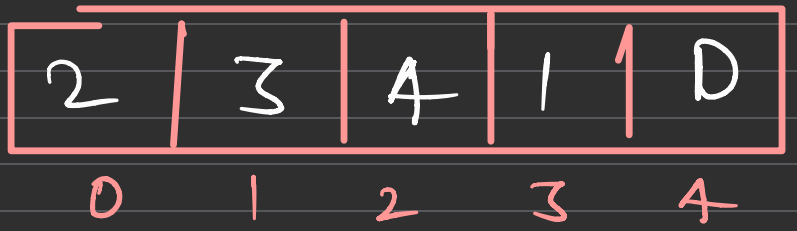
Rich people have small TVs and big libraries, and poor people have small libraries and big TVs.

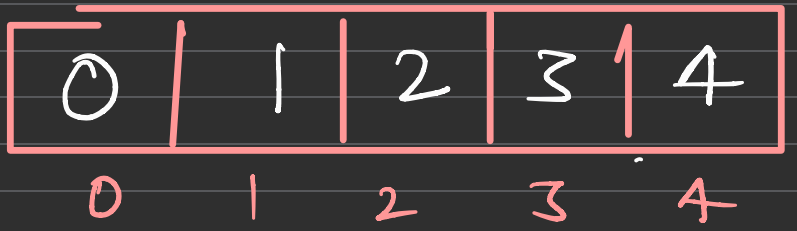
Why we need Hashing

arr



O(N)

arr



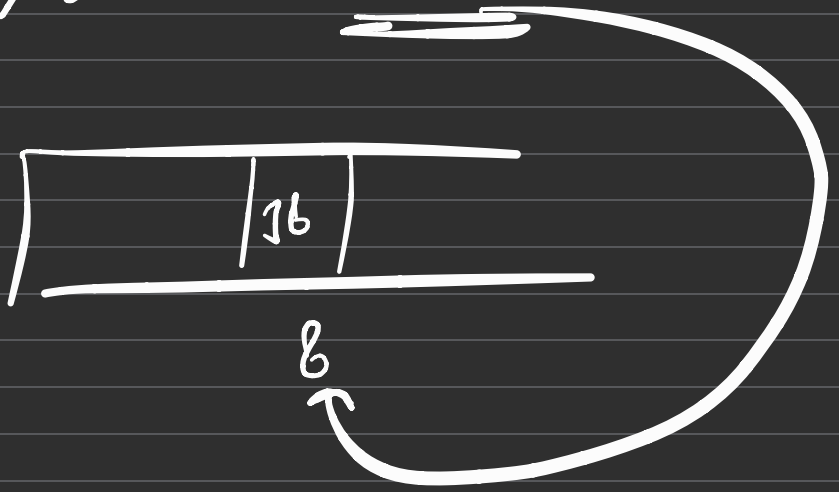
O(1)

70.000 , 80.000, 4000, 70, 60

7777777788 88833321 21

x = 38 → 38 index

index = 38 % 10 = 8



What is Hashing

- Hashing is a technique that efficiently stores and retrieves data in a way that allows for quick access.
- We can perform the operations like, Insert, Delete, Update in O(1) Time
- In hashing, large number is converted into smaller number, and that smaller number is used as index to store the large number.
- This can be done, using the hash function

$$h(x) = x \bmod N$$

X : Key which need to inserted into the table.

N: Size of the table.

Keys: 3, 9, 14, 25, 36, 2, 1, 47, 18, 20

$h(x) = x \% N \quad (N = 10)$

$3 \% 10 = 3$

$9 \% 10 = 9$

$14 \% 10 = 4$

$25 \% 10 = 5$

$36 \% 10 = 6$

$2 \% 10 = 2$

$1 \% 10 =$

$47 \% 10 = 7$

$18 \% 10 = 8$

$20 \% 10 = 0$

Hash Table

Key : 3 9 14 25 36 2 74 46 18 20

$h(x) = x \% N$ $\quad N = 10$

$3 \% 10 = 3$

$9 \% 10 = 9$

$14 \% 10 = 4$

$25 \% 10 = 5$

$36 \% 10 = 6$

$2 \% 10 = 2$

$74 \% 10 = 4$

$46 \% 10 = 6$

$18 \% 10 = 8$

$20 \% 10 = 0$

Collision

Hash Table

9	9
8	18
7	
6	36 (46)
5	25
4	14 (74)
3	3
2	2
1	
0	20

Collisions Techniques

1. Separate chaining
2. Open Addressing
   1. linear probing
   2. Quadratic probing
   3. Double Hashing

Keys: 3, 9, 14, 25, 36, 2, 74, 46, 18, 20

$h(x) = x \% N$ $\quad N = 10$

$3 \% 10 = 3$

$9 \% 10 = 9$

$14 \% 10 = 4$

$25 \% 10 = 5$

$36 \% 10 = 6$

$2 \% 10 = 2$

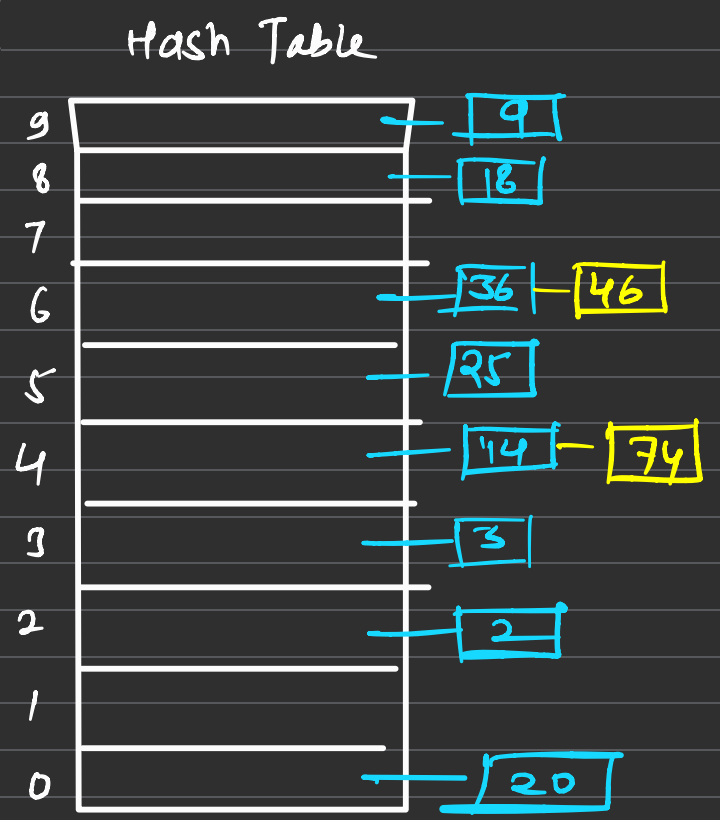
$74 \% 10 = 4$

$46 \% 10 = 6$

$18 \% 10 = 8$

$20 \% 10 = 0$

Separate chaining



Keys: 3, 9, 14, 25, 36, 2, 74, 46, 18, 20

$$h(x) = (x + i) \% N$$

$N = 10$

$(i = 0 \ldots\ldots N)$

$(3+0) \% 10 = 3$

$(9+0) \% 10 = 9$

$(14+0) \% 10 = 4$

$(25+0) \% 10 = 5$

$(36+0) \% 10 = 6$

$(2+0) \% 10 = 2$

$(74+0) \% 10 = 4$

$(74+1) \% 10 = 5$

$(74+2) \% 10 = 6$

$(74+3) \% 10 = 7$

$(46+0) \% 10 = 6$

$(46+1) \% 10 = 7$

$(46+2) \% 10 = 8$

$(18+0) \% 10 = 8$

$(18+1) \% 10 = 9$

$(18+2) \% 10 = 0$

$(20+0) \% 10 = 0$

$(20+1) \% 10 = 1$

linear probing

Hash Table

9	9
8	46
7	74
6	36
5	25
4	14
3	3
2	2
1	20
0	18

Keys: 3, 9, 14, 25, 36, 2, 74, 46, 18, 20

$h(x) = (x + i^2) \% N$

$N = 10$

$(14 + 0^2) \% 10 = 4$

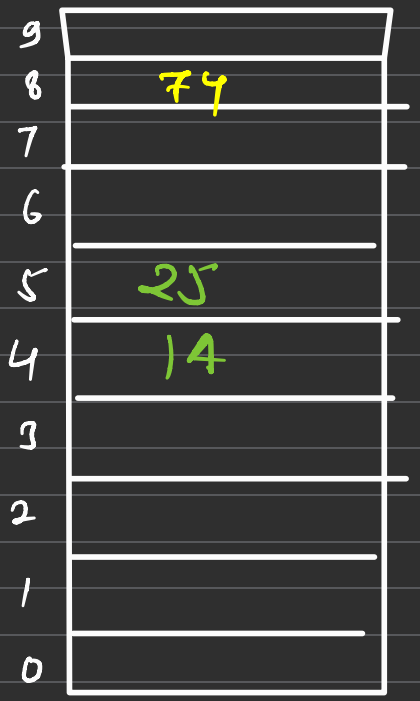
$(74 + 0^2) \% 10 = 4$

$(74 + 1^2) \% 10 = 5$

$(74 + 2^2) \% 10 = 8$

Quadratic probing

Hash Table



Count Occurrence of elements

7 → 3
1 → 2
3 → 1
4 → 2
6 → 1

mp[a[i]]++

mp.put(a[i], mp.getOrDefault(a[i], 0)+1)

3005. Count Elements With Maximum Frequency

Solved

Easy | Topics | Companies | Hint | Re-do

You are given an array `nums` consisting of **positive** integers.

Return *the **total frequencies** of elements in* `nums` *such that those elements all have the **maximum** frequency*.

The **frequency** of an element is the number of occurrences of that element in the array.

Example 1:

```
Input: nums = [1,2,2,3,1,4]
Output: 4
Explanation: The elements 1 and 2 have a frequency of 2 which is the maximum frequency in the array.
So the number of elements in the array with maximum frequency is 4.
```

Example 2:

```
Input: nums = [1,2,3,4,5]
Output: 5
Explanation: All elements of the array have a frequency of 1 which is the maximum.
So the number of elements in the array with maximum frequency is 5.
```

① find freq. of all elements

② find max. frequency

③ check freq matching max & add those

```
map<int,int> mp
mp.put(a[i], mp.getOrDefault(a[i], 0)+1)

for (auto it : arr)
    mp[it]++

maxf = 0
for (auto it : mp)
{   if (it.second > maxf)
        maxf = it.second
}

sum = 0
for (auto it : mp)
{   if (it.second == maxf)
        sum += it.second
```

Given an array **arr[]** of positive integers. Find the number of pairs of integers whose absolute difference equals to a given number **k**.
Note: (a, b) and (b, a) are considered the same. Also, the same numbers at different indices are considered different.

The answer is guaranteed to fit in a 32-bit integer.

Examples:

Input: arr[] = [1, 4, 1, 4, 5], k = 3
Output: 4
Explanation: There are 4 pairs with absolute difference 3, the pairs are {1, 4}, {1, 4}, {4, 1} and {1, 4}.

Input: arr[] = [8, 16, 12, 16, 4, 0], k = 4
Output: 5
Explanation: There are 5 pairs with absolute difference 4, the pairs are {8, 12}, {8, 4}, {16, 12}, {12, 16}, {4, 0}.

Intersection of Two arrays with Duplicate Elements

Difficulty: **Easy** Accuracy: **61.4%** Submissions: **33K+** Points: **2** Average Time: **20m**

Given two integer arrays **a[]** and **b[]**, you have to find the **intersection** of the two arrays. Intersection of two arrays is said to be elements that are common in both arrays. The intersection should not have duplicate elements and the result should contain items in any order.

Note: The driver code will sort the resulting array in increasing order before printing.

Examples:

Input: a[] = [1, 2, 1, 3, 1], b[] = [3, 1, 3, 4, 1]
Output: [1, 3]
Explanation: 1 and 3 are the only common elements and we need to print only one occurrence of common elements.

Input: a[] = [1, 1, 1], b[] = [1, 1, 1, 1, 1]
Output: [1]
Explanation: 1 is the only common element present in both the arrays.

Input: a[] = [1, 2, 3], b[] = [4, 5, 6]
Output: []

Find Only Repetitive Element from 1 to n-1

Difficulty: **Easy** Accuracy: **59.22%** Submissions: **24K+** Points: **2**

Given an array **arr[]** of size **n**, filled with numbers from **1** to **n-1** in random order. The array has **only** one repetitive element. Your task is to find the **repetitive element**.

Note: It is guaranteed that there is a repeating element present in the array.

Examples:

Input: arr[] = [1, 3, 2, 3, 4]
Output: 3
Explanation: The number 3 is the only repeating element.

Input: arr[] = [1, 5, 1, 2, 3, 4]
Output: 1
Explanation: The number 1 is the only repeating element.

Input: arr[] = [1, 1]
Output: 1
Explanation: The array is of size 2 with both elements being 1, making 1 the repeating element.