Data Structures Lab 8

Course: Data Structures (CS2001) **Semester:** Spring 2024
Instructor: Bushra Sattar **SLA:** M Anus

Note:

- Lab manual cover following below Advance sorting algorithms **{Quick Sort, Merge Sort}**
- Maintain discipline during the lab.
- Just raise hand if you have any problem.
- Completing all tasks of each lab is compulsory.
- Get your lab checked at the end of the session.

Quick Sort:

Quick Sort Algorithm is a Divide & Conquer algorithm. It divides input array in two partitions, calls itself for the two partitions (recursively) and performs in-place sorting while doing so. A separate partition () function is used for performing this in-place sorting at every iteration.

There are 2 Phases in the Quick Sort Algorithm.

Division Phase – Divide the array into 2 halves by finding the pivot point to perform the partition of the array.
The in-place sorting happens in this partition process itself.

Recursion Phase –
Call Quick Sort on the left partition
Call Quick Sort on the right partition.

Step by Step Process

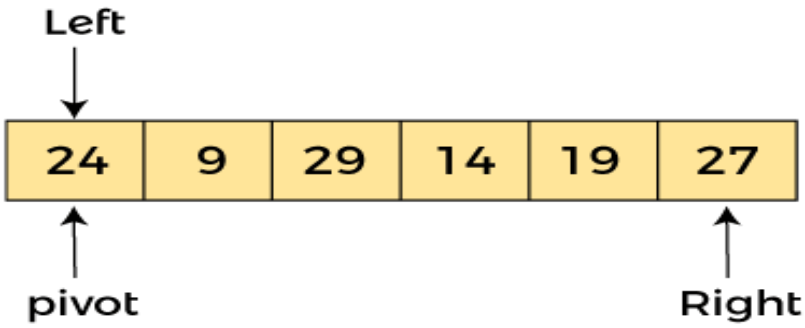
In Quick sort algorithm, partitioning of the list is performed using following steps...

Step 1 - Consider the first element of the list as pivot (i.e., Element at first position in the list).
Step 2 - Define two variables i and j. Set i and j to first and last elements of the list respectively.
Step 3 - Increment i until list[i] > pivot, then stop.
Step 4 - Decrement j until list[j] < pivot, then stop.
Step 5 - If i < j then exchange list[i] and list[j].
Step 6 - Repeat steps 3,4 & 5 until i > j.
Step 7 - Exchange the pivot element with list[j] element.

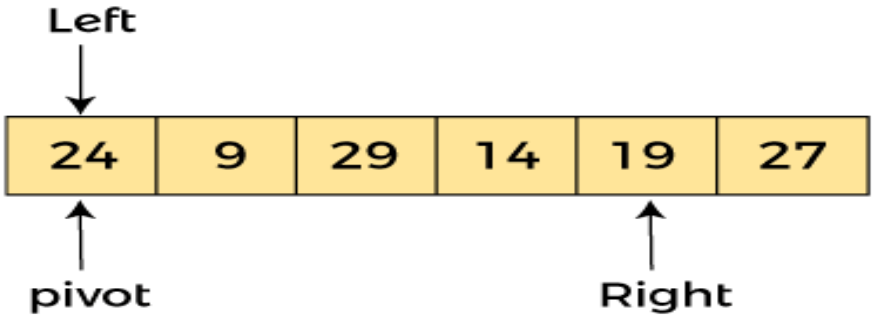
Let the elements of array are -

In the given array, we consider the leftmost element as pivot. So, in this case, a[left] = 24, a[right] = 27 and a[pivot] = 24.

Since, pivot is at left, so algorithm starts from right and move towards left.

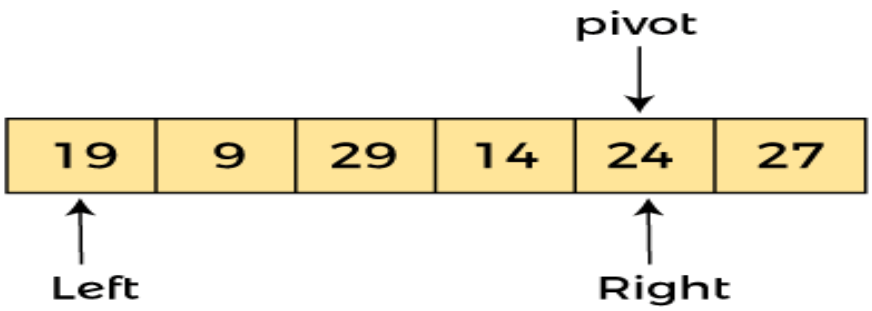


Now, a[pivot] < a[right], so algorithm moves forward one position towards left, i.e. -



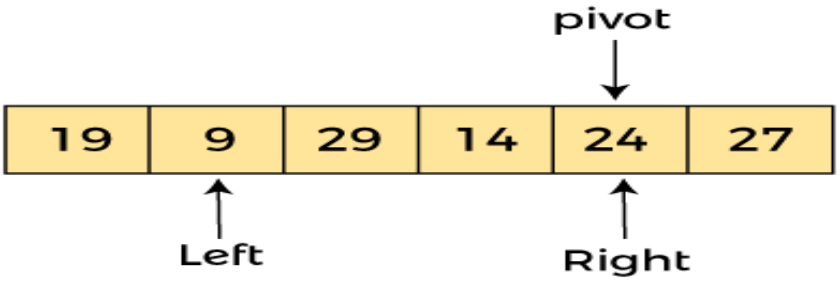
Now, a[left] = 24, a[right] = 19, and a[pivot] = 24.

Because, a[pivot] > a[right], so, algorithm will swap a[pivot] with a[right], and pivot moves to right, as -

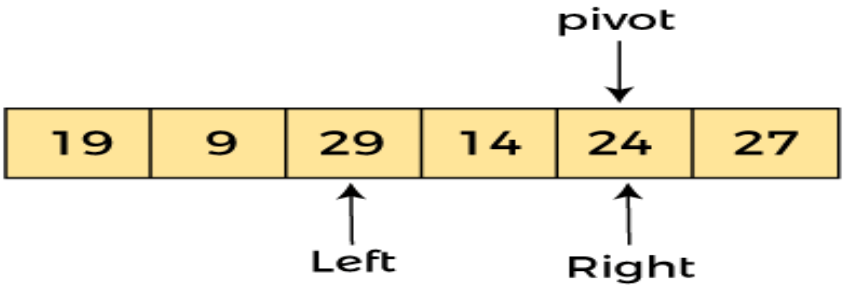


Now, a[left] = 19, a[right] = 24, and a[pivot] = 24. Since, pivot is at right, so algorithm starts from left and moves to right.

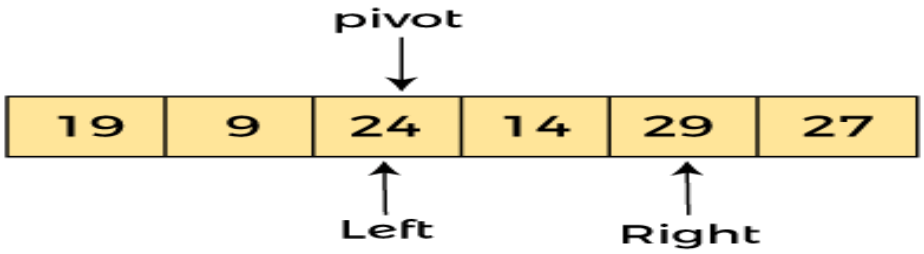
As a[pivot] > a[left], so algorithm moves one position to right as -



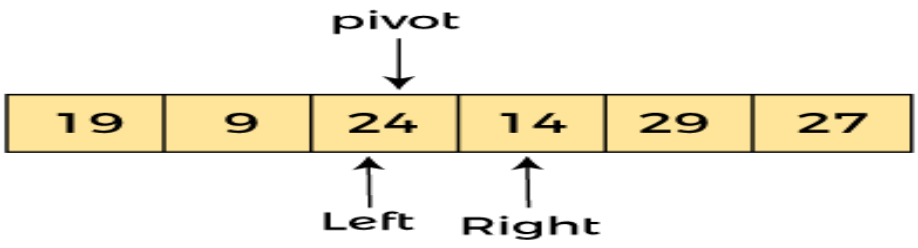
Now, a[left] = 9, a[right] = 24, and a[pivot] = 24. As a[pivot] > a[left], so algorithm moves one position to right as -



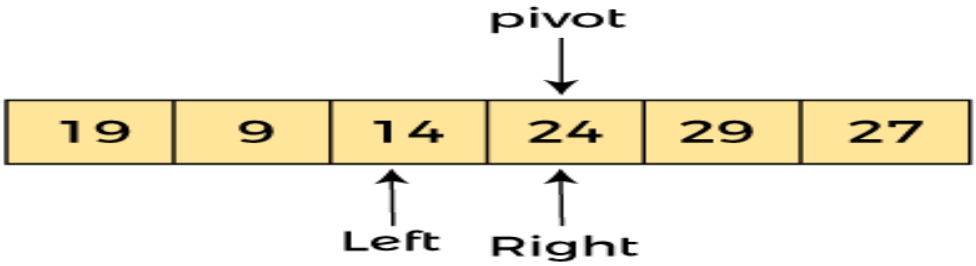
Now, a[left] = 29, a[right] = 24, and a[pivot] = 24. As a[pivot] < a[left], so, swap a[pivot] and a[left], now pivot is at left, i.e. -



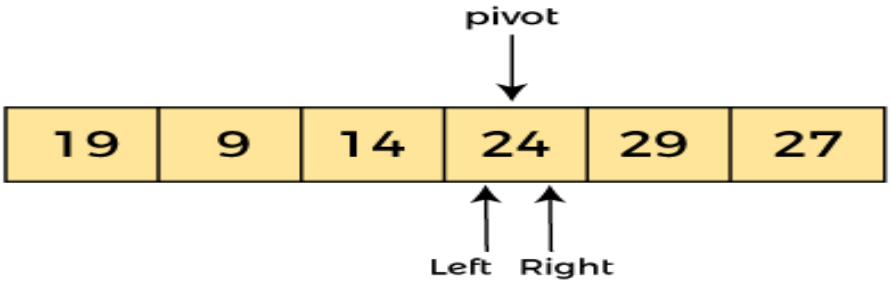
Since, pivot is at left, so algorithm starts from right, and move to left. Now, a[left] = 24, a[right] = 29, and a[pivot] = 24. As a[pivot] < a[right], so algorithm moves one position to left, as -



Now, a[pivot] = 24, a[left] = 24, and a[right] = 14. As a[pivot] > a[right], so, swap a[pivot] and a[right], now pivot is at right, i.e. -



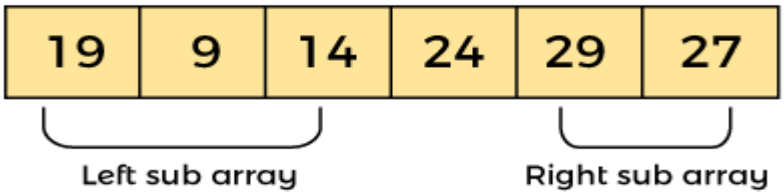
Now, a[pivot] = 24, a[left] = 14, and a[right] = 24. Pivot is at right, so the algorithm starts from left and move to right.



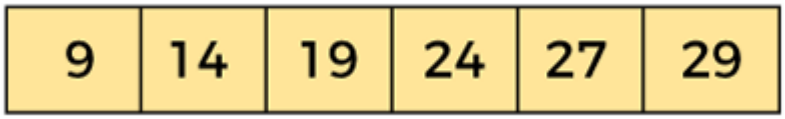
Now, a[pivot] = 24, a[left] = 24, and a[right] = 24. So, pivot, left and right are pointing the same element. It represents the termination of procedure.

Element 24, which is the pivot element is placed at its exact position.

Elements that are right side of element 24 are greater than it, and the elements that are left side of element 24 are smaller than it.

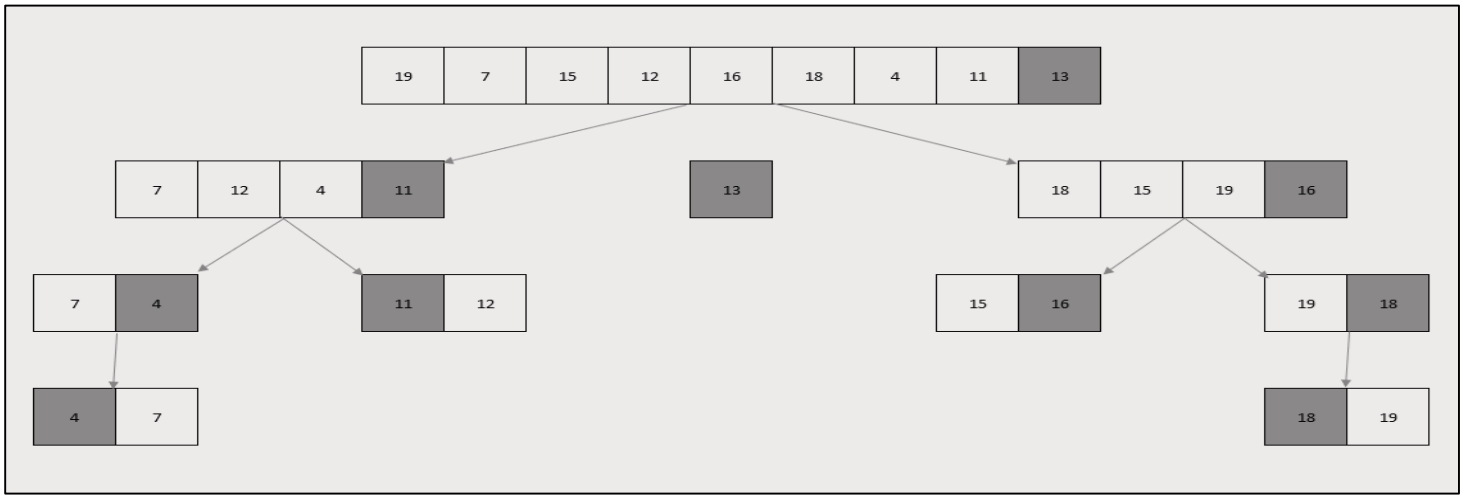


Now, in a similar manner, quick sort algorithm is separately applied to the left and right sub-arrays. After sorting gets done, the array will be -



Time Complexity

Case	Time Complexity
Best Case	O(n*logn)
Average Case	O(n*logn)
Worst Case	$O(n^2)$



Task 1:

Given the array in the above-mentioned figure implement quick sort that simply chooses the middle element as the pivot and sort accordingly.

Merge Sort
Merge sort is defined as a sorting algorithm that works by dividing an array into smaller subarrays, sorting each subarray, and then merging the sorted subarrays back together to form the final sorted array.

In simple terms, we can say that the process of merge sort is to divide the array into two halves, sort each half, and then merge the sorted halves back together. This process is repeated until the entire array is sorted.

How does Merge Sort work?
Merge sort is a recursive algorithm that continuously splits the array in half until it cannot be further divided i.e., the array has only one element left (an array with one element is always sorted). Then the sorted subarrays are merged into one sorted array.

Algorithm
In the following algorithm, arr is the given array, beg is the starting element, and end is the last element of the array.

```
MERGE_SORT(arr, beg, end)
if beg < end
set mid = (beg + end)/2
MERGE_SORT(arr, beg, mid)
MERGE_SORT(arr, mid + 1, end)
MERGE (arr, beg, mid, end)
end of if
END MERGE_SORT
```

The important part of the merge sort is the MERGE function. This function performs the merging of two sorted sub-arrays that are A[beg…mid] and A[mid+1…end], to build one sorted array A[beg…end]. So, the inputs of the MERGE function are A[], beg, mid, and end.

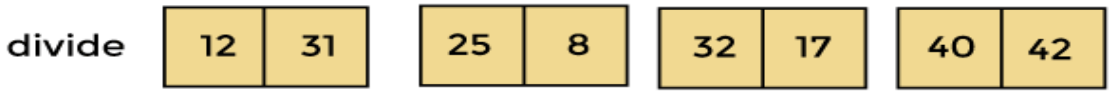
Working of Merge sort Algorithm

Let the elements of array are -

According to the merge sort, first divide the given array into two equal halves. Merge sort keeps dividing the list into equal parts until it cannot be further divided.

As there are eight elements in the given array, so it is divided into two arrays of size 4.

Now, again divide these two arrays into halves. As they are of size 4, so divide them into new arrays of size 2.

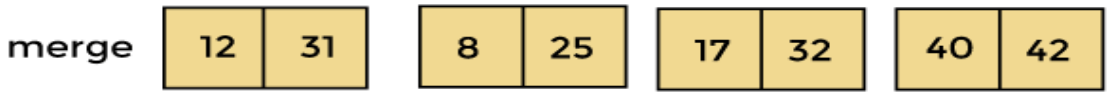


Now, again divide these arrays to get the atomic value that cannot be further divided.

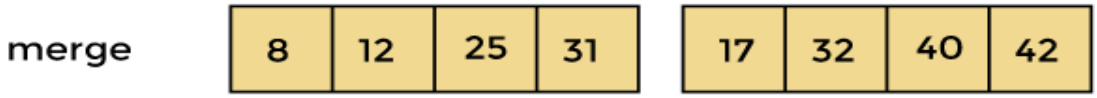
Now, combine them in the same manner they were broken.

In combining, first compare the element of each array and then combine them into another array in sorted order.

So, first compare 12 and 31, both are in sorted positions. Then compare 25 and 8, and in the list of two values, put 8 first followed by 25. Then compare 32 and 17, sort them and put 17 first followed by 32. After that, compare 40 and 42, and place them sequentially.



In the next iteration of combining, now compare the arrays with two data values and merge them into an array of found values in sorted order.



Now, there is a final merging of the arrays. After the final merging of above arrays, the array will look like -

Now, the array is completely sorted.

Merge sort complexity

Now, let's see the time complexity of merge sort in best case, average case, and in worst case. We will also see the space complexity of the merge sort.

Time Complexity

Case	Time Complexity
Best Case	O(n*logn)
Average Case	O(n*logn)
Worst Case	O(n*logn)

Lab Task

Task 2:
You are given an unsorted array of integers: [7, 2, 1, 6, 8, 5, 3, 4]. Apply the Quick Sort algorithm to sort this array in descending order.

Task 3:
Given an array of 10 integers arr[]. Sort the integers using Merge Sort and display the sorted array.

Task 4:
Given an array arr[ ] of length N consisting cost of N toys and an integer K the amount with you.
The task is to find maximum number of toys you can buy with K amount.

Note: Use quick sort to sort the array.
Input:
N = 7
K = 50
arr[] = {1, 12, 5, 111, 200, 1000, 10}
Output: 4
Explanation: The costs of the toys
you can buy are 1, 12, 5 and 10.

Task 5:
Given a singly linked list, implement Merge Sort algorithm to sort the linked list.

Task 6:
Given k sorted arrays of different length, merge them into a single array such that the merged array is also sorted.
Examples:

Input: {{3, 13},
{8, 10, 11}
{9, 15}}
Output: {3, 8, 9, 10, 11, 13, 15}

Input: {{1, 5},
{2, 3, 4}}
Output: {1, 2, 3, 4, 5}