geeksforgeeks.org

Maximum sum of lengths of non-overlapping subarrays with k as the max element.

4-5 minutes

Find the maximum sum of lengths of non-overlapping subarrays (contiguous elements) with k as the maximum element.
Example 1:

```
Input : arr[] = {2, 1, 4, 9, 2, 3, 8, 3, 4}
        k = 4
Output : 5
{2, 1, 4} => Length = 3
{3, 4} => Length = 2
So, 3 + 2 = 5 is the answer
```

Example 2:

```
Input : arr[] = {1, 2, 3, 2, 3, 4, 1}
        k = 4
Output : 7
{1, 2, 3, 2, 3, 4, 1} => Length = 7
```

Example 3:

```
Input : arr = {4, 5, 7, 1, 2, 9, 8, 4, 3, 1}
        k = 4
```

```
Ans = 4
{4} => Length = 1
{4, 3, 1} => Length = 3
So, 1 + 3 = 4 is the answer
```

question source : http://www.geeksforgeeks.org/amazon-interview-experience-set-376-campus-internship/

Algorithm :

```
Traverse the array starting from first element
   Take a loop and keep on incrementing count
   If element is less than equal to k
       if array element is equal to k, then mark
       a flag

   If flag is marked, add this count to answer

   Take another loop and traverse the array
   till element is greater than k
return ans
```

- C++
- Java
- Python

C++

```
#include <bits/stdc++.h>

using namespace std;

int calculateMaxSumLength(int arr[], int n, int k)
```

```
{
    int ans = 0;
    int count = 0;
    int flag = 0;
    for (int i = 0; i < n;) {
        count = 0;
        flag = 0;
        while (arr[i] <= k && i < n) {
            count++;
            if (arr[i] == k)
                flag = 1;
            i++;
        }
        if (flag == 1)
            ans += count;
        while (arr[i] > k && i < n)
            i++;
    }
    return ans;
}
int main()
{
```

```
    int arr[] = { 4, 5, 7, 1, 2, 9, 8, 4, 3, 1 };
    int size = sizeof(arr) / sizeof(arr[0]);
    int k = 4;
    int ans = calculateMaxSumLength(arr, size, k);
    cout << "Max Length :: " << ans << endl;
    return 0;
}
```

Java

```
public class GFG
{
    static int calculateMaxSumLength(int arr[], int
n, int k) {
        int ans = 0;
        int count = 0;
        int flag = 0;
        for (int i = 0; i < n;) {
            count = 0;
            flag = 0;
            while (i < n && arr[i] <= k) {
                count++;
                if (arr[i] == k)
```

```
                    flag = 1;
                i++;
            }
            if (flag == 1)
                ans += count;
            while (i < n && arr[i] > k)
                i++;
        }
        return ans;
    }
    public static void main(String[] args) {
        int arr[] = { 4, 5, 7, 1, 2, 9, 8, 4, 3, 1
};
        int size = arr.length;
        int k = 4;
        int ans = calculateMaxSumLength(arr, size,
k);
        System.out.println("Max Length :: " +
ans);
    }
}
```

Python

```
def calculateMaxSumLength(arr, n, k):
    ans = 0
    for i in range(n):
        count = 0
        flag = 0
        while i < n and arr[i] <= k :
            count = count + 1
            if arr[i] == k:
                flag = 1
            i = i + 1
        if flag == 1:
            ans = ans + count
        while i < n and arr[i] > k :
            i = i + 1
    return ans
arr = [4, 5, 7, 1, 2, 9, 8, 4, 3, 1]
size = len(arr)
k = 4
ans = calculateMaxSumLength(arr, size, k)
print "Max Length ::",ans
```

Output:

```
Max Length :: 4
```

Time Complexity : O(n)
It may look like O(n2), but if you take a closer look, array is traversed only once

This article is contributed by Mandeep Singh. If you like GeeksforGeeks and would like to contribute, you can also write an article using contribute.geeksforgeeks.org or mail your article to contribute@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.