**Approach 1:**

1) The idea is to choose a datastructure that can provide all the operations in O(1) average time.

We start with a list:

Insertion - O(1) : append at last

Deletion - O(n) : find index and then delete

Return random element - O(1) : return any index.

The only problem is deletion.

But what if we look at the problem differently.

When we have to delete the last element, we just pop it -> O(1) operation

So why not apply it here. Move the element to the last and pop it out?

**Problem:**

Find index of element to be deleted which in general will take O(n) operation, can be reduced by use of hashMap that stores the indices of each element.

**Algorithm**

1)Let the class members be a list: to store elements and a hashMap:to store indices of each element

2) InsertOperation:

* Create an entry in map if not already present, add index
* Add val to array
* Return

3)RemoveOperation

* If val not in map: return False
* Get one of the index of val from the map and pop it from map because we will remove ele at this index
* Copy the element at last at this index
* From the map, for the last element remove the entry of last index and instead add a new index which is the index where we remove ‘val’

4)RandomOp

* Return an element from the array/list

**CODE:**

class RandomizedCollection:

def \_\_init\_\_(self):

self.map = defaultdict(set)

self.arr = []

def insert(self, val: int) -> bool:

self.map[val].add(len(self.arr))

self.arr.append(val)

return len(self.map[val]) == 1

def remove(self, val: int) -> bool:

if not self.map[val]: return False

# get index of element

index = self.map[val].pop()

last\_val = self.arr[-1]

# copy

self.arr[index] = last\_val

# update map

self.map[last\_val].add(index)

self.map[last\_val].discard(len(self.arr)-1)

# delete/remove/pop

self.arr.pop()

return True

def getRandom(self) -> int:

return choice(self.arr)

Time Complexity => O(n)

Space COmplexity => O(n)