

Data Structures and Algorithms

Objectives

- In this session, you will learn to:
 - Sort data by using quick sort
 - Sort data by using merge sort
 - Search data by using linear search technique
 - Search data by using binary search technique

Data Structures and Algorithms

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

Data Structures and Algorithms

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.

Data Structures and Algorithms

Implementing Quick Sort Algorithm (Contd.)

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

	0	1	2	3	4	5	6	7
arr	28	55	46	38	16	89	83	30

Data Structures and Algorithms

Implementing Quick Sort Algorithm (Contd.)

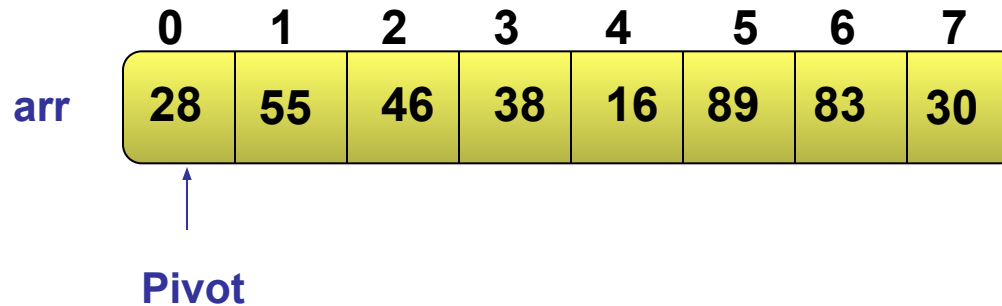
- Let us sort this unsorted list.

	0	1	2	3	4	5	6	7
arr	28	55	46	38	16	89	83	30

Data Structures and Algorithms

Implementing Quick Sort Algorithm (Contd.)

- Select a Pivot.



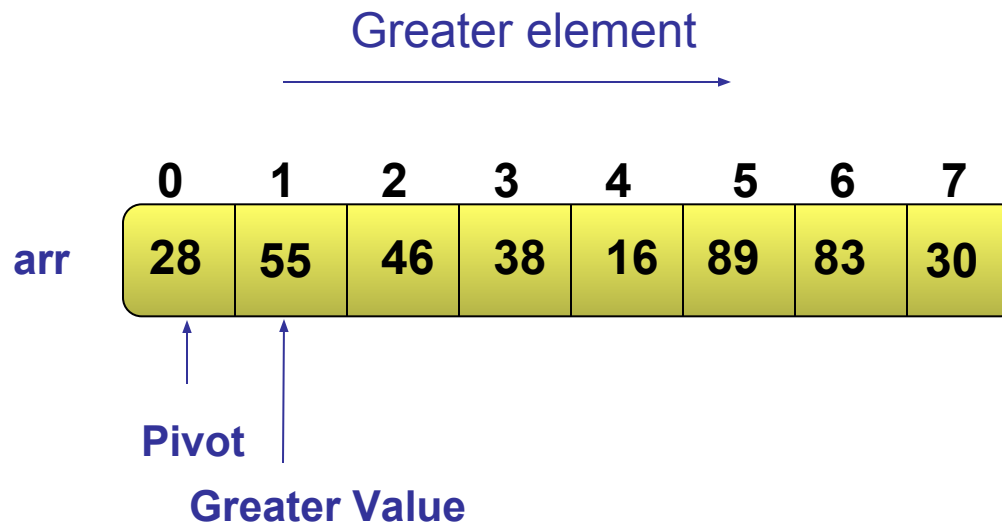
	0	1	2	3	4	5	6	7
arr	28	55	46	38	16	89	83	30

Pivot

Data Structures and Algorithms

Implementing Quick Sort Algorithm (Contd.)

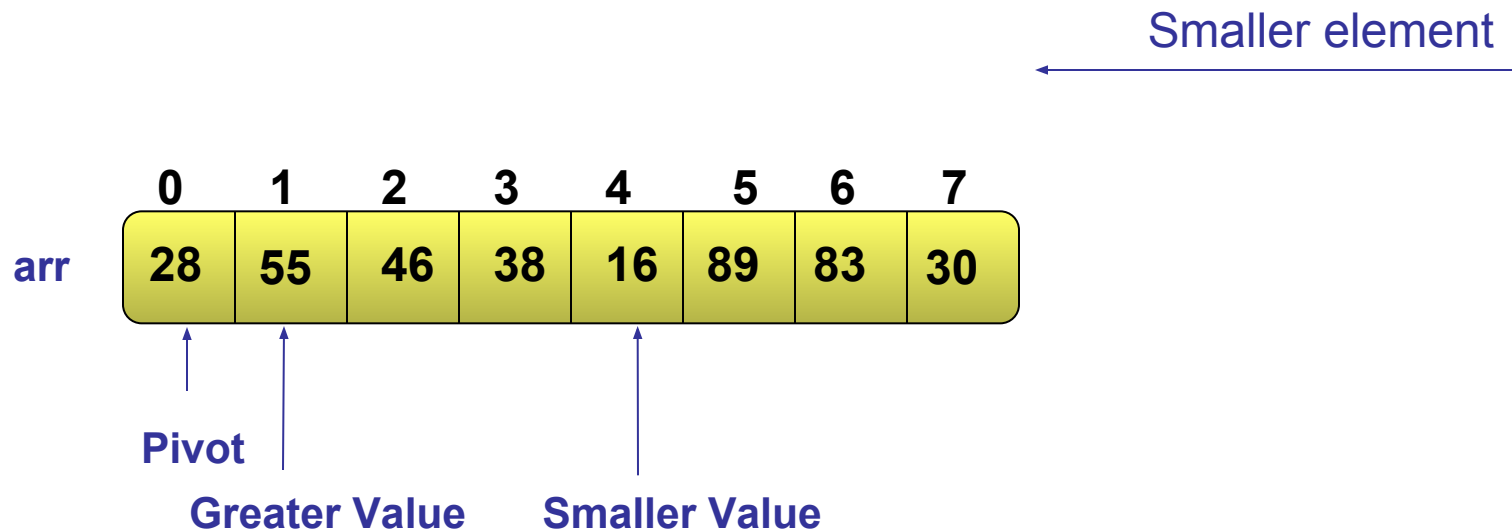
- Start from the left end of the list (at index 1).
- Move in the left to right direction.
- Search for the first element that is greater than the pivot value.



Data Structures and Algorithms

Implementing Quick Sort Algorithm (Contd.)

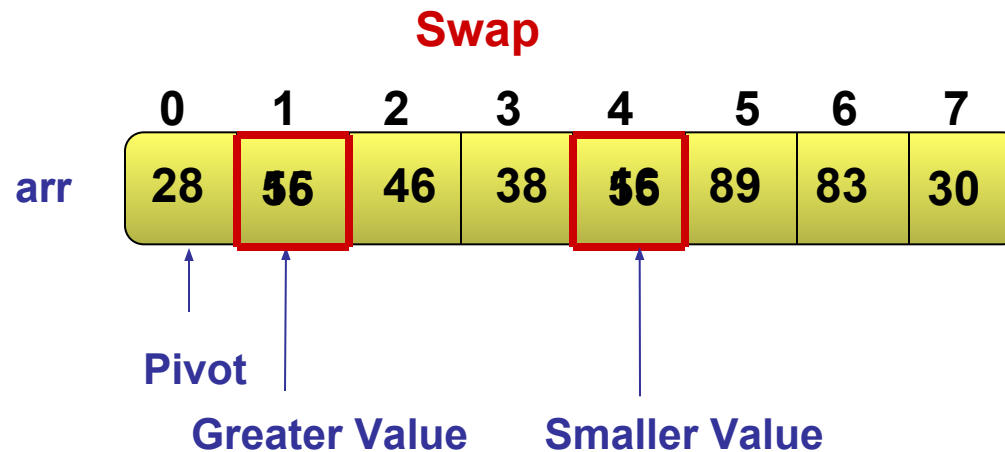
- Start from the right end of the list.
- Move in the right to left direction.
- Search for the first element that is smaller than or equal to the pivot value.



Data Structures and Algorithms

Implementing Quick Sort Algorithm (Contd.)

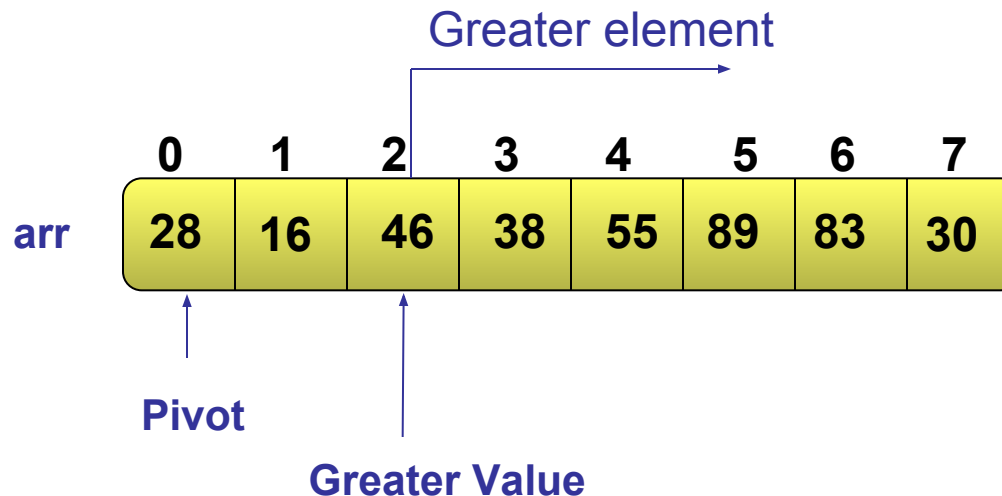
- Interchange the greater value with smaller value.



Data Structures and Algorithms

Implementing Quick Sort Algorithm (Contd.)

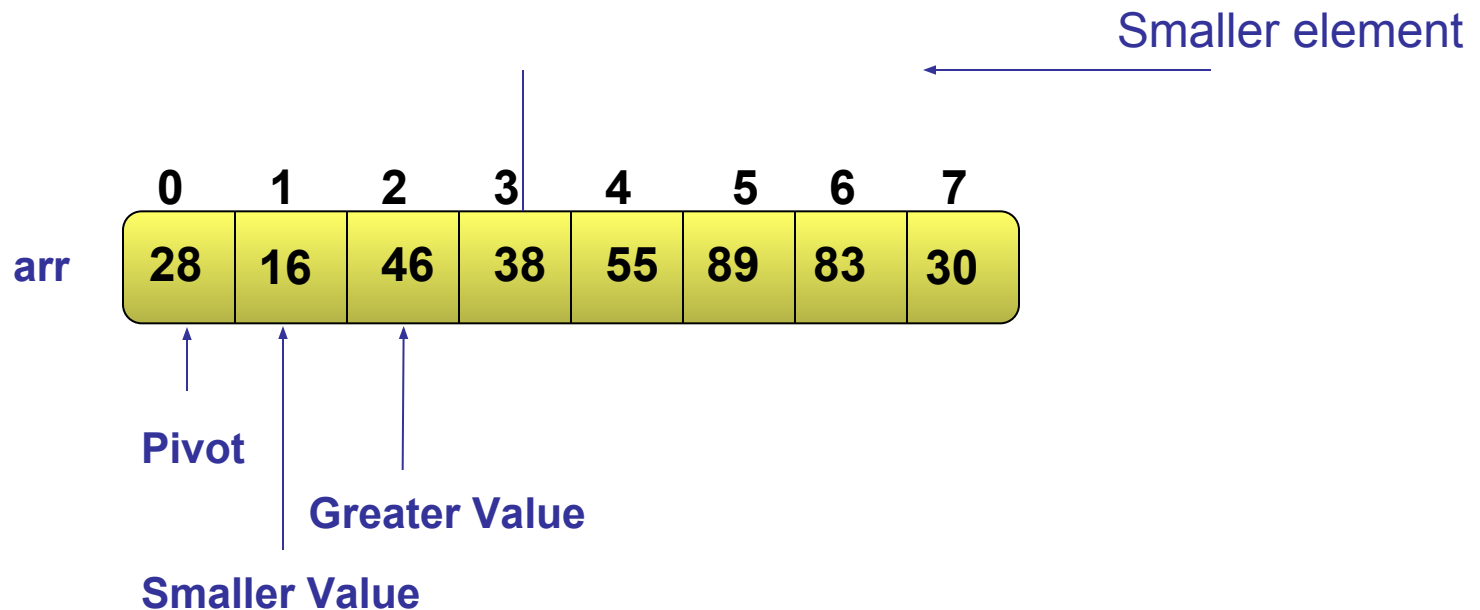
- Continue the search for an element greater than the pivot.
- Start from arr[2] and move in the left to right direction.
- Search for the first element that is greater than the pivot value.



Data Structures and Algorithms

Implementing Quick Sort Algorithm (Contd.)

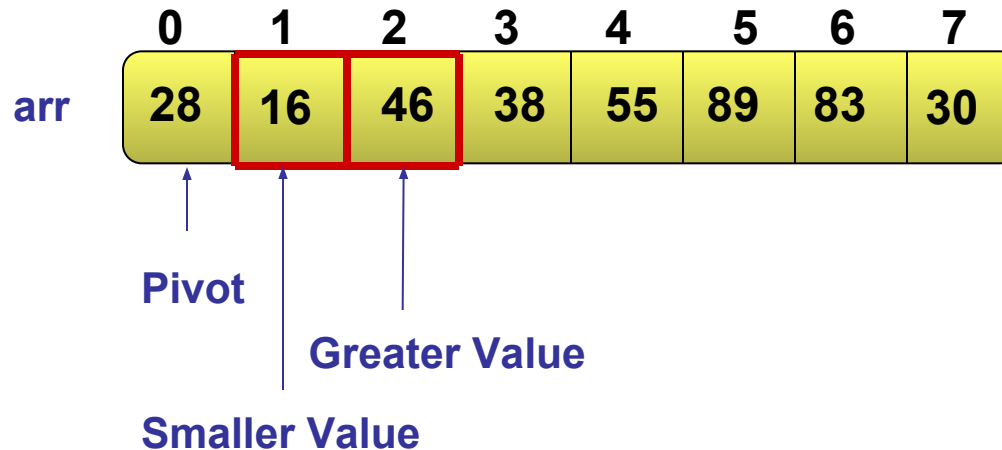
- Continue the search for an element smaller than the pivot.
- Start from `arr[3]` and move in the right to left direction.
- Search for the first element that is smaller than or equal to the pivot value.



Data Structures and Algorithms

Implementing Quick Sort Algorithm (Contd.)

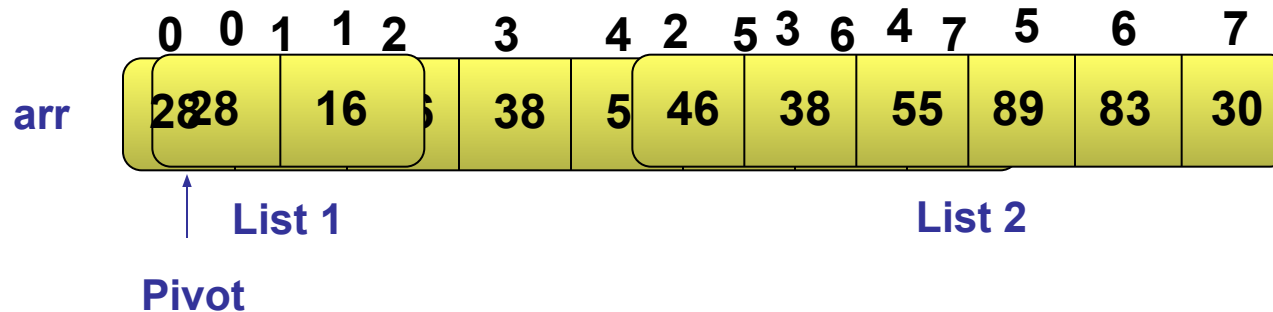
- The smaller value is on the left hand side of the greater value.
- Values remain same.



Data Structures and Algorithms

Implementing Quick Sort Algorithm (Contd.)

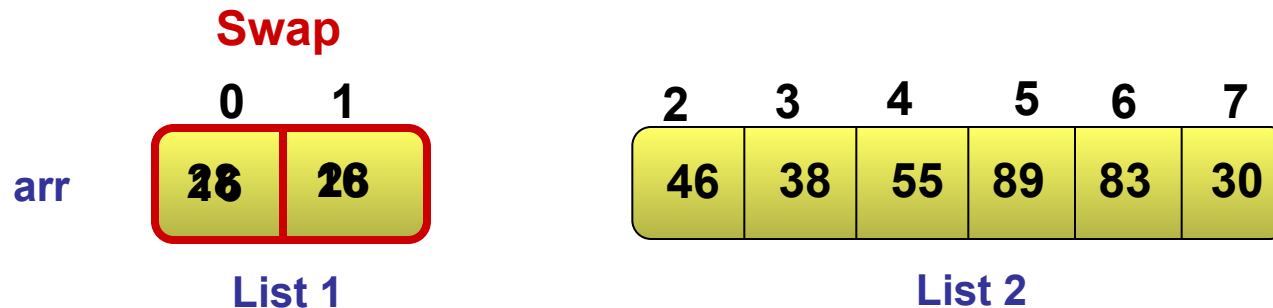
- List is now partitioned into two sublists.
- List 1 contains all values less than or equal to the pivot.
- List 2 contains all the values greater than the pivot.



Data Structures and Algorithms

Implementing Quick Sort Algorithm (Contd.)

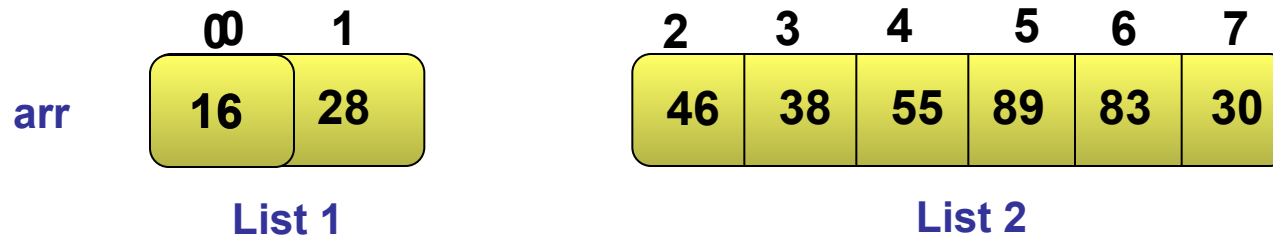
- Replace the pivot value with the last element of List 1.
- The pivot value, 28 is now placed at its correct position in the list.



Data Structures and Algorithms

Implementing Quick Sort Algorithm (Contd.)

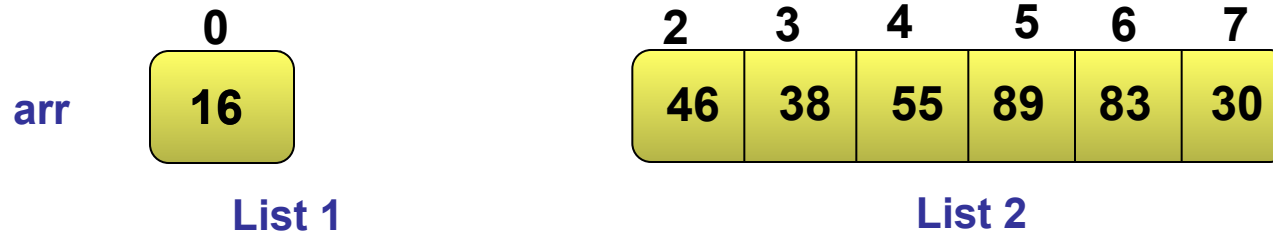
- Truncate the last element, that is, pivot from List 1.



Data Structures and Algorithms

Implementing Quick Sort Algorithm (Contd.)

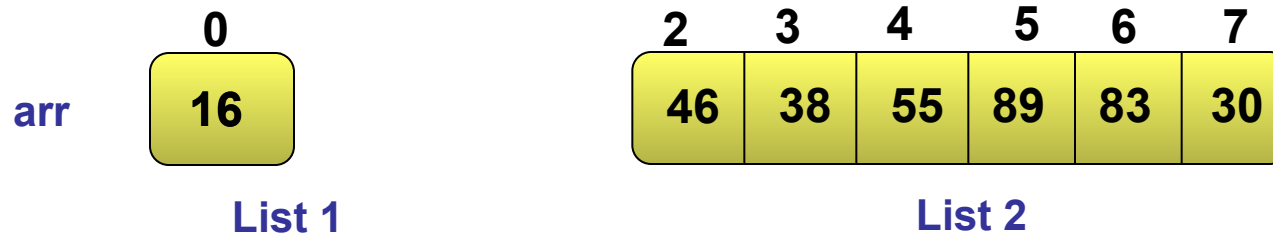
- List 1 has only one element.
- Therefore, no sorting required.



Data Structures and Algorithms

Implementing Quick Sort Algorithm (Contd.)

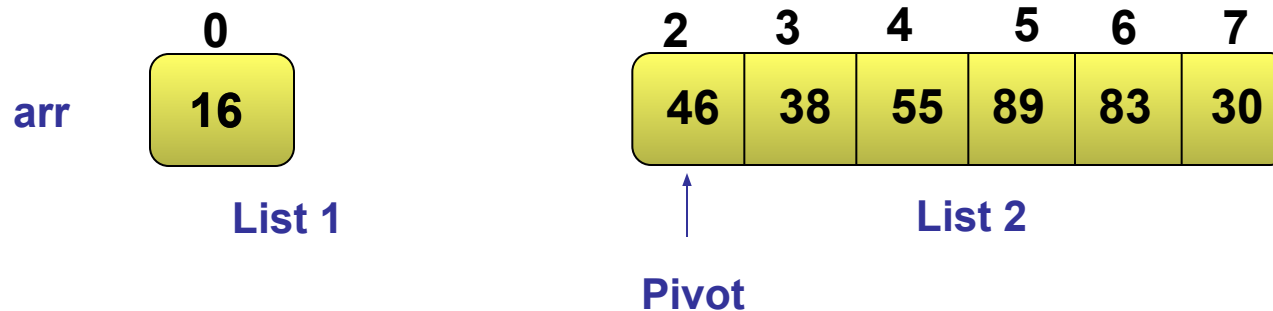
- Sort the second list, List 2.



Data Structures and Algorithms

Implementing Quick Sort Algorithm (Contd.)

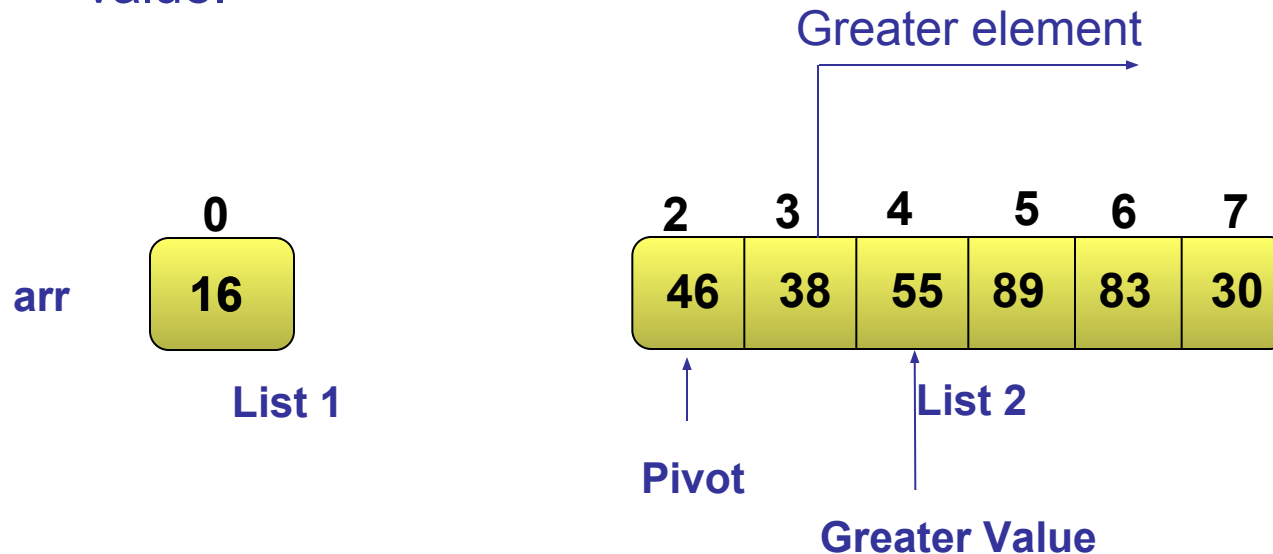
- Select a pivot.
- The pivot in this case will be $\text{arr}[2]$, that is, 46.



Data Structures and Algorithms

Implementing Quick Sort Algorithm (Contd.)

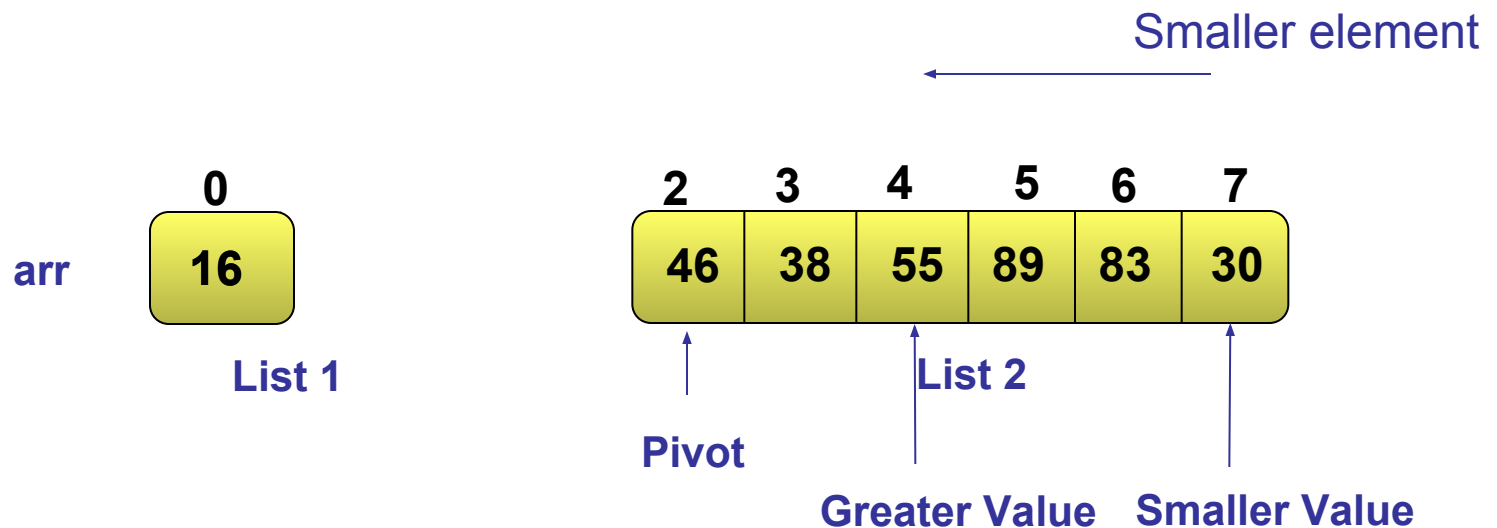
- Start from the left end of the list (at index 3).
- Move in the left to right direction.
- Search for the first element that is greater than the pivot value.



Data Structures and Algorithms

Implementing Quick Sort Algorithm (Contd.)

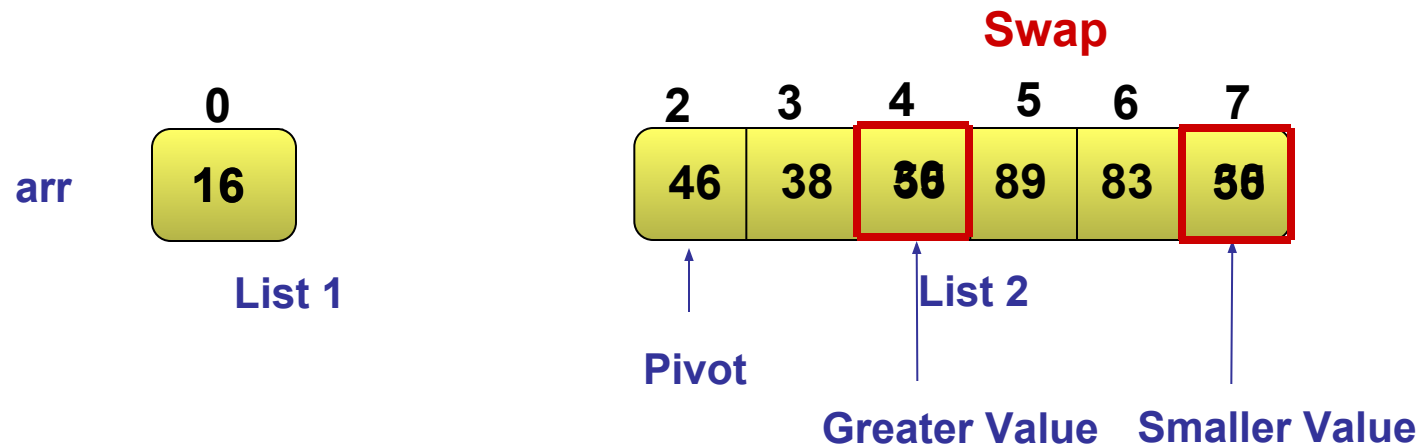
- Start from the right end of the list (at index 7).
- Move in the right to left direction.
- Search for the first element that is smaller than or equal to the pivot value.



Data Structures and Algorithms

Implementing Quick Sort Algorithm (Contd.)

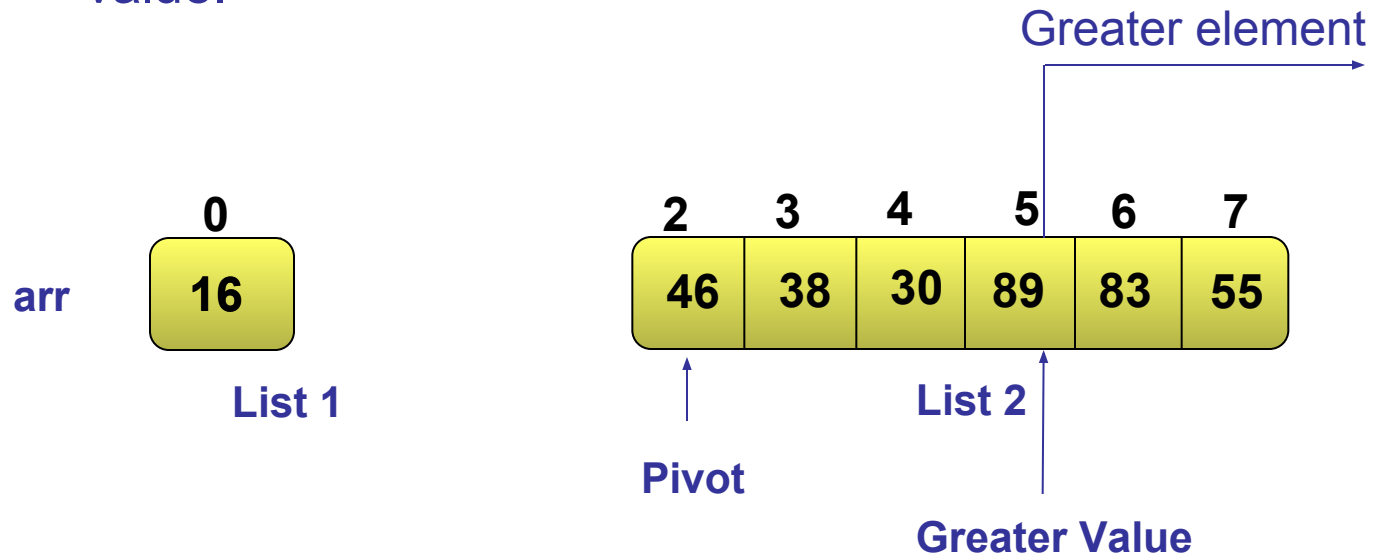
- Interchange the greater value with smaller value.



Data Structures and Algorithms

Implementing Quick Sort Algorithm (Contd.)

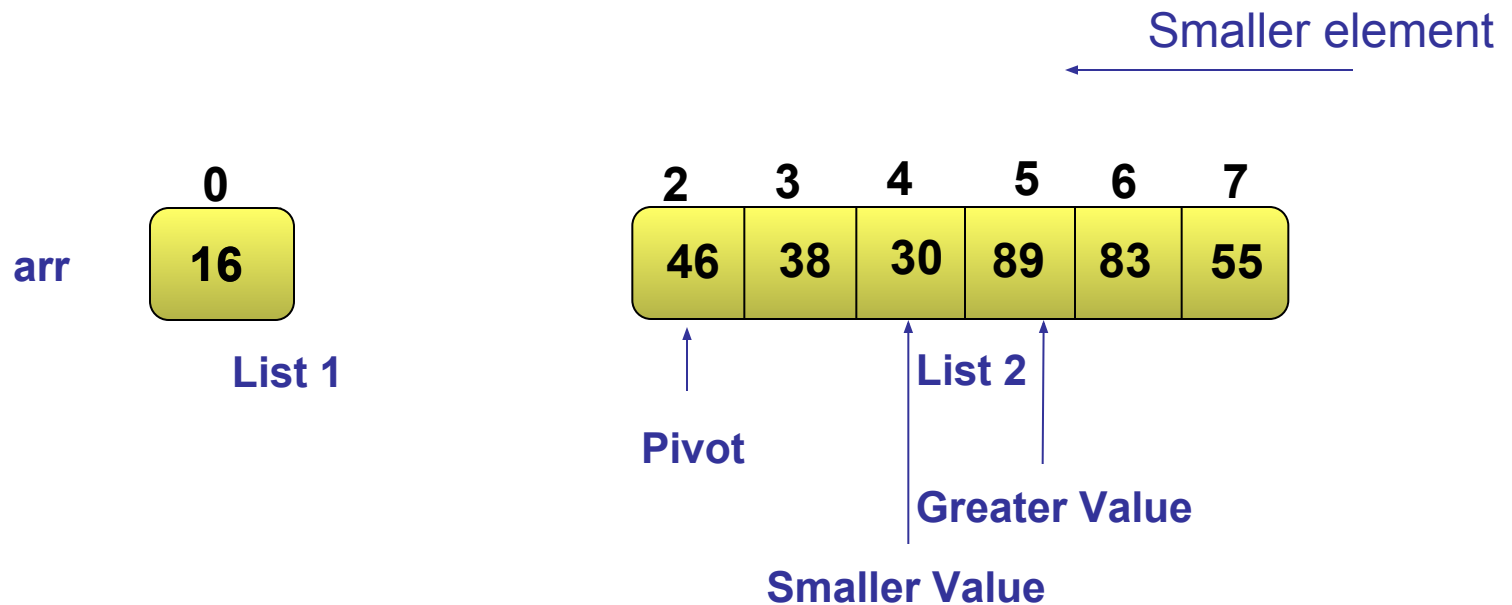
- Continue the search for an element greater than the pivot.
- Start from arr[5] and move in the left to right direction.
- Search for the first element that is greater than the pivot value.



Data Structures and Algorithms

Implementing Quick Sort Algorithm (Contd.)

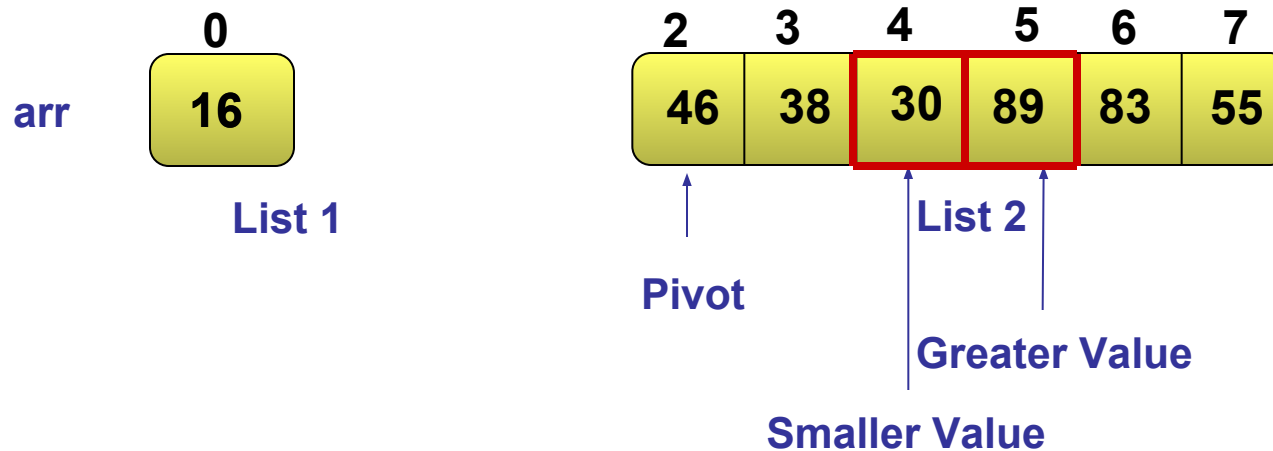
- Continue the search for an element smaller than the pivot.
- Start from arr[6] and move in the right to left direction.
- Search for the first element that is smaller than the pivot value.



Data Structures and Algorithms

Implementing Quick Sort Algorithm (Contd.)

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



Data Structures and Algorithms

Implementing Quick Sort Algorithm (Contd.)

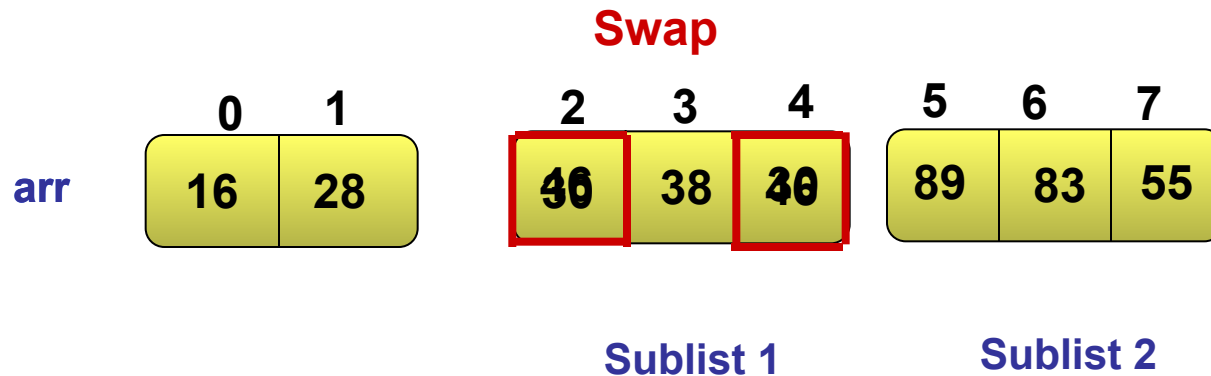
- Divide the list into two sublists.
- Sublist 1 contains all values less than or equal to the pivot.
- Sublist 2 contains all the values greater than the pivot.

arr	0	1	2	3	4	5	6	7
	16	28	46	38	30	89	83	55

Data Structures and Algorithms

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.



Data Structures and Algorithms

Implementing Quick Sort Algorithm (Contd.)

- Write an algorithm to implement quick sort:

QuickSort(low,high)

1. If (low > high):
 - a. Return
2. Set pivot = arr[low]
3. Set i = low + 1
4. Set j = high
5. Repeat step 6 until i > high or arr[i] > pivot // **Search for an element greater than**
//
//
pivot
6. Increment i by 1
7. Repeat step 8 until j < low or arr[j] < pivot // **Search for an element**

// smaller than pivot
8. Decrement j by 1
9. If i < j: // **If greater element is on the left of smaller element**

Data Structures and Algorithms

Implementing Quick Sort Algorithm (Contd.)

10. If $i \leq j$:
 - a. Go to step 5 // **Continue the search**
11. If $low < j$:
 - a. Swap $arr[low]$ with $arr[j]$ // **Swap pivot with last element in**
// first part of the
list
12. QuickSort($low, j - 1$) // **Apply quicksort on list left to pivot**
13. QuickSort($j + 1, high$) // **Apply quicksort on list right to pivot**

Data Structures and Algorithms

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 $O(n \log n)$.

Data Structures and Algorithms

Just a minute

- What is the total number of comparisons for an average case in a quick sort algorithm?
- Answer:
 - $O(n \log n)$

Data Structures and Algorithms

Activity: Sorting Data by Using Quick Sort Algorithm

- Problem Statement:
 - Write a program that stores 10 numbers in an array, and sorts them by using the quick sort algorithm.

Data Structures and Algorithms

Sorting Data by Using Merge Sort

- Merge sort algorithm:
 - Is based on the divide and conquer approach
 - Divides the list into two sublists of sizes as nearly equal as possible
 - Sorts the two sublists separately by using merge sort
 - Merges the sorted sublists into one single list

Data Structures and Algorithms

Implementing Merge Sort Algorithm

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

	0	1	2	3	4	5	6
arr	53	10	30	76	3	57	24

Data Structures and Algorithms

Implementing Merge Sort Algorithm (Contd.)

- Let us sort this unsorted list.

	0	1	2	3	4	5	6
arr	53	10	30	76	3	57	24

Data Structures and Algorithms

Implementing Merge Sort Algorithm (Contd.)

- The first step to sort data by using merge sort is to split the list into two parts.

	0	1	2	3	4	5	6
arr	53	10	30	76	3	57	24

Data Structures and Algorithms

Implementing Merge Sort Algorithm (Contd.)

- The first step to sort data by using merge sort is to split the list into two parts.

	0	1	2	3	4	5	6
arr	53	10	30	76	3	57	24

Data Structures and Algorithms

Implementing Merge Sort Algorithm (Contd.)

- The list has odd number of elements, therefore, the left sublist is longer than the right sublist by one entry.

	0	1	2	3	4	5	6
arr	53	10	30	76	3	57	24

Data Structures and Algorithms

Implementing Merge Sort Algorithm (Contd.)

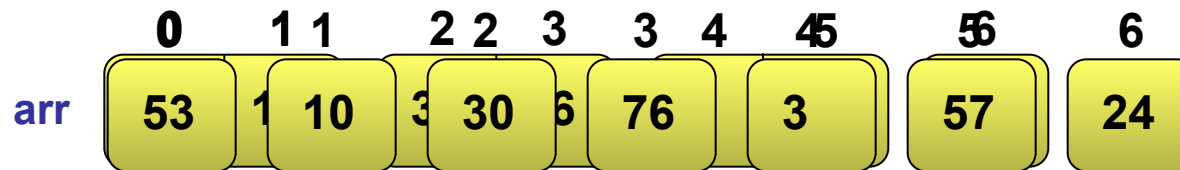
- Further divide the two sublists into nearly equal parts.

	0	1	2	2	3	3	4	4	5	5	6	6
arr	53	10	30	76	1	3	57	4	24			

Data Structures and Algorithms

Implementing Merge Sort Algorithm (Contd.)

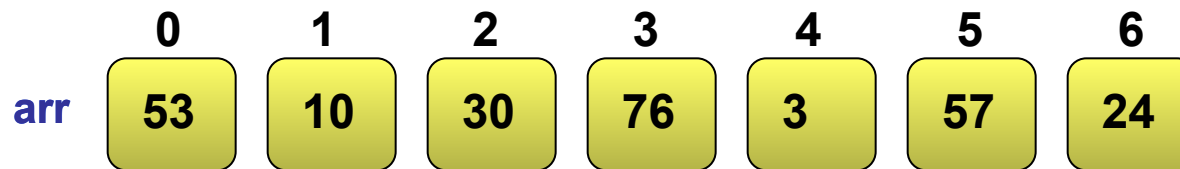
- Further divide the sublists.



Data Structures and Algorithms

Implementing Merge Sort Algorithm (Contd.)

- There is a single element left in each sublist.
- Sublists with one element require no sorting.



The diagram shows an array labeled 'arr' with 7 elements. Each element is represented by a yellow rounded rectangle with a black border. Above each rectangle is its index (0 to 6). The values inside the rectangles are 53, 10, 30, 76, 3, 57, and 24. The background of the slide features a blue gradient on the left and a collage of images (hands, keyboard, person) on the right.

	0	1	2	3	4	5	6
arr	53	10	30	76	3	57	24

Data Structures and Algorithms

Implementing Merge Sort Algorithm (Contd.)

- Start merging the sublists to obtain a sorted list.

arr

0	1	1	2	2	3	3	4	5	6	6
10	53	0	30	76	7	3	57	24	24	

Data Structures and Algorithms

Implementing Merge Sort Algorithm (Contd.)

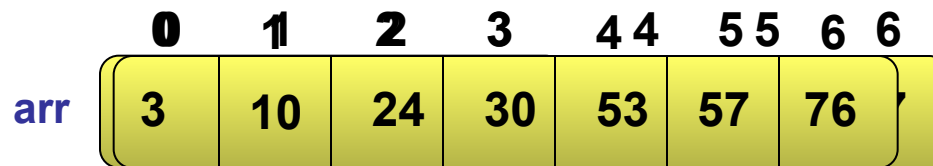
- Further merge the sublists.

	0	1	2	2	3	3	4	4	5	5	6	6
arr	10	30	53	76	5	3	24	57	24			

Data Structures and Algorithms

Implementing Merge Sort Algorithm (Contd.)

- Again, merge the sublists.



	0	1	2	3	4 4	5 5	6 6
arr	3	10	24	30	53	57	76

Data Structures and Algorithms

Implementing Merge Sort Algorithm (Contd.)

- The list is now sorted.

	0	1	2	3	4	5	6
arr	3	10	24	30	53	57	76

Data Structures and Algorithms

Implementing Merge Sort Algorithm (Contd.)

- Write an algorithm to implement merge sort:

MergeSort(low,high)

1. If $(low \geq high)$:
 - a. Return
2. Set $mid = (low + high)/2$
3. Divide the list into two sublists of nearly equal lengths, and sort each sublist by using merge sort. The steps to do this are as follows:
 - a. MergeSort(low, mid
 - b. MergeSort(mid + 1, high)
4. Merge the two sorted sublists:
 - a. Set $i = low$
 - b. Set $j = mid + 1$
 - c. Set $k = low$
 - d. Repeat until $i > mid$ or $j > high$: **// This loop will terminate**

when

// you reach the end of one of the

// two sublists.

Data Structures and Algorithms

Implementing Merge Sort Algorithm (Contd.)

i. If ($\text{arr}[i] \leq \text{arr}[j]$)

 Store $\text{arr}[i]$ at index k in array B

 Increment i by 1

 Else

 Store $\text{arr}[j]$ at index k in array B

 Increment j by 1

ii. Increment k by 1

e. Repeat until $j > \text{high}$: **// If there are still some elements in the**

// second sublist append them to the new list

i. Store $\text{arr}[j]$ at index k in array B

ii. Increment j by 1

iii. Increment k by 1

f. Repeat until $i > \text{mid}$: **// If there are still some elements in the**

// first sublist append them to the new list

i. Store $\text{arr}[i]$ at index k in array B

ii. Increment i by 1

iii. Increment k by 1

5. Copy all elements from the sorted array B into the original array arr

Data Structures and Algorithms

Determining the Efficiency of Merge Sort Algorithm

- To sort the list by using merge sort algorithm, you need to recursively divide the list into two nearly equal sublists until each sublist contains only one element.
- To divide the list into sublists of size one requires $\log n$ passes.
- In each pass, a maximum of n comparisons are performed.
- Therefore, the total number of comparisons will be a maximum of $n \times \log n$.
- The efficiency of merge sort is equal to $O(n \log n)$
- There is no distinction between best, average, and worst case efficiencies of merge sort because all of them require the same amount of time.

Data Structures and Algorithms

Just a minute

- Which algorithm uses the following procedure to sort a given list of elements?
 1. Select an element from the list called a pivot.
 2. Partition the list into two parts such that one part contains elements lesser than the pivot, and the other part contains elements greater than the pivot.
 3. Place the pivot at its correct position between the two lists.
 4. Sort the two parts of the list using the same algorithm.
- Answer:
 - Quick sort

Data Structures and Algorithms

Just a minute

- On which algorithm design technique are quick sort and merge sort based?
- Answer:
 - Quick sort and merge sort are based on the divide and conquer technique.

Data Structures and Algorithms

Performing Linear Search

- Linear Search:
 - Is the simplest searching method
 - Is also referred to as sequential search
 - Involves comparing the items sequentially with the elements in the list

Data Structures and Algorithms

Implementing Linear Search

- The linear search would begin by comparing the required element with the first element in the list.
- If the values do not match:
 - The required element is compared with the second element in the list.
- If the values still do not match:
 - The required element is compared with the third element in the list.
- This process continues, until:
 - The required element is found or the end of the list is reached.

Data Structures and Algorithms

Implementing Linear Search (Contd.)

- Write an algorithm to search for a given employee ID in a list of employee records by using linear search algorithm:
 1. Read the employee ID to be searched
 2. Set $i = 0$
 3. Repeat step 4 until $i > n$ or $\text{arr}[i] = \text{employee ID}$
 4. Increment i by 1
 5. If $i > n$:
 - Display "Not Found"
 - Else
 - Display "Found"

Data Structures and Algorithms

Determining the Efficiency of Linear Search

- The efficiency of a searching algorithm is determined by the running time of the algorithm.
- In the best case scenario:
 - The element is found at the first position in the list.
 - The number of comparisons in this case is 1.
 - The best case efficiency of linear search is therefore, $O(1)$.
- In the worst case scenario:
 - The element is found at the last position of the list or does not exist in the list.
 - The number of comparisons in this case is equal to the number of elements.
 - The worst case efficiency of linear search is therefore, $O(n)$.

Data Structures and Algorithms

Determining the Efficiency of Linear Search (Contd.)

- In the average case scenario:
 - The number of comparisons for linear search can be determined by finding the average of the number of comparisons in the best and worst case.
- The average case efficiency of linear search is $\frac{1}{2}(n + 1)$.

Data Structures and Algorithms

Just a minute

- You have to apply linear search to search for an element in an array containing 5,000 elements. If, at the end of the search, you find that the element is not present in the array, how many comparisons you would have made to search the required element in the given list?
- Answer:
 - 5,000

Data Structures and Algorithms

Activity: Performing Linear Search

- Problem Statement:
 - Write a program to search a given number in an array that contains a maximum of 20 numbers by using the linear search algorithm. If there are more than one occurrences of the element to be searched, then the program should display the position of the first occurrence. The program should also display the total number of comparisons made.

Data Structures and Algorithms

Performing Binary Search

- Binary search algorithm:
 - Is used for searching large lists
 - Searches the element in very few comparisons
 - Can be used only if the list to be searched is sorted

Data Structures and Algorithms

Implementing Binary Search

- Consider an example.
You have to search for the name Steve in a telephone directory that is sorted alphabetically.
 - To search the name Steve by using binary search algorithm:
 - You open the telephone directory at the middle to determine which half contains the name.
 - Open that half at the middle to determine which quarter of the directory contains the name.
 - Repeat this process until the name Steve is not found.
 - Binary search reduces the number of pages to be searched by half each time.

Data Structures and Algorithms

Implementing Binary Search (Contd.)

- Consider a list of 9 elements in a sorted array.

	0	1	2	3	4	5	6	7	8
arr	9	13	17	19	25	29	39	40	47

Data Structures and Algorithms

Implementing Binary Search (Contd.)

- You have to search an element 13 in the given list.

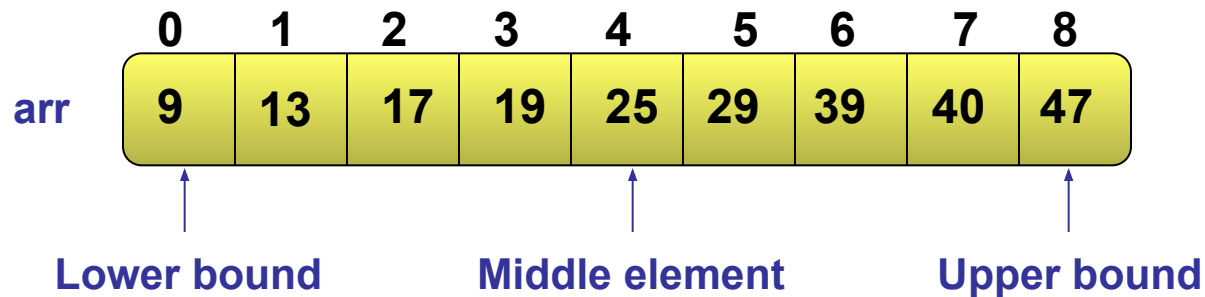
	0	1	2	3	4	5	6	7	8
arr	9	13	17	19	25	29	39	40	47

Data Structures and Algorithms

Implementing Binary Search (Contd.)

- Determine the index of the middlemost element in the list:

$$\begin{aligned}\text{Mid} &= (\text{Lower bound} + \text{Upper bound})/2 \\ &= (0 + 8)/2 \\ &= 4\end{aligned}$$

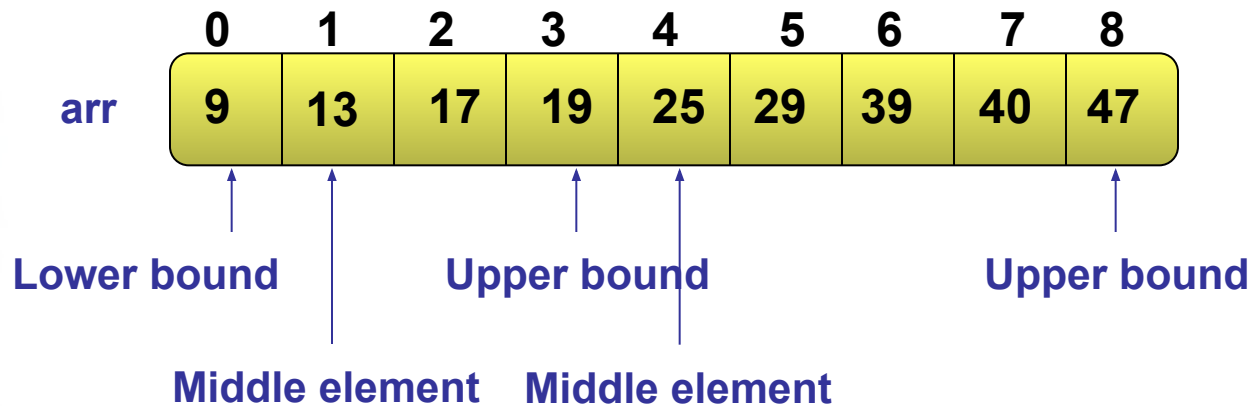


Data Structures and Algorithms

Implementing Binary Search (Contd.)

- 13 is not equal to the middle element, therefore, again divide the list into two halves:

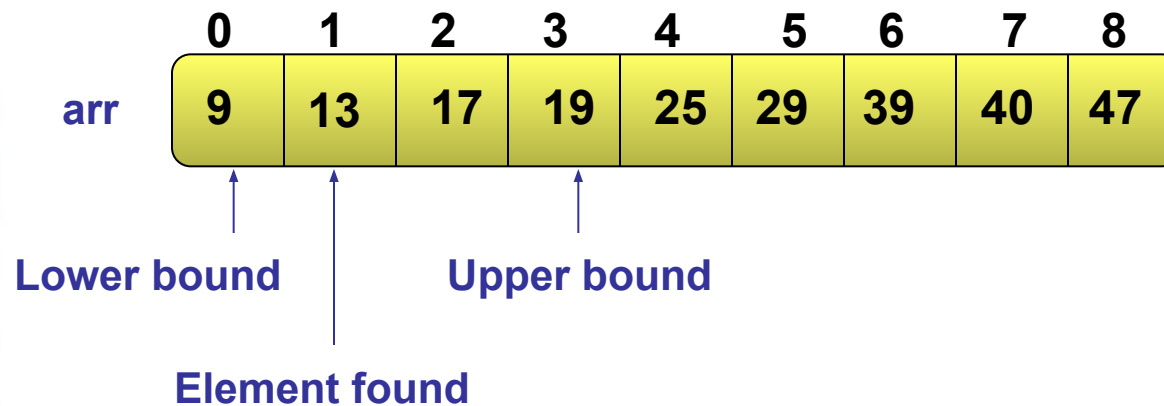
$$\begin{aligned}\text{Mid} &= (\text{Lower bound} + \text{Upper bound})/2 \\ &= (0 + 3)/2 \\ &= 1\end{aligned}$$



Data Structures and Algorithms

Implementing Binary Search (Contd.)

- 13 is equal to middle element.
- Element found at index 1.



- Write an algorithm to implement binary search algorithm.
 1. Accept the element to be searched
 2. Set lowerbound = 0
 3. Set upperbound = $n - 1$
 4. Set mid = $(\text{lowerbound} + \text{upperbound})/2$
 5. If $\text{arr}[\text{mid}] = \text{desired element}$:
 - a. Display "Found"
 - b. Go to step 10
 6. If $\text{desired element} < \text{arr}[\text{mid}]$:
 - a. Set upperbound = $\text{mid} - 1$

Data Structures and Algorithms

Implementing Binary Search (Contd.)

7. If desired element $>$ arr[mid]:
 - a. Set lowerbound = mid + 1
8. If lowerbound \leq upperbound:
 - a. Go to step 4
9. Display "Not Found"
10. Exit

Data Structures and Algorithms

Determining the Efficiency of Binary Search

- In binary search, with every step, the search area is reduced to half.
- In the best case scenario, the element to be search is found at the middlemost position of the list:
 - The number of comparisons in this case is 1.
- In the worst case scenario, the element is not found in the list:
 - After the first bisection, the search space is reduced to $n/2$ elements, where n is the number of elements in the original list.
 - After the second bisection, the search space is reduced to $n/4$ elements, that is, $n/2^2$ elements.
 - After i^{th} bisections, the number of comparisons would be $n/2^i$ elements.

Data Structures and Algorithms

Just a minute

- In _____ search algorithm, you begin at one end of the list and scan the list until the desired item is found or the end of the list is reached.
- Answer:
 - linear

Data Structures and Algorithms

Just a minute

- To implement _____ search algorithm, the list should be sorted.
- Answer:
 - binary

Data Structures and Algorithms

Activity: Performing Binary Search

- Problem Statement:
 - Write a program to search a number in an array that contains a maximum of 20 elements by using binary search. Assume that the array elements are entered in ascending order. If the number to be searched is present at more than one location in the array, the search should stop when one match is found. The program should also display the total number of comparisons made.

Data Structures and Algorithms

Summary

- In this session, you learned that:
 - Quick sort and merge sort algorithms are based on the divide and conquer technique.
 - To sort a list of items by using the quick sort algorithm, you need to:
 - Select a pivot value.
 - Partition the list into two sublists such that one sublist contains all items less than the pivot, and the second sublist contains all items greater than the pivot.
 - Place the pivot at its correct position between the two sublists.
 - Sort the two sublists by using quick sort.

Data Structures and Algorithms

Summary (Contd.)

- The total time taken by the quick sort algorithm depends on the position of the pivot value and the initial ordering of elements.
- The worst case efficiency of the quick sort algorithm is $O(n^2)$.
- The best case efficiency of the quick sort algorithm is $O(n \log n)$.
- To sort a list of items by using merge sort, you need to:
 - Divide the list into two sublists.
 - Sort each sublist by using merge sort.
 - Merge the two sorted sublists.
- The merge sort algorithm has an efficiency of $O(n \log n)$.

Data Structures and Algorithms

Summary (Contd.)

- The best case efficiency of linear search is $O(1)$ and the worst case efficiency of linear search is $O(n)$.
- To apply binary search algorithm, you should ensure that the list to be searched is sorted.
- The best case efficiency of binary search is $O(1)$ and the worst case efficiency of binary search is $O(\log n)$.