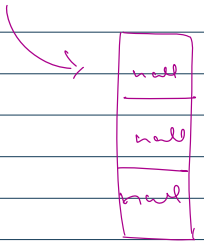
10 August 2025 10:13

Jagged Array

row have different length

```
int[][] arr = new int[3][];
```




```
public class Main {
    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();

        int[][] arr = new int[n][];

        System.out.println(arr);
        System.out.println(arr[0]);

        arr[0] = new int[2];
        arr[1] = new int[3];
        arr[2] = new int[1];

        int[][]  arr2 = {
            {1, 2},
            {2},
            {3, 4}
        };

        for(int i=0; i<arr2.length; i++){
            for(int j=0; j<arr2[i].length; j++){
                System.out.print(arr2[i][j]+" ");
            }
            System.out.println();
        }
    }
}
```

169. Majority Element

Solved

Easy · Topics · Companies

Given an array `nums` of size `n`, return *the majority element*.

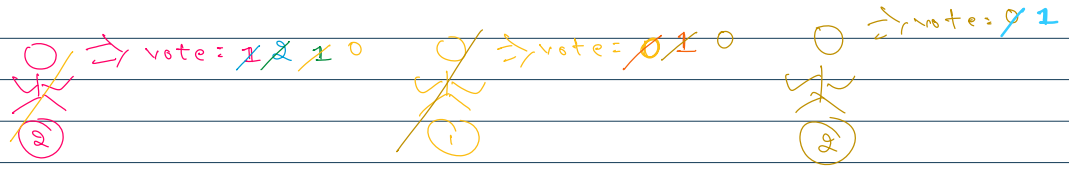
The majority element is the element that appears more than `⌊n / 2⌋` times. You may assume that the majority element always exists in the array.

Moore's Voting Algo

arr → [2, 2, 1, 1, 1, 1, 2, 2]

arr → [2, 2, 1, 1, 1, 2, 2]

arr → [2, 2, 1, 1, 1, 2, 2]



arr → [3, 2, 3]

maj → 3 2

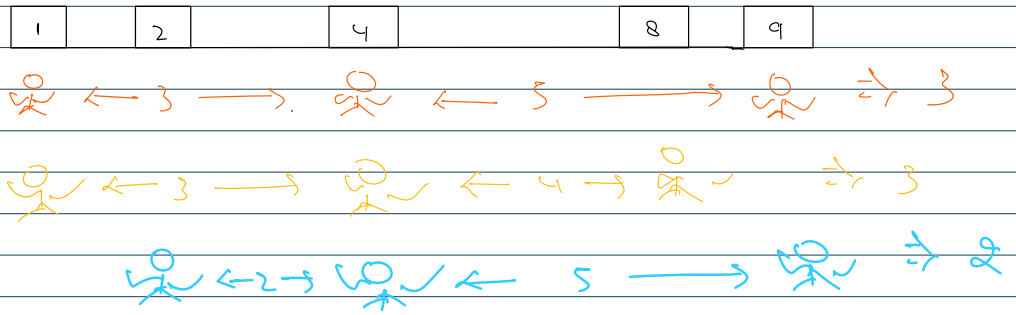
vote → 1 0 -1

Farmer John has built a new long barn, with N (2 ≤ N ≤ 100,000) stalls. The stalls are located along a straight line at positions x1 ... xN (0 ≤ xi ≤ 1,000,000,000).

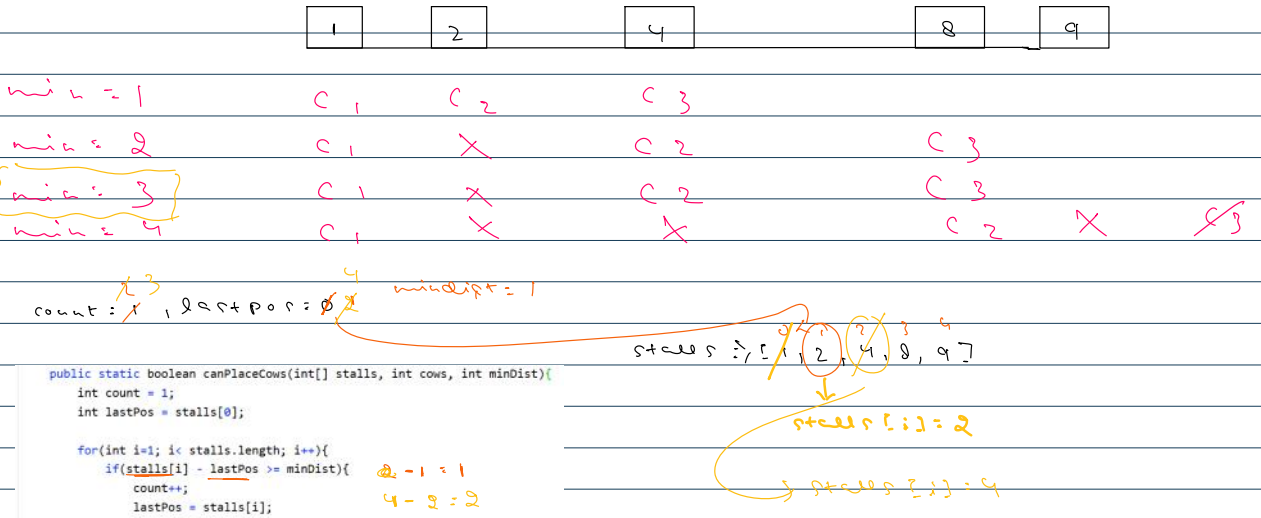
His C (2 ≤ C ≤ N) cows don't like this barn layout and become aggressive towards each other once put into a stall. To prevent the cows from hurting each other, FJ wants to assign the cows to the stalls, such that the minimum distance between any two of them is as large as possible. What is the largest minimum distance?

Input:
1
5 3
1
2
8
4
9

Output:
3



canPlaceCows(stalls, cows, minDist) => Complete.




```
public static boolean canPlaceCows(int[] stalls, int cows, int minDist){
    int count = 1;
    int lastPos = stalls[0];

    for(int i=1; i< stalls.length; i++){
        if(stalls[i] - lastPos >= minDist){
            count++;
            lastPos = stalls[i];

            if(count == cows){
                return true;
            }
        }
    }
    return false;
}
```

2 - 1 = 1
4 - 2 = 2

Binary Search

Searching for => 1 to maxDist
=> 1 to 8 => Sorted → ?

low : 1
high : 8 → mid : 4 => canPlaceCows → ? → X

left side → try smaller values

high : mid - 1 ;

low : 1
high = 4 - 1 = 3 → mid : 2 => canPlaceCows → ? → ✓

one of the answer => try larger values

low : mid + 1 ⇐ right side

low : 2 + 1 : 3
high : 3

3 ⇒ canPlaceCows → ? → ✓

one of the answer ⇒ try larger values

low : mid + 1 ⇐ right side

low : 3 + 1 = 4 ⇒ while (low < high) → ✗
high : 3

```
public class Main {

    public static boolean canPlaceCows(int[] stalls, int cows, int minDist){
        int count = 1;
        int lastPos = stalls[0];

        for(int i=1; i< stalls.length; i++){
            if(stalls[i] - lastPos >= minDist){
                count++;
                lastPos = stalls[i];

                if(count == cows){
                    return true;
                }
            }
        }
        return false;
    }

    public static int agressiveCowsLinear(int[] stalls, int cows){
        Arrays.sort(stalls);
        int n = stalls.length;
        int maxDist = stalls[n-1] - stalls[0];

        int best = 0;

        for(int d=1; d<=maxDist; d++){
            if(canPlaceCows(stalls, cows, d)){
                best = d;
            } else{
                break;
            }
        }

        return best;
    }

    public static int agressiveCowsBinary(int[] stalls, int cows){
```

```
        Arrays.sort(stalls);
        int n = stalls.length;

        int maxDist = stalls[n-1] - stalls[0];

        int low = 1;
        int high = maxDist;

        int best = 0;

        while(low <= high){
            int mid = low + (high-low)/2;

            if(canPlaceCows(stalls, cows, mid)){
                best = mid;
                low = mid + 1;
            } else{
                high = mid - 1;
            }
        }
        return best;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int t = sc.nextInt();
        while(t>0){
            int n = sc.nextInt();
            int c = sc.nextInt();
            int[] stalls = new int[n];
            for(int i=0; i<n; i++){
                stalls[i] = sc.nextInt();
            }

            // int result = agressiveCowsLinear(stalls, c);
            int result = agressiveCowsBinary(stalls, c);
            System.out.println(result);
            t--;
        }
    }
}
```