128. Longest Consecutive Sequence

Nov. 3, 2017 | 125.8K views

Average Rating: 4.58 (69 votes)

Given an unsorted array of integers, find the length of the longest consecutive elements sequence.

Your algorithm should run in O(*n*) complexity.

Example:

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

Approach 1: Brute Force

Intuition

Because a sequence could start at any number in `nums`, we can exhaust the entire search space by building as long a sequence as possible from every number.

Algorithm

The brute force algorithm does not do anything clever - it just considers each number in `nums`, attempting to count as high as possible from that number using only numbers in `nums`. After it counts too high (i.e. `currentNum` refers to a number that `nums` does not contain), it records the length of the sequence if it is larger than the current best. The algorithm is necessarily optimal because it explores every possibility.

Java | Python | Copy

```python
class Solution:
    def longestConsecutive(self, nums):
        longest_streak = 0

        for num in nums:
            current_num = num
            current_streak = 1

            while current_num + 1 in nums:
                current_num += 1
                current_streak += 1

            longest_streak = max(longest_streak, current_streak)

        return longest_streak
```

Complexity Analysis

- Time complexity : $O(n^3)$.

  The outer loop runs exactly n times, and because `currentNum` increments by 1 during each iteration of the `while` loop, it runs in $O(n)$ time. Then, on each iteration of the `while` loop, an $O(n)$ lookup in the array is performed. Therefore, this brute force algorithm is really three nested $O(n)$ loops, which compound multiplicatively to a cubic runtime.

- Space complexity : $O(1)$.

  The brute force algorithm only allocates a handful of integers, so it uses constant additional space.

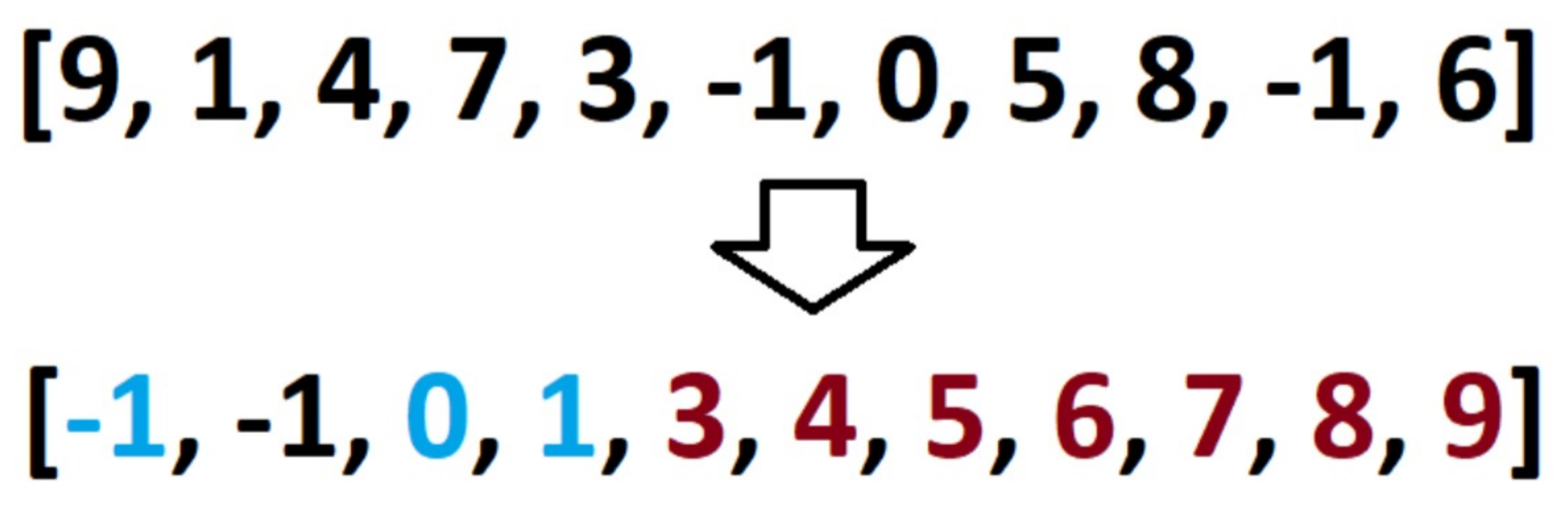
Approach 2: Sorting

Intuition

If we can iterate over the numbers in ascending order, then it will be easy to find sequences of consecutive numbers. To do so, we can sort the array.

Algorithm

Before we do anything, we check for the base case input of the empty array. The longest sequence in an empty array is, of course, 0, so we can simply return that. For all other cases, we sort `nums` and consider each number after the first (because we need to compare each number to its previous number). If the current number and the previous are equal, then our current sequence is neither extended nor broken, so we simply move on to the next number. If they are unequal, then we must check whether the current number extends the sequence (i.e. `nums[i] == nums[i-1] + 1`). If it does, then we add to our current count and continue. Otherwise, the sequence is broken, so we record our current sequence and reset it to 1 (to include the number that broke the sequence). It is possible that the last element of `nums` is part of the longest sequence, so we return the maximum of the current sequence and the longest one.



Here, an example array is sorted before the linear scan identifies all consecutive sequences. The longest sequence is colored in red.

Java | Python | Copy

```python
class Solution:
    def longestConsecutive(self, nums):
        if not nums:
            return 0

        nums.sort()

        longest_streak = 1
        current_streak = 1

        for i in range(1, len(nums)):
            if nums[i] != nums[i-1]:
                if nums[i] == nums[i-1]+1:
                    current_streak += 1
                else:
                    longest_streak = max(longest_streak, current_streak)
                    current_streak = 1

        return max(longest_streak, current_streak)
```

Complexity Analysis

- Time complexity : $O(nlgn)$.

  The main `for` loop does constant work n times, so the algorithm's time complexity is dominated by the invocation of `sort`, which will run in $O(nlgn)$ time for any sensible implementation.

- Space complexity : $O(1)$ (or $O(n)$).

  For the implementations provided here, the space complexity is constant because we sort the input array in place. If we are not allowed to modify the input array, we must spend linear space to store a sorted copy.

Approach 3: HashSet and Intelligent Sequence Building

Intuition

It turns out that our initial brute force solution was on the right track, but missing a few optimizations necessary to reach $O(n)$ time complexity.

Algorithm

This optimized algorithm contains only two changes from the brute force approach: the numbers are stored in a `HashSet` (or `Set`, in Python) to allow $O(1)$ lookups, and we only attempt to build sequences from numbers that are not already part of a longer sequence. This is accomplished by first ensuring that the number that would immediately precede the current number in a sequence is not present, as that number would necessarily be part of a longer sequence.

Java | Python | Copy

```python
class Solution:
    def longestConsecutive(self, nums):
        longest_streak = 0
        num_set = set(nums)

        for num in num_set:
            if num - 1 not in num_set:
                current_num = num
                current_streak = 1

                while current_num + 1 in num_set:
                    current_num += 1
                    current_streak += 1

                longest_streak = max(longest_streak, current_streak)

        return longest_streak
```

Complexity Analysis

- Time complexity : $O(n)$.

  Although the time complexity appears to be quadratic due to the `while` loop nested within the `for` loop, closer inspection reveals it to be linear. Because the `while` loop is reached only when `currentNum` marks the beginning of a sequence (i.e. `currentNum-1` is not present in `nums`), the `while` loop can only run for n iterations throughout the entire runtime of the algorithm. This means that despite looking like $O(n \cdot n)$ complexity, the nested loops actually run in $O(n+n) = O(n)$ time. All other computations occur in constant time, so the overall runtime is linear.

- Space complexity : $O(n)$.

  In order to set up $O(1)$ containment lookups, we allocate linear space for a hash table to store the $O(n)$ numbers in `nums`. Other than that, the space complexity is identical to that of the brute force solution.

Rate this article:

Previous Next

Comments: 59

Sort By

Type comment here... (Markdown is supported)

Preview Post

arboleda1298 ★ 182 March 14, 2019 9:07 PM

Why is the tag for this Union-Find?

150 Share Reply

SHOW 11 REPLIES

cryptoandy ★ 164 December 6, 2018 3:05 AM

For those who got confused by if the last solution is O(n^2) or O(n), please take a close look at the entering of the logic: if(!num_set.contains(num-1)).
That means, for example, 6,5,4,3,2,1 input, only the value 1 is valid for the loop(all other values have its value - 1 in the set), that is O(n).
Another corner example, 2, 5, 6, 7, 9, 11. All of these numbers are the "entrance" for the logic but the

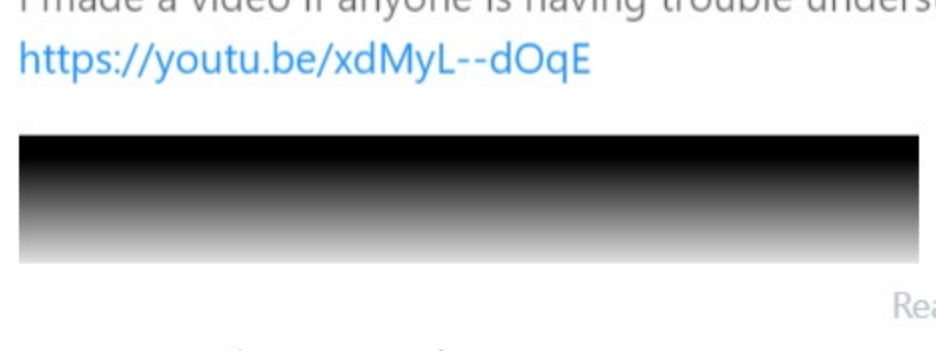
Read More

42 Share Reply

SHOW 4 REPLIES

terrible_whiteboard ★ 627 May 19, 2020 6:24 PM

I made a video if anyone is having trouble understanding the solution (clickable link)
https://youtu.be/xdMyL--dOqE



Read More

20 Share Reply

ABattle ★ 48 January 17, 2019 3:16 AM

Can be reduced to a simple dfs/bfs

```
public class Solution {
    public int longestConsecutive(int[] nums) {
        Set<Integer> seen = new HashSet<>();
```

Read More

17 Share Reply

SHOW 1 REPLY

wcarvalho ★ 36 January 9, 2018 2:19 AM

wouldn't [100,99, 98, ..., 1] (i.e. an array in reverse order) make this run in $$O(n^2)$$ because for every num, it have to go through the entire set so far?

25 Share Reply

SHOW 7 REPLIES

AnishSGM ★ 23 May 22, 2018 7:37 PM

If the list is very long, then the O(n) hash-set solution is a drain on memory so, not an elegant solution. If the list is small, then O(nlogn) shouldn't make much of a difference either. It's a stupid question.

21 Share Reply

SHOW 2 REPLIES

Albert2522 ★ 31 January 1, 2018 1:51 PM

I think, using a HashSet is a bit confusing in terms of the time complexity. For many people (and to myself in particular) it wasn't obvious that we spend O(n).
I prefer using a HashMap here in the following way:

1. Firstly, put every element in a HashMap and mark it as unexplored.

Read More

9 Share Reply

defunator ★ 5 February 2, 2019 12:45 PM

Excuse me, but as I know, HashSet works in AVERAGE O(1) for lookup.
So it is not O(n), it is expected O(n)(in some cases the constant is too big, so it is more appropriate to use set or something else with log but more determined).

5 Share Reply

SHOW 1 REPLY

anmingyu11 ★ 425 December 25, 2017 12:35 PM

class Solution {
public int longestConsecutive(int[] nums) {
final HashSet mySet = new HashSet();
for (int i : nums) mySet.add(i);

Read More

4 Share Reply

SHOW 1 REPLY

afsf11 ★ 10 July 24, 2019 6:00 PM

Why is this hard? Easier than many medium questions

3 Share Reply

< 1 2 3 4 5 6 >