Check if an Array is Sorted

Given an array of size **n**, write a program to check if it is sorted in ascending order or not. Equal values are allowed in an array and two consecutive equal values are considered sorted.

Examples:

```
Input : 20 21 45 89 89 90
Output : Yes

Input : 20 20 45 89 89 90
Output : Yes

Input : 20 20 78 98 99 97
Output : No
```

Recursive approach: The basic idea for the recursive approach:

```
1: If size of array is zero or one, return true.
2: Check last two elements of array, if they are
   sorted, perform a recursive call with n-1
   else, return false.
If all the elements will be found sorted, n will
eventually fall to one, satisfying Step 1.
```

Below is the implementation using recursion:

```
// Recursive approach to check if an
// Array is sorted or not
#include <bits/stdc++.h>
using namespace std;

// Function that returns 0 if a pair
// is found unsorted
int arraySortedOrNot(int arr[], int n)
{
    // Array has one or no element or the
    // rest are already checked and approved.
    if (n == 1 || n == 0)
        return 1;

    // Unsorted pair found (Equal values allowed)
    if (arr[n - 1] < arr[n - 2])
        return 0;

    // Last pair was sorted
    // Keep on checking
    return arraySortedOrNot(arr, n - 1);
}

// Driver code
int main()
{
    int arr[] = { 20, 23, 23, 45, 78, 88 };
    int n = sizeof(arr) / sizeof(arr[0]);
    if (arraySortedOrNot(arr, n))
        cout << "Yes\n";
    else        cout << "No\n";
}
```

Output:

```
Yes
```

Time Complexity: O(n)

Auxiliary Space: O(n) for Recursion Call Stack.

Another Recursive approach:

```
// C++ Recursive approach to check if an
// Array is sorted or not
#include <iostream>
using namespace std;

// Function that returns true if array is
// sorted in non-decreasing order.
```

```
bool arraySortedOrNot(int a[], int n)
{

    // Base case
    if (n == 1 || n == 0)
    {
        return true;
    }

    // Check if present index and index
    // previous to it are in correct order
    // and rest of the array is also sorted
    // if true then return true else return
    // false
    return a[n - 1] >= a[n - 2] &&
     arraySortedOrNot(a, n - 1);
}

// Driver code
int main()
{
    int arr[] = { 20, 23, 23, 45, 78, 88 };
    int n = sizeof(arr) / sizeof(arr[0]);

    // Function Call
    if (arraySortedOrNot(arr, n))
    {
        cout << "Yes" << endl;
    }
    else
    {
        cout << "No" << endl;
    }

    return 0;
}
```

Output

```
Yes
```

Time Complexity: O(n)

Auxiliary Space: O(n) for Recursion Call Stack.

Iterative approach: The idea is pretty much the same. The benefit of the iterative approach is it avoids the usage of recursion stack space and recursion overhead.

Below is the implementation using iteration:

```
// C++ program to check if an
// Array is sorted or not
#include <bits/stdc++.h>
using namespace std;

// Function that returns true if array is
// sorted in non-decreasing order.
bool arraySortedOrNot(int arr[], int n)
{
    // Array has one or no element
    if (n == 0 || n == 1)
        return true;

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

        // Unsorted pair found
        if (arr[i - 1] > arr[i])
            return false;

    // No unsorted pair found
    return true;
}

// Driver code
int main()
{
    int arr[] = { 20, 23, 23, 45, 78, 88 };
    int n = sizeof(arr) / sizeof(arr[0]);
    if (arraySortedOrNot(arr, n))
        cout << "Yes\n";
    else        cout << "No\n";
}
```

Output:

```
Yes
```

Time Complexity: O(n)

Auxiliary Space: O(1)