Quicksort

Josh Wilcox (jw14g24)

May 7, 2025

Table of Contents

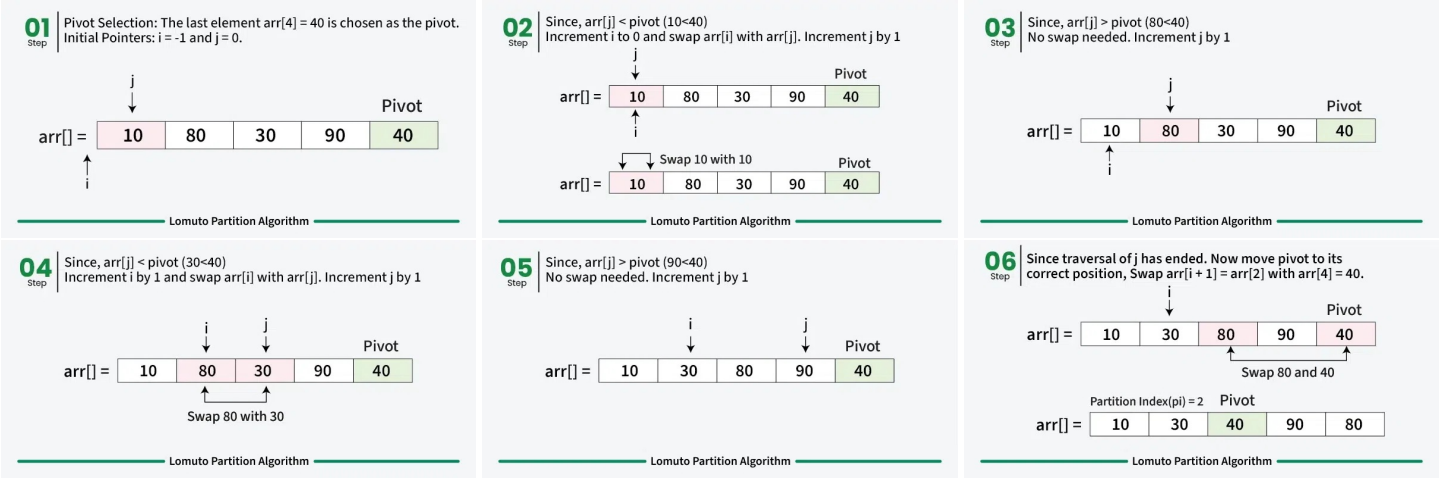
Lomuto Scheme - Right Element Pivot

Overview

- Given an array `arr[]`, we want to partition array by assuming the last element is the **pivot**
  - Elements smaller than the pivot must be to it's left in the new partition array
  - Elements larger **or equal** to the pivot should appear on its right in the new partition array

Lomuto Algorithm

- Always choose the last element of the array as the pivot p
- Initialise pointer `i` at the start of the array
  - This acts as the boundary of elements that are $\leq p$
- Traverse the array with pointer `j` checking each element `arr[j]`
  - If `arr[j]` $\leq p$, swap `arr[i]` with `arr[j]`
  - This moves the element that is smaller to the pivot in the zone bounded by `i`
  - Increment `i` to adjust the boundary to accomodate for this new element
- Finally swap `arr[i]` with the pivot to place it in the correct position



Algorithm Lomuto Partition Scheme

```
1: procedure PARTITION(arr, left, right)
2:     p ← arr[right]                          ▷ Choose rightmost element as pivot
3:     i ← left − 1                            ▷ Initialize boundary of elements ≤ pivot
4:     for j ← left to right − 1 do            ▷ Scan through array
5:         if arr[j] ≤ p then
6:             i ← i + 1                       ▷ Expand the left partition
7:             swap(arr[i], arr[j])            ▷ Move smaller element to left partition
8:         end if
9:     end for
10:    swap(arr[i + 1], arr[right])            ▷ Place pivot in its final position
11:    return i + 1                            ▷ Return the pivot's position
12: end procedure
```

Hoare Partition Scheme

Overview

- Given an array `arr[]`, we want to partition array by assuming the last element is the **pivot**
  - Elements smaller than the pivot must be to it's left in the new partition array
  - Elements larger **or equal** to the pivot should appear on its right in the new partition array

Hoare's Algorithm

- Consider the **first element** as the pivot p and initialise two pointers
  - `i` should be at the start of the array
  - `j` should be at the end of the array
- Move `i` to the right until an element $\geq p$ is found
- Move `j` to the left until an element $\leq p$ is found
- `if (arr[i] >= p && arr[j] <= p):`
  - Swap `arr[i]` and `arr[j]`
- Repeat until `i` and `j` meet or cross

Algorithm Hoare Partition Scheme

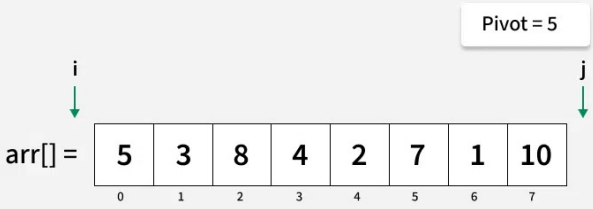
```
1:  procedure PARTITION(arr, pivot, left, right)
2:      i ← left − 1                                  ▷ Initialize left pointer
3:      j ← right + 1                                 ▷ Initialize right pointer
4:      while True do
5:          repeat
6:              j ← j − 1                             ▷ Move right pointer left
7:          until arr[j] ≤ pivot
8:          repeat
9:              i ← i + 1                             ▷ Move left pointer right
10:         until arr[i] ≥ pivot
11:         if i < j then
12:             swap(arr[i], arr[j])                  ▷ Swap elements if pointers haven't crossed
13:         else
14:             return j                              ▷ Return final partition position
15:         end if
16:     end while
17: end procedure
```

Subsections of Hoare Partition Scheme

- Example

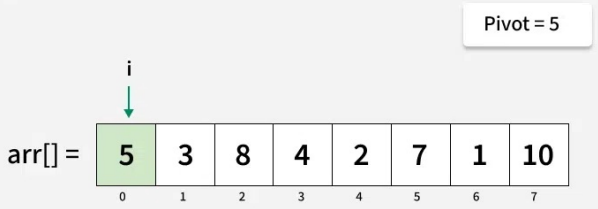
Hoare Partition Scheme - **Example**

01 Step Consider the first element of the array as the pivot.
Initialise i = -1 and j = n.



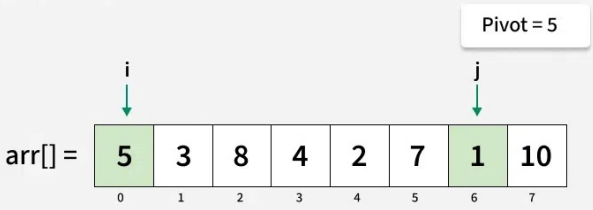
Hoare's Algorithm for Array Partition

02 Step Find the next element greater than equal to pivot from the left.
At i = 0, we get this element.



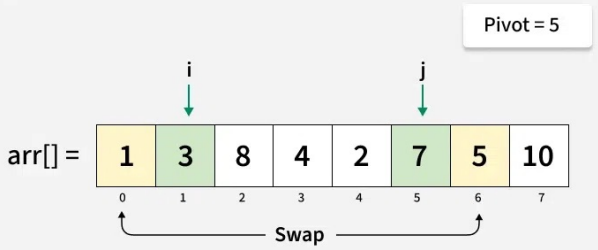
Hoare's Algorithm for Array Partition

03 Step Find the next element smaller than the pivot from the right.
At j = 6, we get this element.



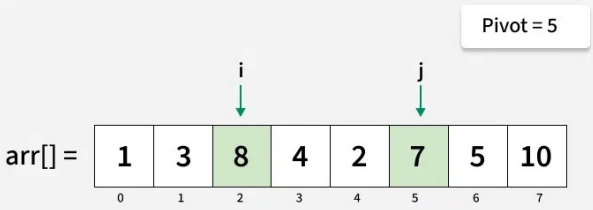
Hoare's Algorithm for Array Partition

04 Step Since i < j, swap arr[i] and arr[j], then increment i and decrement j.



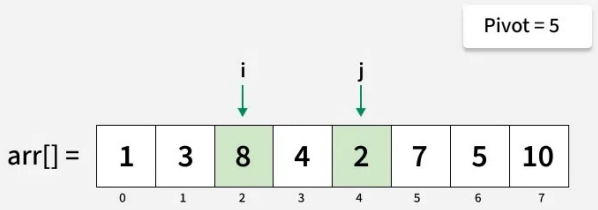
Hoare's Algorithm for Array Partition

05 Step At i = 2, we get the next element greater than the pivot from the left.



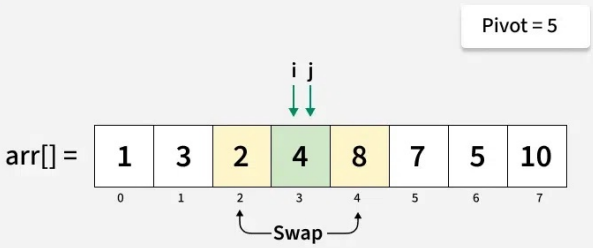
Hoare's Algorithm for Array Partition

06 Step At j = 4, we get the next element smaller than the pivot from the right.



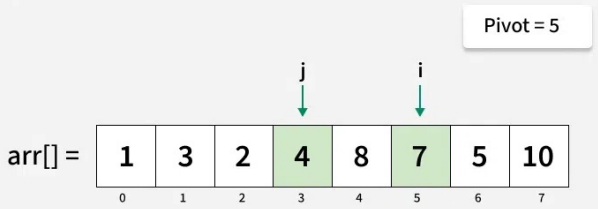
Hoare's Algorithm for Array Partition

07 Step Since i < j, swap arr[i] and arr[j], then increment i and decrement j.



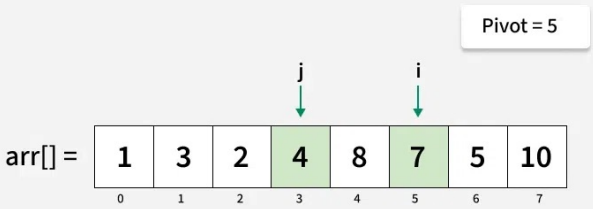
Hoare's Algorithm for Array Partition

08 Step At i = 5, we get the next element larger than the pivot from the left.



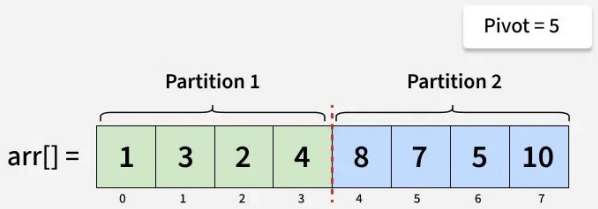
Hoare's Algorithm for Array Partition

09 Step At j = 3, we get the next element smaller than the pivot from the right.



Hoare's Algorithm for Array Partition

10 Step Since i > j, the entire array has been traversed and the partition is completed.



Hoare's Algorithm for Array Partition