Count Strictly Increasing Subarrays

Given an array of integers, count number of subarrays (of size more than one) that are strictly increasing.

Expected Time Complexity : O(n)

Expected Extra Space: O(1)

Examples:

```
Input: arr[] = {1, 4, 3}
Output: 1
There is only one subarray {1, 4}

Input: arr[] = {1, 2, 3, 4}
Output: 6
There are 6 subarrays {1, 2}, {1, 2, 3}, {1, 2, 3, 4}
                {2, 3}, {2, 3, 4} and {3, 4}

Input: arr[] = {1, 2, 2, 4}
Output: 2
There are 2 subarrays {1, 2} and {2, 4}
```

We strongly recommend that you click here and practice it, before moving on to the solution.

A **Simple Solution** is to generate all possible subarrays, and for every subarray check if subarray is strictly increasing or not. Worst case time complexity of this solution would be $O(n^3)$.

A **Better Solution** is to use the fact that if subarray arr[i:j] is not strictly increasing, then subarrays arr[i:j+1], arr[i:j+2], .. arr[i:n-1] cannot be strictly increasing. Below is C++ program based on above idea.

```
// C++ program to count number of strictly
// increasing subarrays
#include<bits/stdc++.h>
using namespace std;

int countIncreasing(int arr[], int n)
{
   // Initialize count of subarrays as 0
   int cnt = 0;

   // Pick starting point
   for (int i=0; i<n; i++)
   {
      // Pick ending point
      for (int j=i+1; j<n; j++)
      {
         if (arr[j] > arr[j-1])
            cnt++;

         // If subarray arr[i..j] is not strictly
         // increasing, then subarrays after it , i.e.,
         // arr[i..j+1], arr[i..j+2], .... cannot
         // be strictly increasing
         else
            break;
      }
   }
   return cnt;
}

// Driver program
int main()
{
  int arr[] = {1, 2, 2, 4};
  int n = sizeof(arr)/sizeof(arr[0]);
  cout << "Count of strictly increasing subarrays is "
       << countIncreasing(arr, n);
  return 0;
}
```

Output:

```
Count of strictly increasing subarrays is 2
```

Time complexity of the above solution is O(m) where m is number of subarrays in output

This problem and solution are contributed by Rahul Agrawal.

An **Efficient Solution** can count subarrays in O(n) time. The idea is based on fact that a sorted subarray of length ‘len’ adds len*(len-1)/2 to result. For example, {10, 20, 30, 40} adds 6 to the result.

```
// C++ program to count number of strictly
// increasing subarrays in O(n) time.
#include<bits/stdc++.h>
using namespace std;

int countIncreasing(int arr[], int n)
{
    int cnt = 0;  // Initialize result

    // Initialize length of current increasing
    // subarray
    int len = 1;

    // Traverse through the array
    for (int i=0; i < n-1; ++i)
    {
        // If arr[i+1] is greater than arr[i],
        // then increment length
        if (arr[i + 1] > arr[i])
            len++;

        // Else Update count and reset length
        else
        {
            cnt += (((len - 1) * len) / 2);
            len = 1;
        }
    }

    // If last length is more than 1
    if (len > 1)
        cnt += (((len - 1) * len) / 2);

    return cnt;
}

// Driver program
int main()
{
  int arr[] = {1, 2, 2, 4};
  int n = sizeof(arr)/sizeof(arr[0]);
  cout << "Count of strictly increasing subarrays is "
       << countIncreasing(arr, n);
  return 0;
}
```

Output:

```
Count of strictly increasing subarrays is 2
```