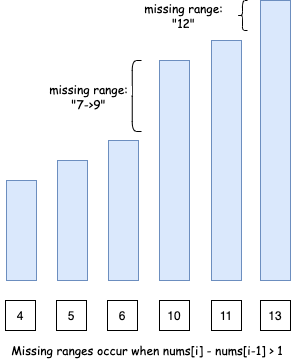
Solution Article

Approach 1: Linear Scan

**Intuition and Algorithm**

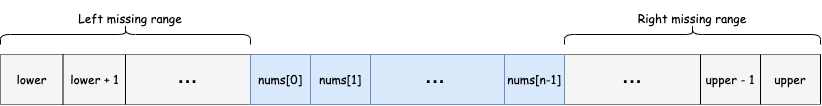
Since the input array, nums, is sorted ascendingly and all the elements in it are within the given [lower, upper] bounds, we can simply check consecutive elements to see if they differ by one or not. If they don't, then we have found a missing range.

* When nums[i] - nums[i-1] == 1, we know that there are no missing elements between nums[i-1] and nums[i].
* When nums[i] - nums[i-1] > 1, we know that the range of elements, [nums[i-1] + 1, nums[i] - 1], is missing.



However, there are two edge cases:

* Edge case 1: If we don't start with lower as the first element of the array, we will need to include [lower, num[0] - 1] as a missing range as well.



* Edge case 2: Similarly, if we don't end with upper as the last element of the array, we will need to include [nums[n-1] + 1, upper] as a missing range as well. Note n here is the length of the input array, nums.

**Complexity Analysis**

Let N*N* be the length of the input array.

* Time complexity : O(N)*O*(*N*).

This is because we are only iterating over the array once, and at each step, we're performing O(1)*O*(1) operations. We treat the string building as O(1)*O*(1) because the strings can never be more than a fixed size.

* Space complexity : O(1)*O*(1).

The output list has a worst case size of O(N)*O*(*N*). This case occurs when we have a missing range between each of the consecutive elements in the input array (for example, if the input array contains all *even* numbers between lower and upper). We aren't using any other additional space, beyond fixed-sized constants that don't grow with the size of the input.

**However**, output space that is simply used to return the output (and not to do any processing) is not counted for the purpose of space complexity analysis. For this reason, the overall space complexity is O(1)*O*(1).

[Report Article Issue](https://github.com/LeetCode-Feedback/LeetCode-Feedback/issues)

class Solution:

def findMissingRanges(self, nums: List[int], lower: int, upper: int) -> List[str]:

# formats range in the requested format

def formatRange(lower, upper):

if lower == upper:

return str(lower)

return str(lower) + "->" + str(upper)

result = []

prev = lower - 1

for i in range(len(nums) + 1):

curr = nums[i] if i < len(nums) else upper + 1

if prev + 1 <= curr - 1:

result.append(formatRange(prev + 1, curr - 1))

prev = curr

return result