Notation

For the following analysis, we have some predefined notations.

- For any array A, any indices l and r, $A[l, r)$ represents the subarray of A in range $[l, r)$:

  $$A[l, r) := (A[l], \dots, A[r-1]).$$

  Note that when $l \geq r$, $A[l, r)$ is empty.

  Similarly,

  $$A[l, r] := (A[l], \dots, A[r]).$$

- For any array A, any indices l and r, and any number $value$, we write $A[l, r) < value$ to represents that every element in the subarray $A[l, r)$ is less than $value$.

  $$A[l, r) < value \quad \Longleftrightarrow \quad \forall x \in A[l, r),\ x < value.$$

  Other comparison operators $(>, \leq, \geq,$ etc.$)$ work the same as above, just replace $<$ with them in the above definition.

  The subarray can also be placed at the right side of the inequation.

  $$A[l, r) < value \quad \Longleftrightarrow \quad value > A[l, r).$$

  Note that $A[l, r) < value$ is a tautology when $A[l, r)$ is empty.

Lomuto Partition

Problem.

Input: An array $A[0, n)$ and two indices l, r where $0 \le l < r \le n$.
Output: A pivot index $i \in [l, r)$ such that $A[l, i) \le A[i] < A[i + 1, r)$.

Algorithm.

```
1 partition(A, l, r)
2     pivot = A[r - 1]
3     i = l
4     for j = l to r - 2
5         if A[j] <= pivot
6             swap(A[i], A[j])
7             i = i + 1
8     swap(A[i], A[r - 1])
9     return i
```

Correctness.

We have the tail $A[r - 1]$ as *pivot*. And in the for loop at line 4-7, we aim to achieve that

$$A[l, i) \le \mathit{pivot} < A[i, r - 1), \quad \text{where} \quad l \le i \le r - 1.$$

Let's proof this claim. Define the loop invariant $\mathcal{I}(j)$ that at the start of the j-th iteration,

- $\mathcal{I}_1$: boundary check
$$l \le i \le j < r$$
- $\mathcal{I}_2$: partition left
$$A[l, i) \le \mathit{pivot}$$
- $\mathcal{I}_3$: partition right
$$\mathit{pivot} < A[i, j)$$

  Note that the subarray $A[j, r - 1)$ is not restricted, since this region is not classified yet.

1. **Initialization:** At the start we have $l = i = j$ and $l < r$ by the input assumption, so $\mathcal{I}_1(l)$ is true. And since $A[l, i)$ and $A[i, j)$ are both empty, $\mathcal{I}_2(l)$ and $\mathcal{I}_3(l)$ are both true.
2. **Maintenance:** Assume that for some index $j \in [l, r - 2]$, $\mathcal{I}(j)$ is true, and we aim to show that $\mathcal{I}(j + 1)$ is also true. We have to consider the two branches at line 5.

(a) If $A[j] \le pivot$, let's first consider the left side of the partition. We swap $A[j]$ with $A[i]$, so $A[i] \le pivot$. And then we update i as $i+1$, so this element is now included in the left partition. Thus, $A[l, i+1) \le pivot$, which means $\mathcal{I}_2(j+1)$ remains true.

   For the right partition, we have to consider whether it is empty at first.

   For the case where $i = j$, both i and j are both going to be incremented by the next iteration, so $A[i, j)$ is still empty, which ensures that $\mathcal{I}_3(j+1)$ is true.

   For the case where $i < j$, we have the original $A[i] > pivot$ rotate upward to $A[j]$, so the right partition remains correct: $A[i+1, j+1) > pivot$. Thus, $\mathcal{I}_3(j+1)$ holds.

   The boundary can be easily check since we have $j \le r-2$ by the loop condition. And based on the assumption of $\mathcal{I}_1(j)$, we can easily deduce that $\mathcal{I}_1(j+1)$ is true:

$$l < i+1 \le j+1 \le r-1 < r.$$

(b) If $A[j] > pivot$, that means this element belongs to the right partition, so no operation needs to be done. $\mathcal{I}_2(j+1)$ remains true since the left partition is not changed. And $\mathcal{I}_3(j+1)$ holds because we add $A[j]$ to the right partition, so $A[i, j+1) > pivot$. For the boundary check, only j is increment, so that $\mathcal{I}_1(j+1)$ is true:

$$l \le i < j+1 \le r-1 < r.$$

In both cases, we achieve that $\mathcal{I}(j+1)$ is true based on the assumption that $\mathcal{I}(j)$ holds.

3. **Termination:** Let's first show that the loop always terminates. We know that j starts at l and increments one each iteration, so it always grows past $r-2$.

   Based on the initialization and maintenance of $\mathcal{I}$, we have that $\mathcal{I}(j)$ is true for all index $j \in [l, r)$. So at the termination where $j = r-1$, we have

$$A[l, i) \le pivot < A[i, r-1), \quad \text{where} \quad l \le i \le r-1.$$

At line 8, we swap the pivot element $A[r-1]$ with $A[i]$, then we get

$$A[l, i) \le A[i] < A[i+1, r).$$

Therefore, the algorithm correctly partition the array and return the desired index.

Hoare Partition

Problem.

Input: A non empty subarray $A[l, r)$ where $0 \le l < r$.
Output: A pivot index $i \in [l, r)$ such that $A[l, i) \le A[i] < A[i + 1, r)$.

Algorithm.

```
 1 partition(A, l, r)
 2     pivot = A[l]
 3     i = l + 1
 4     j = r
 5     while true
 6         while i < j and A[i] <= pivot
 7             i = i + 1
 8         while i < j and pivot < A[j - 1]
 9             j = j - 1
10         if i == j
11             break
12         j = j - 1
13         swap(A[i], A[j])
14         i = i + 1
15     i = i - 1
16     swap(A[l], A[i])
17     return i
```

Correctness.

We choose the first element $A[l]$ as *pivot*, and in the while loop at line 5-14, we aim to partition $A[l + 1, r)$ as

$$A[l + 1, i) \le \mathit{pivot} < A[i, r), \quad \text{where } l + 1 \le i \le r.$$

Let's proof this claim. Define the loop invariant $\mathcal{I}$ that at the start of each iteration,

- $\mathcal{I}_1$: boundary check
$$l + 1 \le i \le j \le r$$
- $\mathcal{I}_2$: partition left
$$A[l + 1, i) \le \mathit{pivot}$$
- $\mathcal{I}_3$: partition right
$$\mathit{pivot} < A[j, r)$$

Note that the subarray $A[i, j)$ is not restricted, since this region is not classified yet.

1. **Initialization:** Before we enter the while loop, we have $i = l + 1$ and $j = r$. And with $l < r$ by the input assumption, $\mathcal{I}$ is true when we enter the while loop because the indices is clear in bounds and both partition region are empty.

2. **Maintenance:** Assume that $\mathcal{I}$ is true at the start of an iteration, we want to show that $\mathcal{I}$ is still true at the end of this iteration.

   For line 6-7, ideally we might want to use another loop invariant to show the behavior of this while loop. But since it is simple enough, let's just conclude that this loop repeatedly checks if the next element can be included in the left partition. The loop stops when either $i = j$ or $\mathit{pivot} < A[i]$. If we denote the original i before this while loop as i', then we have

   $$A[i', i) \leq \mathit{pivot}, \quad i' \leq i.$$

   Combine with the assumption of $\mathcal{I}_2$, we then get

   $$A[l+1, i') \cup A[i', i) = A[l+1, i) \leq \mathit{pivot}, \quad l+1 \leq i \leq j$$

   Similarly, the while loop at line 8-9 ends with either $i = j$ or $A[j-1] \leq \mathit{pivot}$, and we get

   $$\mathit{pivot} < A[j, r), \quad i \leq j \leq r$$

   Let's first shortly discuss the case where $i = j$. This would trigger the termination condition so the iteration ends immediately. We can see the loop invariant $\mathcal{I}$ is perfectly maintained as the above stated.

   Now we can consider the case where $i \neq j$, that is, the two while loops stop as they find elements at the wrong partitions:

   $$\mathit{pivot} < A[i], \quad A[j-1] \leq \mathit{pivot}.$$

   $A[i]$ should be in the right partition, and $A[j-1]$ should be in the left partition. At line 12-14, these two elements are swapped, so now $A[i]$ belongs to the left partition and $A[j-1]$ belongs to the right partition. Thus, we can increase the range of both partition, that is, increment and decrement i and j respectively by one. Then $\mathcal{I}_2$ and $\mathcal{I}_3$ are both tightly satisfied:

   $$A[l+1, i) \leq \mathit{pivot} < A[j, r).$$

   Let's carefully examine the boundary check. One might identify the mistake that with $i \neq j$, if we were to have $i = j - 1$, then increment and decrement i, j by one would lead to $i = j + 1$, making i surpass j. However, this actually never occur because we have *pivot* $< A[i]$ and $A[j-1] \leq$ *pivot*, so the two elements are different. Thus, after we increase the range, we always still have $i \leq j$.

To show that they still stay in the $[l+1, r)$ region, originally we have $l+1 \leq i$ and $j \leq r$, and we only increment i and decrement j. Combine with the above $i \leq j$, the boundary check $\mathcal{I}_1$ is perfectly maintained.

3. **Termination:** Let's first check that the loop always terminates. Since we always increment i and decrement j, so the while loops at line 6-7 and 8-9 cannot be infinite loops, and we always reach the condition $i = j$.

   Based on the initialization and maintenance of $\mathcal{I}$, at termination of the while loop we have

$$A[l+1, i) \leq \mathit{pivot} < A[i, r), \quad \text{where } l+1 \leq i \leq r.$$

At line 14 we decrement i by one so that $A[i]$ is the rightmost element of the left partition. And at line 15 we swap this element with the pivot element $A[l]$, so now we get

$$A[l, i) \leq A[i] < A[i+1, r), \quad \text{where } l \leq i < r.$$

Finally, we return the pivot index i, and the subarray $A[l, r)$ is correctly partitioned. Therefore, the algorithm is correct.