Longest Consecutive Sequence in an Array

Problem Statement: You are given an array of ‘N’ integers. You need to find the length of the longest sequence which contains the consecutive elements.

Examples:

```
Example 1:

Input: [100, 200, 1, 3, 2, 4]

Output: 4

Explanation: The longest consecutive subsequence is 1, 2, 3, and 4.

Input: [3, 8, 5, 7, 6]

Output: 4

Explanation: The longest consecutive subsequence is 5, 6, 7, and 8.
```

Solution

Disclaimer: Don’t jump directly to the solution, try it out yourself first.

Solution 1: (Brute force)

Approach: We can simply sort the array and run a for loop to find the longest consecutive sequence

```
public static int longestConsecutive(int[] nums) {
    if(nums.length == 0 || nums == null){
      return 0;
    }
    Arrays.sort(nums);
    int ans = 1;
    int prev = nums[0];
    int cur = 1;

    for(int i = 1;i < nums.length;i++){
      if(nums[i] == prev+1){
        cur++;
      }
      else if(nums[i] != prev){
        cur = 1;
      }
      prev = nums[i];
      ans = Math.max(ans, cur);
    }
    return ans;
  }
```

Time Complexity: We are first sorting the array which will take O(N * log(N)) time and then we are running a for loop which will take O(N) time. Hence, the overall time complexity will be O(N * log(N)).
Space Complexity: The space complexity for the above approach is O(1) because we are not using any auxiliary space

Solution 2: (Optimal Approach)

Approach: We will first push all are elements in the HashSet. Then we will run a for loop and check for any number(x) if it is the starting number of the consecutive sequence by checking if the HashSet contains (x-1) or not. If ‘x’ is the starting number of the consecutive sequence we will keep searching for the numbers y = x+1, x+2, x+3, ….. And stop at the first ‘y’ which is not present in the HashSet. Using this we can calculate the length of the longest consecutive subsequence.

```
public static int longestConsecutive(int[] nums) {
        Set < Integer > hashSet = new HashSet < Integer > ();
        for (int num: nums) {
            hashSet.add(num);
        }
        int longestStreak = 0;
        for (int num: nums) {
            if (!hashSet.contains(num - 1)) {
                int currentNum = num;
                int currentStreak = 1;
                while (hashSet.contains(currentNum + 1)) {
                    currentNum += 1;
                    currentStreak += 1;
                }
                longestStreak = Math.max(longestStreak, currentStreak);
            }
        }
        return longestStreak;
    }
```

Time Complexity: The time complexity of the above approach is O(N) because we traverse each consecutive subsequence only once.

Space Complexity: The space complexity of the above approach is O(N) because we are maintaining a HashSet.