2

Greatest Till Me

Make a [redacted]x array of size **N** such that at the **kth** index of the [redacted]x array store the greatest element from the left till the **kth** index of the given array.

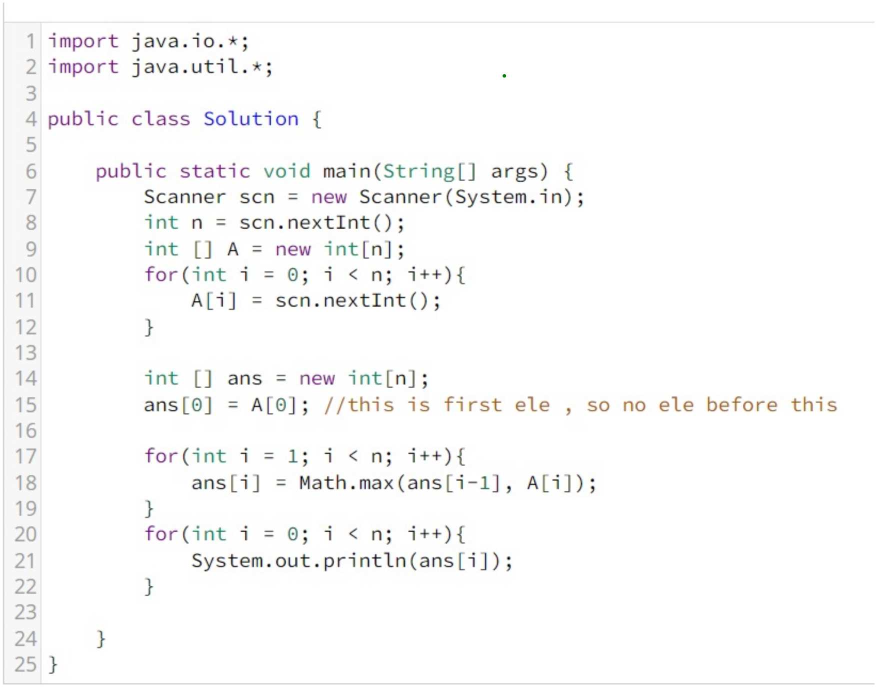
ans[0] = A[0].

n = 10.

A	2	4	3	7	6	5	5	8	2	3
	0	1	2	3	4	5	6	7	8	9
ans	2	4	4	7	7	7	7	8	8	8

2 4 4 7 7 7 7 8 8 8.

$$ans[i] = Max(ans[i-1], A[i])$$



```java
import java.io.*;
import java.util.*;

public class Solution {

    public static void main(String[] args) {
        Scanner scn = new Scanner(System.in);
        int n = scn.nextInt();
        int [] A = new int[n];
        for(int i = 0; i < n; i++){
            A[i] = scn.nextInt();
        }

        int [] ans = new int[n];
        ans[0] = A[0]; //this is first ele , so no ele before this

        for(int i = 1; i < n; i++){
            ans[i] = Math.max(ans[i-1], A[i]);
        }
        for(int i = 0; i < n; i++){
            System.out.println(ans[i]);
        }

    }
}
```

Print Prefix Sum between L and R

Take an integer input **l** and **r** such that `l,r<=array.length`. Given an array. Make a prefix sum array from this. The print the sum of the elements inside the array starting from the **l-index** till the **r-index(l and r both inclusive)**.

Sample Input 0

```
5  → n
1
2
8
4
10
1
3
```

n=5

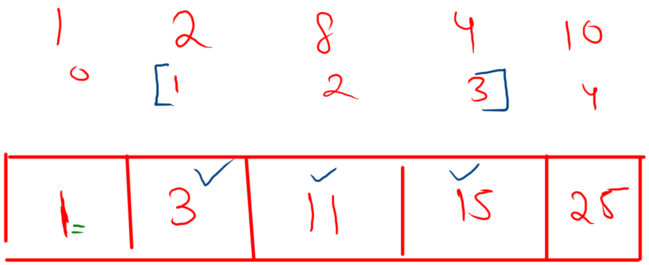
✓ psa.

Sample Output 0

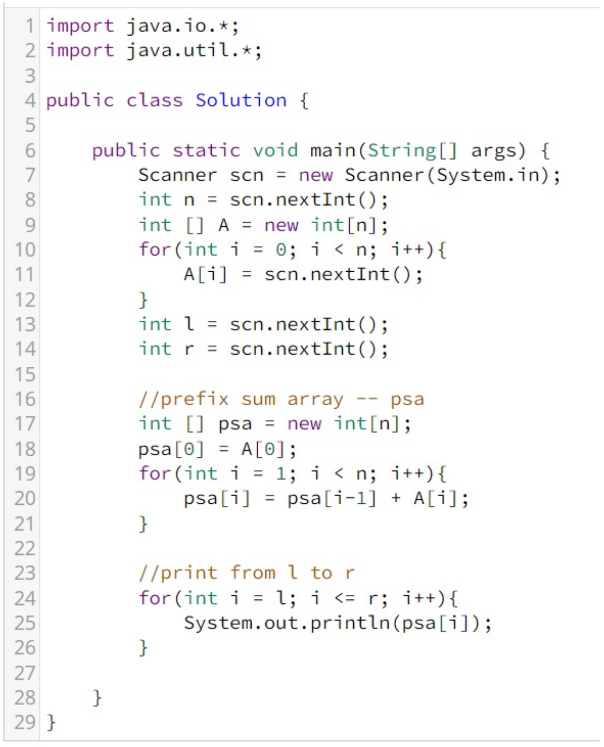
```
3
11
15
```

l=1

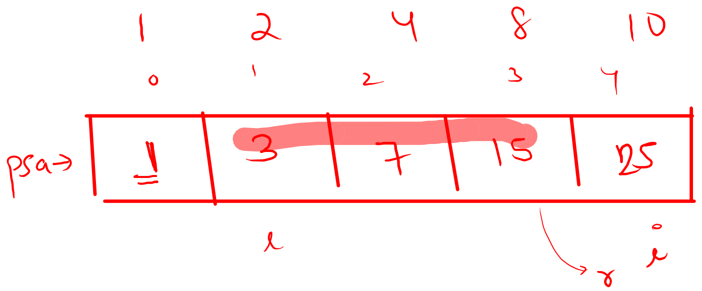
r=3



$l=1$ $r=3$ $n=5$



```java
import java.io.*;
import java.util.*;

public class Solution {

    public static void main(String[] args) {
        Scanner scn = new Scanner(System.in);
        int n = scn.nextInt();
        int [] A = new int[n];
        for(int i = 0; i < n; i++){
            A[i] = scn.nextInt();
        }
        int l = scn.nextInt();
        int r = scn.nextInt();

        //prefix sum array -- psa
        int [] psa = new int[n];
        psa[0] = A[0];
        for(int i = 1; i < n; i++){
            psa[i] = psa[i-1] + A[i];
        }

        //print from l to r
        for(int i = l; i <= r; i++){
            System.out.println(psa[i]);
        }

    }
}
```



$i \le r$

$1 \le 3$ ✓

$4 \le 3$ ✗

3

7

15

Find Pivot Index

Given an array of integers nums, calculate the pivot index of this array.

The pivot index is the index where the sum of all the numbers strictly to the left of the index is equal to the sum of all the numbers strictly to the index's right.

If the index is on the left edge of the array, then the left sum is 0 because there are no elements to the left. This also applies to the right edge of the array.

Return the leftmost pivot index. If no such index exists, return -1.

Sample Input 0

```
6
1 7 3 6 5 6
```

Sample Output 0

```
3
```

n=6

1	7	3	6	5	6
0	1	2	3	4	5

0	1	8	11	17	22

i

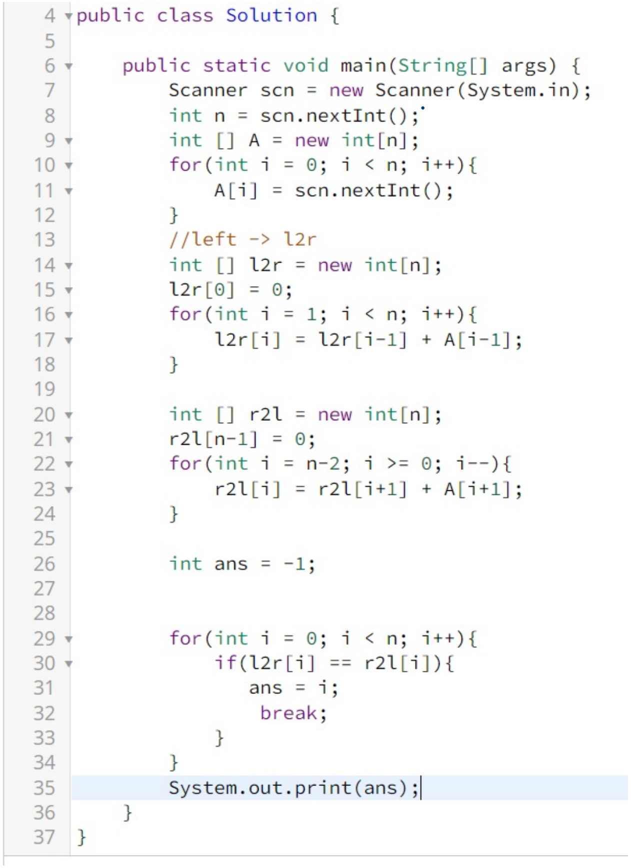
right

27	20	17	11	6	0

3

```java
public class Solution {

    public static void main(String[] args) {
        Scanner scn = new Scanner(System.in);
        int n = scn.nextInt();
        int [] A = new int[n];
        for(int i = 0; i < n; i++){
            A[i] = scn.nextInt();
        }
        //left -> l2r
        int [] l2r = new int[n];
        l2r[0] = 0;
        for(int i = 1; i < n; i++){
            l2r[i] = l2r[i-1] + A[i-1];
        }

        int [] r2l = new int[n];
        r2l[n-1] = 0;
        for(int i = n-2; i >= 0; i--){
            r2l[i] = r2l[i+1] + A[i+1];
        }

        for(int i = 0; i < n; i++){
            if(l2r[i] == r2l[i]){
                System.out.print(i);
            }
        }


        System.out.print(-1);



    }
}
```

stop



```java
public class Solution {

    public static void main(String[] args) {
        Scanner scn = new Scanner(System.in);
        int n = scn.nextInt();
        int [] A = new int[n];
        for(int i = 0; i < n; i++){
            A[i] = scn.nextInt();
        }
        //left -> l2r
        int [] l2r = new int[n];
        l2r[0] = 0;
        for(int i = 1; i < n; i++){
            l2r[i] = l2r[i-1] + A[i-1];
        }

        int [] r2l = new int[n];
        r2l[n-1] = 0;
        for(int i = n-2; i >= 0; i--){
            r2l[i] = r2l[i+1] + A[i+1];
        }

        int ans = -1;


        for(int i = 0; i < n; i++){
            if(l2r[i] == r2l[i]){
               ans = i;
                break;
            }
        }
        System.out.print(ans);
    }
}
```

Print Freq of Alphabet in String

✓ hashmap. →

John is a software engineer who is passionate about programming. One day, he stumbled upon a challenging problem in an online coding platform. The problem required him to find the **frequency** of each alphabet in a given **string** and print the frequency of each alphabet present in the string.

help John and write a program that return the frequency of each element of string using **array** ~~as hashmap~~.

Sample Input 0

```
abcdaccd
```

Sample Output 0

```
a-2 ✓
b-1
c-3
d-2
```

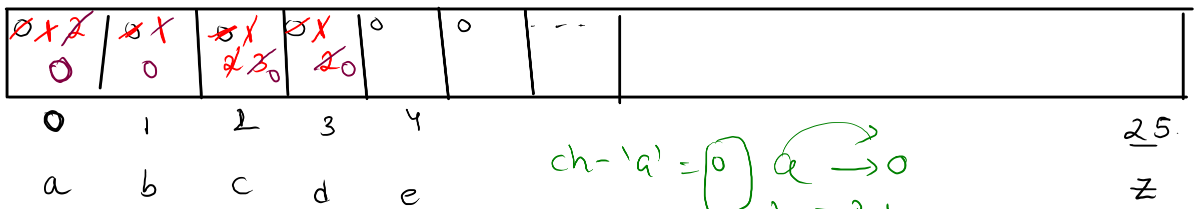
$8 \rightarrow$ " a b c d a c c d "

a	b	c	d	a	c	c	d
0	1	2	3	4	5	6	7

print ans.

a - 2
b - 1
c - 3
d - 2

store freq → array (26) →



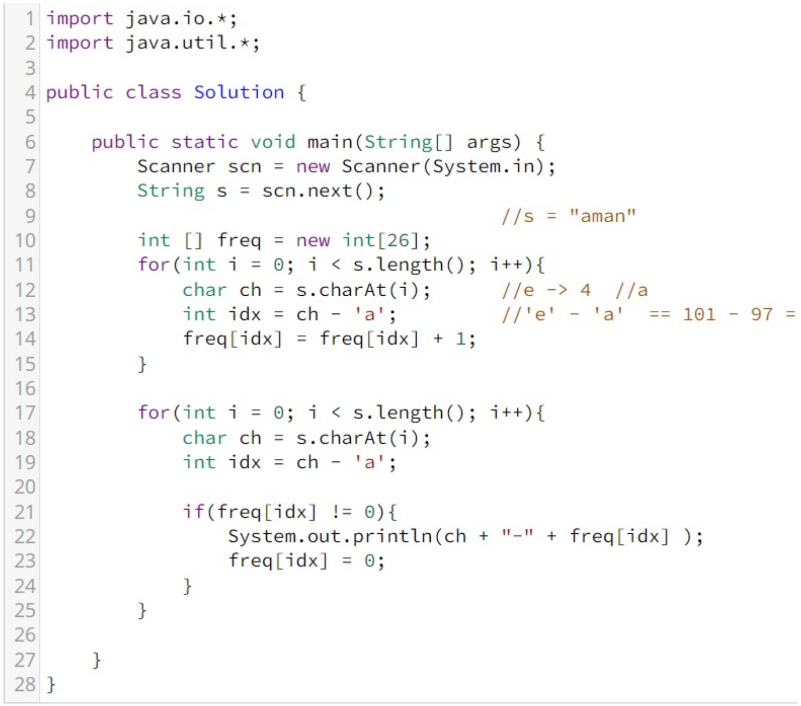
$ch - 'a' = \boxed{0}$

a → 0
b → 1
c → 2

ch

'b' − 'a' = 1
98 − 97 = 1
'c' − 'a'
99 − 97 = 2

8 → "b a cc aa"



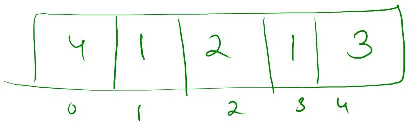
```java
import java.io.*;
import java.util.*;

public class Solution {

    public static void main(String[] args) {
        Scanner scn = new Scanner(System.in);
        String s = scn.next();
                                            //s = "aman"
        int [] freq = new int[26];
        for(int i = 0; i < s.length(); i++){
            char ch = s.charAt(i);          //e -> 4  //a
            int idx = ch - 'a';             //'e' - 'a'  == 101 - 97 =
            freq[idx] = freq[idx] + 1;
        }

        for(int i = 0; i < s.length(); i++){
            char ch = s.charAt(i);
            int idx = ch - 'a';

            if(freq[idx] != 0){
                System.out.println(ch + "-" + freq[idx] );
                freq[idx] = 0;
            }
        }

    }
}
```

Int with Maximum Freq

Mark is a data analyst who is trying to analyze the customer data of a retail company. One of the tasks he needs to perform is to find the most common digit in the customer IDs. The IDs are represented as an array of **single-digit** integers from 0-9. Mark needs to find the **digit that occurs the most in the array** in order to identify patterns in customer behavior.

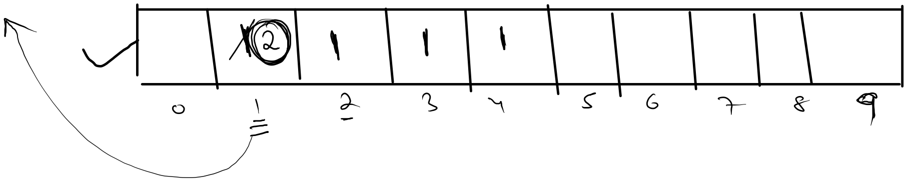
Help Mark and find the digit form the array that occurs maximum number of times.

n = 5



which single digit is occuring number of time?

ans = 1



1. freq. array.

freq →	0	2	1	1	1	0	0	0	0	0
	0	1	2	3	4	5	6	7	8	9

i

```
int maxIdx = 0;
for(int i = 0; i < 10; i++){
    if(freq[maxIdx] < freq[i]){
        maxIdx = i;
    }
}
```

$f[1] < f[3]$

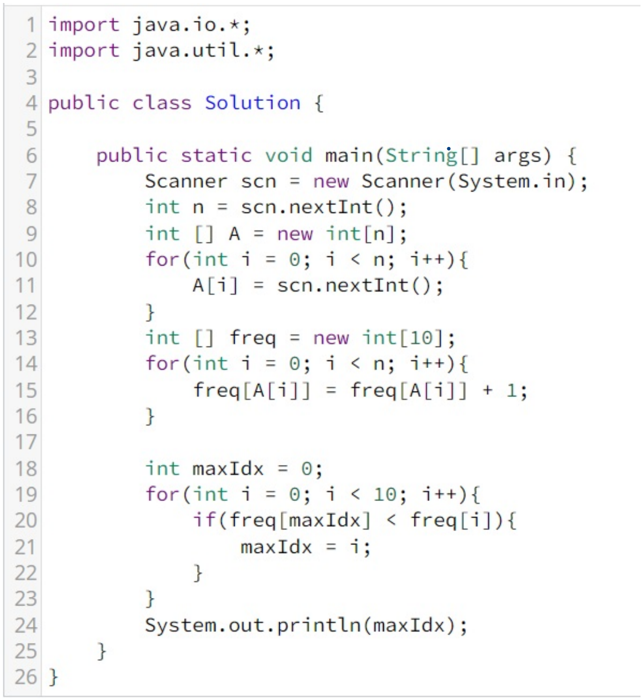
$2 < 1$ false.

mI = ~~0~~ 1

ans = 3.

n = 5

3 2 3 1 4



```java
import java.io.*;
import java.util.*;

public class Solution {

    public static void main(String[] args) {
        Scanner scn = new Scanner(System.in);
        int n = scn.nextInt();
        int [] A = new int[n];
        for(int i = 0; i < n; i++){
            A[i] = scn.nextInt();
        }
        int [] freq = new int[10];
        for(int i = 0; i < n; i++){
            freq[A[i]] = freq[A[i]] + 1;
        }

        int maxIdx = 0;
        for(int i = 0; i < 10; i++){
            if(freq[maxIdx] < freq[i]){
                maxIdx = i;
            }
        }
        System.out.println(maxIdx);
    }
}
```