17. Longest Consecutive Sequence

128. Longest Consecutive Sequence

Medium 15.4K 631

Companies

Given an unsorted array of integers `nums`, return *the length of the longest consecutive elements sequence.*

You must write an algorithm that runs in `O(n)` time.

Example 1:

```
Input: nums = [100,4,200,1,3,2]
Output: 4
Explanation: The longest consecutive elements sequence is [1, 2, 3, 4].
Therefore its length is 4.
```

Example 2:

```
Input: nums = [0,3,7,2,5,8,4,6,0,1]
Output: 9
```

Longest Consective Sequence:

arr[] = [102, 4, 100, 1, 101, 3, 2, 1, 1]

1 2 3 4 ⇒ leng ⇒ (4) ⇒ max return it.

100 101 102 ⇒ length ⇒ 3

1) Brute Solution

→ Pick an element 'a' then look for a+1, a+2

x, x+1, x+2

→ Start iterating from array and count = 1

arr[] = [102, 4, 100, 1, 101, 3, 2, 1, 1]

cnt = 1

x = ~~102~~ → next is 103 (So look for it in the array) as it's not then move to next element

x = 4, (same process / linear search)
↙ 5

x = 100
↙ 101 (it's there in arr so increase the counter

cnt = 2
↙ 102 (it's there so)

cnt = 3
103 (not there so counter will be 3)

Now x = 1
(Same process you look for 2, 3, 4 ...)
then return the max cnter.

```
longest = 1
for (i = 0, i < n ; i ++)
{
    x = arr [i]
    cnt = 1 (Initially one element)
    while (linear search (arr, x + 1) == true)
    {
        x = x + 1 ;
        cnt = cnt + 1 ;
    }
}
ls (arr , num)
{ for (i = 0, i < n ; i ++)
  { if (arr [i] == n)
        return true
  }
```

```
    }
    return false;
  }
```

T.C ⇒ $O(n^2)$ S.C ⇒ $O(1)$

2) Better Solution

arr[] = {100, 102, 100, 101, 101, 4, 3, 2, 3, 2, 1, 1, 1, 2}

↓ sort

{1, 1, 1, 2, 2, 2, 3, 3, 4, 100, 100, 101, 101, 102}

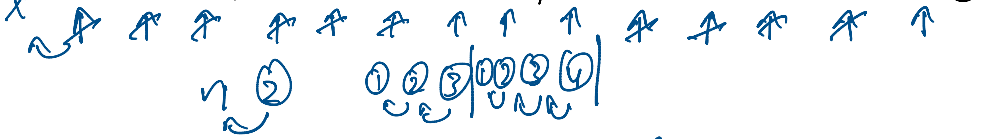
{1, 2, 3, 4} {100, 101, 102}

Initution for sorting is they are kind of club together somewhere together.

cntCurr = ~~0 1 2 3 4~~ lastSmaller = ~~INT_MIN~~ ~~1 2 3 4 100~~ longest = ~~1 2 3~~ 4

↳ (take sequences) ~~= 3~~ ↳ (keep track of last smallest guy that is seen) ~~101 102~~ ↳ return

{1, 1, 1, 2, 2, 2, 3, 3, 4, 100, 100, 101, 101, 102}



100

it should be 99

So cnt current = 1

start from fresh

→ Once iteration is compled return the longest.

sort( ), longest = 1, cntcurr = 0, lastSmaller = INT_MIN

sort( ), longest = 1, cnt curr = 0, last_Smaller = INT_MIN

for (i = 0 ⟶ n)

{

if (arr[i] − 1 == last_smaller)

{

cnt = cnt + 1

lastSmaller = arr[i];

}

else if (arr[i] == last_smaller)

{

// don't do anything (E.g. 1, 1, 1) ← Don't update

}

else (arr[i] != last_smaller)

{

cnt = 1

lastSmaller = arr[i];

(E.g. 1, 101) (99 is missing) (Start new sequence)

}

longest = max(longest, currCnt)

}

Code Studio

```
int longestSuccessiveElements(vector<int>&nums) {
    if(nums.size() == 0)  return 0;
    sort(nums.begin(), nums.end());
    int n = nums.size();

    int lastSmaller = INT_MIN;
    int longest = 1;
    int count = 0;
    for(int i = 0;i<n;i++){
        if(nums[i] - 1 == lastSmaller){
            count += 1;
            lastSmaller = nums[i];
        }
        else if(lastSmaller != nums[i]){
            count = 1;
            lastSmaller = nums[i];
        }
        longest = max(longest, count);
    }
    return longest;

}
```

Leet Code

```
class Solution {
public:
    int longestConsecutive(vector<int>& nums) {
    if(nums.size() == 0)  return 0;
    sort(nums.begin(), nums.end());
    int n = nums.size();

    int lastSmaller = INT_MIN;
    int longest = 1;
    int count = 0;
    for(int i = 0;i<n;i++){
        if(nums[i] - 1 == lastSmaller){
            count += 1;
            lastSmaller = nums[i];
        }
        else if(lastSmaller != nums[i]){
            count = 1;
            lastSmaller = nums[i];
        }
        longest = max(longest, count);
    }
    return longest;

    }
};
```

GFG

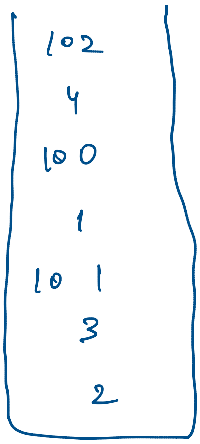
```
class Solution{
  public:
    // arr[] : the input array
    // N : size of the array arr[]

    //Function to return length of longest subsequence of consecutive integers.
    int findLongestConseqSubseq(int arr[], int n)
    {
        //Your code here
        sort(arr,arr+n);
        int ct=1;
        int ans=INT_MIN;
        for(int i=0;i<n;i++){
            if(arr[i+1]-arr[i]==0){

            }
            else if(arr[i+1]-arr[i]==1){
                ct++;
            }
            else{
                ct=1;
            }
            ans=max(ans,ct);

        }
        return ans;
    }
};
// } Driver Code Ends
```

T.C $\Rightarrow O(N \log N) + O(N)$

3rd Approach

arr[] = [102, 4, 100, 1, 101, 3, 2, 1, 1]

- Put everything in set (unique element) data structure
- Unordered set in C++
- Start iterating element by element in set d.s.



→ ✗ 102 → it's not starting point, as 101 is there in d.s.
✓ 101

→ 4 (Same 3 is already there)

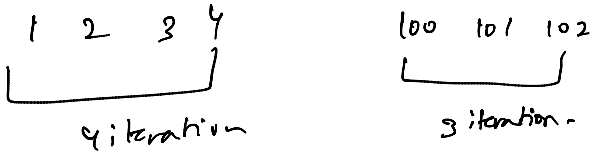
→ 100 (99 is not there, so this is probably start point and check if 101 is there, 102, 103)
(100, 101, 102)
cnt = 3 → longest as of now

→ 1 ('0' is not there, so might be starting point & check 2, 3, are there in d.s or not
(1, 2, 3, 4) cnt = 4 → longest

→ 101 (no need to do for this as it's mid, unnecessary event)
100, 101, 102

→ 3 ( ) 2 [3] 4

→ 2 ( ) 1 [2] 3



→ Always start looking from 1st element.

```cpp
int longestSuccessiveElements(vector<int>&a) {
    int n = a.size();
    if(n == 0) return 0;
    int longest = 1;
    unordered_set<int> st;
    for(int i = 0;i<n;i++){
        st.insert(a[i]);
    }

    // iterate over the set
    for(auto it: st){
        if(st.find(it - 1) == st.end()){
            int cnt = 1;
            int x = it;
            while(st.find(x+1) != st.end()){
                x = x + 1;
                cnt = cnt + 1;
            }
            longest = max(longest, cnt);
        }
    }
    return longest;
}
```

O(N)

```cpp
class Solution {
public:
    int longestConsecutive(vector<int>& a) {
        int n = a.size();
        if(n == 0) return 0;
        int longest = 1;
        unordered_set<int> st;
        for(int i = 0;i<n;i++){
            st.insert(a[i]);
        }

        // iterate over the set
        for(auto it: st){
            if(st.find(it - 1) == st.end()){
                int cnt = 1;
                int x = it;
                while(st.find(x+1) != st.end()){
                    x = x + 1;
                    cnt = cnt + 1;
                }
                longest = max(longest, cnt);
            }
        }
        return longest;
    }
};
```

```cpp
class Solution{
    public:
    // arr[] : the input array
    // N : size of the array arr[]
    int findLongestConseqSubseq(int arr[], int n)
    {
        //Your code here
        unordered_set<int> s;
        for(int i=0;i<n;i++)
        {
            s.insert(arr[i]);
        }
        int longestStreak=0;
        for(int i=0;i<n;i++)
        {
            int currentElement=arr[i];
            int previousElement=currentElement-1;
            int currentStreak=1;
            if(s.find(previousElement)==s.end())
            {
                while(s.find(currentElement+1)!=s.end())
                {
                    currentElement++;
                    currentStreak++;
                }
            }
            longestStreak=max(currentStreak,longestStreak);

        }
        return longestStreak;
    }
};
```

T.C ⇒ O(1) + O(N)

↓

unordered set (for best/average)

O(N) for worst

S.C ⇒ O(N)