128. Longest Consecutive Sequence (Medium)

Given an array of integers `nums`, return *the length* of the longest consecutive sequence of elements that can be formed.

A *consecutive sequence* is a sequence of elements in which each element is exactly `1` greater than the previous element. The elements do *not* have to be consecutive in the original array.

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

Example 1:

```
Input: nums = [2,20,4,10,3,4,5]

Output: 4
```

Explanation: The longest consecutive sequence is `[2, 3, 4, 5]`.

Example 2:

```
Input: nums = [0,3,2,5,4,6,1,1]

Output: 7
```

Constraints:

- `0 <= nums.length <= 1000`
- `10^9 <= nums[i] <= 10^9`

▼ 思路:

作法一：排序後比對

T:*O*(*nlogn*) , S:*O*(1)

作法二：unordered_map紀錄每段區間長度值

T:$O(n)$, S:$O(n)$

作法一

```
class Solution {
public:
    int longestConsecutive(vector<int>& nums) {
        if(nums.size()<1) return 0; //notice
        sort(nums.begin(),nums.end());
        int lcs=1,tmp=1;
        for(int i=0;i<nums.size()-1;i++){
            if(nums[i+1] == nums[i]+1) tmp+=1;
            else if(nums[i+1] == nums[i]){}
            else{
                lcs = (tmp>lcs) ? tmp:lcs;
                tmp = 1;
            }
        }
        lcs = (tmp>lcs) ? tmp:lcs;
        return lcs;
    }
};
```

作法二

```
class Solution {
public:
    int longestConsecutive(vector<int>& nums) {
        if(nums.size()<1) return 0; //notice
        unordered_map<int,int> umap;
        int lcs=1;
        for(int num:nums){
            if(umap[num]) continue;//exsist
            //umap[num] records the length of consecutive nums
            if(umap.find(num-1)==umap.end() && umap.find(num+1)==umap.end()){
                umap[num]=1;
```

```
            continue;
        }//non-num-non
        else if(umap.find(num+1)==umap.end()){
            umap[num] = umap[num-umap[num-1]] = umap[num-1]+1;
            lcs = max(umap[num],lcs);
            continue;
        }//exsist-num-non
        //renew the leftest value
        else if(umap.find(num-1)==umap.end()){
            umap[num] = umap[num+umap[num+1]] = umap[num+1]+1;
            lcs = max(umap[num],lcs);
            continue;
        }//non-num-exsist
        //renew the rightest value
        umap[num] = umap[num-umap[num-1]] = umap[num+umap[num+1]]
        = umap[num-1] + umap[num+1] +1; //exsist-num-exsist
        //renew the leftest and the rightest value
        lcs = max(umap[num],lcs);
    }
    return lcs;
  }
};
```

作法二精簡

```
class Solution {
public:
  int longestConsecutive(vector<int>& nums) {
    if(nums.size()<1) return 0; //notice
    unordered_map<int,int> umap;
    int lcs=1;
    for(int num:nums){
      if(!umap[num]){
        umap[num] = umap[num-1]+umap[num+1]+1;
        umap[num-umap[num-1]] = umap[num];
        umap[num+umap[num+1]] = umap[num];
```

```
            }
            lcs = max(umap[num],lcs);
        }
        return lcs;
    }
};
```