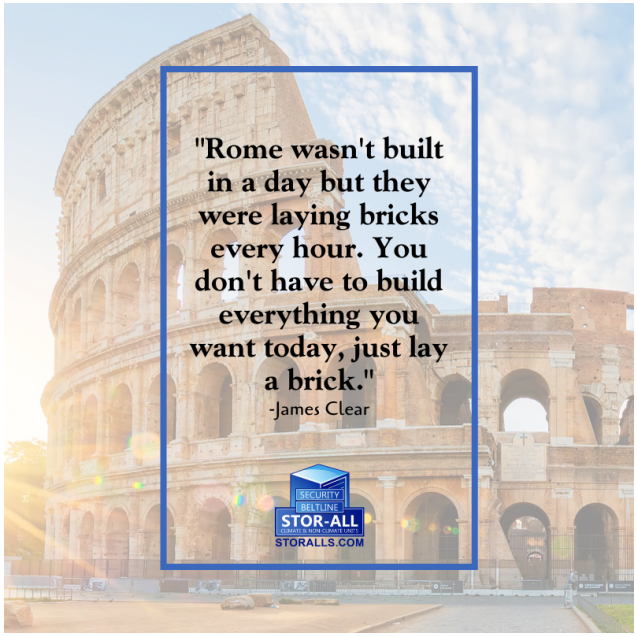
NOTES:

# HASHMAP

Good Evening



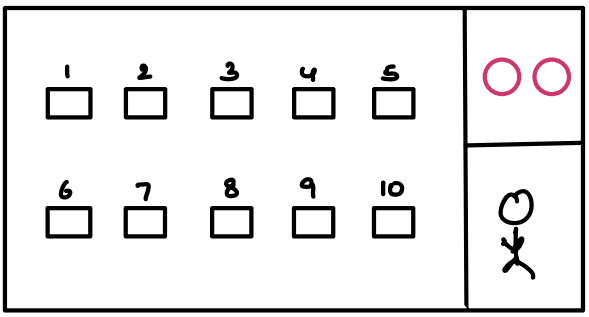

Note :- Concept will be correct

Syntax → look from internet

Today's Agenda

- → Hashmap Introduction
- → Hashset Introduction
- → Frequency of each query
- → First non repeating ele
- → distinct elements
- → Subarray with sum = 0 (Interview)

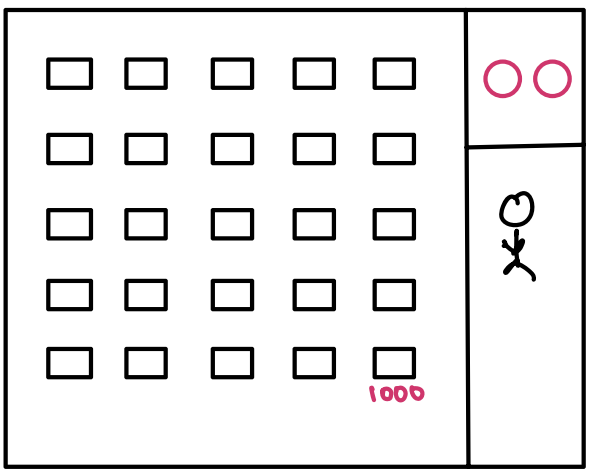
Yogi + Sachin




Register

1 → Occupied
2 → Not occupied
3
4
5 → Occupied
⋮
10 → Occupied

1000 rooms

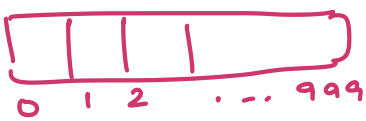


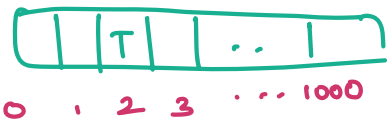

Room no. → {1 1000}

boolean array [ ] = 1001

01. 999 → 

X

02 1000 → 


03 1001 → 


Room 2 is occuppied → mark 2nd index as True

Room 5 is available? → If value is false, then its available

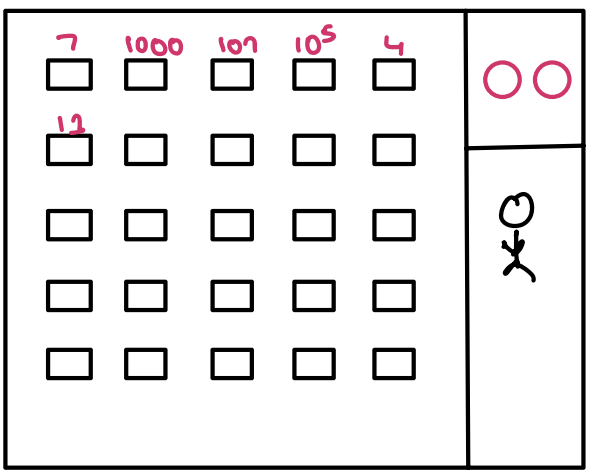
To check if particular room is available or not ?

TC : O(1)

* Covid → Business is going down

Numerologist → [1 – $10^9$]

1000 rooms




One on one mapping

boolean [] checkIn = $10^9+1$

Issues → For 1000 rooms

↳ $10^9+1$ memory spare

Advantage → Access the information → O(1) time

HashMap<Key, Value>

7 → False
1000 → False
107 → False
$10^5$ → True
5 → False

Size of HM = 1000

check(5)

↓

TC: O(1)

Searching in HM is always O(1) irrespective of its size

How the TC is O(1)
How do implement HashMap

} Advance content

* Obs → Keys are always unique

Value can be anything

01. Store population of every country

Key : County name {String}

Value : Population {Long}

HashMap<String, Long> hm;

02. No. of states of each country

Key : Country name {String}

Value : No. of states {Integer}

HashMap<String, Integer> hm;

03. Name of states of each country

Key : Country name → {String}

Value : All state names → List<String>

HashMap<String, List<String>> hm;

04. For every country, store population of each state

Key : Country name → {Strings}

Value : population of each state → HM<String, Long>

HashMap<String, HM<String, Long>> hm;

HashMap → To store key-value pairs

Hashset → To only store unique keys

| HashMap functionality {key, value} | Hashset functionality |
|---|---|
| 01. size() → {No. of keys in hashMap} | 01. size() → {No. of keys in hashset} |
| 02 insert (key, value) | 02. add(key) |
| 03. search (key) | 03. search(key) |
| 04. remove (key) | 04 remove(key) |
| 05. update → insert (key, value) | 05 update → No sense in hashset |
| 06. get (key) → Provide the value | 06 Iterate on hashset → to get all keys |
| 07. iterate on hm → to get all keys | |

A single operation in HashMap / Hashsed → O(1)

01 If we insert n {key, value}

TC: O(n)

SC: O(n)

01. India → 20 ✓

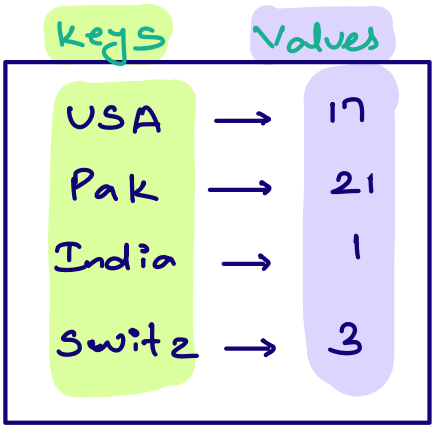
02 USA → 17

03. Pak → 21

04. Swit → 3

```
HashMap<String, Int> hm;
hm.insert (India, 20)
hm. insert (Pak, 21)
hm. insert (Swit, 3)
hm. insert (USA, 17)
hm. insert (India, 1)
```

HM




TC: O(n)
SC: O(n)
} To insert n entries

Q → **Find frequency of numbers**

Given N array elements & Q queries, find frequency of elements (given in queries) from the array.

ar[10] = {2 6 3 8 2 8 2 3 8 10}

Queries: 4

**Brute force** → For each query, iterate on array look for the query ele & get the count

TC: Q * N

SC: O(1)

02 Sorting the array

TC: O(nlogn + Q*n)

SC: O(1)

* Optimised Approach → Store the data in HM

Key → Distinct ele of array

Values → Frequency of every distinct ele

ar[10] = {2 6 3 8 2 8 2 3 8 10}

HM

| key (Distinct ele) | Value (Freq) |
|---|---|
| 2 | 3 |
| 6 | 1 |
| 3 | 2 |
| 8 | 3 |
| 10 | 1 |

Now, to answer a single query, O(1) time

Queries: 4

2 → 3

8 → 3

3 → 2

5 → 0

```
void printquery ( int [] ar, int [] Q )
{
    HashMap < Integer, Integer > hm;
    // Iterate on array & populate hm
    for( i=0; i<n; i++)
    {
        if ( hm.search (ar[i]) == true){
            int of = hm.get(ar[i]);
            int nf = of +1 ;
            hm.insert (ar[i], nf )
        }else {
            hm.insert (ar[i], 1);
        }
    }
    //Iterate on queries & answer using hashmap
    for (i=0; i<Q.length ; i++) {
        int ele = Q[i];
        if ( hm.search (ele) == true)
            print ( hm.get (ele));
        else
            print (0);
    }
}
```

O(n)

O(Q)

TC: O(N+Q)

SC: O(n)

02. Find the first non repeating element.

↳ first ele from start which is not repeating

arr = { 4 3 3 2 5 6 4 5 } Ans = 2

arr = { 2 6 8 4 7 2 9 } Ans = 6

arr = { 1, 2, 3, 1, 2, 5 } Ans = 3

arr = { 4, 3, 1, 3 } Ans = 4

BF → Choose each element & for that particular ele, iterate on complete array & check if it has duplicate or not

TC: $O(n^2)$

SC: $O(1)$

02. Sorting is not possible → Order matters

03. Using hashMap Key → Distinct ele

Value → Frequency

arr = { 1, 2, 3, 1, 2, 5 }

0 1 2 3 4 5

HashMap

Keys Values

01. Insert all ele in HM

5 ⟶ 1

1 ⟶ 2

2 ⟶ 2

3 ⟶ 1

Iterate on arr again & look for the first ele with val(freq) as 1

Code → {TODO}

TC: O(n)
SC: O(n)

03. Given an arr[N] ele, find no. of distinct elements.

arr[5] = {3 5 6 5 4} Ans = 4

arr[5] = {1 1 1 2 2} Ans = 2

arr[3] = {3 3 3} Ans = 1

Solution ⟶ Hashset

Note :- Inserting the same key in hashset is not going to affect the hashset.

Hashset will only contain 1 occurrence

```
Hashset < Integer > set ;

for ( i=0 ; i<n ; i++ ) {
    set . add ( ar[i] );
}

return set . size();
```

TC: O(n)
SC: O(n)

Q → Given an array, check if all elements are distinct or not.

ar[5] = { 6 8 3 2 7 } → True

```
Hashset < Integer > set ;

for ( i=0 ; i<n ; i++ ) {
    set . add ( ar[i] );
}

if (set . size() == ar. length) return True

else return false;
```

TC: O(n)
SC: O(n)

Q5. Given ar[N] elements, check if there exists a subarr with sum = 0

ar[10] =

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

Ans = True

Brute force → Consider every subarr & check if sum=0 or not

3 nested loops

TC → $O(n^3)$

SC → $O(n)$

Prefix sum

TC: $O(n^2)$

SC: $O(n)$

Carry forward

TC: $O(n^2)$

SC: $O(1)$

Optimisation → Prefix sum for subarr sum calculation

ar[10] =

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

Pf[10] =

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| 2 | 4 | 5 | 2 | 6 | 9 | 10 | 8 | 5 | 7 |

Pf[2] = sum[0-2] = 5

Pf[8] = sum[0-8] = 5

= sum[0-2] + sum[3-8] = 5

→ 5 + sum[3-8] = 5

sum[3-8] = 5 - 5 = 0

= x = * = x

Pf[0] = sum[0-0] = 2

Pf[3] = sum[0-3] = 2

= sum[0-0] + sum[1 3] = 2

2 + sum[1 3] = 2

sum[1 3] = 2-2 = 0

**Conclusion** → If pf array is containg same values then we have a subarray with sum = 0

| | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| arr = | 2 | -5 | 3 | 6 |
| Pf = | 2 | -3 | 0 | 6 |

Note :- If 0 is present in pf array, then there exist a subarr with sum = 0

**One way** → if pf[i] == 0 return true

**Other way** → Explicitly add a 0 by yourself

Find Idea → We will insert pf array elements in hashset & check if a value is already present in hashset or not.

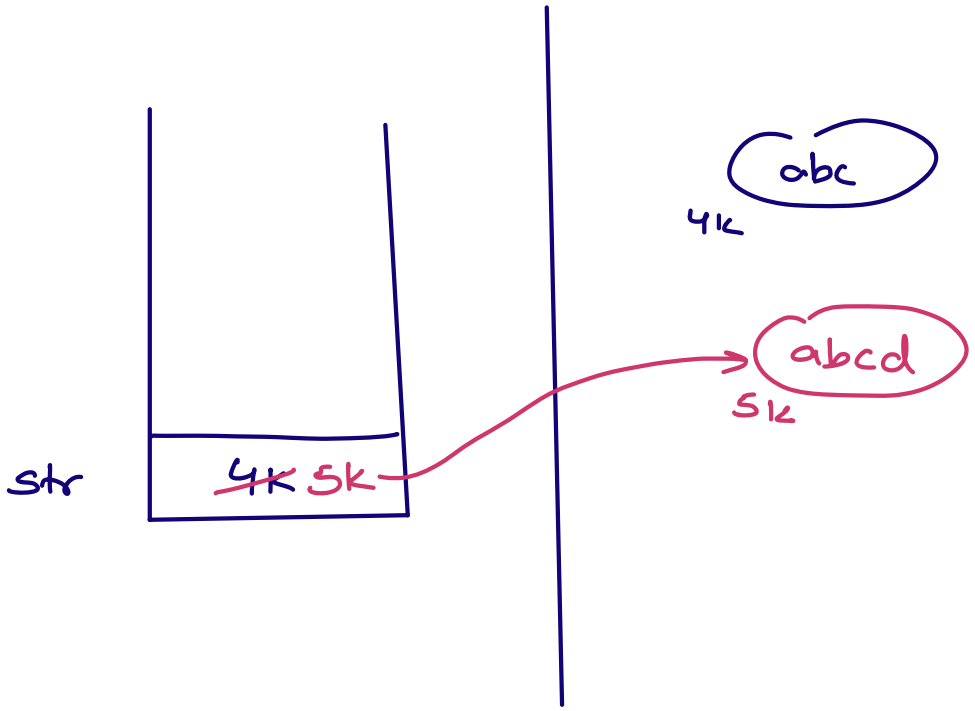
```
boolean subarrzero (int ar[])
{
    int pf[n];
    Hashset<Integer> set;
        set.add(0);
    for(i=0; i<n; i++){
        if (set.search(pf[i])==true) return true;
        else set.add(pf[i]);
    }
}
```

→ Can be optimsed by using carry forward

Strings in Java → Immutable

String str = "abc"

str = str + 'd' ⟶ abcd




Immutable strings in Java

long sum = 0

$$\left\{ \begin{array}{l} 1 \le len(arr) \le 10^5 \\ 1 \le ar[i] \le 10^9 \end{array} \right\}$$

min sum = $\boxed{1}$ = 1

max sum = $\frac{10^9}{0} \; \frac{10^9}{1} \; \frac{10^9}{2} \; \frac{10^9}{3} \; \cdots \; \frac{10^9}{10^5}$

$$= \underbrace{10^9 + 10^9 + 10^9 + \cdots + 10^9}_{10^5 \text{ times}} = \underline{\underline{10^{14}}}$$