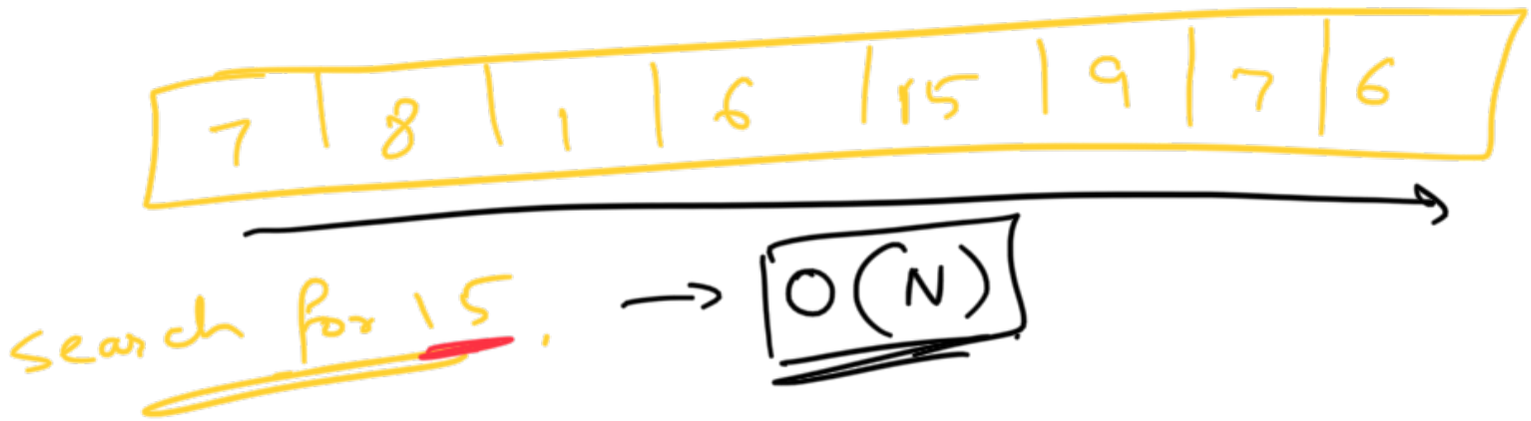
Hashing - 1



Set

↳ search/find → O(1)

Advanced

Hashcode

Self balancing B.S.T.

↳ ordered_set

Java / C#

Hashset

C++ (STL)

unordered_set

Python / Ruby / JS

Set

C

✗

set → get / find (x)

↳ add / insert (x)

↳ size()

↳ delete / erase (x)

} O(1)

A : [1, 2, 1, 2, 1, 2, 3, 3, 4, 1]

Set only stores unique elements

Q.1 Given an array, count no. of distinct elements in the array.

eg. A = [7, 3, 2, 1, 3, 7, 5] ⇒ Ans = 5

```
{   set S = {}
    for (i = 0 → n-1)
    {
        S.insert(A[i])} → O(1)

    }
return S.size()
}
```

T.C ⇒ O(N)

S.C (extra) ⇒ O(N)

Q. Given an array & Q queries.
In every query, return the freq of an element.

A: [7, 2, 7, 1, 2, 8, 1, 0, 8]

Q → 7 ⇒ 2 ⇒ O(N)
→ 2 ⇒ O(N)

} T.C ⇒ O(QN)

1 ⇒ 2 ⇒ O(N)

0 ⇒ 1 ⇒ O(N)

7 ⇒ 2 ⇒ O(N)

⋮ ⋮

Q. Given an array of strings.
Q queries having a string in each query.
Return the sum of ascii values of the char.

N → strings
M → length of every string

A : [abc, bcd, a, e, c]

Q ⇒ bcd ⇒ 98+99+100 ⇒ 297 ⇒ O(M)

a ⇒ 97 ⇒ O(M)

e ⇒ 101 ⇒ O(M)

bcd ⇒ 98+99+100 ⇒ 297

⋮

O(QM)

abc ⇒ 294

(Q >> N)

Map

Java	C++	Python	Ruby/Js	C
HashMap	unordered_map	dict	map	✗

Map < Key , value >
↑ unique

set < Key >
↑ unique

Q. store countries → Capital

Map < string , string >

Q. country → population

Map < string , Long >

Q. country → all states of the country.

Map < string , Array [string] >

set <string>

vector < string >

Q. Given country name & a state name | Return population of that state of that country.

Map < string , map < string , long > >

string → country name

string → state

long → population

country-name state

Q. store marks scored by student

Map < ~~string~~ , Double >

name marks

↳ Roll number

Keys in Hashmap are always unique

Hashmap

- get / find (K) //Returns value associated with 'K'.
- add / insert (K, V) // adds <K,V> to map
- size() // returns size

O(1)

- delete / erase (K) // removes the value.
- update (K, V) // updates value for the 'K'.
- isPresent (K) / ContainsKey (K) // checks if K is present as key.

A: [abc , bcd , acd]

N = 3

M = 3

Q >> N

Q: [abc] → 1 → 97+98+99 → 294

[bcd] → 1 → 98+99+100 → 297

[abc] → 97+98+99 → 294

[abc] → 97+98+99 → 294

Q = 100

3 * 100

Q . M

100

map[abc] =! 294

Map < string , int >

map [abc] = 97+98+99 = 294

map [bcd] = ... = 297

map [acd] = ... = _

9 + 100

(N*M) + Q

set/Map

S.C ⇒ O(keys)

S.C ⇒ O(N)

T.C ⇒ O(NM+Q)

Break : 10:10

A : [ABF , BCD]

map [ABF] ⇒ 300

map [BCD] ⇒ 300

Q → ABF

↳ BCD

T.A :

Amazon

Q. Given an array of size N.
Find the first non-repeating element in the array.

A: 8, 2, 8, 3, 1, 2, 6, 5, 8

Ans ⇒ 3

Quiz: A: [1, 2, 3, 1, 2, 5] ⇒

Ans ⇒ 3.

Brute: 2 for loops ⇒ $O(N^2)$.

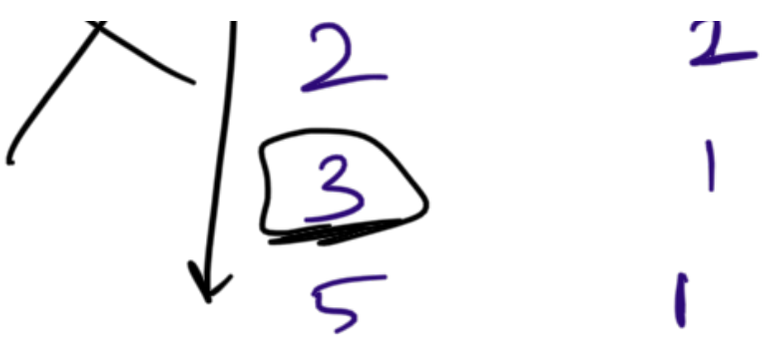
Steps:

① Build hashmap to store the freq of every element

Map

Key	value
1	2

② Iterate over the array & return the first element with freq = 1



A : 1, 2, 3, 1, 2, 5

HashMap < Int , Int > map ;

A[i] freq

map

Key	value
<1	2 >
<2	2 >
<3	1 >
<5	1 >

```
for ( i=0 ; i<N ; i++)
{
  // if A[i] is present in map
  if (map.containsKey(A[i]))
  (
      map.put(A[i], map.get(A[i]) + 1)
  )
  else // A[i] is not present in map
  {
      map.put(A[i], 1)
```

→ map[A[i]]++;

O(N)

```
    }
}
```

T.C ⇒ O(N)

S.C ⇒ O(N) (extra)

```
// find first non-repeating element!
for( i=0 ; i<N ; i++)
    if( map.get(A[i]) == 1)
        return A[i];
```

O(N)

Amazon
M.S.
Google

Q. Given an array of size N.
Return true if there exists a subarray with sum = 0.

A : [2, 2, 1, -3, 4, 3, 1, -2, -3]

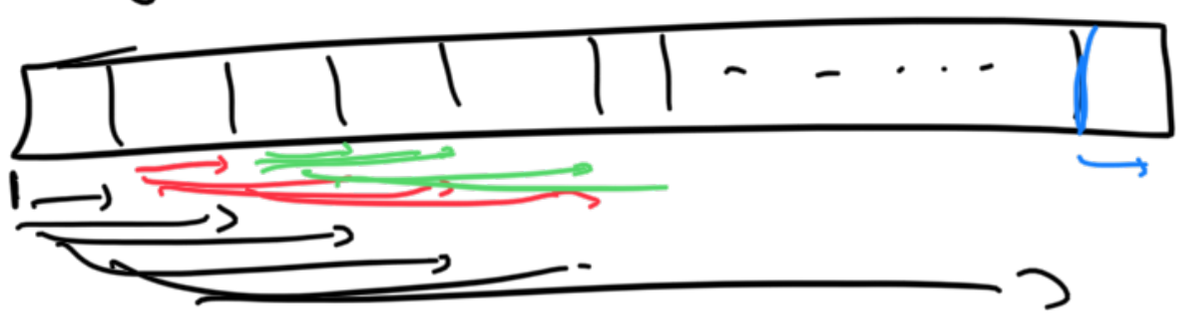
Ans - True

No. of subarrays in array of size N

$= N + (N-1) + (N-2) + \ldots + 1$

$= \frac{N(N+1)}{2}$

$= \boxed{O(N^2)}$



Approach ①

```
for( i=0 ; i<N ; i++)
{
    for( j=i ; j<N ; j++)
    {
        sum=0
        for( k=i ; k<=j ; k++)
        {
            sum += A[k]
        }
        if(sum==0)
            return true
    }
}
```

sum $= (ps[j] - ps[i-1])$

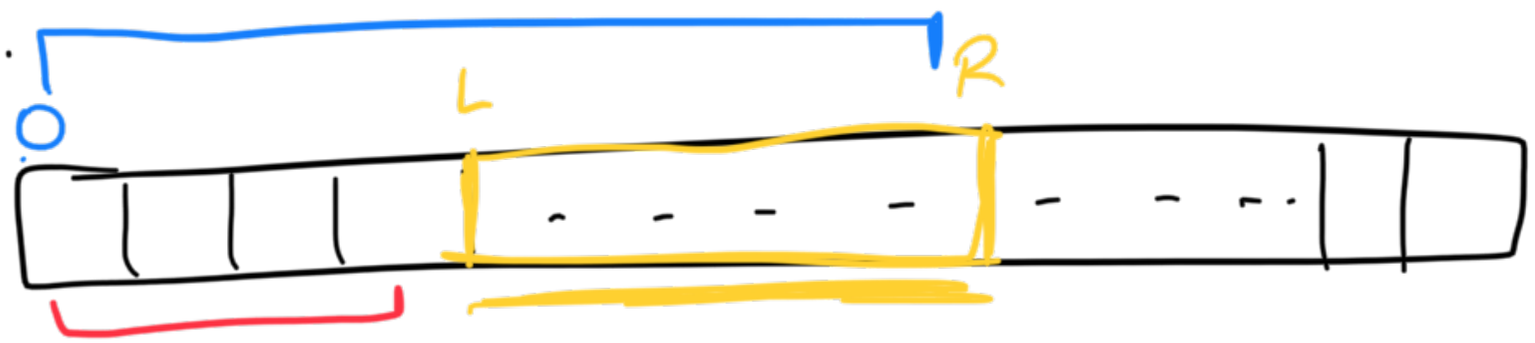
$O(N^3)$ → use P.S. → $O(N^2)$

Constraint $N \leq 10^6$

return false

N → 10^5

Approach (2)

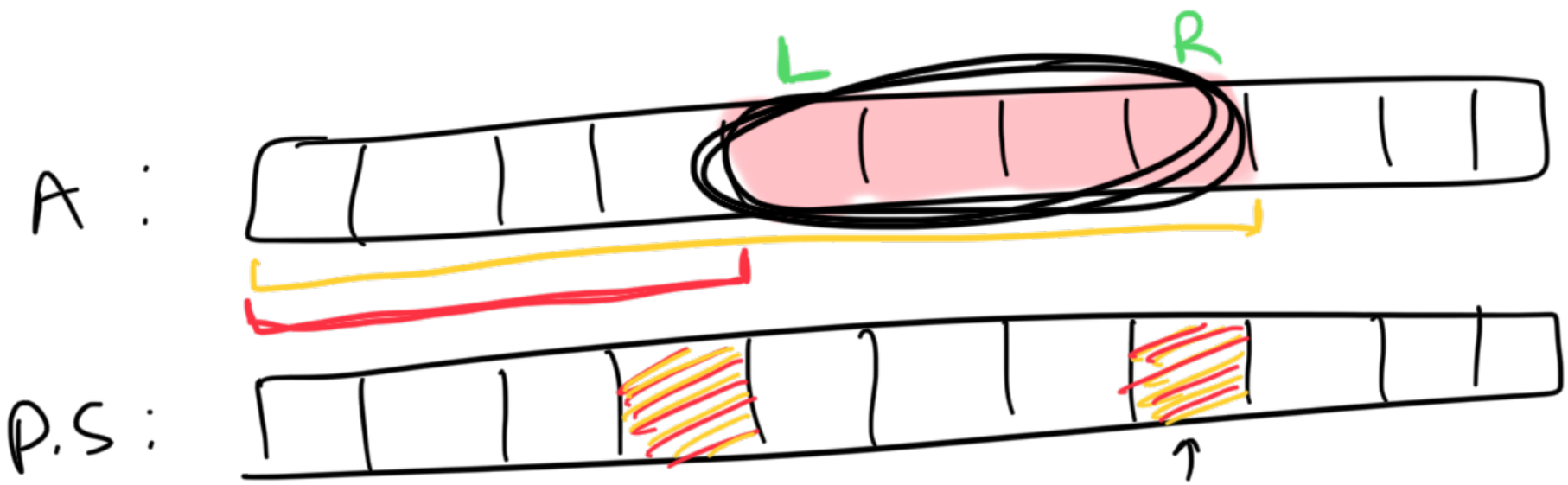


$$sum[L \to R] = PS[R] - PS[L-1]$$

if sum(L, R) is 0 (Assume)

$$0 = PS[R] - PS[L-1]$$

$$PS[R] = PS[L-1]$$



$$Sum(0 \to R) = sum(0, L-1) + \cancel{sum(L, R)}$$

if $sum(L, R) = 0$

$$\underline{Sum(0 \to R) = sum(0, L-1)}$$

$$\boxed{PS[R] = PS[L-1]}$$

A : [2, 2, 1, -3, 4, 3, 1, -2, -3]

P.S : [2, 4, 5, 2, 6, 9, 10, 8, 5]

2 4, 5

If any two values repeat in P.S. array

Ans is True

H.W

Calculate how many subarray exist with sum=0

① Create P.S. array.

② Use set to iterate & find duplicate

T.C ⇒ O(N)

S.C ⇒ O(N) (extra)

implement !

Hashing 2

Recorded

20 read → code