Previous Next

35. Search Insert Position

March 1, 2020 | 16.7K views

Average Rating: 4.76 (21 votes)

Given a sorted array and a target value, return the index if the target is found. If not, return the index where it would be if it were inserted in order.

You may assume no duplicates in the array.

Example 1:

```
Input: [1,3,5,6], 5
Output: 2
```

Example 2:

```
Input: [1,3,5,6], 2
Output: 1
```

Example 3:

```
Input: [1,3,5,6], 7
Output: 4
```

Example 4:

```
Input: [1,3,5,6], 0
Output: 0
```

Solution

Approach 1: Binary Search

Intuition

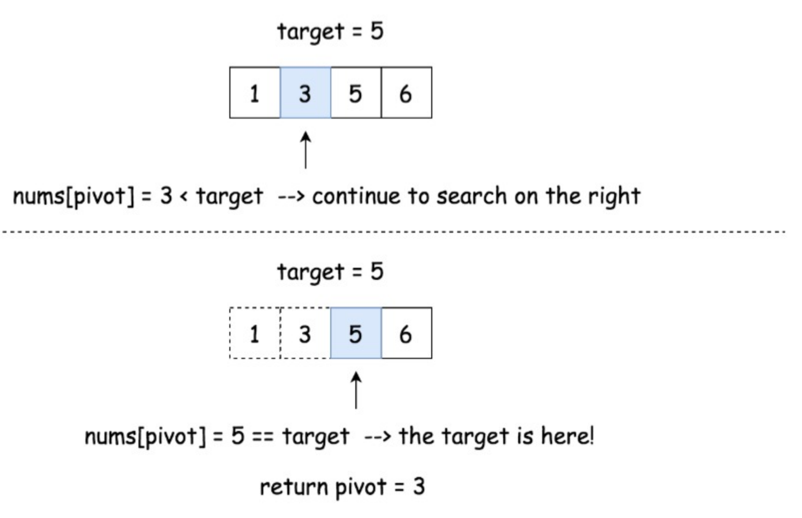
Based on the description of the problem, we can see that it could be a good match with the binary search algorithm.

> Binary search is a search algorithm that find the position of a target value within a *sorted* array.

Usually, within binary search, we compare the target value to the middle element of the array at each iteration.

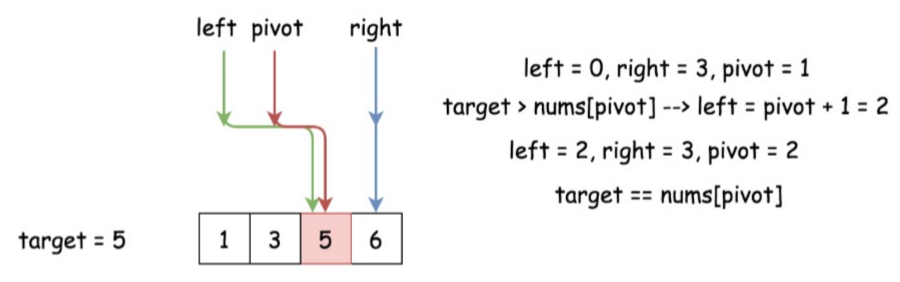
- If the target value is equal to the middle element, the job is done.
- If the target value is less than the middle element, continue to search on the left.
- If the target value is greater than the middle element, continue to search on the right.

Here we showcase a simple example on how it works.



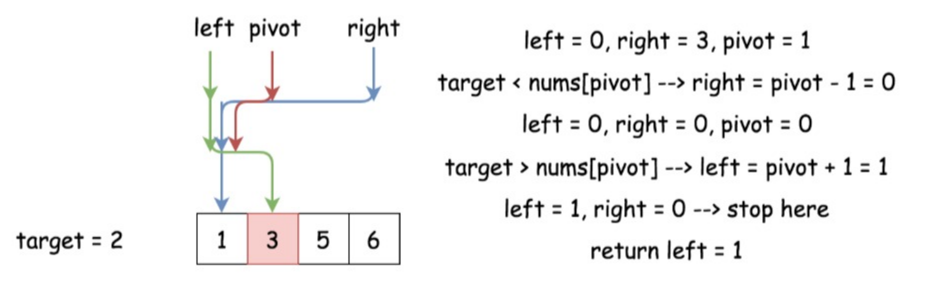
To mark the search boundaries, one could use two pointers: `left` and `right`. Starting from `left = 0` and `right = n - 1`, we then move either of the pointers according to various situations:

- While `left <= right`:
  - Pivot index is the one in the middle: `pivot = (left + right) / 2`. The pivot also divides the original array into two subarray.
  - If the target value is equal to the pivot element: `target == nums[pivot]`, we're done.
  - If the target value is less than the pivot element `target < nums[pivot]`, continue to search on the left subarray by moving the right pointer `right = pivot - 1`.
  - If the target value is greater than the pivot element `target > nums[pivot]`, continue to search on the right subarray by moving the left pointer `left = pivot + 1`.



> What if the target value is not found?

In this case, the loop will be stopped at the moment when `right < left` and `nums[right] < target < nums[left]`. Hence, the proper position to insert the target is at the index `left`.



Integer Overflow

Let us now stress the fact that `pivot = (left + right) // 2` works fine for Python3, which has arbitrary precision integers, but it could cause some issues in Java and C++.

If `left + right` is greater than the maximum int value $2^{31} - 1$, it overflows to a negative value. In Java, it would trigger an exception of `ArrayIndexOutOfBoundsException`, and in C++ it causes an illegal write, which leads to memory corruption and unpredictable results.

Here is a simple way to fix it:

C++ Java Python Copy

```
pivot = (left + right) // 2
```

and here is a bit more complicated but probably faster way using the bit shift operator.

C++ Java Python Copy

```
pivot = (left + right) >> 1
```

Algorithm

- Initialize the `left` and `right` pointers : `left = 0`, `right = n - 1`.
- While `left <= right`:
  - Compare middle element of the array `nums[pivot]` to the target value `target`.
    - If the middle element *is* the target, *i.e.* `target == nums[pivot]`: return `pivot`.
    - If the target is not here:
      - If `target < nums[pivot]`, continue to search on the left subarray. `right = pivot - 1`.
      - Else continue to search on the right subarray. `left = pivot + 1`.
- Return `left`.

Implementation

C++ Java Python Copy

```
class Solution:
    def searchInsert(self, nums: List[int], target: int) -> int:
        left, right = 0, len(nums) - 1
        while left <= right:
            pivot = (left + right) // 2
            if nums[pivot] == target:
                return pivot
            if target < nums[pivot]:
                right = pivot - 1
            else:
                left = pivot + 1
        return left
```

Complexity Analysis

- Time complexity : $\mathcal{O}(\log N)$.

  Let us compute the time complexity with the help of master theorem $T(N) = aT\left(\frac{N}{b}\right) + \Theta(N^d)$. The equation represents dividing the problem up into a subproblems of size $\frac{N}{b}$ in $\Theta(N^d)$ time. Here at each step there is only one subproblem *i.e.* `a = 1`, its size is a half of the initial problem *i.e.* `b = 2`, and all this happens in a constant time *i.e.* `d = 0`. As a result, $\log_b a = d$ and hence we're dealing with case 2 that results in $\mathcal{O}(n^{\log_b a} \log^{d+1} N)$ = $\mathcal{O}(\log N)$ time complexity.

- Space complexity : $\mathcal{O}(1)$ since it's a constant space solution.

Rate this article:

Previous Next

Comments: 7 Sort By

Type comment here... (Markdown is supported)

Preview Post

little_late 51 June 10, 2020 3:14 PM

I always mess up lower nd upper_bound conditions

17 Share Reply

SHOW 2 REPLIES

rust10 30 June 10, 2020 9:46 PM

Little simplified, one less "if" check,

+And also, I really think anyone should not worry about Integer overflow for array Size. Imagine an array length of bigger than 2^30 roughly ~1 bln elements in array. Unless specific case no-one uses it. 1 bln integers is about 4 GB memory.

Read More

2 Share Reply

panamavan 2 June 11, 2020 2:14 AM Report

Why is bisect.bisect_left for Python so slow here. Isn't it still log N?

1 Share Reply

SHOW 1 REPLY

sunheyanhobbit 1 June 10, 2020 3:35 PM

so fresh

1 Share Reply

SuperX 0 June 29, 2020 6:08 AM

recursive version

```
class Solution:
    def searchInsert(self, nums: List[int], target: int) -> int:
        def _search(n):
```

Read More

0 Share Reply

Mugdha_Bajjuri 0 June 26, 2020 2:48 PM Report

Solution: (python 3)

```
class Solution:
    def searchInsert(self, nums: List[int], target: int) -> int:
        if target>nums[-1]:
```

Read More

0 Share Reply

oj_glove 2 June 13, 2020 3:24 AM

I am failing the test case [1,3,5,7], 1. I return 1 - which happens to be a correct solution, but LC only expects 0 (which is also a correct answer. How do I get credit?

0 Share Reply

SHOW 1 REPLY