```
import java.util.*;
 public class Solution {
 public static void main(String[] args) {
 Scanner s= new Scanner(System.in);
 int n=s.nextInt()
; int array[]=new int [n];
 for(int i=0;i<n;i++)
array[i]=s.nextInt();
Map<Integer, Integer> map = new HashMap<>();
 List<Integer> outputArray = new ArrayList<>();
 // Assign elements and their count in the list and map
for (int current : array) {
int count = map.getOrDefault(current, 0);
 map.put(current, count + 1);
 outputArray.add(current);
}
// Compare the map by value
SortComparator comp = new SortComparator(map);
 // Sort the map using Collections
 CLass Collections.sort(outputArray, comp);
 // Final Output
for (Integer i : outputArray) {
 System.out.print(i + " ");
} } }
 // Implement Comparator Interface to sort the values
class SortComparator implements Comparator<Integer> {
 private final Map<Integer, Integer> freqMap;
// Assign the specified map
 SortComparator(Map<Integer, Integer> tFreqMap) {
this.freqMap = tFreqMap;
```

```
} //
 Compare the values
@Override
public int compare(Integer k1, Integer k2) {
 // Compare value by frequency
 int freqCompare = freqMap.get(k2).compareTo(freqMap.get(k1));
 // Compare value if frequency is equal
int valueCompare = k1.compareTo(k2);
 // If frequency is equal, then just compare by value, otherwise –
// compare by the frequency.
if (freqCompare == 0)
return valueCompare;
else
 return freqCompare; } }
```

Game of death

Send Feedback

There are n people standing in a circle (numbered clockwise 1 to n) waiting to be executed. The counting begins at point 1 in the circle and proceeds around the circle in a fixed direction (clockwise). In each step, a certain number of people are skipped and the next person is executed. The elimination proceeds around the circle (which is becoming smaller and smaller as the executed people are removed), until only the last person remains, who is given freedom.
Given the total number of persons n and a number k which indicates that k-1 persons are skipped and kth person is killed in circle.
Find the position of last person to survive.
Sample input 1:

```
4 2
```

Sample output 1:

```
1
(Skip 1 , kill 2 , Skip 3 , Kill 4)
(Skip 1 , Kill 4, So 1 survives)
```

Sample input 2:

```
50 10
```

Sample output 2:

```
36
```

```
import java.io.*;
 import java.util.*;
 class Solution {
public static void main(String[] args) {
 Scanner s=new Scanner(System.in);
 int n=s.nextInt();
```

```
 int k=s.nextInt();
System.out.println(find(n,k));
 }
public static int find(int n,int k){
 if(n==1)
return 1;
 return (find(n-1,k)+k-1)%n+1;
}
}
```

Rainbow array

Send Feedback

An array is Rainbow if it has the following structure:

1. First a1 elements equal 1.
2. Next a2 elements equal 2.
3. Next a3 elements equal 3.
4. Next a4 elements equal 4.
5. Next a5 elements equal 5.
6. Next a6 elements equal 6.
7. Next a7 elements equal 7.
8. Next a6 elements equal 6.
9. Next a5 elements equal 5.
10. Next a4 elements equal 4.
11. Next a3 elements equal 3.
12. Next a2 elements equal 2.
13. Next a1 elements equal 1.

(Here a1,a2,a3... are number of elements).

14. ai can be any non-zero positive integer.
15. There are no other elements in array.

Find out if the given array is a Rainbow Array or not.

Input:

The first line of contains an integer N, denoting the number of elements in the given array.
The second line contains N space-separated integers A1, A2, ..., AN denoting the elements of array.

Output:

Output a line containing "yes" or "no" (without quotes) corresponding to the case if the array is rainbow array or not.

Constraints:

$7 \leq N \leq 100$
$1 \leq Ai \leq 10$

Sample Input 1:

19
1 2 3 4 4 5 6 6 6 7 6 6 6 5 4 4 3 2 1

Sample output 1:

Yes

Sample input 2:

12
1 2 3 4 5 6 6 5 4 3 2 1

Sample output 2:

No
(has no elements with value 7 after elements with value 6.)

Sample output 3:

14
1 1 2 3 4 5 6 7 6 5 4 3 2 1

Sample output 3:

No
(On the left we have two occurrences of 1, whereas on the right only one occurrence).

```
class Solution{ public static void israinbow(int [] arr){
```

```
 int N=arr.length;
if(N<13)
 { System.out.println("no");
 return;
}
int start = 0;
 int end = N-1;
 boolean isValid = true;
int cur = 0;
while(start != end && start < end) {
 if(arr[start] != arr[end]) {
 isValid = false; break;
 }
if(arr[start] < 1 || arr[start] > 7) {
 isValid = false; break; }
if(arr[start] != cur) {
 if(arr[start] != cur + 1) {
 isValid = false; break;
}
else { cur = arr[start];
 }
 }
start++; end--;
 }
if((arr[start] == 7 || cur == 7) && isValid) {
 System.out.println("yes");
 }
 else { System.out.println("no"); } } }
```

Reverse the Linked List

Send Feedback

Given a linked list of size N. You need to reverse every k nodes (where k is an input to the function) in the linked list.

Input:

First line contains length of linked list and next line contains the linked list elements.

Output:

Single line of output which contains the linked list with every k group elements reversed.

Example:

Input:

8 1 2 2 4 5 6 7 8 4

Output:

4 2 2 1 8 7 6 5

Explanation:

Since, k = 4. So, we have to reverse every group of two elements. Modified linked list is as 4, 2, 2, 1, 8, 7, 6, 5.

```
public class Solution{

   public static Node reverse(Node head, int k)

   {

      //Your code here

      //Make change in the linked list only

      //Return the head of the new Linked list

      Node back =null;

      Node curr = head;

      int n =k;



      while(curr!=null && n>0){

         Node next = curr.next;

         curr.next = back;

         back=curr;

         curr= next;

         n--;



         if(n==0){

            head.next= reverse(curr,k);

         }

      }

      return back;
```

```
    }

}
```

Binomial Expansion

Send Feedback

Given three integers A, X, and n, the task is to print terms of below binomial expression series.
(A+X)^n = a0*X^0 + a1*X^1 +….+an*X^n.

So at each position find the value of the general term and print that term(Print a0,a1*x*,*a2*x^2,....,an*x^n).

Input Format:
3 space seperated integers-> A,X,n.

Output Format:
The output is the terms of the binomial expression series.

Sample input:
1 2 6

Sample Output:
1 12 60 160 240 192 64

```
import java.io.*;

import java.util.*;

public class Solution{

 // function to print the series

 static void series(int A, int X, int n) {

 // Calculating and printing first

 // term

int term = (int)Math.pow(A, n);

 System.out.print(term + " ");

// Computing and printing

// remaining terms

 for (int i = 1; i <= n; i++) {

 // Find current term using

// previous terms We increment

 // power of X by 1, decrement

// power of A by 1 and compute

// nCi using previous term by

 // multiplying previous term
```

```
 // with (n - i + 1)/i
term = term * X * (n - i + 1) / (i * A);
System.out.print(term + " ");
 } }
 // main function started
 public static void main(String[] args) {
 Scanner s=new Scanner(System.in);
int A =s.nextInt();
 int X=s.nextInt();
int n=s.nextInt();
series(A, X, n); } }
```