Hashing

Count element of array

1	2	1	3	2

Count the appearance

1 → 2
3 → 1
4 → 0
2 → 2
10 → 0

1st Approach

~~int cnt = 0~~

int f(int n, a[]) f(1, a[]) // for '1'; f(3, a[]) // for '3'.

```
{ cnt=0
 for(i=0; i<n; i++)
 { if (arr[i] == number)
        cnt++;
   }
 return cnt;
}
```

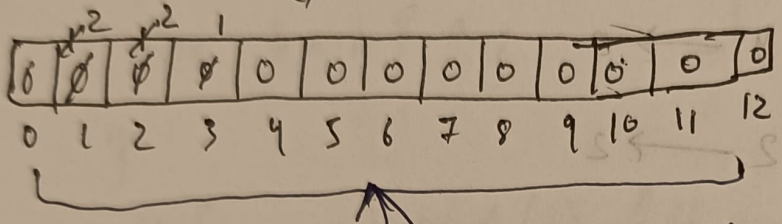
T.C ⇒ O(n) ≃ O(θ(N))

2nd Approach

Hashing → Prestoring / fetching

1	2	1	3	2

n=13 hash array



1 → 2
3 → 1
4 → 0
2 → 2
10 → 0
⋮
12 → 0

pre-calculation

hash[1] = 2

hash[3] = 1

hash[4] = 0

Code

```
int main ()
{
  int n;
  cin >> n;
  int arr[n];
  for (int i = 0; i < n; i++)
{   cin >> arr[i];
}
```

// pre-compute ⇒ next page

Test Case

5 ⇒ array

1 3 2 1 3

5 ⇒ no of values

1
4
2
3
} numbers

12

↳ find the count

```
// taking all the input
int q;
cin>>q;
while(q--){
    int number;
    cin>>number;
    // fetch
}
```

```
// precomute - Code
int hash[13] = {0};  // iteration of array
for(int i=0; i<n; i++){
    // for every number increase the value
    hash[arr[i]] += 1;
}
```

```
// fetch - code
cout<<hash[number]<<endl;
```

o/p
1 - 2
4 - 0
2 - 1
3 - 1 12 - 0

Maximum hash array size (Main vs Global)

Suppose max array element = 10^9

can we declare array of size arr[10^9+1]

NO

max you can do only arr[10^6] → inside main

if you 10^7

then segementation fault.

BUT

~~after the declaration~~

after the space next line you declare (Global)

int arr[1e7] ⇒ then you can use 10^7

Character Hashing P-2

$S = "abcdabefc"$

a → 2

c → 2

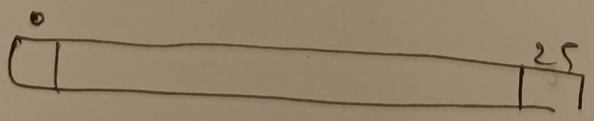
2 → 0

```
f(char c, s)
{
    cnt = 0
    for(int i=0; i<=n; i++)
    {
        if(s[i] == c)
            cnt++
    }
    return cnt
}
```

T.C ⇒ O(0 × n)

→ Can you hash them into arrays? Yes

let lower case letter a ... z → 26



$S = "abcdabefc"$

a → 0

b → 1

⋮

z → 25

ASCII Value will be used

upper A ⇒ 65

lower a ⇒ 97

int x = 'a'

↓

97

lower b ⇒ 127

'f' − 'a'

102 − 97 = 5 (so for ~~this~~ the 'f' this is the array value)

So, 'a' − 'a' = 0

'b' − 'a' = 1

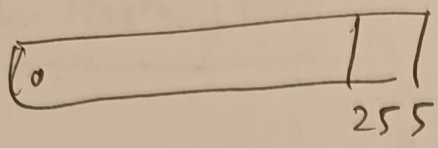
'c' − 'a' = 2

So the formula is

ch − 'a' ⇒ corresponding array

easy to hash now

for all char



arr[256]

'a' - 97 ⇒ a

Code

```
int main()
{
    string s
    cin >> s
    // pre-compute
    int hash[26] = {0};
    for(int i=0; i<s.size(); i++){
        hash[s[i] - 'a']++;
    }
    int q
    cin>>q;
    while(q--){
        char c;
        cin>>c;
        // fetch
    }  cout<< hash[c-'a'] <<endl
```

↳ indexing

Test Case

a b c d a b e h f → q

5

a → 2

g → 0

h → 1

b → 2

c → 1

for all Character

```
int has[256]={0}
//pre compute
for( ){
has[s[i]]++; //auto cast
}
//fetch
cout << hash[c]
     <<endl;
```

Number hashing

STL / Collection

- map
- unordered_map → Hash Map

↳ contain unique keys but not in sorted order

map

arr = 1, 2, 3, 1, 3, 2

mpp[1]++

(3 → 2)
(2 → 1)
(1 → 2)

map < key , value >

key → number
value → frequency

⇒ mpp[arr[i]++]

mpp4 → 0

If doesn't store itself as it's not available

Code

```
int main(){
  int n;
  cin>>n;
  int arr[n];
for(int i=0; i<n; i++)
{  cin >> arr[i]; }
```

```
//pre-comute
map<int, int> mpp;
for(int i=0; i<n; i++)
    mpp[arr[i]]++;
}
int q
cin>>q
while(q--){
    int num;
    cin>>num;
    //fetch
    cout<<mpp[number]<<endl
}return 0;
}
```

7

1 2 3 1 3 2 12

5

1 → 2

2 → 2

3 → 2

4 → 2

12 → 0

TC → Map

Storing } → (log N)
Fetching }

{ over but worst }

unordered_map

Storing } → O(1) { average but
Fetching }

→ O(N) worst

↳ no of element in the map.

Leetcode 125

```
  l
  ↓ ↓
* a b : a ≥ b a .
          ↑ ↑
          r r
```

1st Approach

both l, r will point only to the valid character inside the while loop.

Once, it's a valid character you will check if l == r if it's true then l++ ; r--.

T.C ⇒ O(N)

2nd Approach

* take a string and form vector of character (arr)
* Start passing from 1st character as whenever valid character we will push that character into a array.

```
  ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓
* a b : a , b a .
l       ↓   ↓r
arr | a | b | a | b | a |
```

⇒ * then normal approach

+ This need two traversal

T.C ⇒ O(N)

3rd Approach

Using alphanumeric.

→ Move 2 pointer from each end (while)

→ increment left pointer if not alphanumeric

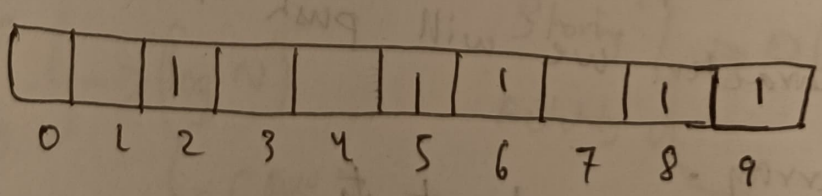
→ decrement right pointer if not alphanumeric

→ Exit and return if not match.

Divison Method

if the arr size is more than 10 we will use this method only till arr[9] is allocated

{2, 5, 16, 28, 139}



arr[i] % 10

16 % 10 = 6

28 % 10 = 8

2 % 10 = 2

5 % 10 = 5

139 % 10 = 9

now i was able to hash it.

Now if someone ask how many times '139' occured we will do

$$139 \% 10 = 9$$

now value at 9 is '1' so only '1' it appear.

@ → what if on modulo 10 they are same numbers

{2, 5, 16, 28, 139, 38, 48, 28, 18}

↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓

2 5 6 8 9 8 8 8 8

Chaining Method

0

1

2 → 2

3

4

5 → 5

6 → 16

7

8 → 18 → 28 → 38 → 48

9 → 139

How many times '28' appears, as it will sorted we can search it using binary search.

Collision Method

18, 28, 38, 48, ... 108

8 ⟶ 18, 28, 38

↳ collision is this as everything want to arr[8].

→ all your keys ends up at index of 8.

→ It's a worst case.

→ O(N).

*

In unordered-map, you can only have individual data type.

par<int, int>, int → Not allowed

↳ key

But in map → it's allowed.