

[Articles](#) > 35. Search Insert Position ▾[◀ Previous \(/articles/sort-characters-by-frequency/\)](/articles/sort-characters-by-frequency/) [Next ▶ \(/articles/third-maximum-number/\)](/articles/third-maximum-number/)

## 35. Search Insert Position [↗](/problems/search-insert-position/) (/problems/search-insert-position/)

March 1, 2020 | 15.7K 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    ≡ Articles > 35. Search Insert Position ▼  
**Output:** 4

**Example 4:**

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

## Solution

### Approach 1: Binary Search

#### Intuition

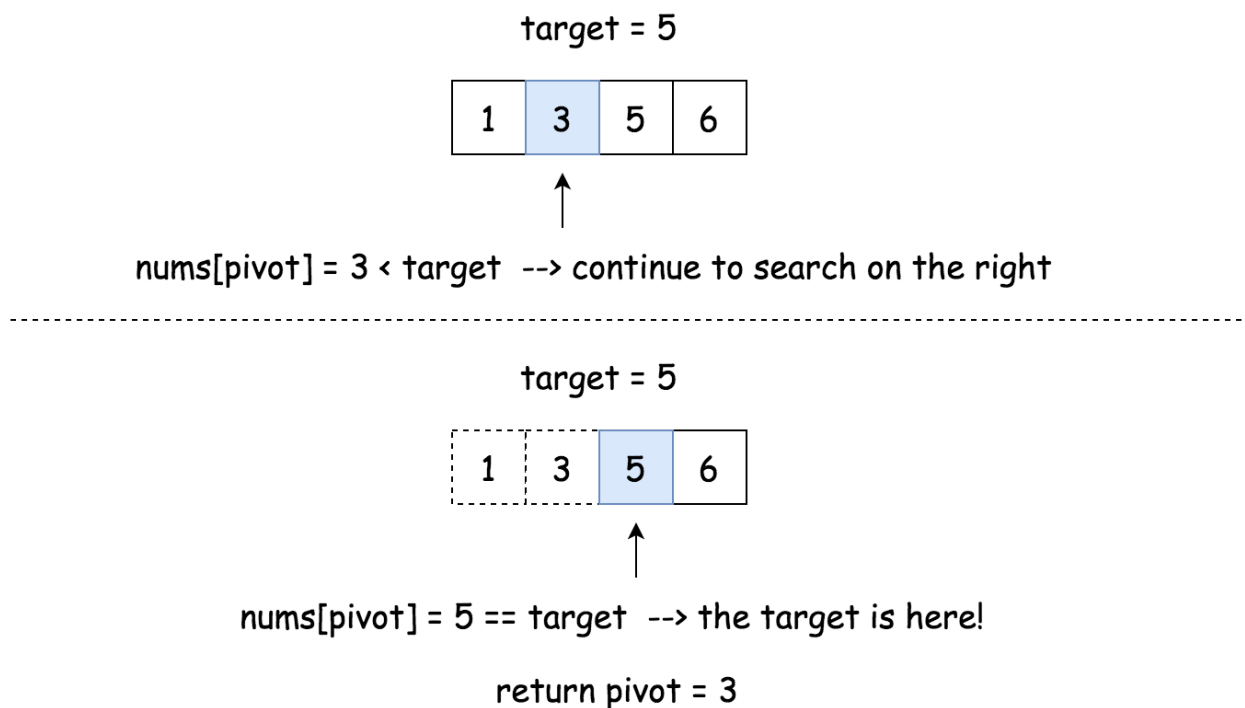
Based on the description of the problem, we can see that it could be a good match with the binary search ([https://en.wikipedia.org/wiki/Binary\\_search\\_algorithm](https://en.wikipedia.org/wiki/Binary_search_algorithm)) algorithm.

Binary search is a search algorithm that find the position of a target value within a ~~Yr wK~~ 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.

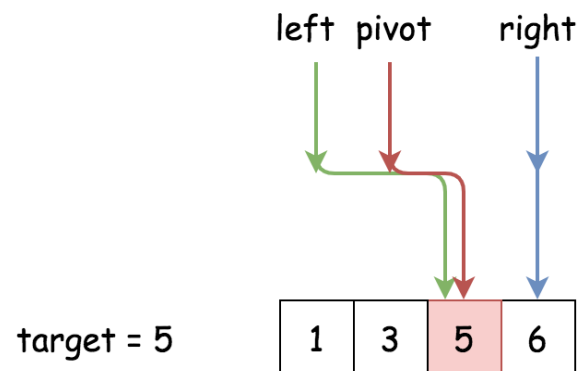
Here we showcase a simple example on how it works.



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`:

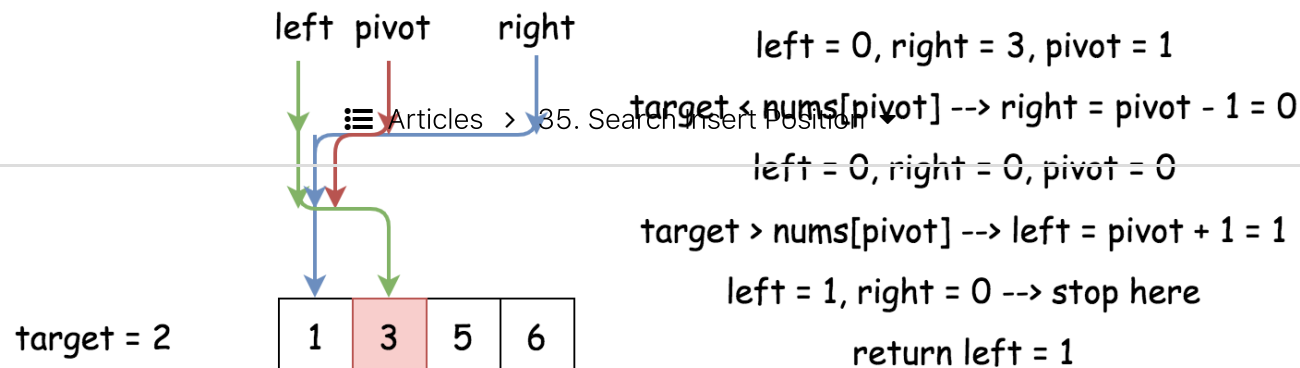
- Pivot index is the one in the middle:  $\text{pivot} = (\text{left} + \text{right}) / 2$ . The pivot also divides the original array into two subarray.
- If the target value is equal to the pivot element:  $\text{target} == \text{nums}[\text{pivot}]$ , we're done.
- If the target value is less than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



$\text{left} = 0, \text{right} = 3, \text{pivot} = 1$   
 $\text{target} > \text{nums}[\text{pivot}] \rightarrow \text{left} = \text{pivot} + 1 = 2$   
 $\text{left} = 2, \text{right} = 3, \text{pivot} = 2$   
 $\text{target} == \text{nums}[\text{pivot}]$

What if the target value is not found?

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



## Integer Overflow

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

If  $\text{left} + \text{right}$  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++JavaPython

Copy

1 pivot = left + (right - left) / 2;

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

C++JavaPython

Copy

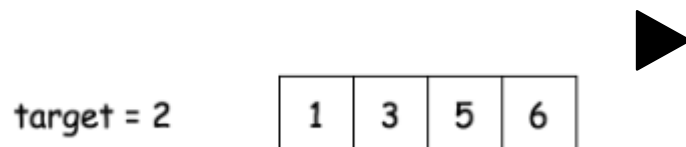
1 pivot = (right + left) >>> 1;

## Algorithm

- Initialize the left and right pointers:  $\text{left} = 0$ ,  $\text{right} = n - 1$ .

- While `left <= right`:
  - Compare middle element of the array `nums[pivot]` to the target value `target`.
    - If the middle element is the target, ~~do~~ `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

```

1 class Solution {
2     public int searchInsert(int[] nums, int target) {
3         int pivot, left = 0, right = nums.length - 1;
4         while (left <= right) {
5             pivot = left + (right - left) / 2;
6             if (nums[pivot] == target) return pivot;
7             if (target < nums[pivot]) right = pivot - 1;
8             else left = pivot + 1;
9         }
10        return left;
11    }
12 }

```

## Complexity Analysis

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

Let us compute the time complexity with the help of master theorem


([https://en.wikipedia.org/wiki/Master\\_theorem\\_\(analysis\\_of\\_algorithms\)](https://en.wikipedia.org/wiki/Master_theorem_(analysis_of_algorithms)))  $T(N) = aT\left(\frac{N}{b}\right) + \Theta(N^d)$ . 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 so  $a = 1$ , its size is a half of the initial problem so  $b = 2$ , and all this happens in a constant time so  $d = 0$ . As a result,  $\log_b a = d$  and hence we're dealing with case 2

([https://en.wikipedia.org/wiki/Master\\_theorem\\_\(analysis\\_of\\_algorithms\)#Case\\_2\\_example](https://en.wikipedia.org/wiki/Master_theorem_(analysis_of_algorithms)#Case_2_example)) 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 (</articles/sort-characters-by-frequency/>)

Next  (</articles/third-maximum-number/>)