

`l, r = 0, len(nums)` ----> This is a "**left-closed, right-open**" interval: `[l, r)`

`l, r = 0, len(nums) - 1` ----> This is a "**fully closed**" interval: `[l, r]`

This means you'll need to **update the loop condition and possibly other parts** of the code.

---

### **Changes Required for `[l, r]` Interval:**

class Solution:

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

```
    l, r = 0, len(nums) - 1
```

```
    while l <= r:
```

```
        mid = l + (r - l) // 2
```

```
        if nums[mid] == target:
```

```
            return mid
```

```
        elif target < nums[mid]:
```

```
            r = mid - 1
```

```
        else:
```

```
            l = mid + 1
```

```
    return l
```

---

### **Key Differences:**

Aspect	<code>[l, r)</code> version ( <code>r = len(nums)</code> )	<code>[l, r]</code> version ( <code>r = len(nums) - 1</code> )
Loop Condition	<code>while l &lt; r</code>	<code>while l &lt;= r</code>
Shrink Right Bound	<code>r = mid</code>	<code>r = mid - 1</code>
Purpose	Works for lower-bound type insert position	Works directly with index insert logic
When Loop Ends	<code>l == r</code> is exit point	<code>l &gt; r</code> is exit point

---

## Final Output Logic:

In both cases, returning `l` gives you the correct **insertion index** — the position where `target` should go to maintain sorted order.

---

Would you like a **visual step-through** example comparing both versions?

You said:

what is the condition for "left closed, right open" and "fully closed"

*ChatGPT said:*

Great! Let's clearly break down what "**left-closed, right-open**" (`[l, r)`) and "**fully closed**" (`[l, r]`) intervals mean — especially in **binary search** logic.

---

## Interval Terminology:

### 1. Left-Closed, Right-Open — `[l, r)`

- Includes the **left index** (`l`)
- **Excludes** the right index (`r`)
- Meaning: you consider all indices `i` such that `l <= i < r`

### ✓ Binary Search Logic for `[l, r)`:

```
l = 0
r = len(nums)          # `r` is one past the last index

while l < r:            # because r is exclusive
    mid = l + (r - l) // 2
    if nums[mid] == target:
        return mid
    elif target < nums[mid]:
        r = mid          # exclude mid
    else:
        l = mid + 1      # exclude mid

return l
```

---

## 2. Fully Closed — $[l, r]$

- Includes **both**  $l$  and  $r$
- Meaning: you consider all indices  $i$  such that  $l \leq i \leq r$

### ✓ Binary Search Logic for $[l, r]$

```
l = 0
r = len(nums) - 1    # because r is inclusive now

while l <= r:         # must include r
    mid = l + (r - l) // 2
    if nums[mid] == target:
        return mid
    elif target < nums[mid]:
        r = mid - 1   # exclude mid
    else:
        l = mid + 1   # exclude mid

return l
```

---

### 🧠 Summary Table:

Feature	$[l, r)$ (Left-closed, right-open)	$[l, r]$ (Fully closed)
Initial $r$	<code>len(nums)</code>	<code>len(nums) - 1</code>
Loop condition	<code>l &lt; r</code>	<code>l &lt;= r</code>
When shrinking $r$	<code>r = mid</code>	<code>r = mid - 1</code>
When shrinking $l$	<code>l = mid + 1</code>	<code>l = mid + 1</code>
Loop ends when	<code>l == r</code>	<code>l &gt; r</code>
Typical use	Lower bound search (insertion)	Exact match or insert index

---

### 📌 Rule of Thumb:

- Use  $[l, r)$  if you prefer math-style "**range-end exclusive**" logic (like Python's `range`).

- Use `[ l, r ]` if you prefer more **intuitive range** (includes both ends), but must be careful with loop boundaries.

#### Q.35: Search Insert Position

Return the index of target if found, otherwise the index where it should be inserted to maintain order.

✓ Goal:

Find the first position  $i$  where  $\text{nums}[i] \geq \text{target}$

Always return a valid index, whether or not the target is found.

✓ Best Interval Style:

Left-closed, Right-open `[left, right)`

Works perfectly for lower bound problems

✓ Typical Code:

```
def searchInsert(nums, target):
    left, right = 0, len(nums)
    while left < right:
        mid = (left + right) // 2
        if nums[mid] < target:
            left = mid + 1
        else:
            right = mid
    return left
```

#### Q.704: Binary Search

Return the index of the target if found, else return -1.

✓ Goal:

Find the exact position of target

Return -1 if not found

✓ Best Interval Style:

Fully closed `[left, right]`

Includes the last index, safely checks exact matches

✓ Typical Code:

```
def search(nums, target):
    left, right = 0, len(nums) - 1
    while left <= right:
        mid = (left + right) // 2
        if nums[mid] == target:
            return mid
        elif nums[mid] < target:
            left = mid + 1
        else:
            right = mid - 1
    return -1
```

#### Summary Table

Feature / Question

Target found?

Target not found?

Goal

Return Range

Preferred Interval

Loop Condition

Suitable for Lower Bound

Q.35: Insert Position

Return index

Return insert position

Find first  $i$  such that  $\text{nums}[i] \geq \text{target}$

Always `[0, len(nums))`

`[left, right)`

`left < right`

✓ Yes

Q.704: Binary Search

Return index

Return -1

Find exact target

-1 if not found

`[left, right]`

`left <= right`

✗ No

Rule of Thumb:  
Task Type  
Insert Position / Lower Bound  
Exact Match Search

Use Interval  
[left, right)  
[left, right]