LONGEST CONSECUTIVE SEQUENCE

☆ In this problem, we are supposed to return the length of the longest sequence which has consecutive numbers.

Brute force solution is to look for a consecutive sequence for each number and record the one with the highest length.

Better solution is to sort the entire array and then start iterating through the entire array We track the last minimum, current sequence length and max length. We start by checking if the element at our pointer is one more than last minimum, if it is, we add 1 to the current length, change it to the last maximum. If it isn't, we start fresh by resetting length to 1 and changing last maximum. Although if both are equal we just skip it.

Pseudocode :

```
LargestConsecutiveSequence(arr, N) {
    length, max length, current = 0, 1, INT_MIN
```

```
for(int i = 0 → N)  {
    if (arr[i] - 1 == current) {
        length += 1
        current = arr[i]
    } else if (arr[i] == current) {
        continue
    } else {
        maxlength = max(length, maxlength)
        current = arr[i]
        length = 1
    }
    maxlength = max(length, maxlength)
}
}
```

Optimal Solution is to put all elements in a set. Then we iterate throughout the set. For each element, instead of looking for the element ahead, we look for the element behind it. If it exists, we keep looking back, after which we quit because now we know the element will exist and will be acted later upon. If the element behind it doesn't exist we look forward.

Pseudocode :

```
longestConsecutiveSequence(arr, N) {
    longest = 1
    set st
    for(i = 0 → N)  {
```

```
        st.insert(arr[i])
    }
    for(auto it : set) {
        if(st.find(it - 1) == st.end()) {
            cnt = 1
            x = it
            while(st.find(x + 1) != st.end()) {
                x += 1
                cnt += 1
            }
            longest = max(longest, cnt)
        }
    }
    return longest
}
```

Our principle is to skip those iterations where we know previous element exists as we will get back to it anyway