Faculty of Computer and Artificial Intelligence
Sadat University

Analysis and Design of Algorithms (CS 302)

Lecture :5

“Divide and Conquer”

Dr.Sara A Shehab

Divide-and-Conquer

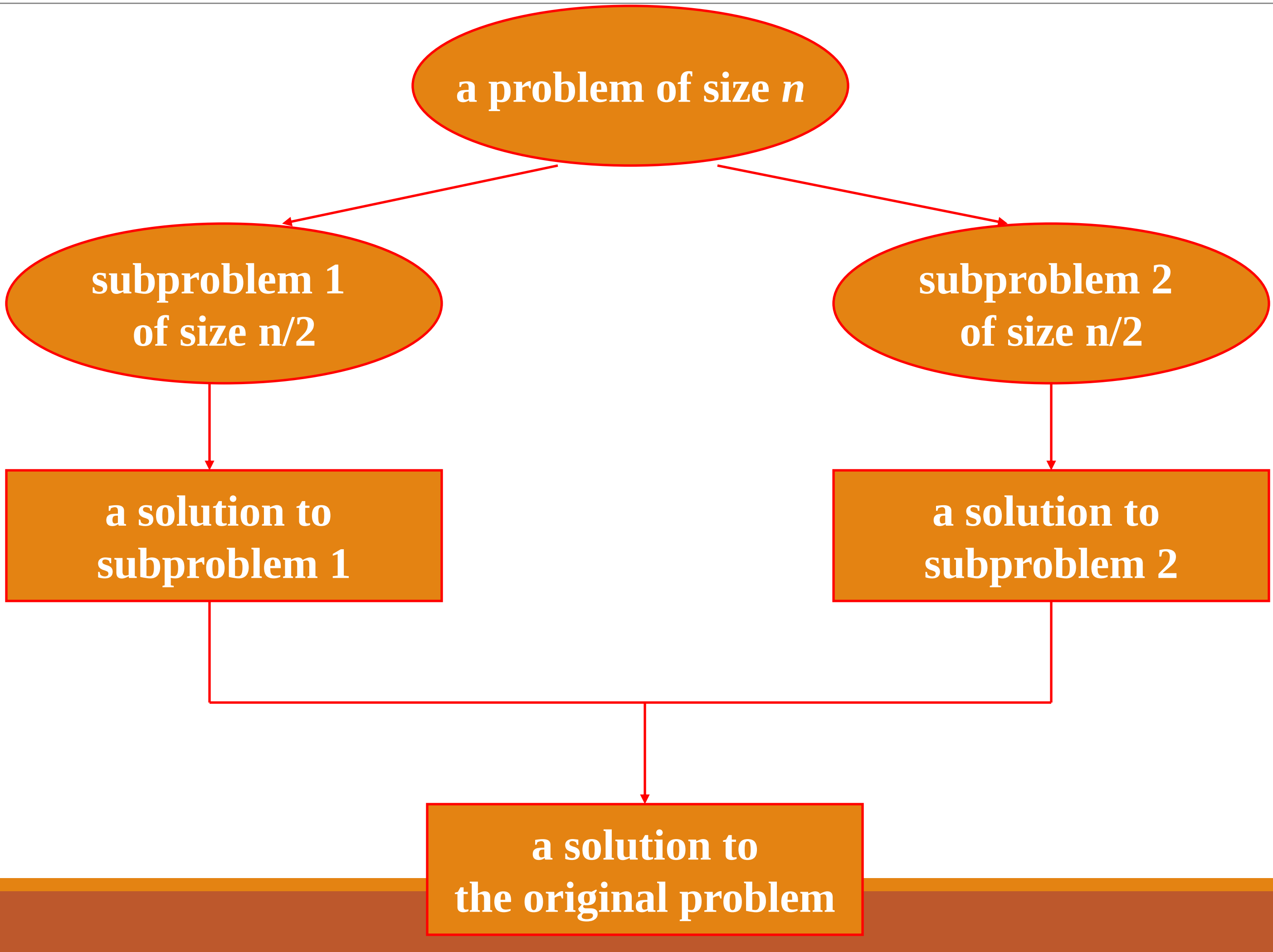
The most-well known algorithm design strategy:

1. Divide instance of problem into two or more smaller instances

2. Solve smaller instances recursively

3. Obtain solution to original (larger) instance by combining these solutions

Divide-and-Conquer Technique (cont.)



Divide-and-Conquer Examples

Sorting:

1. mergesort
2. quicksort

Mergesort

Split array A[0..*n*-1] in two about equal halves and make copies of each half in arrays B and C

Sort arrays B and C recursively

Merge sorted arrays B and C into array A as follows:

- Repeat the following until no elements remain in one of the arrays:
  - compare the first elements in the remaining unprocessed portions of the arrays
  - copy the smaller of the two into A, while incrementing the index indicating the unprocessed portion of that array
- Once all elements in one of the arrays are processed, copy the remaining unprocessed elements from the other array into A.

Pseudocode of Merge sort

ALGORITHM *Mergesort*$(A[0..n-1])$

//Sorts array $A[0..n-1]$ by recursive mergesort
//Input: An array $A[0..n-1]$ of orderable elements
//Output: Array $A[0..n-1]$ sorted in nondecreasing order
if $n > 1$
copy $A[0..\lfloor n/2 \rfloor - 1]$ to $B[0..\lfloor n/2 \rfloor - 1]$
copy $A[\lfloor n/2 \rfloor..n-1]$ to $C[0..\lceil n/2 \rceil - 1]$
Mergesort$(B[0..\lfloor n/2 \rfloor - 1])$
Mergesort$(C[0..\lceil n/2 \rceil - 1])$
Merge(B, C, A)

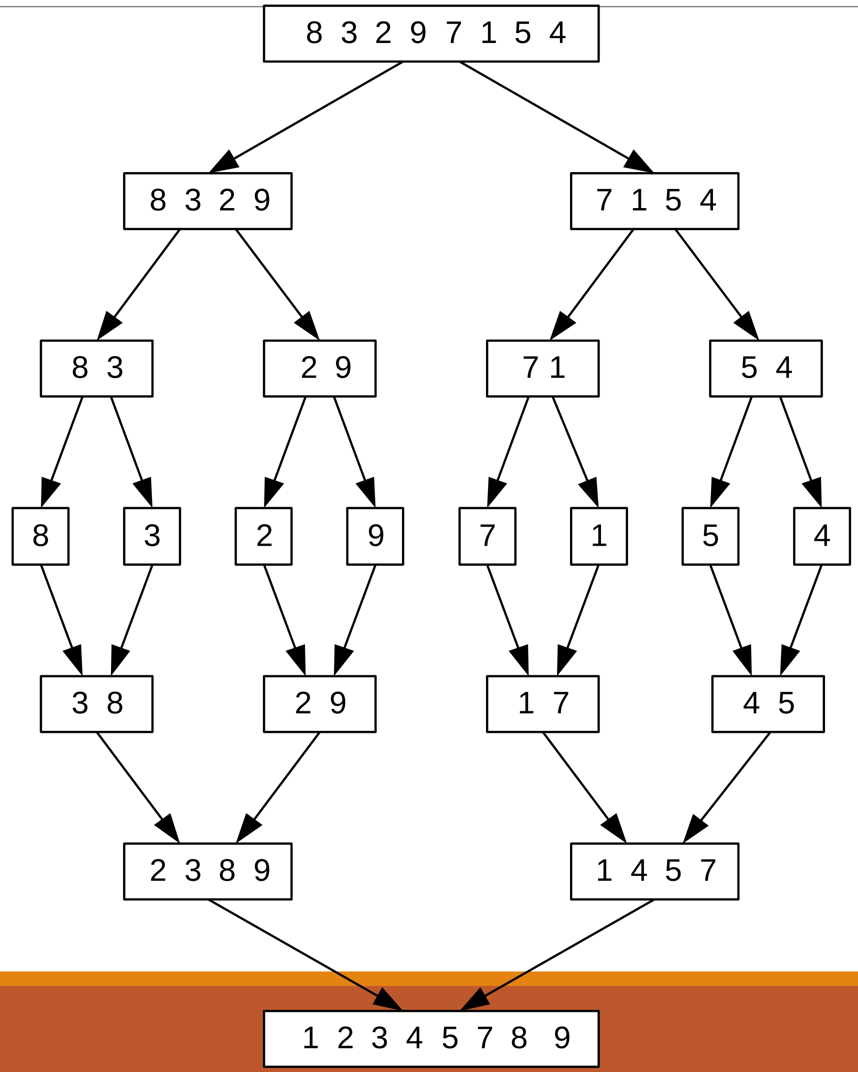
Pseudocode of Merge

ALGORITHM $Merge(B[0..p-1], C[0..q-1], A[0..p+q-1])$

//Merges two sorted arrays into one sorted array
//Input: Arrays $B[0..p-1]$ and $C[0..q-1]$ both sorted
//Output: Sorted array $A[0..p+q-1]$ of the elements of B and C

$i \leftarrow 0;\ j \leftarrow 0;\ k \leftarrow 0$
while $i < p$ **and** $j < q$ **do**
    if $B[i] \leq C[j]$
        $A[k] \leftarrow B[i];\ i \leftarrow i + 1$
    else $A[k] \leftarrow C[j];\ j \leftarrow j + 1$
    $k \leftarrow k + 1$
if $i = p$
    copy $C[j..q-1]$ to $A[k..p+q-1]$
else copy $B[i..p-1]$ to $A[k..p+q-1]$

Mergesort Example



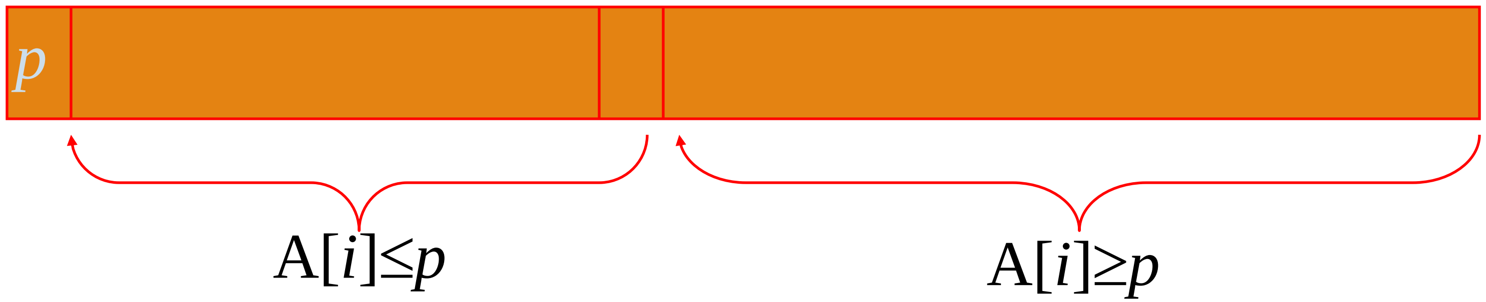
Analysis of Mergesort

All cases have same efficiency: $\Theta(n \log n)$

Quicksort

Select a *pivot* (partitioning element) – here, the first element

Rearrange the list so that all the elements in the first *s* positions are smaller than or equal to the pivot and all the elements in the remaining *n-s* positions are larger than or equal to the pivot (see next slide for an algorithm)



Exchange the pivot with the last element in the first (i.e., $\leq$) subarray — the pivot is now in its final position

Sort the two subarrays recursively

```
Algorithm Partition(A[l..r])
//Partitions a subarray by using its first element as a pivot
//Input: A subarray A[l..r] of A[0..n − 1], defined by its left and right
//        indices l and r (l < r)
//Output: A partition of A[l..r], with the split position returned as
//        this function's value
p ← A[l]
i ← l;   j ← r + 1
repeat
    repeat i ← i + 1 until A[i] ≥ p
    repeat j ← j − 1 until A[j] · p
    swap(A[i], A[j])
until i ≥ j
swap(A[i], A[j])   //undo last swap when i ≥ j
swap(A[l], A[j])
return j
```

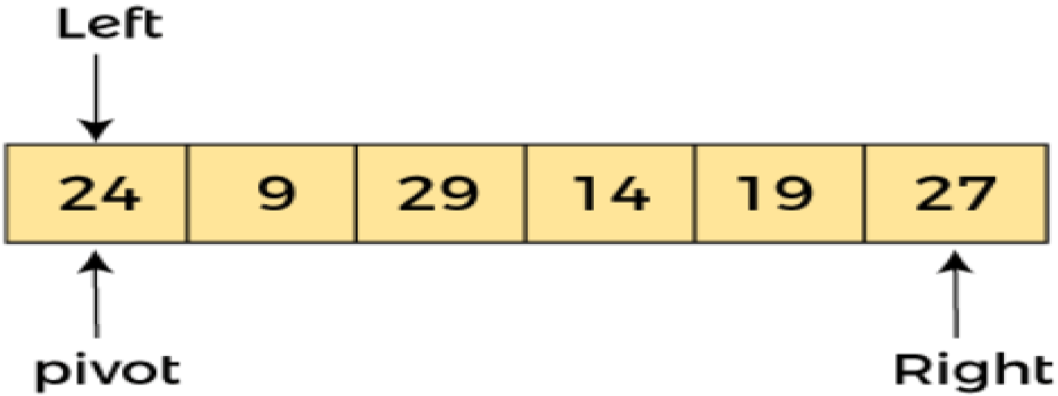
Quick Sort Example

- To understand the working of quick sort, let's take an unsorted array.
- It will make the concept more clear and understandable.
- Let the elements of array are -

24	9	29	14	19	27

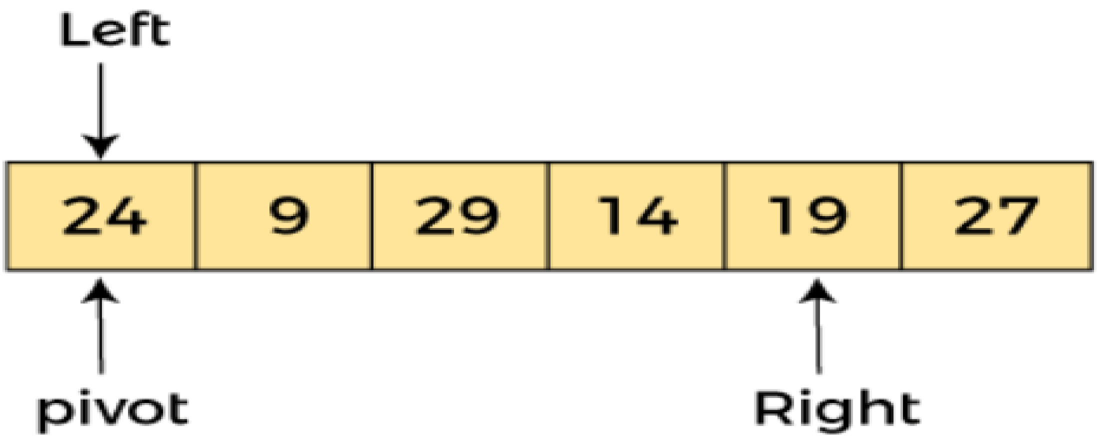
- In the given array, we consider the leftmost element as pivot. So, in this case, a[left] = 24, a[right] = 27 and a[pivot] = 24.
- Since, pivot is at left, so algorithm starts from right and move towards left.

Quick Sort Example



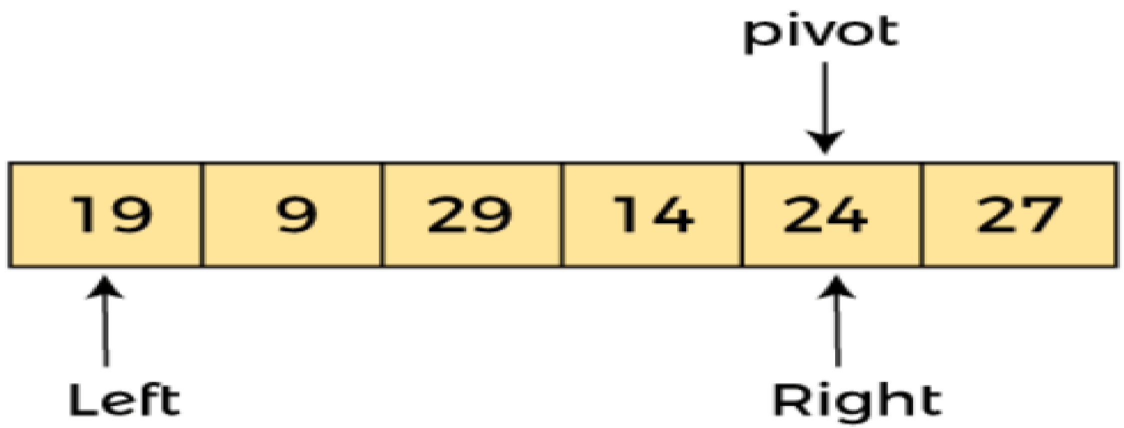
- Now, a[pivot] < a[right]
- So algorithm moves forward one position towards left, i.e. -

Quick Sort Example



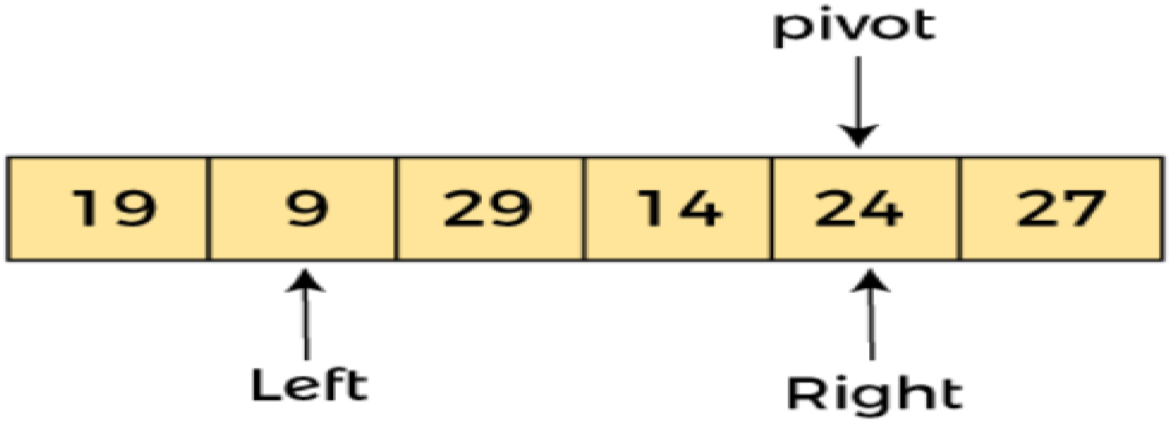
- Now, a[left] = 24, a[right] = 19, and a[pivot] = 24.
- Because, a[pivot] > a[right],
- so, algorithm will swap a[pivot] with a[right], and pivot moves to right, as -

Quick Sort Example



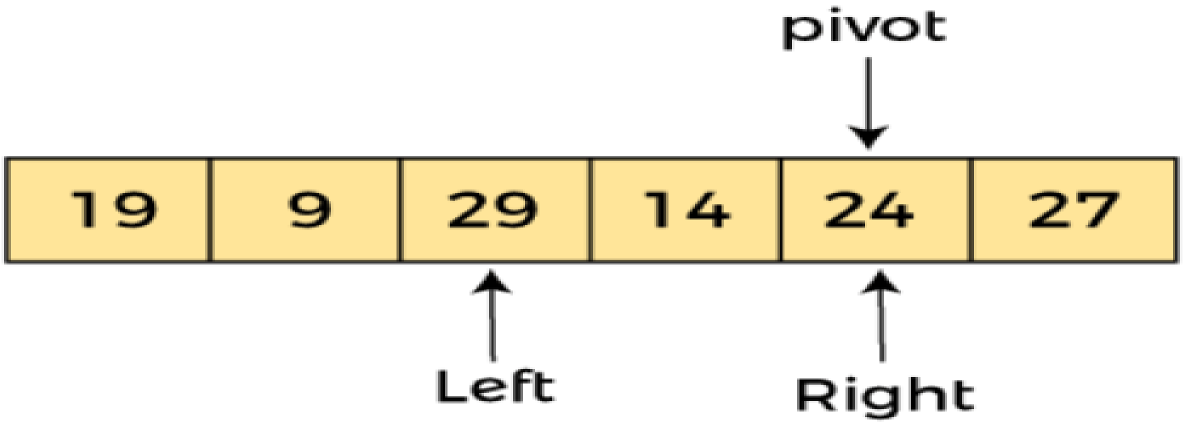
- Now, a[left] = 19, a[right] = 24, and a[pivot] = 24.
- Since, pivot is at right, so algorithm starts from left and moves to right.
- As a[pivot] > a[left], so algorithm moves one position to right as -

Quick Sort Example



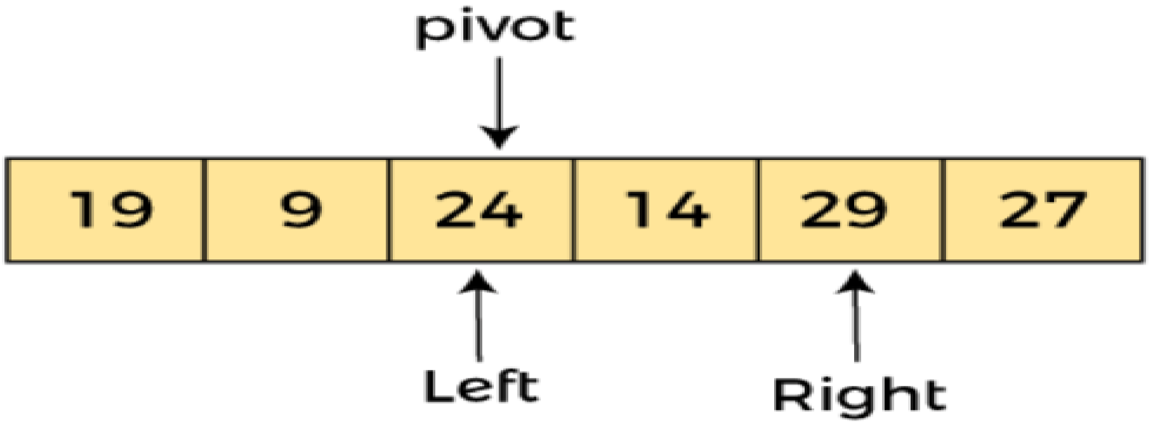
- Now, a[left] = 9, a[right] = 24, and a[pivot] = 24.
- As a[pivot] > a[left],
- so algorithm moves one position to right as -

Quick Sort Example



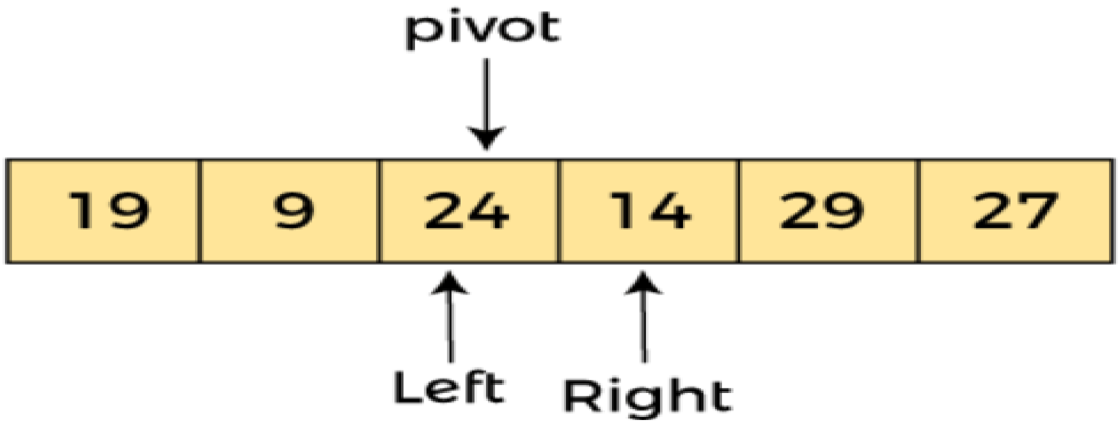
- Now, a[left] = 29, a[right] = 24, and a[pivot] = 24.
- As a[pivot] < a[left],
- so, swap a[pivot] and a[left], now pivot is at left, i.e. -

Quick Sort Example



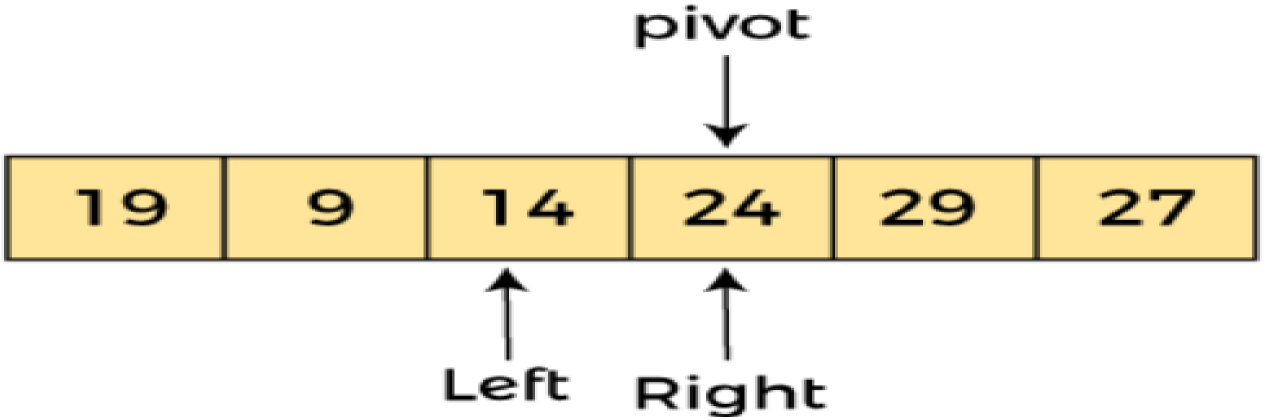
- Since, pivot is at left,
- so algorithm starts from right, and move to left.
- Now, a[left] = 24, a[right] = 29, and a[pivot] = 24.
- As a[pivot] < a[right],
- so algorithm moves one position to left, as -

Quick Sort Example



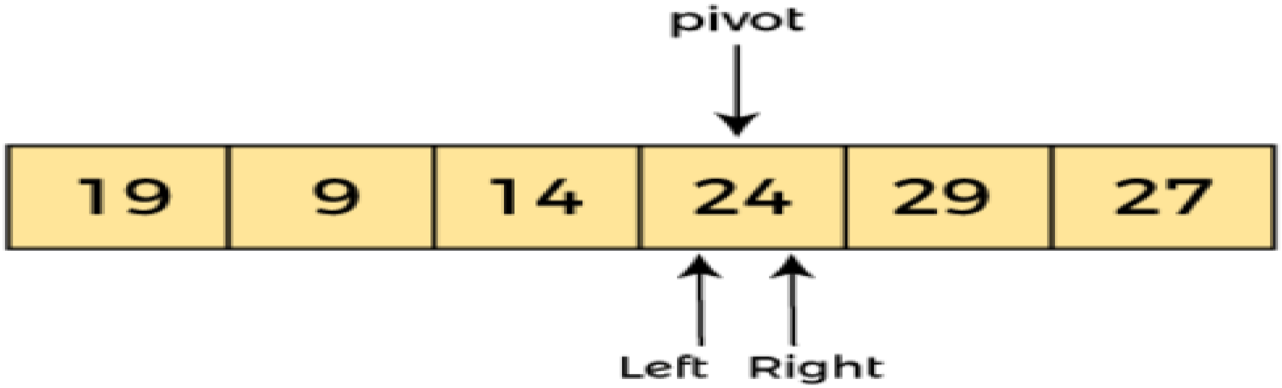
- Now, a[pivot] = 24, a[left] = 24, and a[right] = 14.
- As a[pivot] > a[right],
- so, swap a[pivot] and a[right], now pivot is at right, i.e. -

Quick Sort Example



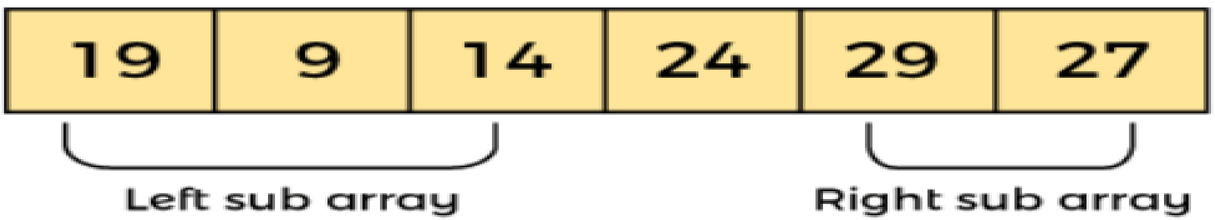
- Now, a[pivot] = 24, a[left] = 14, and a[right] = 24.
- Pivot is at right,
- so the algorithm starts from left and move to right.

Quick Sort Example

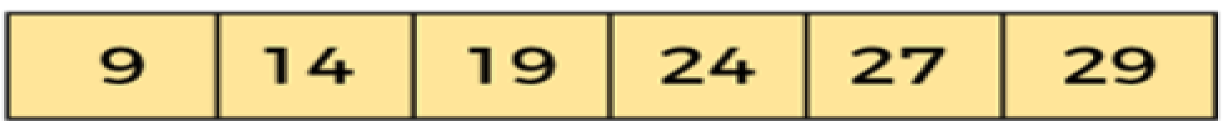


- Now, a[pivot] = 24, a[left] = 24, and a[right] = 24.
- So, pivot, left and right are pointing the same element.
- It represents the termination of procedure.
- Element 24, which is the pivot element is placed at its exact position.

Quick Sort Example



- 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 -



Quicksort Example

5 3 1 9 8 2 4 7

Analysis of Quicksort

Best case: split in the middle — $\Theta(n \log n)$

Worst case: sorted array! — $\Theta(n^2)$

Average case: random arrays — $\Theta(n \log n)$


The End