Expt No: 2 - Design a program to Implement Quick sort and Merge sort

Divide and Conquer Strategy

Using the Divide and Conquer technique, we divide a problem into subproblems. When the solution to each subproblem is ready, we 'combine' the results from the subproblems to solve the main problem.

Divide

If q is the half-way point between p and r, then we can split the subarray *A[p..r]* into two arrays *A[p..q]* and *A[q+1, r]*.

Conquer

In the conquer step, we try to sort both the subarrays *A[p..q]* and *A[q+1, r]*. If we haven't yet reached the base case, we again divide both these subarrays and try to sort them.

Combine

When the conquer step reaches the base step and we get two sorted subarrays *A[p..q]* and *A[q+1, r]* for array *A[p..r]*, we combine the results by creating a sorted array *A[p..r]* from two sorted subarrays *A[p..q]* and *A[q+1, r]*.

(i) QUICK SORT

It is one of the most efficient sorting algorithms and is based on the splitting of an array (partition) into smaller ones and swapping (exchange) based on the comparison with 'pivot' element selected. Due to this, quick sort is also called as "Partition Exchange" sort. It falls into the category of divide and conquer approach of problem-solving methodology.

Concept

Step 1 − Make the right-most index value pivot
Step 2 − partition the array using pivot value
Step 3 − quicksort left partition recursively
Step 4 − quicksort right partition recursively

Partition 1	Pivot element	Partition 2

Elements less than pivot **Elements greater than pivot**

1. A pivot element is chosen from the array. You can choose any element from the array as the pivot element. Here, I have taken the rightmost (ie. the last element) of the array as the pivot element.

2. The elements smaller than the pivot element are put on the left and the elements greater than the pivot element are put on the right.

The above arrangement is achieved by the following steps.

a. A pointer is fixed at the pivot element. The pivot element is compared with the elements beginning from the first index. If the element greater than the pivot element is reached, shift it to the rightside of pivot element.

b. Now, the pivot element is compared with the other elements. If an element smaller than the pivot element is reached no need to shift (Since I have chosen the last element as pivot)

c. The process goes on until the first iteration is completed.

d. Now the left and right subparts of this pivot element are taken for further processing in the steps below.

3. Pivot elements are again chosen for the left and the right sub-parts separately. Which means here we are dividing this complete array into sub arrays. This process is known as Partitiong. This is the backbone of Quick sort. Within these sub-parts, the pivot elements are placed at their right position. Then, step 2 is repeated.

4. The sub-parts are again divided into smaller sub-parts (partitions) until each subpart is formed of a single element.

5. Finally, the array will be sorted.

Example:

Problem Statement

Consider the following array: 8, 3, 1, 7, 0, 10, 2

We need to sort this array using quick sort.

Iteration- I

First Partition

Step 1:

8	3	1	7	0	10	**2**

Array[0] = lower bound (lb)

array[6] = upper bound (ub)

Total no: of elements =7

Choosing a[6] (*rightmost element*) as Pivot element.

So here **Pivot=2**

Step 2:

Compare first element with pivot element

8 > 2 so put it into right side of pivot element

10	3	1	7	0	**2**	**8**

Step 3:

Take next first element in the array 10 > 2 put it into right side of pivot element

0	3	1	7	**2**	**10**	8

0 < 2 it's already in the left of pivot so no change

Step 4:

compare pivot with next element a[1]

3 > 2 so put it into right side of pivot element

0	**7**	1	**2**	**3**	10	8

Step 5:

compare pivot with next element

7 > 2 so put it into right side of pivot element

0	**1**	**2**	**7**	3	10	8

Step 6:

1 < 2 it's already in the left of pivot so no change.

0	1	**2**	7	3	10	8
Less than 2		**Pivot**				Greater than 2

1st iteration is completed. Here we can see pivot divides the array into two halves in such a way that elements in the left half are smaller than pivot & elements in the right are greater than pivot.

Iteration- II

Second Partition

Again partition into two sub partition

Sub Partition 1

0	**1**

Take pivot element = 1

Step 1:

Compare with low element 0 < 1 it's already in the left of pivot so no change.

Pivot = 2 (a[3] it is fixed)

Sub Partion 2

7	3	10	**8**

Choose pivot = 8

Step 1:

Compare with low element 7 < 8 it's already in the left of pivot so no change.

Take next element 3 < 8 again no change

10 > 8 so put it into right side of pivot element

7	3	**8**	**10**
Less than 8		**Pivot**	Greater than 8

2nd iteration is completed. Here we can see pivot divides the array into two halves in such a way that elements in the left half are smaller than pivot & elements in the right are greater than pivot.

Iteration- III

Third Partition

Again partition into sub sub partitions

7	**3**

Here pivot element is 3

Compare 7 > 3 so put it into right side of pivot element

3	**7**

3rd iteration is completed.

So the final sorted array is :

0	1	2	3	7	8	10

Illustration

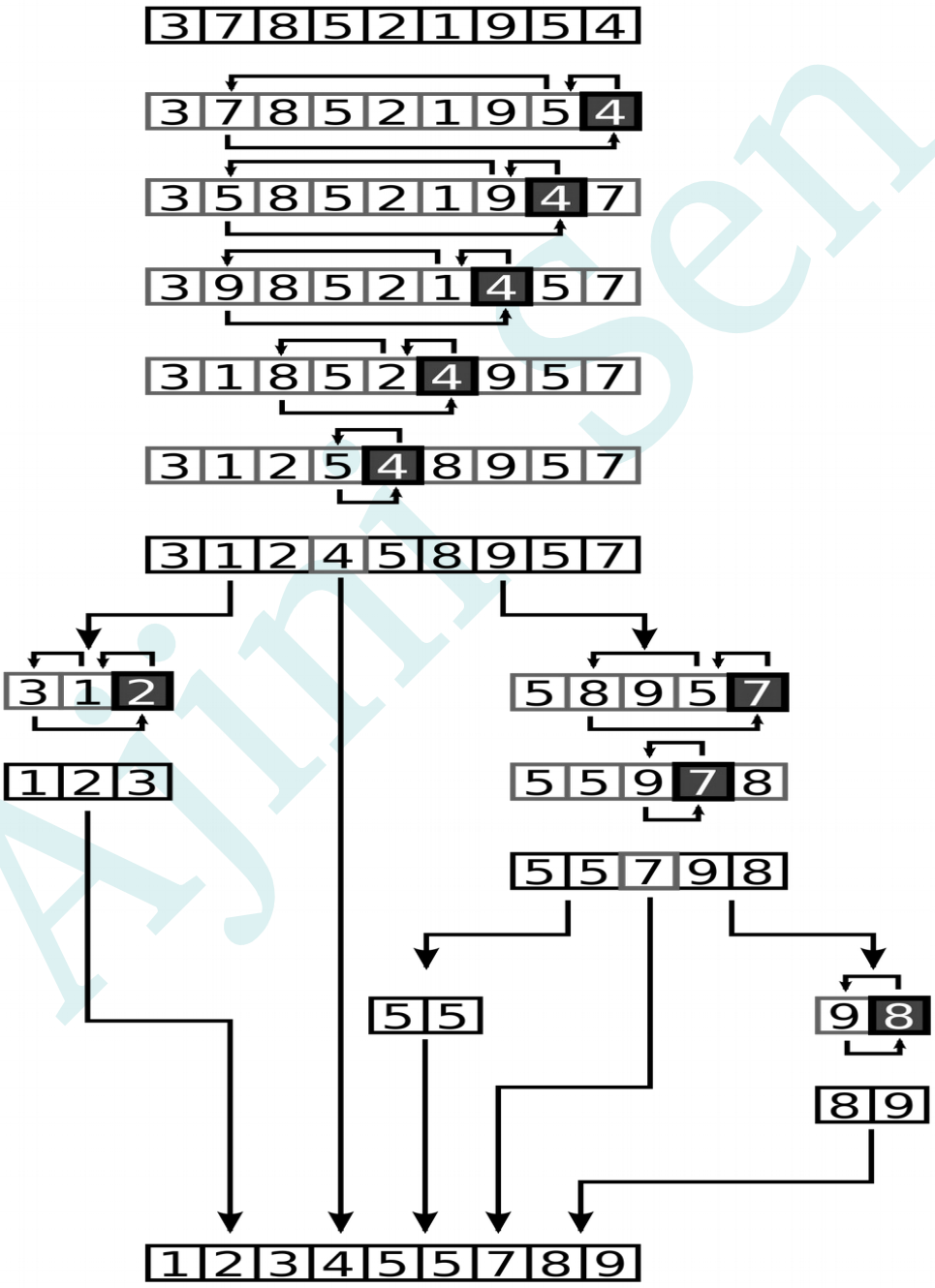


Fig (a) : Quick sort

(ii) MERGE SORT

Merge sort is one of the most efficient sorting algorithms. It is general-purpose, comparison-based sorting algorithm. It works on the principle of Divide and Conquer. Merge sort repeatedly breaks down a list into several sublists until each sublist consists of a single element and merging those sublists in a manner that results into a sorted list.

Conceptually, a merge sort works as follows:

1. Divide the unsorted list into *n* sublists, each containing one element (a list of one element is considered sorted).
2. Repeatedly merge sublists to produce new sorted sublists until there is only one sublist remaining. This will be the sorted list.

Example:

Take an unsorted array as the following :

Step 1:

First divide the whole array iteratively into two equal sub arrays unless the atomic values are achieved. For that find the mid position and then divide.

Using the formula: (lower bound + upper bound)/2

Here an array of 8 items is divided into two arrays of size 4.

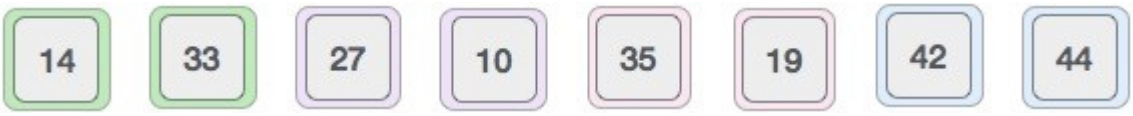
This does not change the sequence of appearance of items in the original.

Step 2:

Now we divide these arrays into halves. (Since each array has 4 elements)

Step 3:

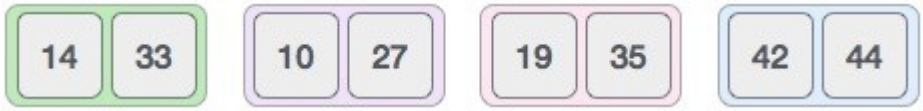
We further divide these arrays and we achieve atomic value which can no more be divided.



Step 4:

Now, we ***combine*** them in exactly the same manner as they were broken down.

We first compare the element for each list and then combine them into another list in a sorted manner. We see that 14 and 33 are in sorted positions. We compare 27 and 10 and in the list of 2 values we put 10 first, followed by 27. We change the order of 19 and 35 whereas 42 and 44 are placed sequentially.



Step 5:

In the next iteration of the combining phase, we compare lists of two data values, and merge them into a list of found data values placing all in a sorted order.

Step 6:

After the final merging, the list should look like this

10	14	19	27	33	35	42	44

Illustration

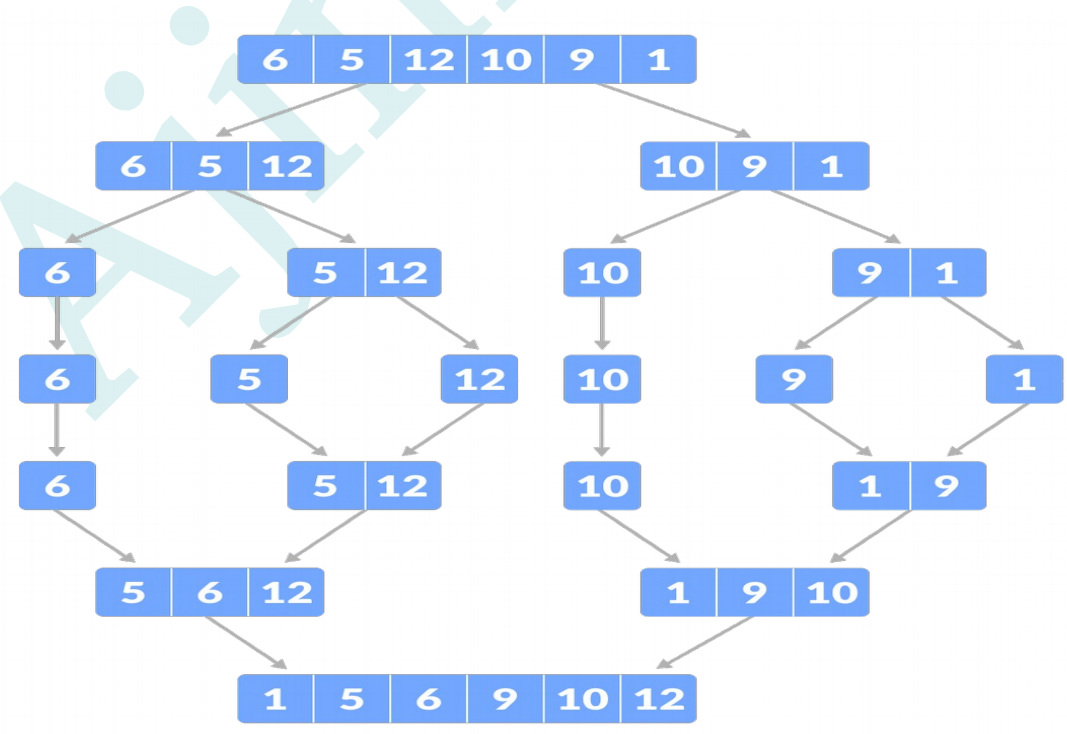


Fig (b) : Merge Sort