Practical 1.2

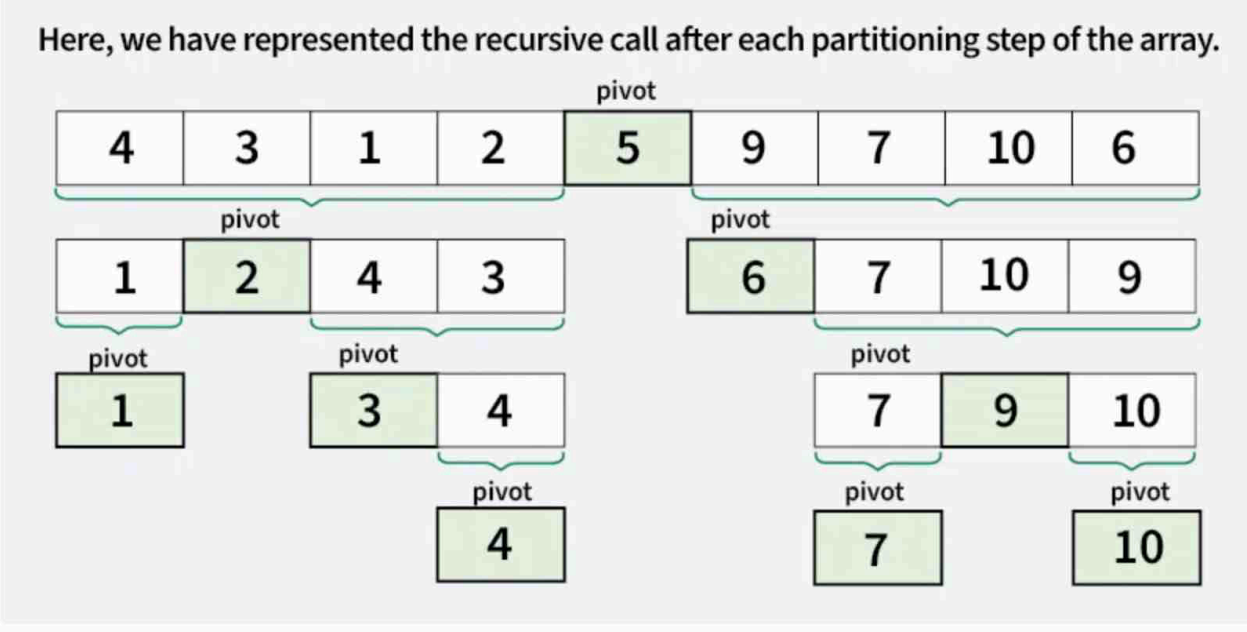
Aim: Implementation of Quick Sort Algorithm using Divide and Conquer Method

Theory:

QuickSort is a sorting algorithm based on the Divide and Conquer that picks an element as a pivot and partitions the given array around the picked pivot by placing the pivot in its correct position in the sorted array. .

There are mainly three steps in the algorithm:

1. **Choose a Pivot:** Select an element from the array as the pivot. The choice of pivot can vary (e.g., first element, last element, random element, or median).
2. **Partition the Array:** Re arrange the array around the pivot. After partitioning, all elements smaller than the pivot will be on its left, and all elements greater than the pivot will be on its right.
3. **Recursively Call:** Recursively apply the same process to the two partitioned sub-arrays.
4. **Base Case:** The recursion stops when there is only one element left in the sub-array, as a single element is already sorted.



Choice of Pivot

There are many different choices for picking pivots.

- Always pick the first (or last) element as a pivot. The below implementation picks the last element as pivot. The problem with this approach is it ends up in the worst case when array is already sorted.
- Pick a random element as a pivot. This is a preferred approach because it does not have a pattern for which the worst case happens.
- Pick the median element is pivot. This is an ideal approach in terms of time complexity as we can find median in linear time and the partition function will always divide the input array into two halves. But it takes more time on average as median finding has high constants.

Partition Algorithm

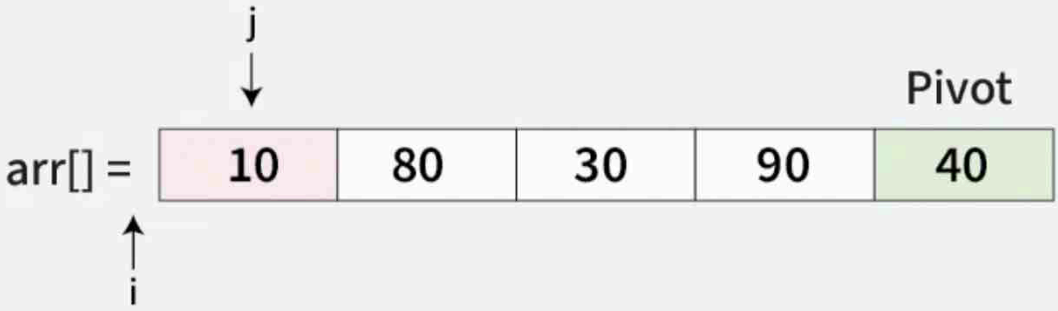
The key process in **quickSort** is a **partition().** There are three common algorithms to partition. All these algorithms have O(n) time complexity.

1. **Naive Partition**: Here we create copy of the array. First put all smaller elements and then all greater. Finally we copy the temporary array back to original array. This requires O(n) extra space.
2. **Lomuto Partition**: We have used this partition in this article. This is a simple algorithm, we keep track of index of smaller elements and keep swapping. We have used it here in this article because of its simplicity.
3. **Hoare's Partition**: This is the fastest of all. Here we traverse array from both sides and keep swapping greater element on left with smaller on right while the array is not partitioned. Please refer Hoare's vs Lomuto for details.

Working of Lomuto Partition Algorithm with Illustration

01 Step

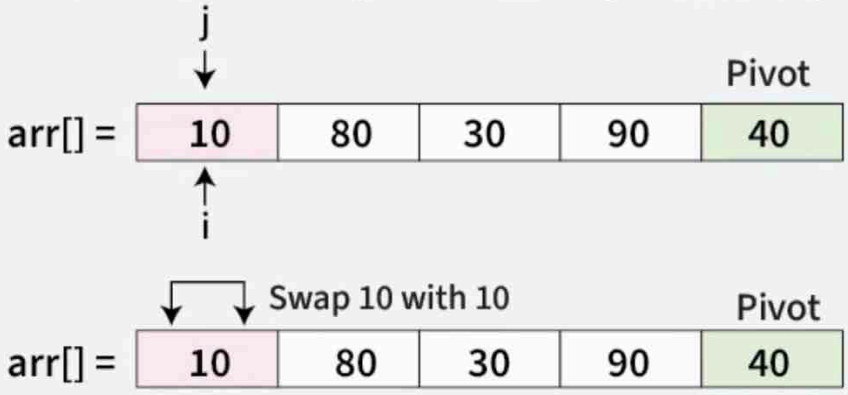
Pivot Selection: The last element arr[4] = 40 is chosen as the pivot.
Initial Pointers: i = -1 and j = 0.



Quick sort

02 Step

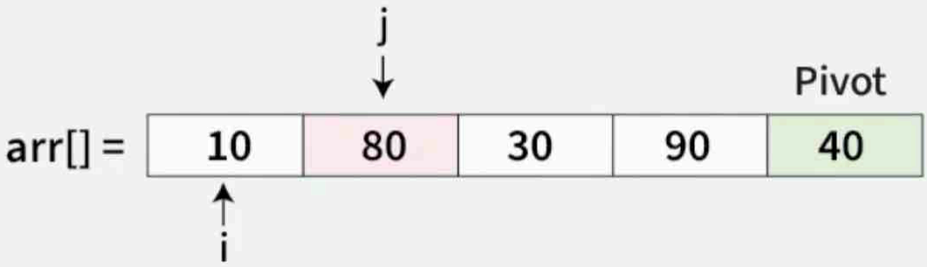
Since, arr[j] < pivot (10<40)
Increment i to 0 and swap arr[i] with arr[j]. Increment j by 1



Quick sort

03 Step

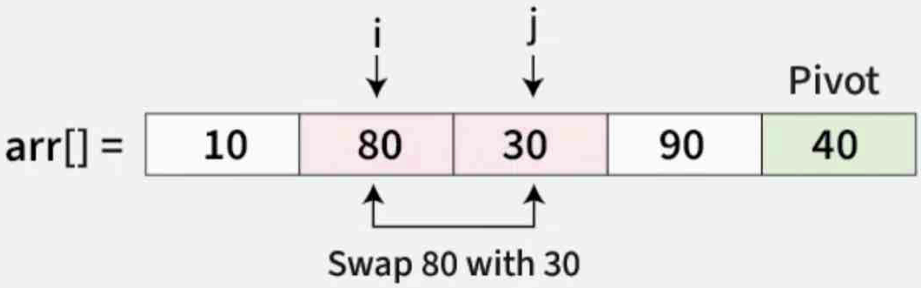
Since, arr[j] > pivot (80>40)
No swap needed. Increment j by 1



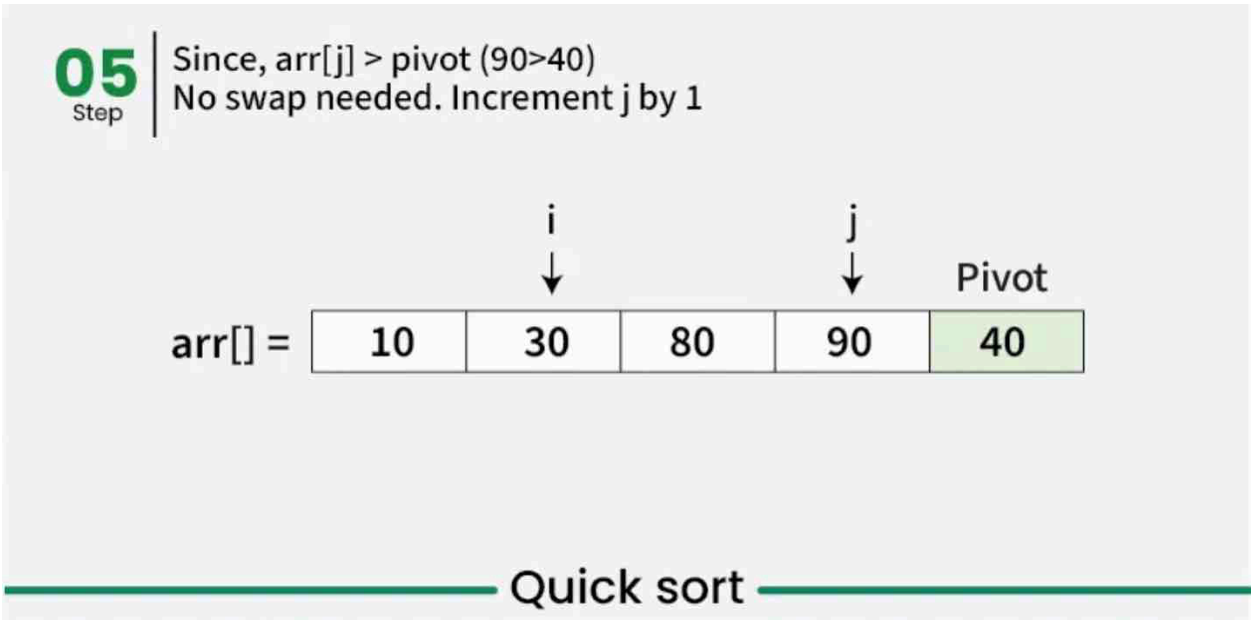
Quick sort

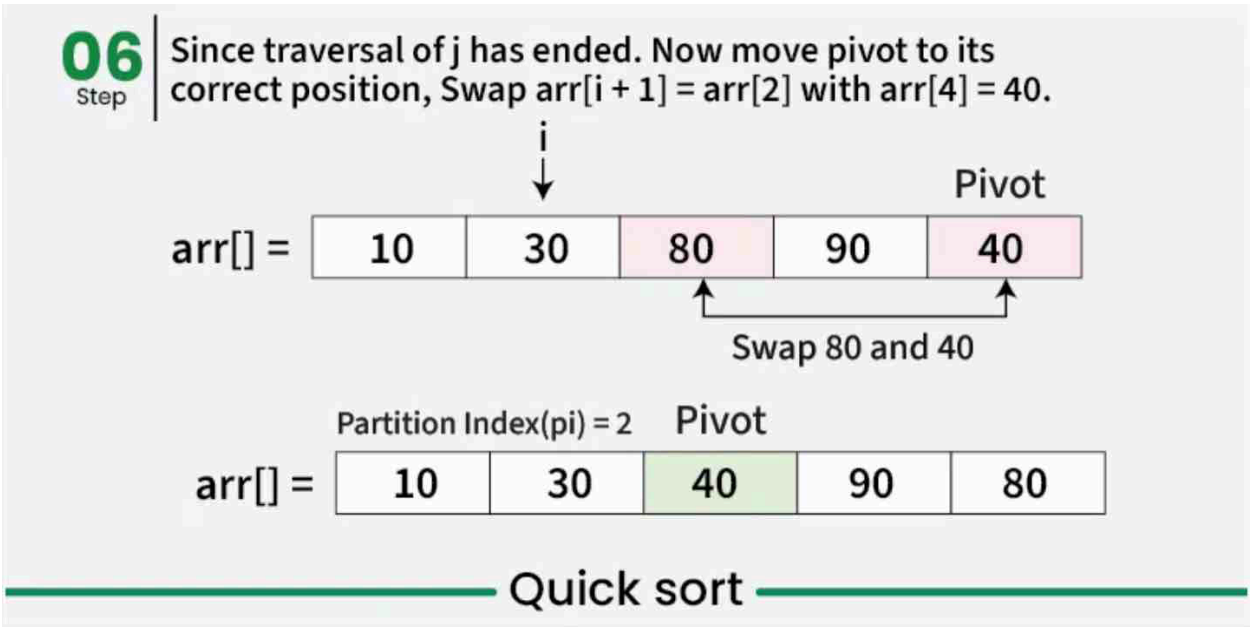
04 Step

Since, arr[j] < pivot (30<40)
Increment i by 1 and swap arr[i] with arr[j]. Increment j by 1



Quick sort





Algorithm: Quick Sort (Divide and Conquer)

1. Select an element from the array as the **pivot**.
2. Rearrange the array so that all elements smaller than the pivot are placed on the left side and all elements greater than the pivot are placed on the right side (partitioning).
3. Recursively apply Quick Sort to the left sub-array.
4. Recursively apply Quick Sort to the right sub-array.
5. When all sub-arrays are sorted, the entire array becomes sorted.

Conclusion

Quick Sort is a fast sorting algorithm based on the Divide and Conquer technique. It is efficient for large datasets and performs in-place sorting.