Hashing Concept

1	2	1	3	2

element → No. of times appears in array

1 → 2

2 → 2

3 → 1

Brute force Approach

```
count = 0

for i in range(len(arr)):

    if arr[i] == num:

        count += 1

return count
```

→ Time complexity O(n)

Running loops for n times on linear search

5 × O(n)

9 × O(n)

O($10^5 \times 10^5$)

O(10^5)

we know 10^8 = 1 sec

10^{10} = 100 sec

$\frac{1}{10^8} \times 10^{10} = 10^2 = 100$

That why we need → Hashing

pre storing → arr fetching

n = 13

2 2 1

Pre calculation

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

→ Hash Array

has[1] → 2

has[2] → 2

This Approach more optimized

→ only single loop required

Counting frequency problem statement

```
hash[0] * 13

n = int(input())

arr = list(map(int, input().split()))

# precompute

for num in arr:

    hash[num] += 1
```

#fetching

```
q = int(input())

while q!=0:

    number = int(input())

    print(hash[number])

    q -= 1
```

Maximum size of hash

int → max size = [10^6]

→ for local variable

for global variable memory allocate upto

→ 10^7

Character hashing problem statement

s = 'abcdabefc'

How we know 26 chars → memory location upto → 0-25

Somehow we need to map value → 0, 1, 2, 3, ...

→ This with a, b, c

→ Ascii value

97 to 122

a - a = 0 → 97 - 97 = 0

b - a = 1 → 98 - 97 = 1

c - a = 2 → 99 - 97 = 2

```
s = input()
hash = [0] * 26
for char in s:
    hash[ord(char) - ord('a')] += 1
q = int(input())
while q!=0:
    c = input()
    print(hash[ord(c) - ord('a')])
    q-=1
```

Collision in hashing

When the same key is assign → with different or multiple index

→ collision occurs

Hashing Method

division method

2 5 7 16 28 139

2%10, 5%10, 7%10, 28%, 139%10

2 5 7 8 9

chaining occurs

↓

If collision can occurs in division method also

Remainder → same → same index