Longest Consecutive Subsequence

Given an array of integers, find the length of the longest sub-sequence such that elements in the subsequence are consecutive integers, the consecutive numbers can be in any order.

Examples

```
Input: arr[] = {1, 9, 3, 10, 4, 20, 2};
Output: 4
The subsequence 1, 3, 4, 2 is the longest subsequence
of consecutive elements

Input: arr[] = {36, 41, 56, 35, 44, 33, 34, 92, 43, 32, 42}
Output: 5
The subsequence 36, 35, 33, 34, 32 is the longest subsequence
of consecutive elements.
```

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

One Solution is to first sort the array and find the longest subarray with consecutive elements. Time complexity of this solution is O(nLogn). Thanks to Hao.W for suggesting this solution here.

We can solve this problem in O(n) time using an **Efficient Solution**. The idea is to use Hashing. We first insert all elements in a Hash. Then check all the possible starts of consecutive subsequences. Below is complete algorithm.

```
1) Create an empty hash.
2) Insert all array elements to hash.
3) Do following for every element arr[i]
....a) Check if this element is the starting point of a
       subsequence.  To check this, we simply look for
       arr[i] - 1 in hash, if not found, then this is
       the first element a subsequence.

       If this element is a first element, then count
       number of elements in the consecutive starting
       with this element.

       If count is more than current res, then update
       res.
```

Below is C++ implementation of above algorithm.

C/C++ Java Python

```
// C++ program to find longest contiguous subsequence
#include<bits/stdc++.h>
using namespace std;

// Returns length of the longest contiguous subsequence
int findLongestConseqSubseq(int arr[], int n)
{
    unordered_set<int> S;
    int ans = 0;

    // Hash all the array elements
    for (int i = 0; i < n; i++)
        S.insert(arr[i]);

    // check each possible sequence from the start
    // then update optimal length
    for (int i=0; i<n; i++)
    {
        // if current element is the starting
        // element of a sequence
        if (S.find(arr[i]-1) == S.end())
        {
            // Then check for next elements in the
            // sequence
            int j = arr[i];
            while (S.find(j) != S.end())
                j++;

            // update  optimal length if this length
            // is more
            ans = max(ans, j - arr[i]);
        }
    }
    return ans;
}

// Driver program
int main()
{
    int arr[] =  {1, 9, 3, 10, 4, 20, 2};
    int n = sizeof arr/ sizeof arr[0];
    cout << "Length of the Longest contiguous subsequence is "
         << findLongestConseqSubseq(arr, n);
    return 0;
}
```

Run on IDE

Output:

```
Length of the Longest contiguous subsequence is 4
```

Time Complexity: At first look, time complexity looks more than O(n). If we take a closer look, we can notice that it is O(n) under the assumption that hash insert and search take O(1) time. The function S.find() inside the while loop is called at most twice for every element. For example, consider the case when all array elements are consecutive. In this case, the outer find is called for every element, but we go inside the if condition only for the smallest element. Once we are inside the if condition, we call find() one more time for every other element.