P-1: Let A be an array of N elements. Print all unique numbers in A along with their repetition count.

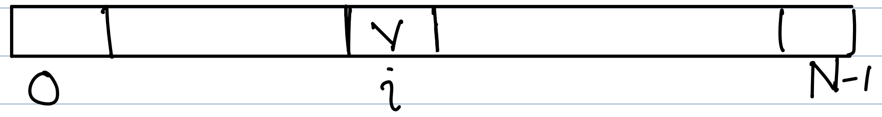
A:

10	5	7	10	8	13	7	5	5	8	25

Output:

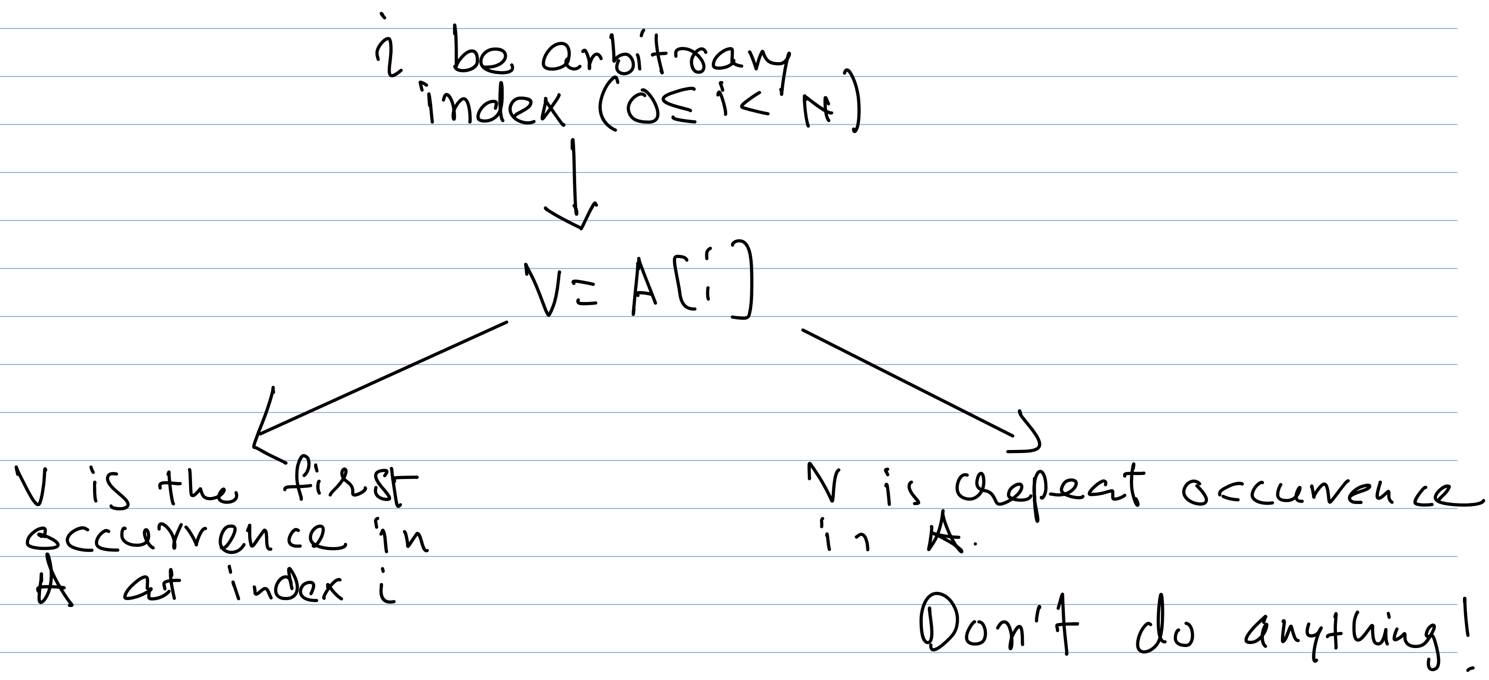
10 → 2
5 → 3
7 → 2
8 → 2
13 → 1
25 → 1.

Constraint: You are allowed to allocate counter variables as much as you require. But you will not duplicate array or use structure or any other memory.



let $v = A[i]$ and let i be the first index with value v.

```
count = 0;
for (k = i; k < N; ++k)
    if (a[k] == a[i])
        ++count;
```



i be arbitrary index (0≤i<N)

V= A[i]

V is the first occurrence in A at index i

V is repeat occurrence in A.

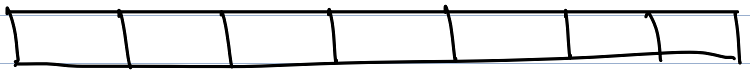
Don't do anything!

```
count = 0;
for (k=i; k<N; ++k)
    if (a[k]==a[i])
        ++count;
```

Problem: let A be the array of N elements where (N ≥ 0). An element can repeat itself in arra.

Write an algo to print number of unique arrangements of array elements?

1 2 4 3 2 3



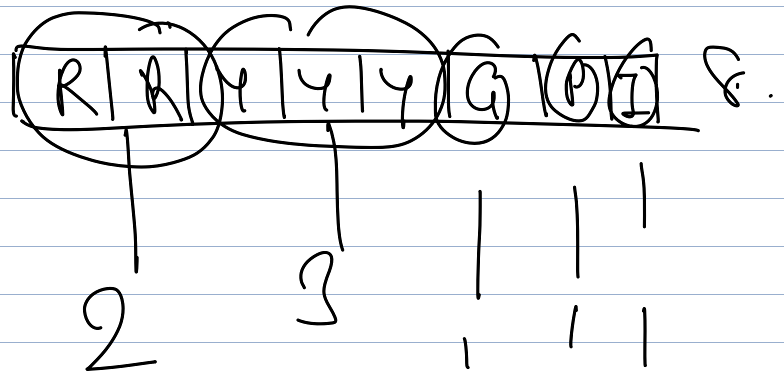
— N-size

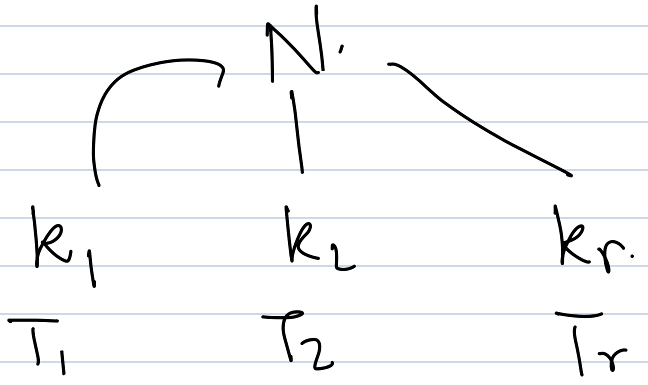
N- different object

R	G	B	Y	V	I	P	Grey

8

8 $1 \times 2 \times 3 \times 4 \times 5 \times 6 \times 7 \times 8$





$k_1 + k_2 + — + k = N$

total number of arrangements $= \dfrac{N!}{k_1! \times k_2! \times k3! \times \ldots \ldots \times k_r!}$

$$= \frac{N!}{k_1! \cdot k2! \ldots k_r!}$$

Total Playing balls = 8

T_1 T_2 T_3 T_4 T_5

(R) (Y) (G) (P) (I)

2 3 1 1 1

$$\frac{8!}{2! \times 3! \times 1! \times 1! \times 1!}$$

$$= \frac{8!}{2! \times 3!} = \frac{8 \times 7 \times 6 \times 5 \times \overset{2}{\cancel{4}} \cancel{\times 3 \times 2 \times 1}}{\cancel{2 \times 1} \times \cancel{3 \times 2 \times 1}}$$

$$= 8 \times 7 \times 6 \times 5 \times 2$$

$$= 56 \times 6 \times 10$$

$$= 56 \times 60$$

$$= \boxed{3360}$$

N element का Array

unique elem.	E_1	E_2	—	—	E_r
Repeat count	k_1	$k2$			kr.

$$\frac{N!}{k_1! \times k2! \text{ — - } \times kr!}$$

unsigned long long Nr = fact (N)

Nr = Nr / factorial(repeat_count)

✓ Coding Practice DSA App.	Math basics Di M.	Practice without time constraint / open book ?'	Practice

C/C++ : Lang. knowledge — 1/2

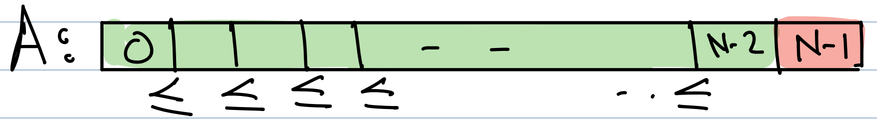
DSA : 2/3 50% 60%

OS/ O.O.P. : 1
design Part

P-2 : - Let A be an array of N elements.

- All but last element are sorted (ascending).
- Develop an algorithm to sort entire array!

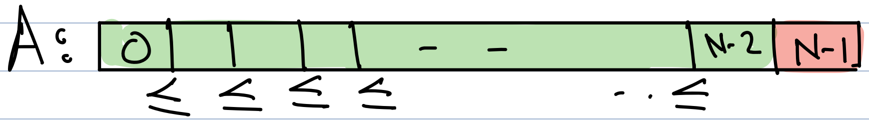
Constraint : Only counter variable
Single loop.

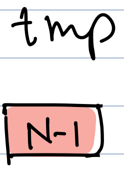


Algo : Complete A: sort.

Move last element in temporary integer variable.

tmp ← A[N-1]





Algo : Complete A: sort.