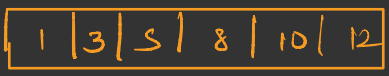
Hashing



Arrays → adding O(1)

→ removing O(n)

→ searching O(n)

Hashing → mapping large amount of data in smaller table using hash function

→ addition, removal and searching can be done in constant time

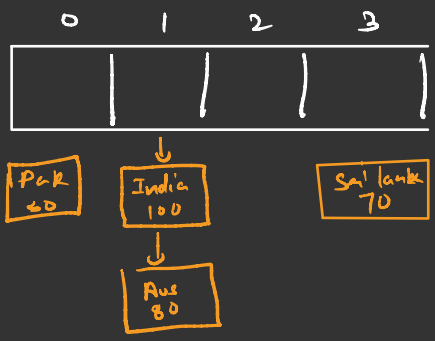
→ Hash Map

Key	Value
India	100
Pak	50
Srilanka	70
Aus	80

→ put
→ remove
→ get
→ containsKey
→ size

} O(1)

```
hm.put("Aus", 80);
hm.put("India", 100);
```



key → Hash Function → Random +ve integer

→ generates uniformly

→ generate index for the given array

n → count of all the elements in HM

N → size of the array

$\frac{n}{N} = \lambda$ (load factor) $\leq$ k

k → Constant Integer

but → Same for all the methods

get index	O(1)
search λ elements	O(λ)

update if present else add at the end → O(1)

TC → O(λ)

N = 4

n = ~~4~~ ~~5~~ ~~6~~ 9

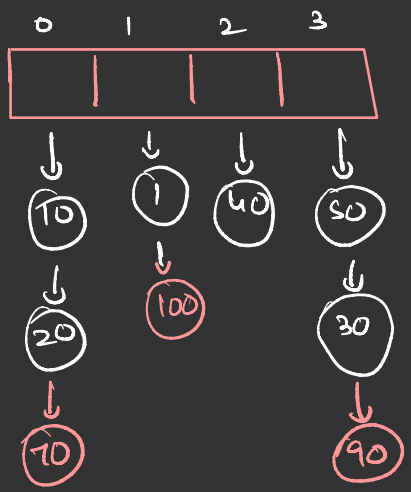
n = ~~5~~ ~~7~~ 7

$\frac{n}{N} = \frac{7}{4} = 2.2$

hm.put(100, 100)

hm.put(70, 60)

hm.put(90, 70)



100 → 1

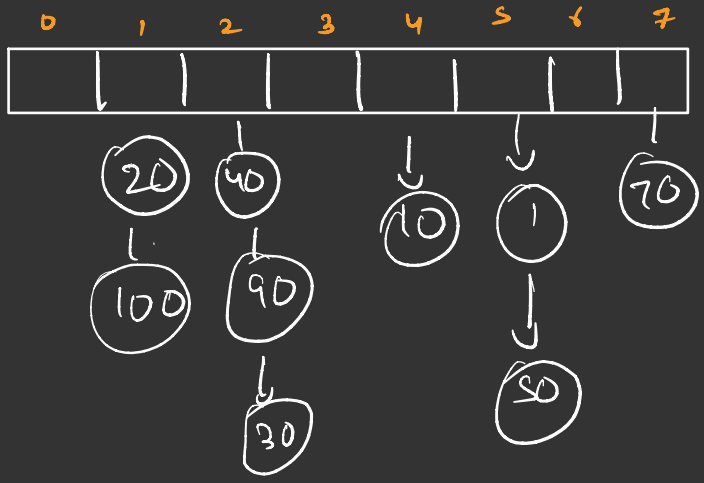
70 → 0

90 → 3

k = 2

my Load factor > k

perform rehashing



Ques Length of the longest subarray with sum 0

15	-2	2	-8	1	7	10	23

Brute Force → Find all subarray — $O(n^2)$

Calculate Sum — $O(n)$

TC → $O(n^2)$

0	1	2	3	4	5	6	7
15	-2	2	-8	1	7	10	23

psum= ~~0~~ ~~15~~ ~~13~~ ~~15~~ ~~7~~ ~~8~~ ~~15~~ ~~25~~ 48

ans = ~~0~~ ~~5~~

TC → O(n)

SC → O(n)

psum	Index
0	-1
15	0
13	1
7	3
8	4
25	6
48	7

```
int maxLen(int arr[], int n)
{
  HashMap<Integer, Integer> map = new HashMap<>();
  map.put(0 , -1);
  int psum = 0, ans = 0;

  for(int i = 0; i < arr.length; i++) {
      psum += arr[i];
      if(map.containsKey(psum)) {
          ans = Math.max(ans, i - map.get(psum));
      }else {
          map.put(psum, i);
      }
  }
  return ans;
}
```