4.2 QUICK SORT

Quick Sort is a divide-and-conquer sorting algorithm widely used due to its efficiency and simplicity. It works by selecting a "pivot" element, partitioning the array into two subarrays (elements less than the pivot and elements greater than the pivot), and then recursively sorting the subarrays.

Key Steps of Quick Sort:

1. **Choose a Pivot**: Select an element as the pivot (commonly the first, last, or middle element).
2. **Partitioning**: Rearrange the array so that:
    - Elements smaller than the pivot are on the left.
    - Elements larger than the pivot are on the right.
3. **Recursive Sorting**: Apply the same process to the left and right subarrays.

Time Complexity:

- **Best/Average Case**: $O(n\log n)$

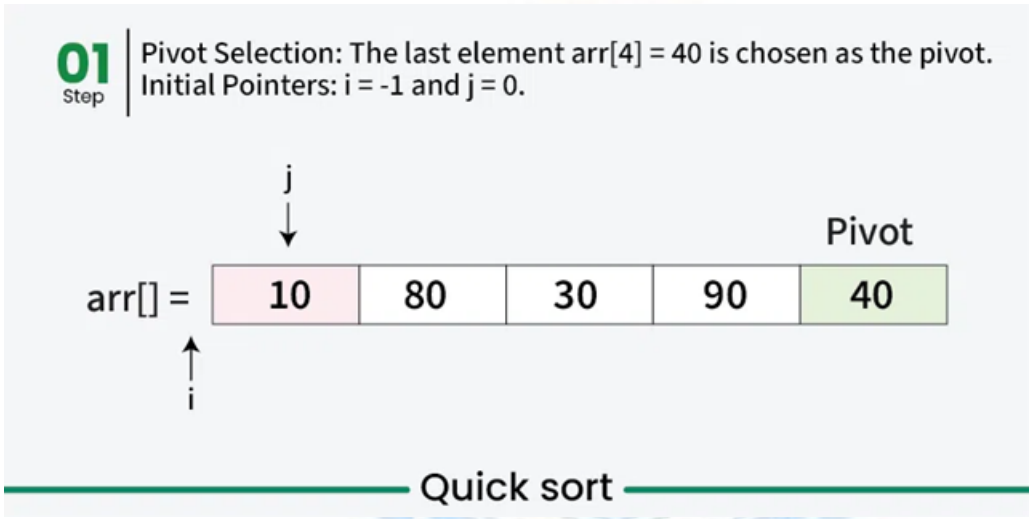
Code

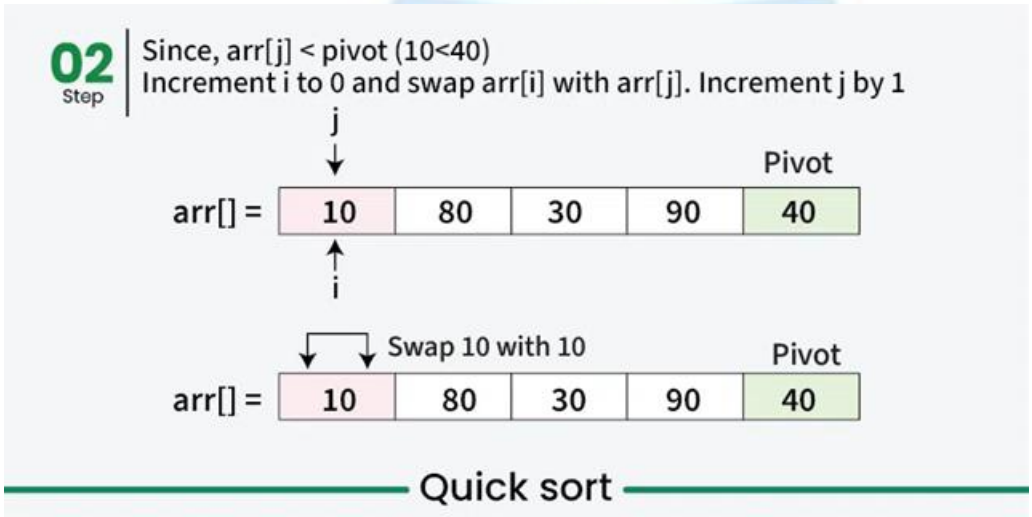
```
def partition(arr, low, high):
  pivot = arr[high] # Choosing the last element as pivot
  i = low - 1 # Index of smaller element
  for j in range(low, high):
     if arr[j] < pivot:
        i += 1
        swap(arr, i, j)
  swap(arr, i + 1, high)
  return i + 1
# swap function
def swap(arr, i, j):
   arr[i], arr[j] = arr[j], arr[i]

def quick_sort(arr, low, high):
  if low < high:
```

```
    pi = partition(arr, low, high) # Partition index
    quick_sort(arr, low, pi - 1) # Sort left sub-array
    quick_sort(arr, pi + 1, high) # Sort right sub-array
# Example usage
arr = [10, 7, 8, 9, 1, 5]
quick_sort(arr, 0, len(arr) - 1)
print("Sorted array:", arr)
```

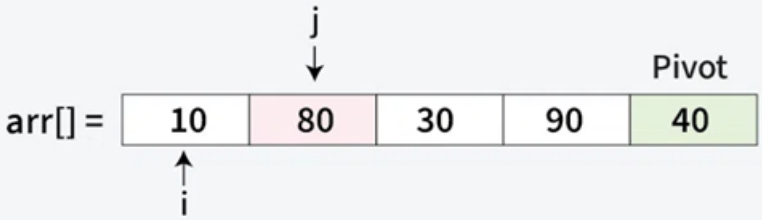
Example:

Array elements= 10,80,30,90,40



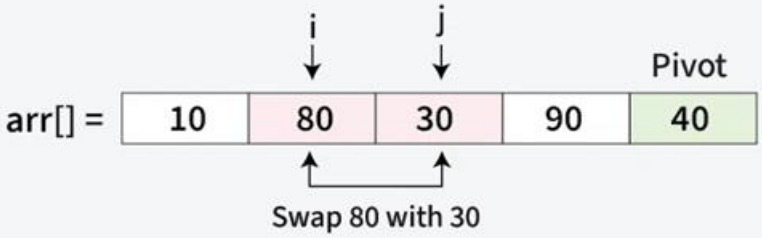


03 Step | Since, arr[j] > pivot (80>40)
No swap needed. Increment j by 1



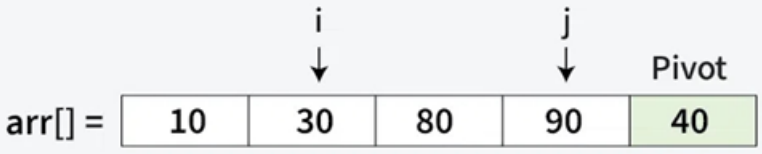
Quick sort

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

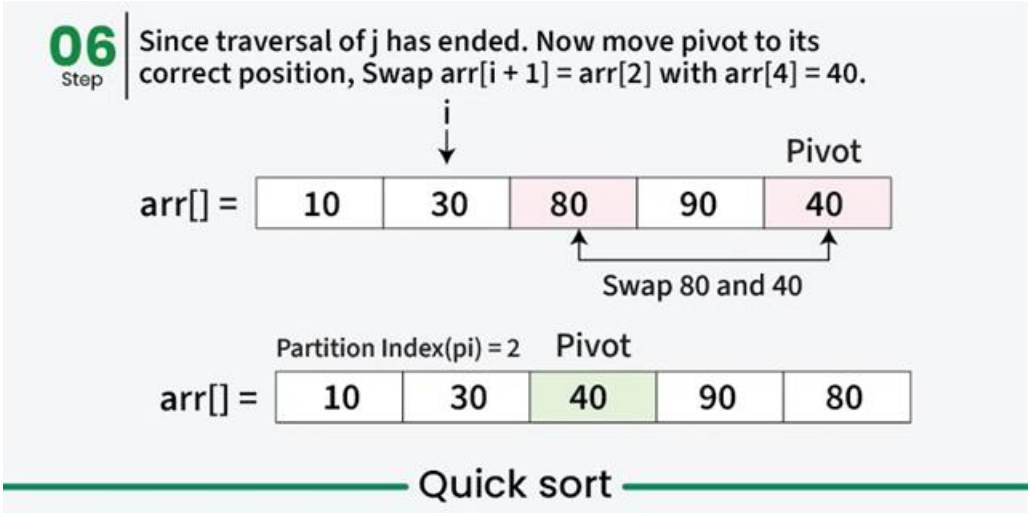


Quick sort

05 Step | Since, arr[j] > pivot (90>40)
No swap needed. Increment j by 1



Quick sort



In the previous step, we looked at how the partitioning process rearranges the array based on the chosen pivot. Next, we apply the same method recursively to the smaller sub-arrays on the left and right of the pivot. Each time, we select new pivots and partition the arrays again. This process continues until only one element is left, which is always sorted. Once every element is in its correct position, the entire array is sorted.

Advantages of Quick Sort

- It is a divide-and-conquer algorithm that makes it easier to solve problems.
- It is efficient on large data sets.

Disadvantages of Quick Sort

- It has a worst-case time complexity of $O(n^2)$, which occurs when the pivot is chosen poorly.
- It is not a good choice for small data sets.