Group multiple occurrence of array elements ordered by first occurrence

Given an unsorted array with repetitions, the task is to group multiple occurrence of individual elements. The grouping should happen in a way that the order of first occurrences of all elements is maintained.

Examples:

```
Input: arr[] = {5, 3, 5, 1, 3, 3}
Output:        {5, 5, 3, 3, 3, 1}

Input: arr[] = {4, 6, 9, 2, 3, 4, 9, 6, 10, 4}
Output:        {4, 4, 4, 6, 6, 9, 9, 2, 3, 10}
```

We strongly recommend to minimize your browser and try this yourself first.

Simple Solution is to use nested loops. The outer loop traverses array elements one by one. The inner loop checks if this is first occurrence, if yes, then the inner loop prints it and all other occurrences.

```
// A simple C++ program to group multiple occurrences of individual
// array elements
#include<iostream>
using namespace std;

// A simple method to group all occurrences of individual elements
void groupElements(int arr[], int n)
{
    // Initialize all elements as not visited
    bool *visited = new bool[n];
    for (int i=0; i<n; i++)
        visited[i] = false;

    // Traverse all elements
    for (int i=0; i<n; i++)
    {
        // Check if this is first occurrence
        if (!visited[i])
        {
            // If yes, print it and all subsequent occurrences
            cout << arr[i] << " ";
            for (int j=i+1; j<n; j++)
            {
                if (arr[i] == arr[j])
                {
                    cout << arr[i] << " ";
                    visited[j] = true;
                }
            }
        }
    }

    delete [] visited;
}

/* Driver program to test above function */
int main()
{
    int arr[] = {4, 6, 9, 2, 3, 4, 9, 6, 10, 4};
    int n = sizeof(arr)/sizeof(arr[0]);
    groupElements(arr, n);
    return 0;
}
```

Output:

```
4 4 4 6 6 9 9 2 3 10
```

Time complexity of the above method is $O(n^2)$.

Binary Search Tree based Method: The time complexity can be improved to O(nLogn) using self-balancing binary search tree like Red-Black Tree or AVL tree. Following is complete algorithm.
1) Create an empty Binary Search Tree (BST). Every BST node is going to contain an array element and its count.
2) Traverse the input array and do following for every element.
……..a) If element is not present in BST, then insert it with count as 0.
……..b) If element is present, then increment count in corresponding BST node.
3) Traverse the array again and do following for every element.
…….. If element is present in BST, then do following
……….a) Get its count and print the element 'count' times.
……….b) Delete the element from BST.

Time Complexity of the above solution is O(nLogn).

Hashing based Method: We can also use hashing. The idea is to replace Binary Search Tree with a Hash Map in above algorithm.

Below is Java Implementation of hashing based solution.

```
/* Java program to group multiple occurrences of individual array elements */
import java.util.HashMap;

class Main
{
    // A hashing based method to group all occurrences of individual elements
    static void orderedGroup(int arr[])
    {
        // Creates an empty hashmap
        HashMap<Integer, Integer> hM = new HashMap<Integer, Integer>();

        // Traverse the array elements, and store count for every element
        // in HashMap
        for (int i=0; i<arr.length; i++)
        {
           // Check if element is already in HashMap
           Integer prevCount = hM.get(arr[i]);
           if (prevCount == null)
                prevCount = 0;

           // Increment count of element element in HashMap
           hM.put(arr[i], prevCount + 1);
        }

        // Traverse array again
        for (int i=0; i<arr.length; i++)
        {
            // Check if this is first occurrence
            Integer count =  hM.get(arr[i]);
            if (count != null)
            {
                // If yes, then print the element 'count' times
                for (int j=0; j<count; j++)
                   System.out.print(arr[i] + " ");

                // And remove the element from HashMap.
                hM.remove(arr[i]);
            }
        }
    }

    // Driver method to test above method
    public static void main (String[] args)
    {
        int arr[] = {10, 5, 3, 10, 10, 4, 1, 3};
        orderedGroup(arr);
    }
}
```

Output:

```
10 10 10 5 3 3 4 1
```

Time Complexity of the above hashing based solution is Θ(n) under the assumption that insert, search and delete operations on HashMap take O(1) time.