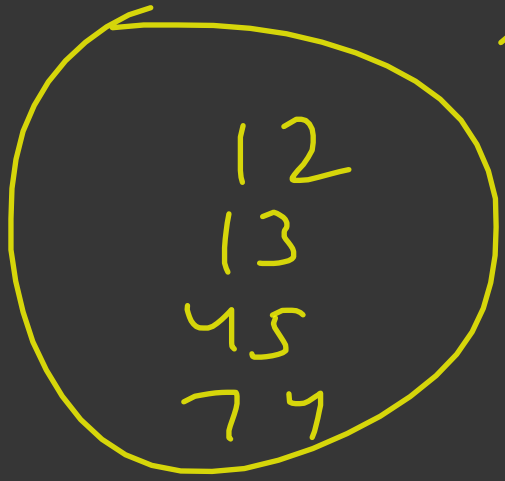
~~{4, 4}~~



-1 3
-1 2
-4 5
-7 4



negate both first & second



first	second
(-ve) inc	(-ve) inc
(-ve) inc	(+ve) dec
(+ve) dec	(+ve) dec
(+ve) dec	(-ve) inc

deque → doubly ended queue

```
deque<int> dq;

dq.push_front(1);
   push_front(2);
   push_back(3);
```

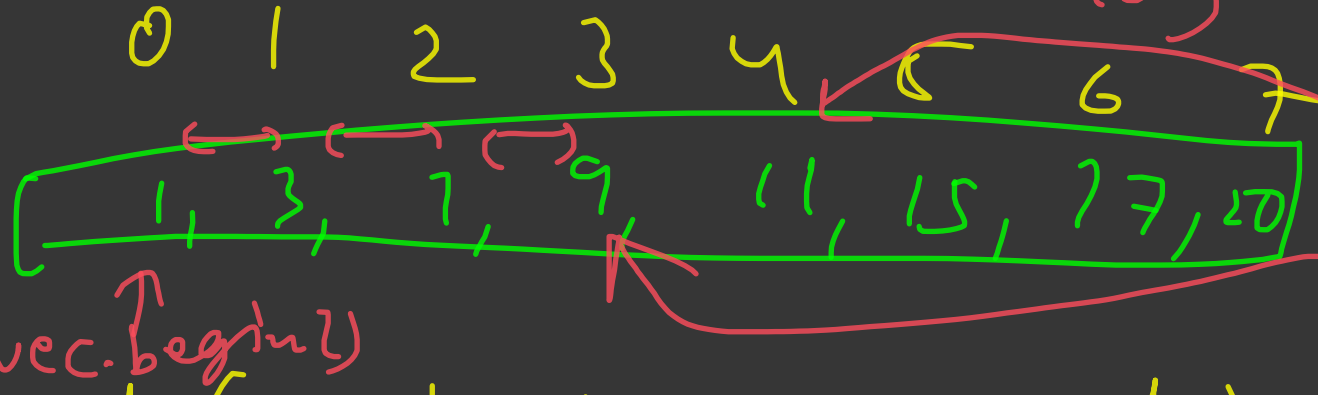
deque

pop_front()

POP_back()

⑧ Binary search:

search in sorted array/vector in log time



O(log n)

① binary_search(vec.begin(), vec.end(), x)

② lower_bound (vec.begin(), vec.end(), 9), → ≥ x

③ upper_bound (" " " , 9), → > x

8

int index = [redacted] - vec.begin();