

*

Suppose you have an array

2 2 3 2 2

check how many times "2" appears?

```
function (numbers) {
  count = 0
  for (i = 0 to size of array - 1)
    if (i == number)
      count++
  }
  return count;
}
```

TC: $O(n)$

*

So to find ^{count} all the element's appearance

~~TC: $O(n^2)$~~ for ex. $Q = 0, 1, 2, 3, 4, 5, 6, \dots$

so then TC would be $(Q \times O(n))$

↑
num. of elements

for example

$Q = 10^5$

Num. of array elements = 10^5

TC = $Q \times O(n)$

TC = $10^5 \times 10^5$

TC = 10^{10}

$10^8 \approx 1 \text{ sec.}$

$10^9 \approx 10 \text{ sec.}$

$10^{10} \approx 100 \text{ sec.}$

one operation will take almost 2 minutes which is very time taking.

So here comes Hashing

(Not Recommended)

* Hashing :- storing & fetching

question

Problem statement
for example array will have elements at max numbers 12.

prestoring

Hash array :- 0 1 2 3 4 5 6 7 8 9 10 11 12

0	0	0	0	0	0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---	---	---	---	---	---

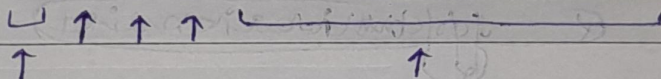
n = 13.

array

precalculation :-

1	2	1	3	2
---	---	---	---	---

0	2	2	3	4	5	6	7	8	9	10	11	12
0	2	2	1	0	0	0	0	0	0	0	0	0



[number of appearance]

frequency

* check How many times 2 appears? Hash[1] = 2 ✓ fetching

~~TC~~

TC: O(n)

SC: O(n)

Maxsize :- of Hash array

Inside of Int mem(c)

10⁶

→ 1909000

outside of Int mem(c)

10⁷

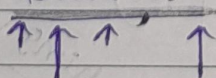
→ 19090000

(Globally)

* Character Hashing

question

S = "abcdabefc"



frequency of

a	→	2
c	→	2
2	→	0

again same logic

```

function (char c, string s)
{
    count = 0
    for (int i = 0; i < s.length(); i++)
    {
        if (s[i] == c)
        {
            count++;
        }
    }
    return count;
}

```

If I do for n characters

(in main)

for (char i = 'a'; i <= 'z'; i++)

again TC would be $2 \times O(n)$

so

s = "abcabcabc"

lowercase, :- a, b, c, ..., z \rightarrow 26 letters

Hash Array \rightarrow 0 0 0 ... 0 0 0 0
0 2 2 3 ... 23 24 25
a b c d ... x y z

ASCII value :- 97 to 122
a z

to store f = ASCII of f - ASCII of a
= 102 - 97

f = 5

a = ASCII of a - ASCII of a
= 97 - 97

a = 0

hash[s[i]] = 'a';

hash['a']

If you have smaller & capital letters both you have to use Hash array of 256 length and then you can directly do as you have

maxlength \rightarrow globally 10^4 , we can't use size for $10^8, 10^9, 10^{10} \dots$ array (X)
input you have to use STL

Basic Unordered Mapping

Working of unordered map

map (key, value) arr = [1 | 2 | 3 | 2 | 3 | 2 | 12]

(12 \rightarrow 1)
(3 \rightarrow 2)
(2 \rightarrow 2)
(1 \rightarrow 2)

add (bucket)

It doesn't store whole array of elements.

It saves memory.

only numbers appears atleast once are only get stored.

If I check arr[12] \rightarrow It will directly give me.

Task:- No Number Hashing using maps.

TC:- (of maps) $O(\log n)$ in all cases

(best) unordered map

TC:- $O(1)$ {avg best worst} worst $O(n)$

Recommended:- unordered map (1st preference)
map (2nd preference)

Working of unordered map with

hash[s[i]]++; instead of hash[s[i]] = 'a';

Summary

* Count freq. using Naive nested for loops

TC: $O(n^2)$

- (1) Number Hashing using Hash array } better
(2) character Hashing using Hash array }

↳ limitations:- 10^7 (globally) maximum input size
so better option ✓

- (3) Number & character Hashing using C++ STL → maps → $O(\log n)$

→ unordered

maps

Best
avg

Worst

$O(1)$ $O(n)$

अब आओ...

map

multimap

unordered map

- sorted & ordered
- each key containing unique elements
- logn
- Implementation: Balanced BST (Red black tree)

- each key can contain duplicate values
- sorted/ordered

- each key containing unique values
- unordered
- Best & Avg. $O(1)$
- Worst $O(n)$
- Implementation: Hash table.

Inner Working of unordered maps

Before that let's learn Hash methods (Division method)

array = [11, 29, 88, 76, 54, 53, 91]

$h(x) = x \% 10$

Bucket	
0 →	
1 →	11, 91
2 →	
3 →	53
4 →	54
5 →	
6 →	76
7 →	
8 →	88
9 →	29

← collision

load factor = $\frac{n}{b}$

number of elements
num of keys

Handle
to ~~separate~~ collision
2 methods

separate chaining
open addressing
↓
probing

unresolved
means
used
this

separate
chaining

creates chain values using ll (linkedList)
means avg case $O(1)$
worst case $O(n)$

load factor here is (0.7) means each bucket is on an avg 70% full, typically good balance, it is.

If it is Higher than 1 → you have to do rehashing.

Hashing
In
unresolved
maps

