Quick Sort

Sorting Data by Using Quick Sort

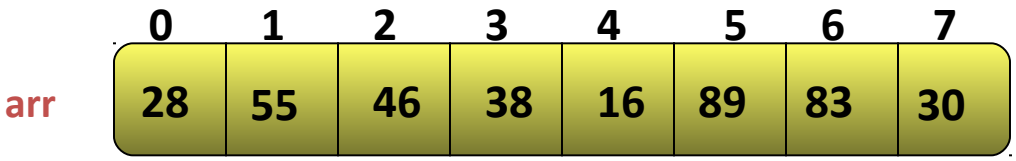
- Quick sort algorithm:
  - Is one of the most efficient sorting algorithms
  - Is based on the divide and conquer approach
  - Successively divides the problem into smaller parts until the problems become so small that they can be directly solved

Implementing Quick Sort Algorithm

- In quick sort algorithm, you:
  - Select an element from the list called as pivot.
  - Partition the list into two parts such that:
    - **All the elements towards the left end of the list are smaller than the pivot.**
    - **All the elements towards the right end of the list are greater than the pivot.**
  - Store the pivot at its correct position between the two parts of the list.
- You repeat this process for each of the two sublists created after partitioning.
- This process continues until one element is left in each sublist.

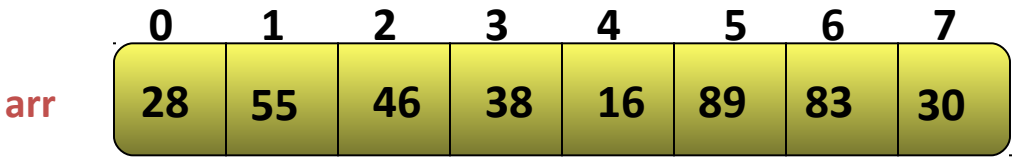
Implementing Quick Sort Algorithm (Contd.)

- To understand the implementation of quick sort algorithm, consider an unsorted list of numbers stored in an array.



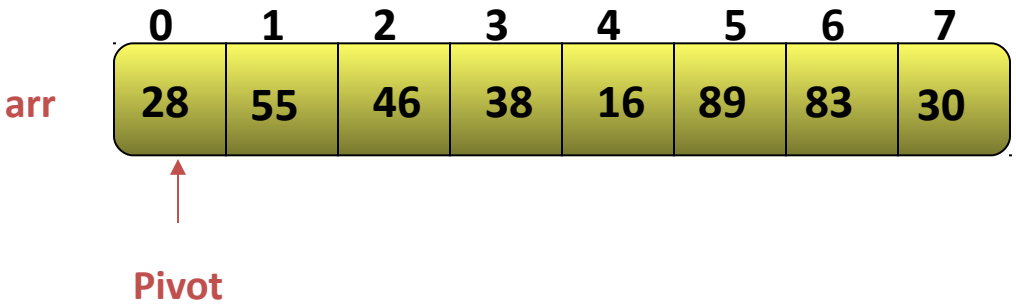
Implementing Quick Sort Algorithm (Contd.)

- Let us sort this unsorted list.



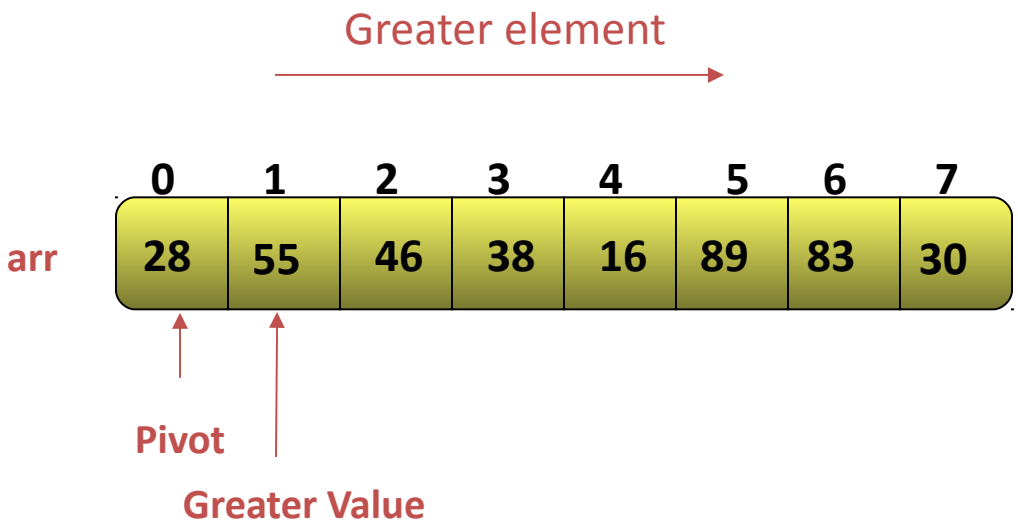
Implementing Quick Sort Algorithm (Contd.)

- Select a Pivot.



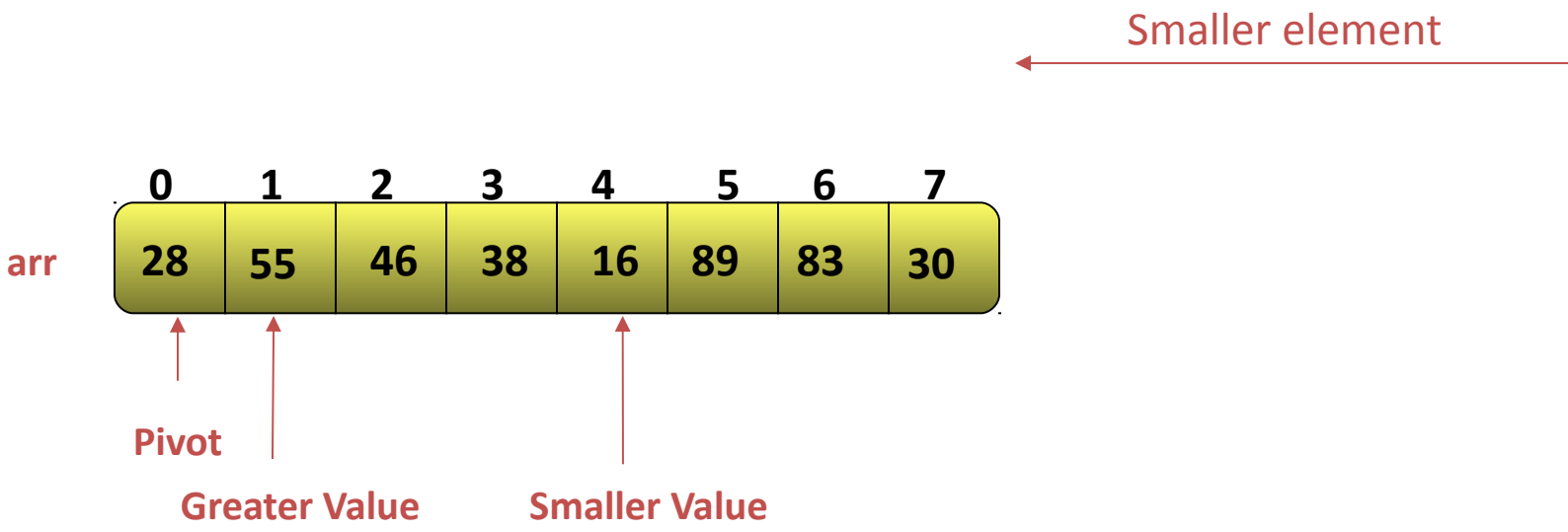
Implementing Quick Sort Algorithm (Contd.)

- Start from the left end of the list (at index 1).
- Move in the left to right direction.
- Search for the first element that is greater than the pivot value.



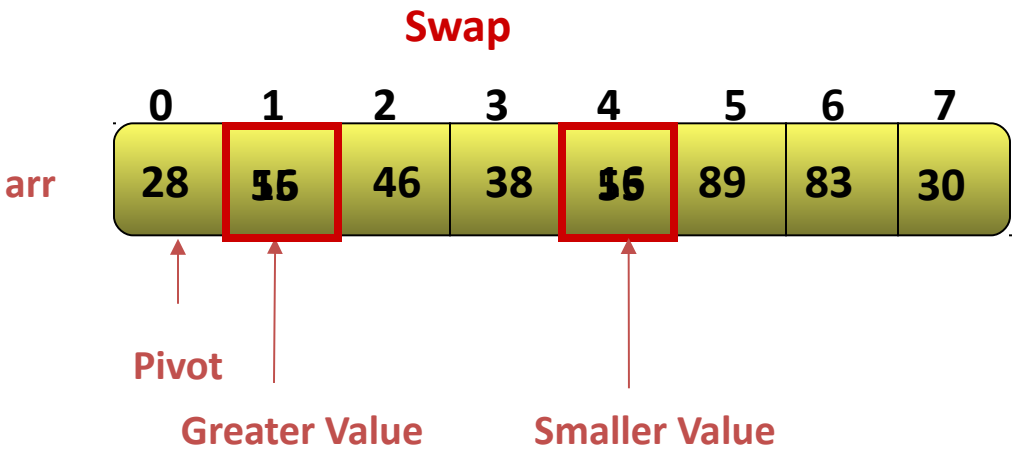
Implementing Quick Sort Algorithm (Contd.)

- Start from the right end of the list.
- Move in the right to left direction.
- Search for the first element that is smaller than or equal to the pivot value.



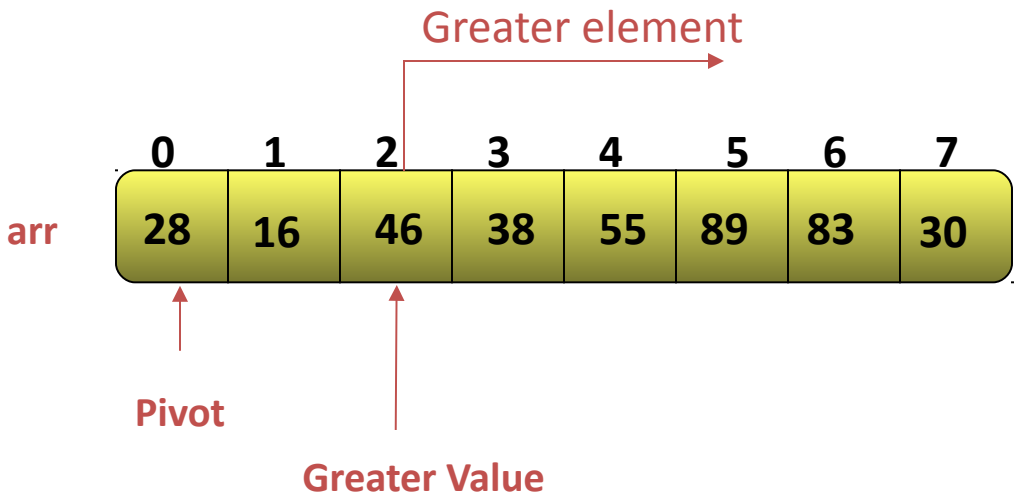
Implementing Quick Sort Algorithm (Contd.)

- Interchange the greater value with smaller value.



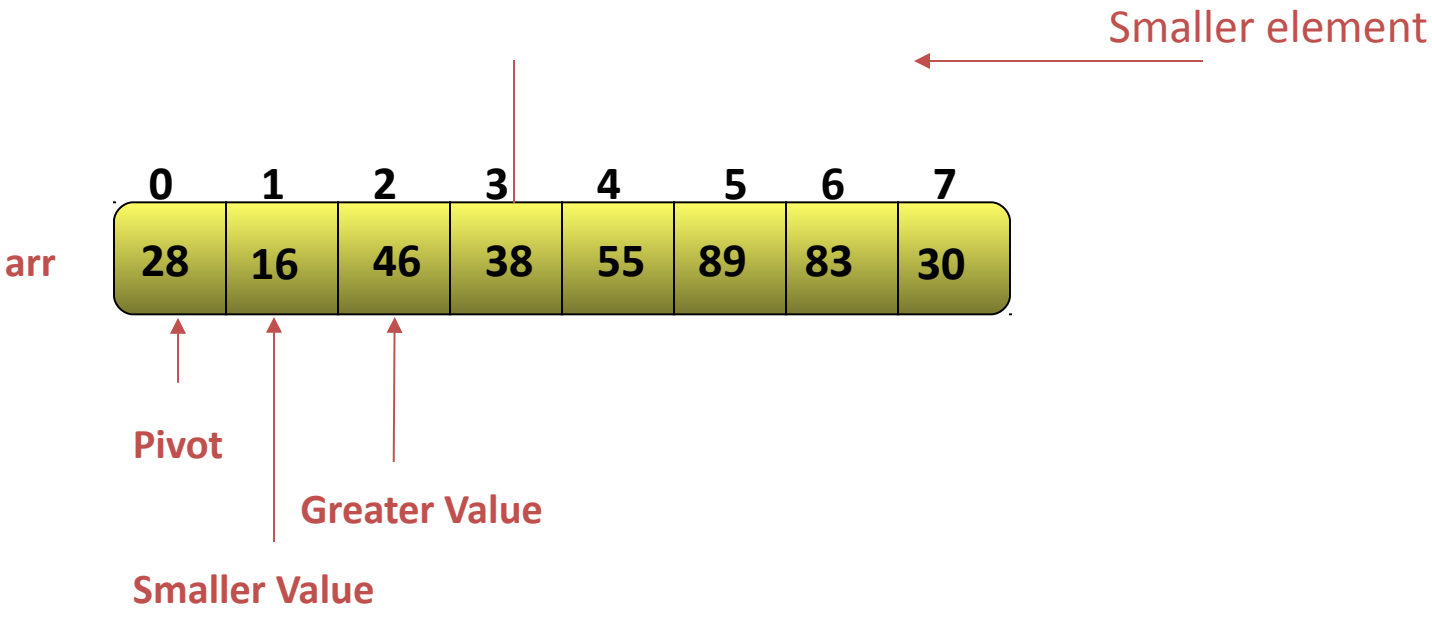
Implementing Quick Sort Algorithm (Contd.)

- Continue the search for an element greater than the pivot.
- Start from arr[2] and move in the left to right direction.
- Search for the first element that is greater than the pivot value.



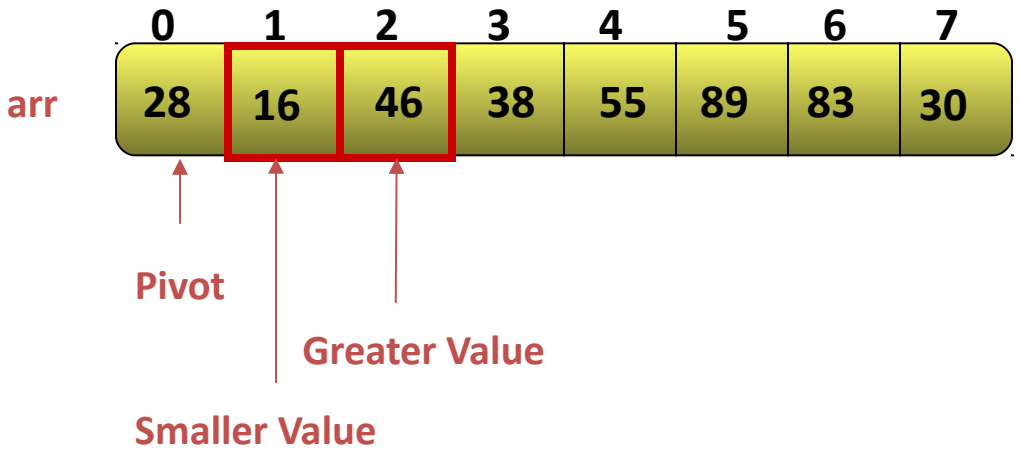
Implementing Quick Sort Algorithm (Contd.)

- Continue the search for an element smaller than the pivot.
- Start from arr[3] and move in the right to left direction.
- Search for the first element that is smaller than or equal to the pivot value.



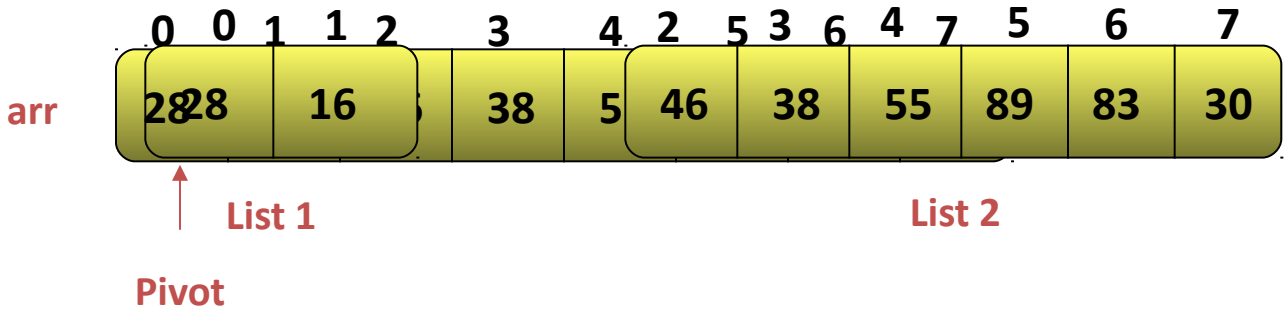
Implementing Quick Sort Algorithm (Contd.)

- The smaller value is on the left hand side of the greater value.
- Values remain same.



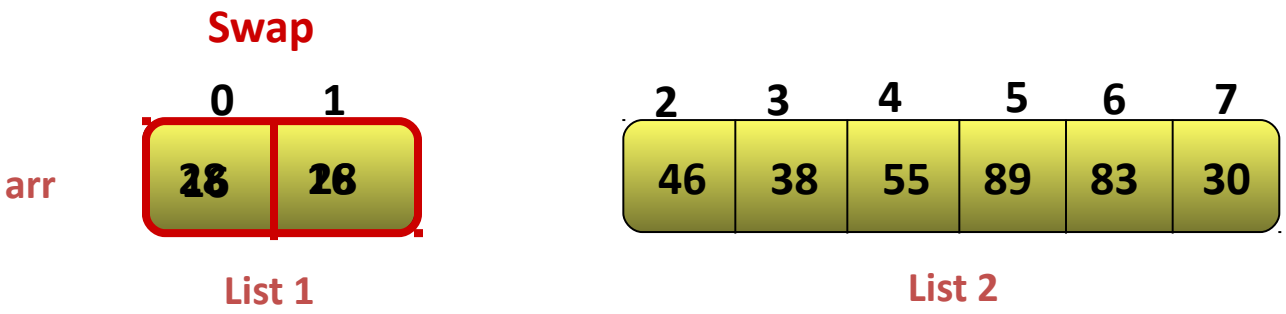
Implementing Quick Sort Algorithm (Contd.)

- List is now partitioned into two sublists.
- List 1 contains all values less than or equal to the pivot.
- List 2 contains all the values greater than the pivot.



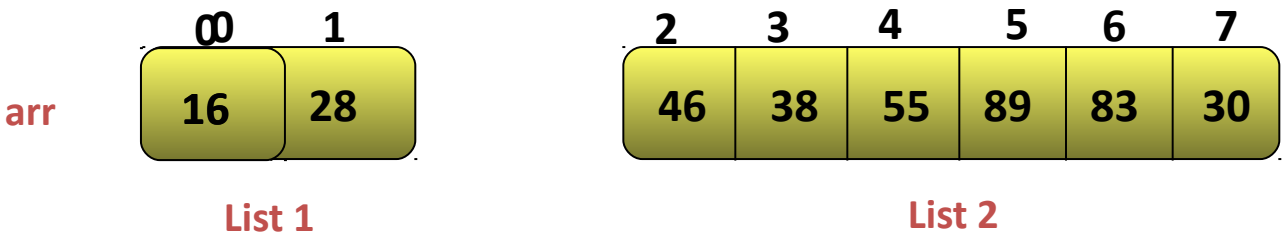
Implementing Quick Sort Algorithm (Contd.)

- Replace the pivot value with the last element of List 1.
- The pivot value, 28 is now placed at its correct position in the list.



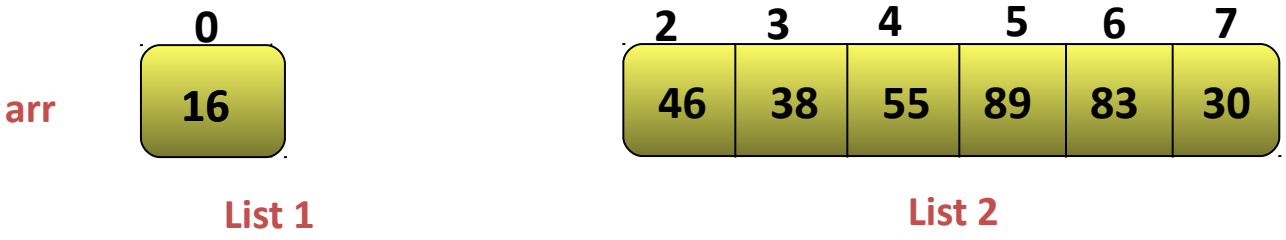
Implementing Quick Sort Algorithm (Contd.)

- Truncate the last element, that is, pivot from List 1.



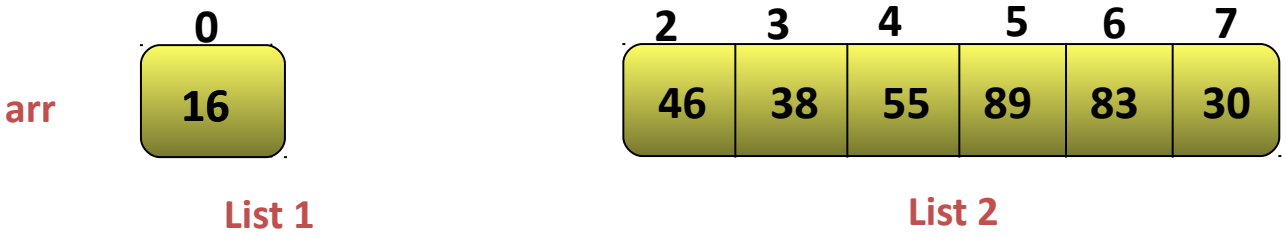
Implementing Quick Sort Algorithm (Contd.)

- List 1 has only one element.
- Therefore, no sorting required.



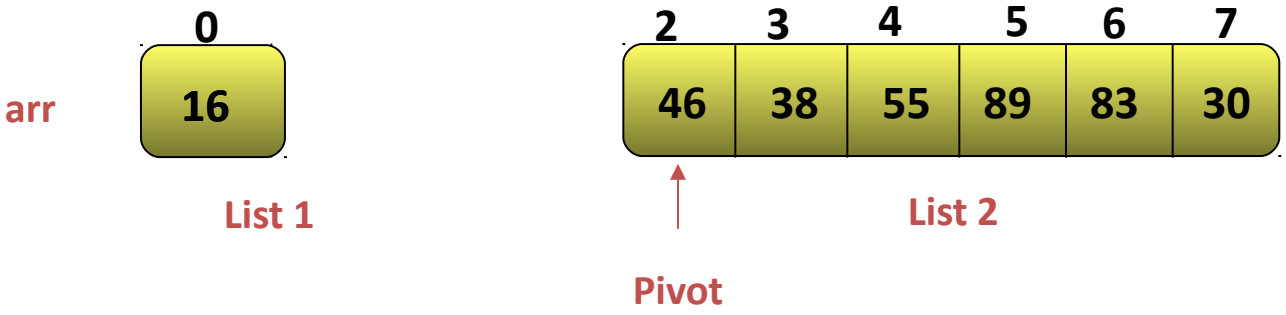
Implementing Quick Sort Algorithm (Contd.)

- Sort the second list, List 2.



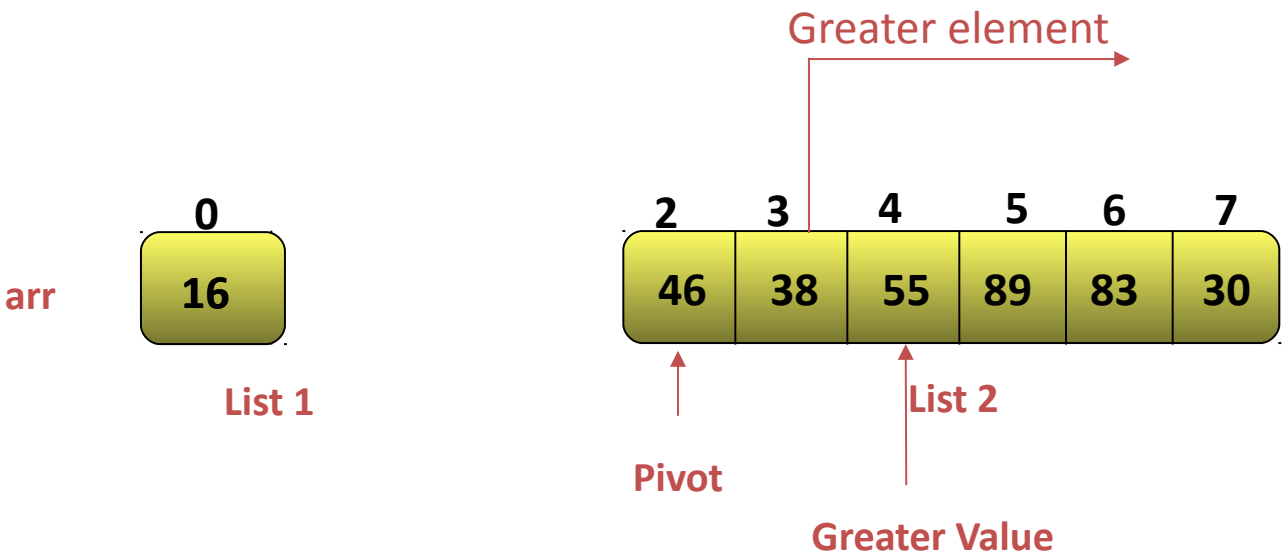
Implementing Quick Sort Algorithm (Contd.)

- Select a pivot.
- The pivot in this case will be arr[2], that is, 46.



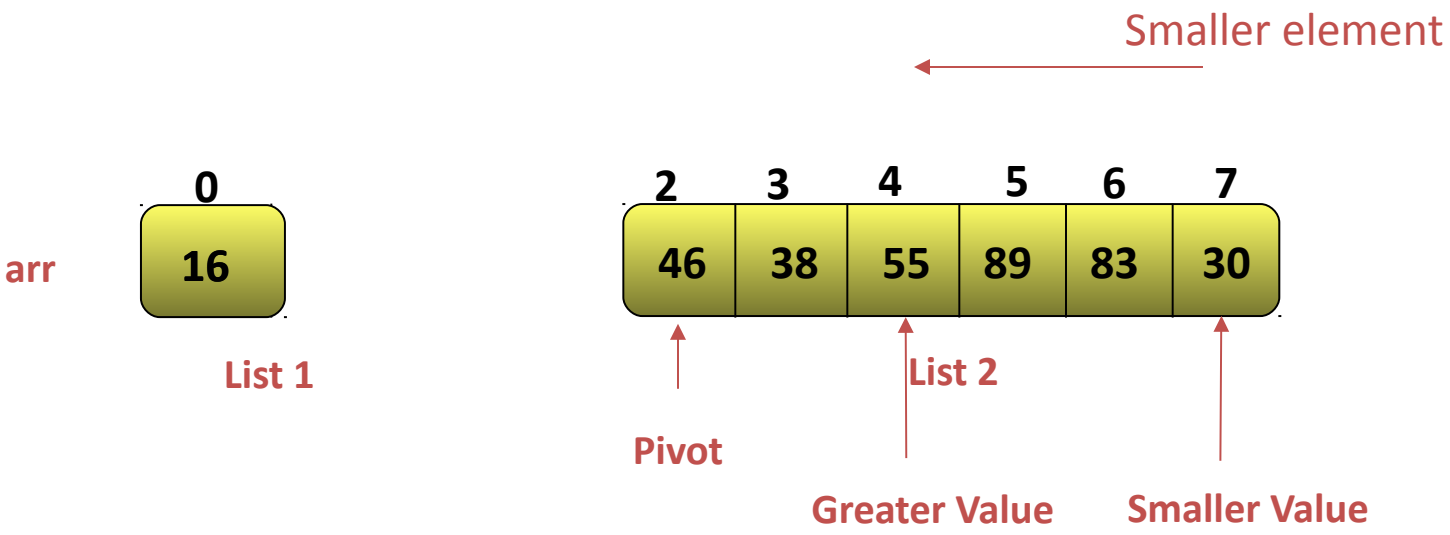
Implementing Quick Sort Algorithm (Contd.)

- Start from the left end of the list (at index 3).
- Move in the left to right direction.
- Search for the first element that is greater than the pivot value.



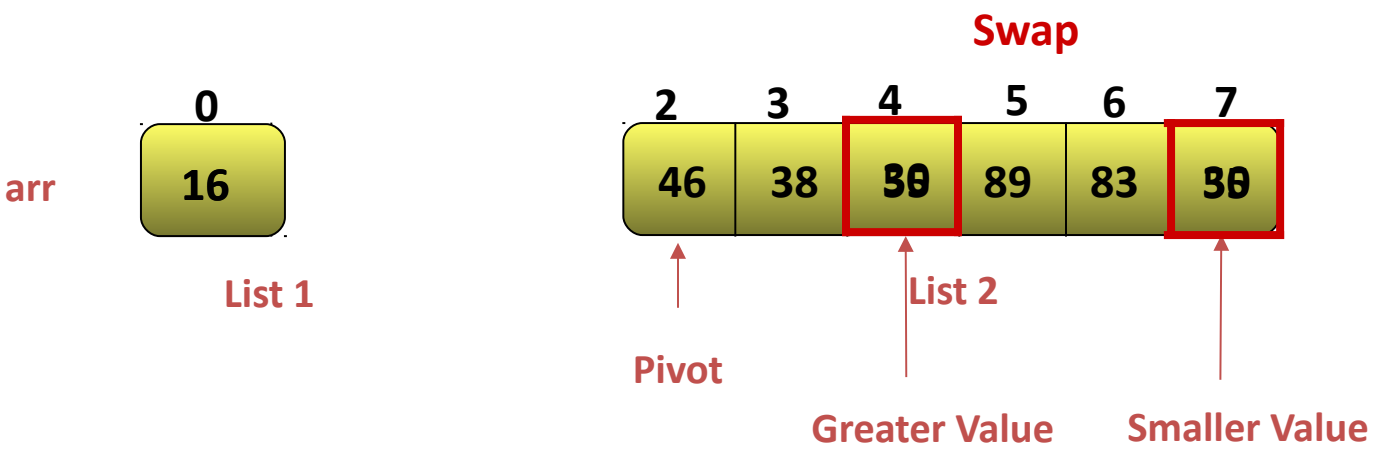
Implementing Quick Sort Algorithm (Contd.)

- Start from the right end of the list (at index 7).
- Move in the right to left direction.
- Search for the first element that is smaller than or equal to the pivot value.



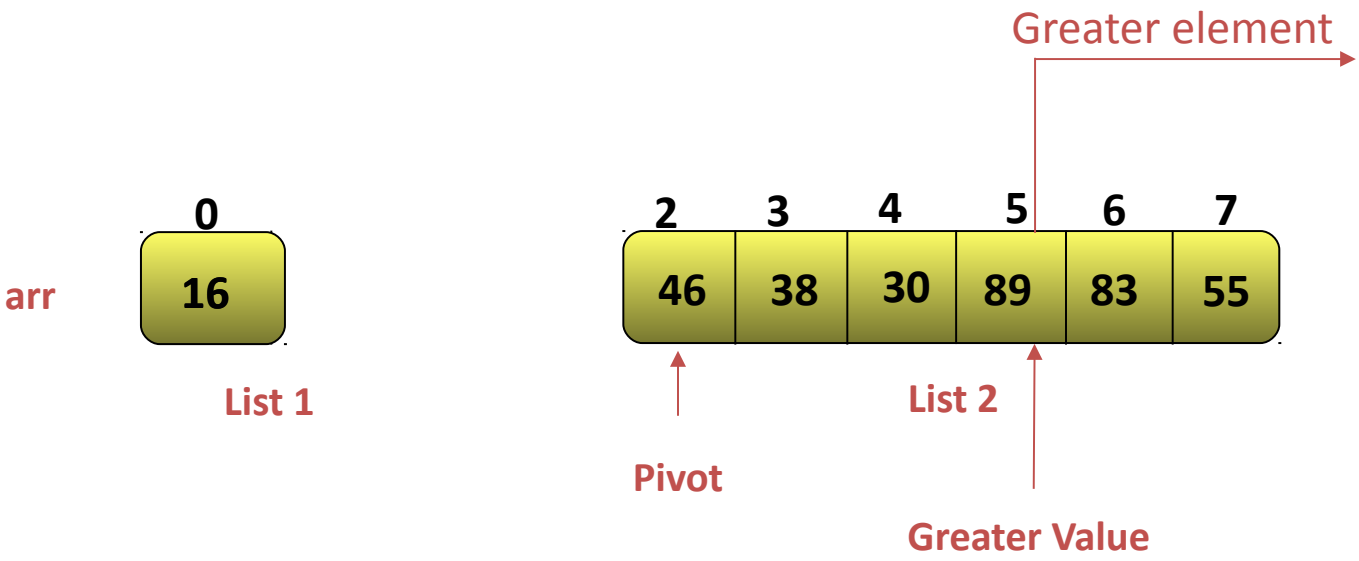
Implementing Quick Sort Algorithm (Contd.)

- Interchange the greater value with smaller value.



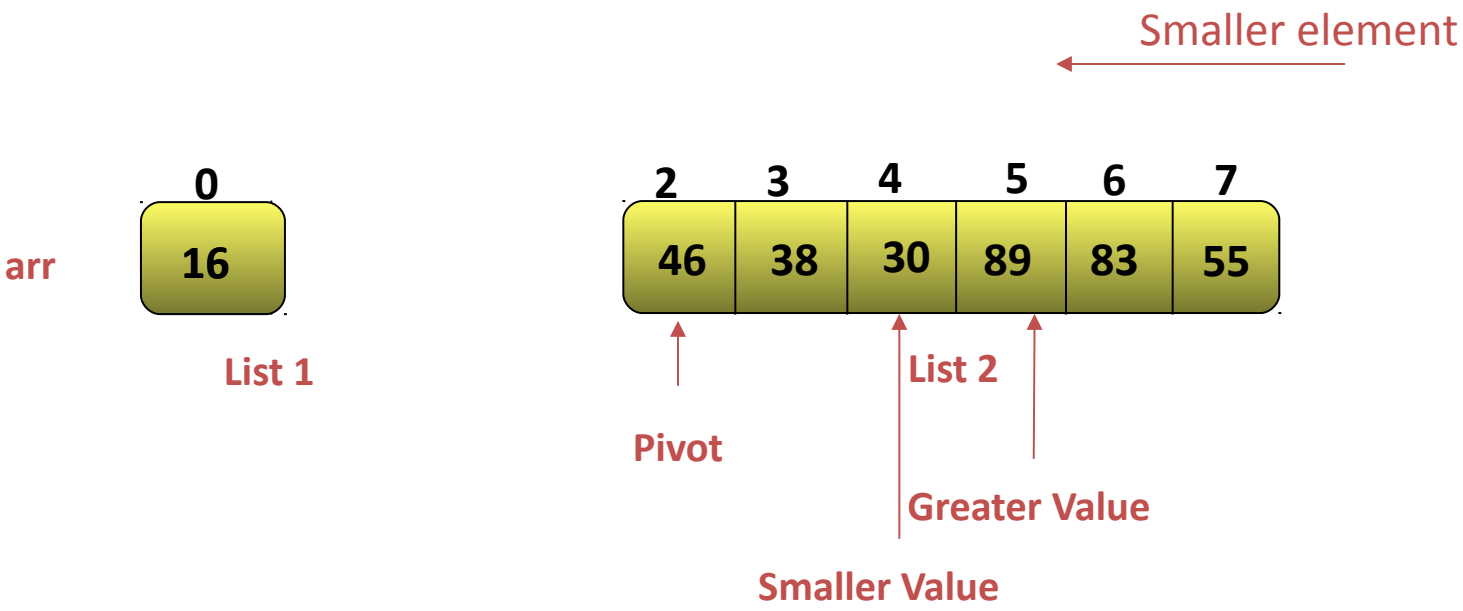
Implementing Quick Sort Algorithm (Contd.)

- Continue the search for an element greater than the pivot.
- Start from arr[5] and move in the left to right direction.
- Search for the first element that is greater than the pivot value.



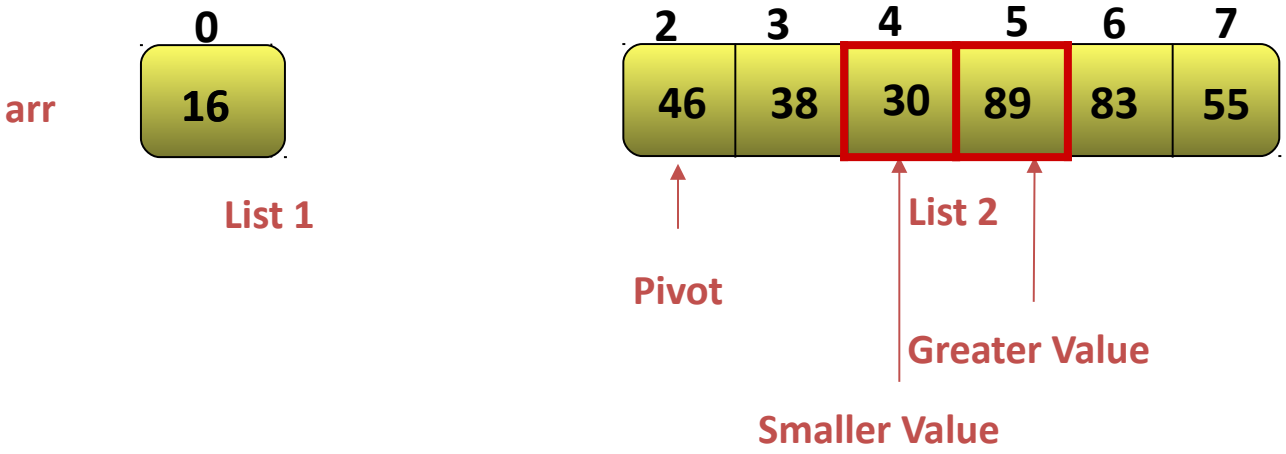
Implementing Quick Sort Algorithm (Contd.)

- Continue the search for an element smaller than the pivot.
- Start from arr[6] and move in the right to left direction.
- Search for the first element that is smaller than the pivot value.



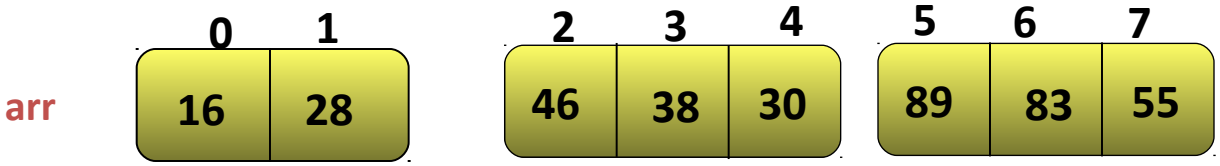
Implementing Quick Sort Algorithm (Contd.)

- The smaller value is on the left hand side of the greater value.
- Values remain same.



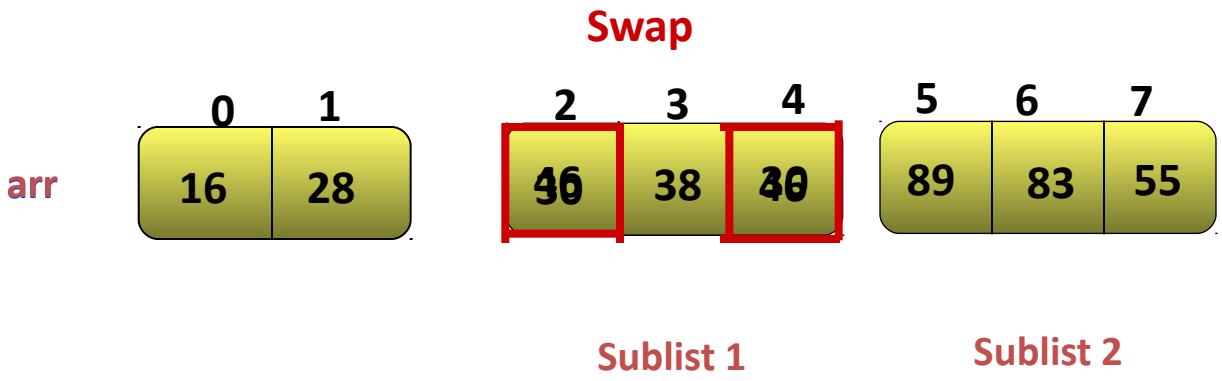
Implementing Quick Sort Algorithm (Contd.)

- Divide the list into two sublists.
- Sublist 1 contains all values less than or equal to the pivot.
- Sublist 2 contains all the values greater than the pivot.



Implementing Quick Sort Algorithm (Contd.)

- Replace the pivot value with the last element of Sublist 1.
- The pivot value, 46 is now placed at its correct position in the list.
- This process is repeated until all elements reach their correct position.



To review

- In quick sort algorithm, you:
  - Select an element from the list called as pivot.
  - Partition the list into two parts such that:
    - **All the elements towards the left end of the list are smaller than the pivot.**
    - **All the elements towards the right end of the list are greater than the pivot.**
  - Store the pivot at its correct position between the two parts of the list.
- You repeat this process for each of the two sublists created after partitioning.
- This process continues until one element is left in each sublist.

```
Algorithm  QuickSort (arr [ n], low, high)
{    //  arr is an array of n elements, low is lower bound of an array and high  is upper bound.
. If (low >= high): Return;
. else
  3. pivot = arr [low]
  4. i = low + 1
  5. j = high
  6. do {
       6.1 while ( arr [i]<pivot and i<=high) // search element higher than pivot
                          6.1.1 i=i+1
       6.2 while ( arr [j]>pivot and j>=low ) // Search for an element smaller than pivot
            6.2.1 j=j-1
          6.3 If (i < j)
            6.3.1 Swap arr [i] with arr [j]
            6.3.2   i = i+ 1
            6.3.3   j = j - 1  // to start searching from next element  after swapping
    } while (i <= j);
  7.Swap arr [low] with arr [j]     //Swap pivot with last element in first part of the list
  8.QuickSort( arr [n], low, j – 1)   // Apply quicksort on list left to pivot
  9.QuickSort(arr [n], j + 1, high)    // Apply quicksort on list right to pivot
  }
```

Determining the Efficiency of Quick Sort Algorithm

- The total time taken by this sorting algorithm depends on the position of the pivot value.
- The worst case occurs when the list is already sorted.
- If the first element is chosen as the pivot, it leads to a worst case efficiency of $O(n^2)$.
- If you select the median of all values as the pivot, the efficiency would be O(n log n).

Just a minute

- What is the total number of comparisons for an average case in a quick sort algorithm?

- Answer:
  - O(n log n)