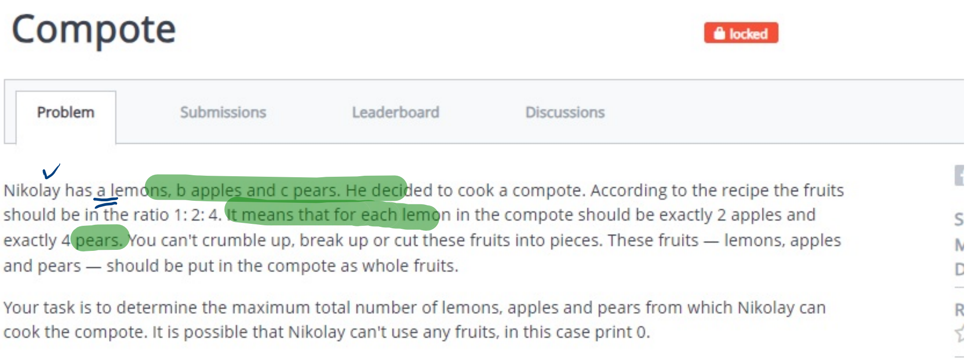


Compote

Nikolay has a lemons, b apples and c pears. He decided to cook a compote. According to the recipe the fruits should be in the ratio 1: 2: 4. It means that for each lemon in the compote should be exactly 2 apples and exactly 4 pears. You can't crumble up, break up or cut these fruits into pieces. These fruits — lemons, apples and pears — should be put in the compote as whole fruits.

Your task is to determine the maximum total number of lemons, apples and pears from which Nikolay can cook the compote. It is possible that Nikolay can't use any fruits, in this case print 0.

ans = 7

eg. 2 5 7

a → lemons
b → apples
c → pears.

a : b : c
1 : 2 : 4

eg.
a ~~3~~ ~~2~~ 1 lemon
b ~~12~~ ~~10~~ 8 apples
c ~~8~~ ~~4~~ 0 pears
→

1: 2: 4

1lemon
2 apples
4 pears

+1
+2
+4

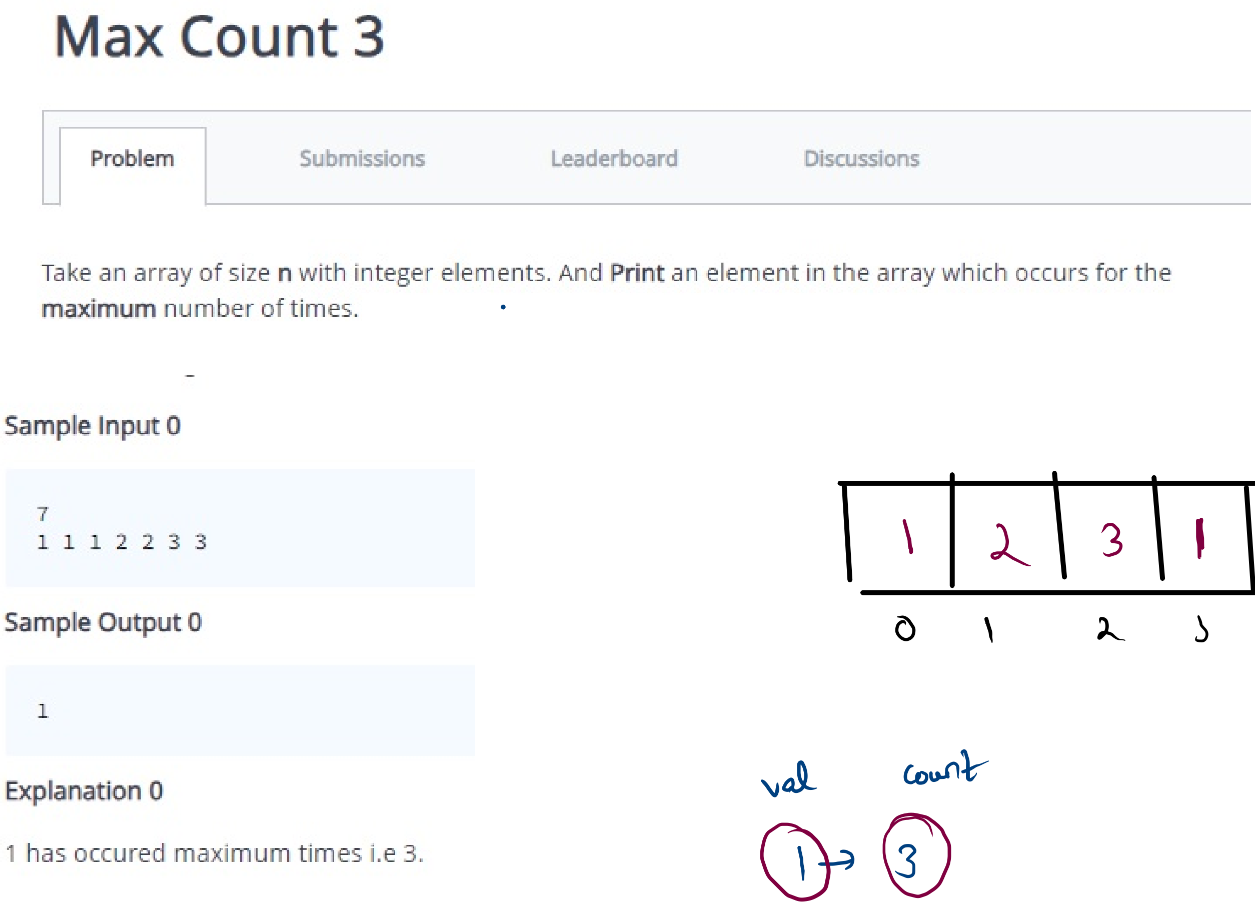
compote.
a && b && c.

pack
1 ✓
2 ✓
4 ✓
7

+

pack
1
2
4
7

fruits = 14.



Max Count 3

Take an array of size **n** with integer elements. And **Print** an element in the array which occurs for the **maximum** number of times.

Sample Input 0

```
7
1 1 1 2 2 3 3
```

Sample Output 0

```
1
```

Explanation 0

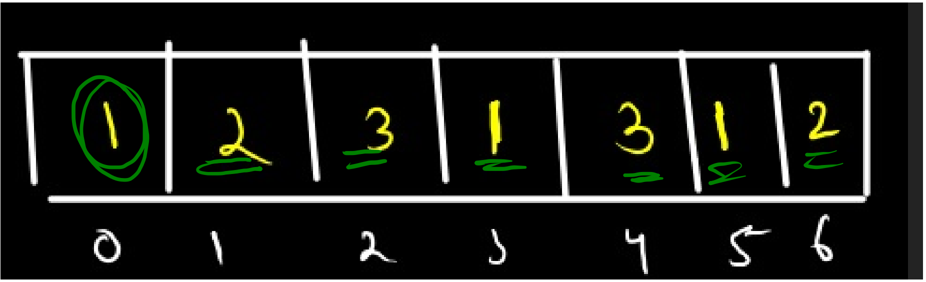
1 has occured maximum times i.e 3.

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

val	count
1 →	3
2 →	2
3 →	2

5	3	5	5	2	5	2	5	5

5



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

1 → 3

2 → 2

3 → 2

1 → 3

3 → 2

1 → 3

2 → 2

maxfreq = ~~3~~ 8

ans = ~~1~~ 3

2 ~~>~~ maxfreq

8 > maxfreq

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

n=7

maxfreq = ~~0~~ ~~2~~ 3

ans = ~~0~~ ~~2~~ 1

i = ~~0~~ ~~1~~ ~~2~~ 3   0< 7 ✓

1<7

2<7

3<7

curr = 1

count = ~~0~~ 3 ✓

1   3 times.

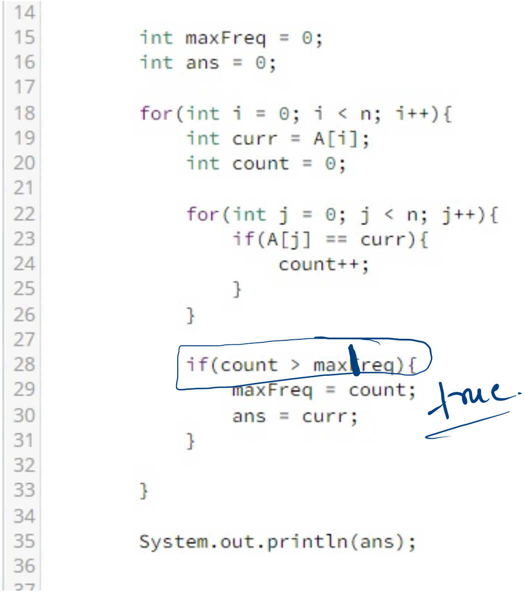
curr = 3

count = 2

3>3

count > maxfreq

maxfreq = count;



```
int maxFreq = 0;
int ans = 0;

for(int i = 0; i < n; i++){
    int curr = A[i];
    int count = 0;

    for(int j = 0; j < n; j++){
        if(A[j] == curr){
            count++;
        }
    }

    if(count > maxFreq){
        maxFreq = count;
        ans = curr;
    }

}

System.out.println(ans);
```

true.

Find Duplicate 3

Problem | Submissions | Leaderboard | Discussions

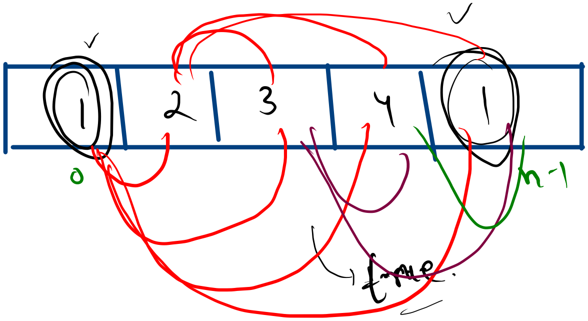
Take an array of size n with integer input. And Print **"true"** if the array contains a duplicate element and print **"false"**. if the array doesn't contain a duplicate element.

Sample Input 0

```
5   n
1
2
3
4
1
```

Sample Output 0

```
true
```



1 2
1 3
1 4

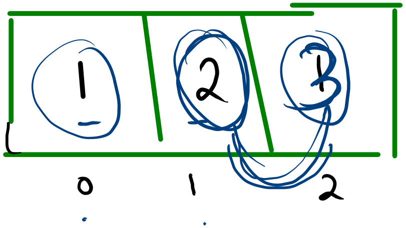
2 3
2 4
2 1

3 4
3 1

4 1

1 1 ? → true.

n=3



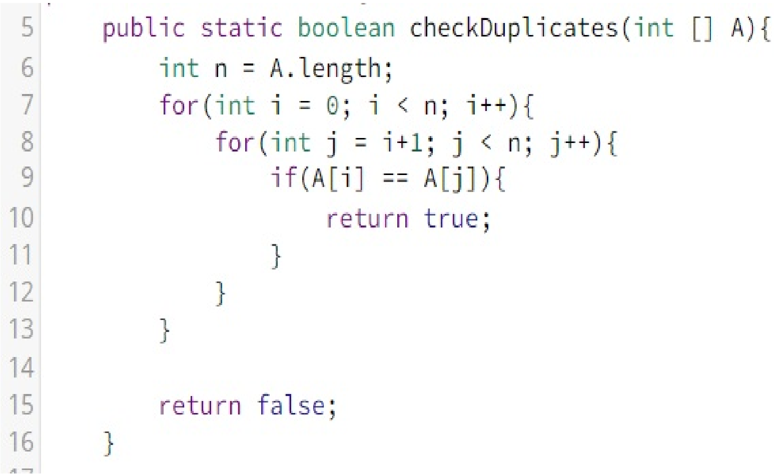
i = ~~1~~ ~~2~~ 3

2<3 ✓

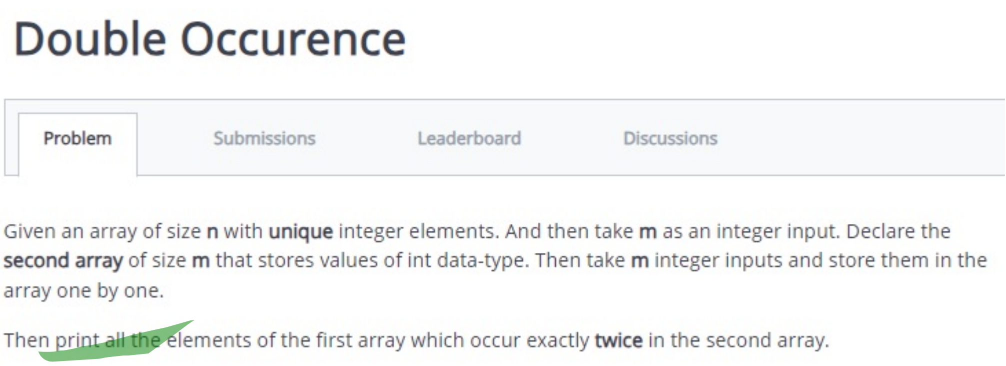
3<3 ✗

3<3 ✗

j = 3



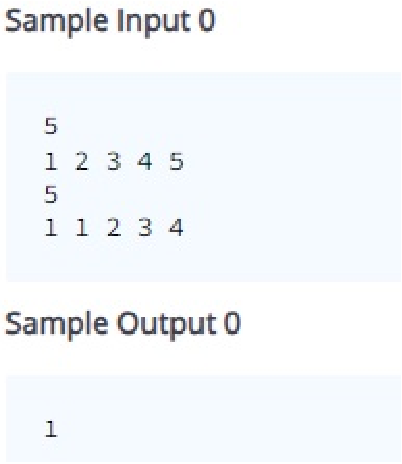
```
    public static boolean checkDuplicates(int [] A){
        int n = A.length;
        for(int i = 0; i < n; i++){
            for(int j = i+1; j < n; j++){
                if(A[i] == A[j]){
                    return true;
                }
            }
        }

        return false;
    }
```



Double Occurence

Given an array of size **n** with **unique** integer elements. And then take **m** as an integer input. Declare the **second array** of size **m** that stores values of int data-type. Then take **m** integer inputs and store them in the array one by one.

Then print all the elements of the first array which occur exactly **twice** in the second array.

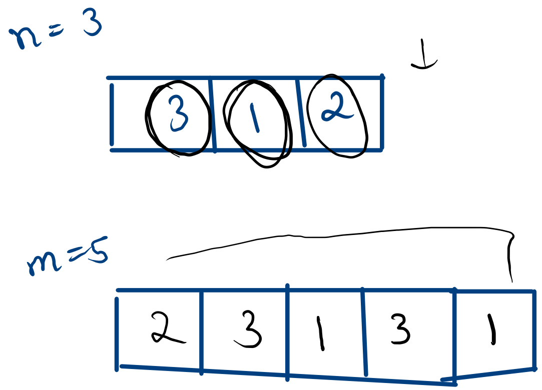


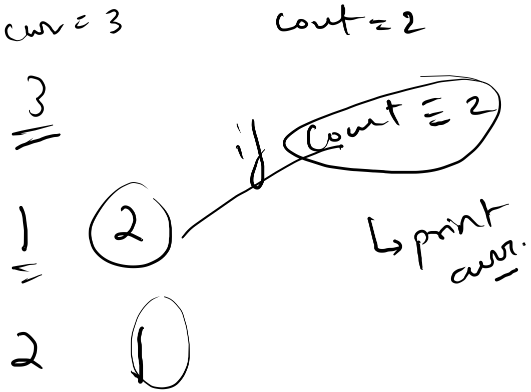
Sample Input 0

```
5
1 2 3 4 5
5
1 1 2 3 4
```

Sample Output 0

```
1
```



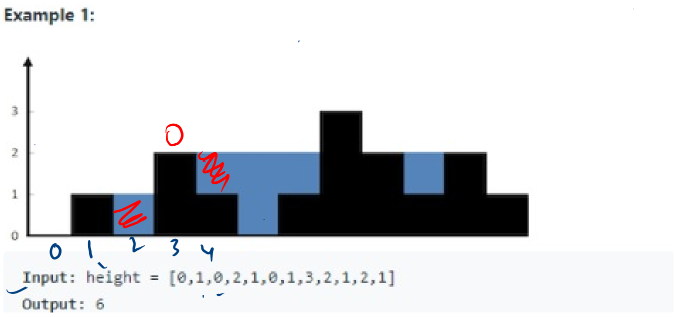


min(Lmax, Rmax) - h[i]

min (2, 3)

(2, 2)

2 - 3 = -1



Example 1:

Input: height = [0,1,0,2,1,0,1,3,2,1,2,1]
Output: 6

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

2-1=1

Step1

left max including me.

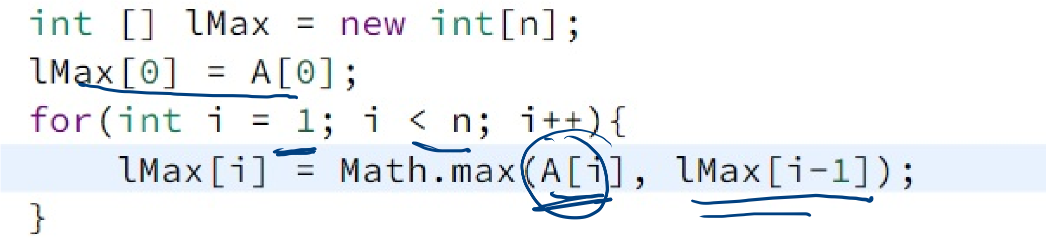
0	1	1	2	2	2	2	3	3	3	3	3
3	3	3	3	3	3	3	3	2	2	2	1

right max including me.

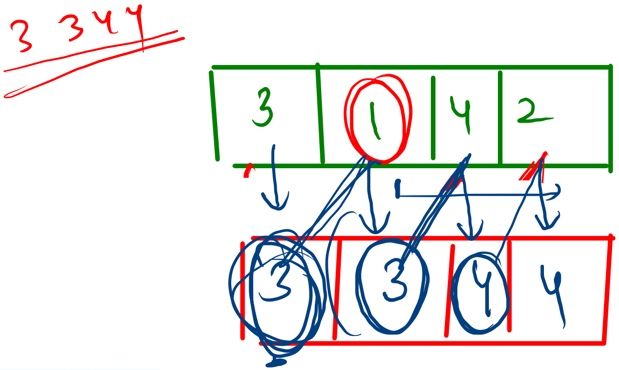
ans = 0 +1

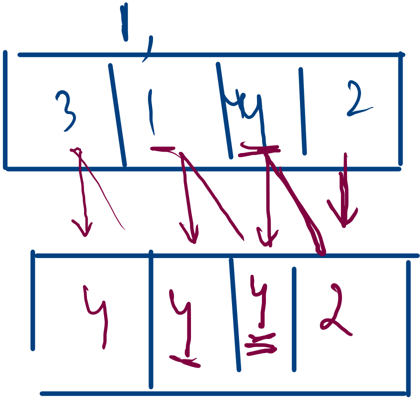
1 - 0 = 1

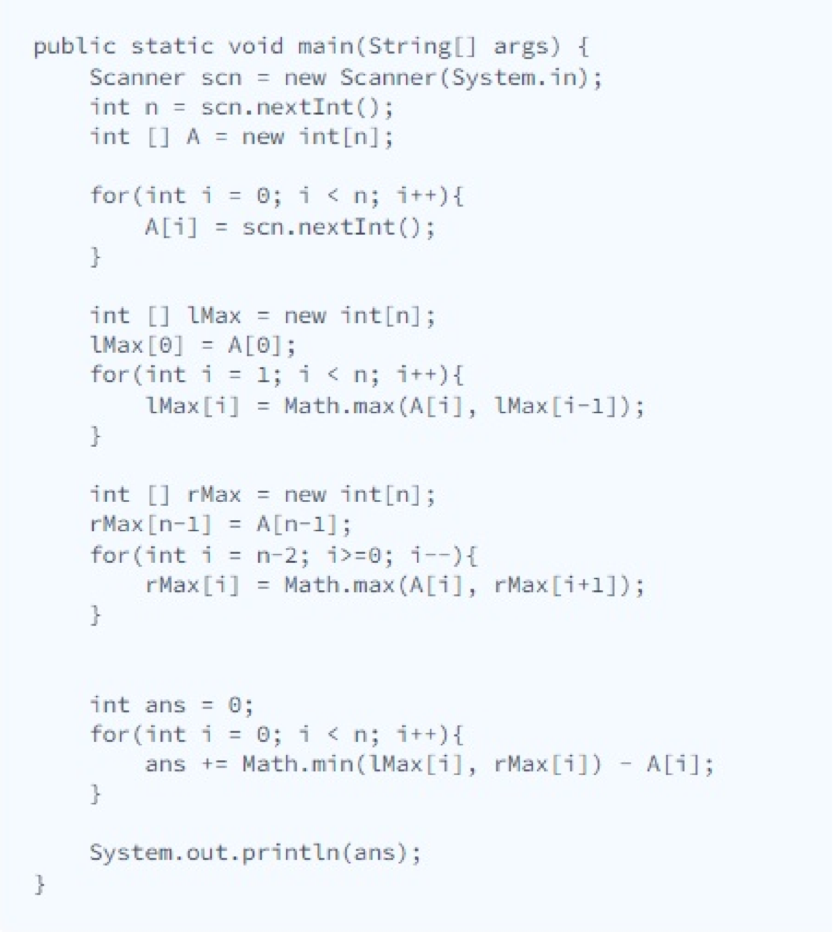
2 - 2 = 0



```
int [] lMax = new int[n];
lMax[0] = A[0];
for(int i = 1; i < n; i++){
    lMax[i] = Math.max(A[i], lMax[i-1]);
}
```







```
  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 [] lMax = new int[n];
      lMax[0] = A[0];
      for(int i = 1; i < n; i++){
          lMax[i] = Math.max(A[i], lMax[i-1]);
      }

      int [] rMax = new int[n];
      rMax[n-1] = A[n-1];
      for(int i = n-2; i>=0; i--){
          rMax[i] = Math.max(A[i], rMax[i+1]);
      }


      int ans = 0;
      for(int i = 0; i < n; i++){
          ans += Math.min(lMax[i], rMax[i]) - A[i];
      }

      System.out.println(ans);
  }
```

```
}
```