```
package Striver;
//**************************Nth Root of an Integer************************************** */
public public class Solution {
     public static double isMuliply(double mid,int n){
       double ans=1.0;
       for(int i=1;i<=n;i++){
          ans=ans*mid;
       }
       return ans;
   }
   public static double findNthRootOfM(int n, int m) {
          // Write your code here.
       double low=1.0;
       double high=m;
       double eps=1e-7;

       while(high-low>eps){
          double mid=(low+high)/2;
          if(isMultiply(mid,n)<m){
             low=mid;
          }
          else{
             high=mid;
          }
       }
       return low;
     }
}
//********************Median
Matrix********************************************************************** */
public class Solution {
   public int findMedian(ArrayList<ArrayList<Integer>> A) {
   int low=1;
   int high=1000000000;
   int N=A.size();
   int M=A.get(0).size();
   while(low<=high){
      int mid=(low+high)/2;
      int cnt=0;
      for(int i=0;i<N;i++){
         cnt+=countEleLessThanVal(A.get(i),mid);
      }
      if(cnt<=((N*M)/2))
      {
         low=mid+1;
      }
      else{
       high=mid-1;
      }
```

```
    }
    return low;
    }
    public int countEleLessThanVal(ArrayList<Integer> J,int target){
        int low=0;
        int high=J.size()-1;
        while(low<=high){
            int mid=(low+high)/2;
            if(J.get(mid)<=target){
                low=mid+1;
            }
            else{
                high=mid-1;
            }
        }
        return low;
    }
}
//******************Find the element that occurs only once
**********************************************/
//***************Brute Force XOR
all********************************************************************** */
class Solution {
    public int singleNonDuplicate(int[] nums) {
        int low=0,high=nums.length-2;
        while(low<=high){
            int mid=(low+high)/2;
            if(mid%2==0){
                if(nums[mid]==nums[mid+1]){
                    low=mid+1;
                }
                else{
                    high=mid-1;
                }
            }
            else{
                if(nums[mid]==nums[mid-1]){
                    low=mid+1;
                }
                else{
                    high=mid-1;
                }
            }
        }
            return nums[low];
    }
}
//******************Search in Rotated Sorted
Array******************************************************** */
```

```
class Solution {
    public int search(int[] nums, int target) {
       int low=0,high=nums.length-1;
       while(low<=high)
        {
            int mid=(low+high)/2;
            if(nums[mid]==target){
                return mid;
            }
            if(nums[low]<=nums[mid]){
                if(nums[low]<=target&&nums[mid]>=target){
                    high=mid-1;
                }
                else{
                    low=mid+1;
                }
            }
            else{
            if(nums[high]>=target&&nums[mid]<=target){
                    low=mid+1;
                }
                else{
                    high=mid-1;
                }
            }
        }
        return -1;
    }
}
//********************Median of Two Sorted Arrays(LeetCode Hard)
*****************************/
//********************Kth Elemenet of Two Sorted
Arrays************************************* */
//********************Allocate Minimum of
Pages******************************************** */
public class Solution {
    public int books(ArrayList<Integer> A, int B) {
        if(A.size()<B)
            return -1;
        int low=A.get(0),high=0;
        for(int i=0;i<A.size();i++){
            high=high+A.get(i);
            low=Math.min(A.get(i),low);
        }
        int res=-1;
        while(low<=high){
            int mid=(low+high)/2;
            if(isPossible(A,mid,B)){
                res=mid;
```

```
                high=mid-1;
            }
            else{
                low=mid+1;
            }
        }
        return low;
    }
    public boolean isPossible(ArrayList<Integer> A,int pages,int B){
        int sumAllocated=0,cnt=0;
        for(int i=0;i<A.size();i++){
            if(sumAllocated+A.get(i)>pages)
            {
                cnt++;
                sumAllocated=A.get(i);
                if(sumAllocated>pages){
                    return false;
                }
            }
            else{
                sumAllocated+=A.get(i);
            }
        }
    if(cnt<B){
        return true;
    }
    return false;
    }
}
//********************Min number of Cows*****************************************************
*/
class TUF {
    static boolean isPossible(int a[],int n,int cows,int minDist) {
        int cntCows=1;
        int lastPlacedCow=a[0];
        for(int i=1;i<n;i++) {
            if (a[i]-lastPlacedCow>=minDist){
                cntCows++;
                lastPlacedCow=a[i];
            }
        }
        if (cntCows>=cows)
        return true;
        else
        return false;
    }
    public static void main(String args[]) {
        int n = 5, cows = 3;
        int a[]={1,2,8,4,9};
```

```
        Arrays.sort(a);
        int low=1,high=a[n-1]-a[0];
        while(low<=high) {
            int mid=(low+high)/2;
            if(isPossible(a,n,cows,mid)){
                low=mid+1;
            }else{
                high=mid-1;
            }
        }
        System.out.println("The largest minimum distance is " + high);


    }
}
```