Unit-1

FaceLook D D M M Y Y Y Y

Divide and conquer

ue- Describe algorithm with Example and Explain algorithm analysis

⇒

Algorithm :-

- Algorithm is well defined computationaly Procedure that takes some value, or a set of values as input and Produce set of output it is sequence of computational steps.
- An algorithm is a well-defined sequential computational technique that accepts a value or a collection of values as input and produces the output(s) needed to solve a problem.
- Every Algorithm must statisfy following Criteria :-

1) input :- It is must be supplied externally.
2) output :- Their will be atleast one output
3) Definiteness :- Algorithm must be clear and an ambiguous.
4) Finiteness :- Algorithm must be terminate after finite numbers of steps.
5) Effectiveness :- Algorithm must be sufficiently Basic.

Need of the Algorithms :-

- Algorithms are used to solve problems or automate tasks in a systematic and efficient manner.
- They are a set of instructions or rules that guide the computer or software in performing a particular task or solving a problem.

There are several reasons why we use algorithm:-

1) Efficiency
2) consistency
3) scalability
4) Automation
5) standardization.

- Algorithm Design Approach:-
  1) Top-down Approach
  2) Bottom-up Approach

Example:-
merge sort:-
-A divide-and-conquer sorting algorithm that works by dividing the unsorted list into n sub-lists, sorting each sub-list and then merging them back into a single sorted list.

- Algorithm analysis:-
-Algorithm analysis is the process of determining the computational complexity of algorithms, that is the amount of resources needed for running an algorithm.
-This includes analyzing the time complexity, space complexity and some cases, other resources like network usage or power consumption.

Complexities:-

1) Time complexity:-

- The amount of time an algorithm takes to complete as a function of the length of the input.
- Big O Notation (O):- Describes the upper bound of the time complexity, focusing on the worst-case scenario.

2) Space complexity:-
- The amount of memory an algorithm uses in relation to the input size.
- Big O Notation (O):- Describes the upper bound of the space complexity, focusing on the worst-case scenario.

Que- Difference between Linear search and Binary search.

⇒

Linear search	Binary search
1) In Linear search input data need not to be in sorted	1) In binary search input data need to be in sorted order.
2) It is also called a sequential search.	2) It is also called a half-interval search
3) The time complexity of linear search O(n)	3) The time complexity of Binary search O(logn)
4) Multidimensional array can be used.	4) only single dimensional array is used.

5] Linear search performs equality comparisons.	5] Binary search performs ordering comparisons
6] It is less complex	6] It is more complex
7] It is very slow process	7] It is very fast process.
8] Algorithm Type is sequential	8] Algorithm Type is divide and conquer.
9] Less efficient for large lists	10] More efficient for large lists.

Que- Explain Binary search with example.

⇒

Binary search :-

- Binary search is the search technique that works efficiently on sorted lists.
- Hence, to search an element into some list using the binary search technique, we must ensure that the list is sorted.
- Binary search follows the divide and conquer approach in which the list is divided into two halves, and the item is compared with the middle element of the list.
- If the match is found then, the location of the middle element is returned.
- otherwise, we search into either of the halves depending upon the result produced through the match.

- There are two methods to implement the binary search algorithm:-
  1) Iterative method
  2) Recursive method
- The Recursive method of binary search follows the divide and conquer approach.

- Binary search Algorithm:-

1) Divide the search space into two halves by finding the middle index "mid".
2) Compare the middle element of the search space with the key.
3) If the key is found at middle element, the process is terminated.
4) If the key is not found at middle element, choose which half will be used as the next search space.
   i) If the key is smaller than the middle element, then the left side is used for next search.
   ii) If the key is larger than the middle element, then the right side is used for next search.
5) This process is continued until the key is found or the total search space is exhausted.

Example:-

0	1	2	3	4	5	6	7	8
10	12	24	29	39	40	51	56	69

key (k) = 56

Time complexity = O(log n)

mid = (beg + end)/2

mid = (0+8)/2 = 4

0	1	2	3	4	5	6	7	8
10	12	24	29	39	40	51	56	69

A[mid] = 39

A[mid] < k (39 < 56)

so, beg = mid + 1 = 5, end = 8

mid = (5+8)/2 = 13/2 = 6

0	1	2	3	4	5	6	7	8
10	12	24	29	39	40	51	56	69

A[mid] = 51

A[mid] < k (51 < 56)

so, beg = mid + 1 = 7, end = 8

mid = (7+8)/2 = 15/2 = 7

0	1	2	3	4	5	6	7	8
10	12	24	29	39	40	51	56	69

A[mid] = 56

A[mid] = k (56 = 56)

So, location = mid

Element found at 7th location of the array.

- Advantages :-

1) Binary search is faster than linear search, especially for large arrays.
2) More efficient than other searching algorithms with a similar time complexity, such as interpolation search or exponential search.
3) Efficiency.

4) memory usage

• Disadvantages :-
1) Requires sorted Data
2) Not suitable for small Datasets
3) Complexity in Recursive implementation
4) No Guarantee of constant Time for Access.

• Applications :-
1) Searching in Databases
2) Finding Elements in sorted arrays
3) Dictionary Lookups
4) Intersection of Intervals
5) Finding Boundaries.

Que- Explain Finding the maximum and minimum number algorithm with example.

⇒

Max-Min Problem :-
-The max-min problem in algorithm analysis is finding the maximum and minimum value in an array.

Solution :-
-To find the maximum and minimum numbers in a given array numbers[] of size n, the following algorithm can be used.
• divide and conquer approach.

• Divide and Conquer Approach :-
-In this approach, the array is divided into

two halves.
- Then using recursive approach maximum and minimum numbers in each halves are found.
- Later, return the maximum of two maxima of each half and the minimum of two minima of each half.

• Algorithm :-

```
max-min (x, y)
 if y - x ≤ 1 then
    return (max (numbers[x], numbers[y]),
            min ((numbers[x], numbers[y]))
 else
    (max1, min1) := maxmin (x, ⌊((x+y)/2)⌋)
    (max2, min2) := maxmin (⌊((x+y)/2)+1)⌋, y)
return
    (max(max1, max2), min(min1, min2))
```

Example :-

1	2	3	4	5	6	7	8
9	6	4	7	10	14	8	11

mid = $\frac{1+8}{2} = 4$

1	2	3	4	5	6	7	8
9	6	4	7	10	14	8	11

mid = $\frac{1+4}{2} = 2$ mid = $\frac{5+8}{2} = 6$

9	6	4	7	10	14	8	11

max = 9 max = 7 max = 14 max = 11
min = 6 min = 4 min = 10 min = 8

max = 9
min = 4

max = 14
min = 8

max = 14
min = 4

- ~~Advantages~~

- Advantages :-
1) Simplicity
2) Efficiency
3) Memory Efficiency
4) Data validation

- Disadvantages :-
1) Limited information
2) No Data sorting
3) Not suitable for Dynamic Data.

- Applications :-

1) Data Analysis and statistics
2) Optimization Problem
3) Financial analysis
4) supply chain management.

Que- Explain Merge sort with example.

⇒

Merge sort :-
- Merge sort is a sorting algorithm that follows the divide-and-conquer approach.
- It works by recursively dividing the input array into smaller subarrays and sorting those subarrays then merging them back together to obtain the 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.

• Merge sort Working :-
- merge sort is a popular sorting algorithm known for its efficiency and stability.
- It follows the divide-and-conquer approach to sort a given array of elements.

Steps :-
1) Divide :- Divide the list or array recursively into two halves until it can no more be divided.
2) Conquer :- Each subarray is sorted individually using the merge sort algorithm.
3) merge :- The sorted subarrays are merged back together in sorted order. The process continues until all element from both subarrays have been merged.

• Algorithm :-

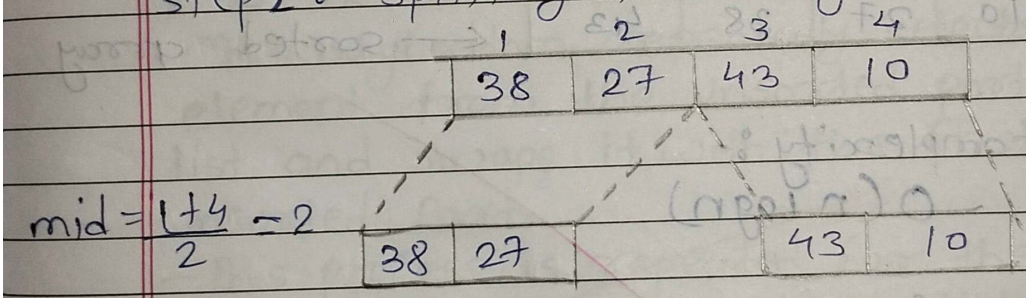
```
Mergesort (l, h)
 if (l < h)
 {
   mid = (l+h)/2

   Mergesort (l, mid)
   Mergesort (mid+1, h)
    merge (l, mid, h)
 }
```

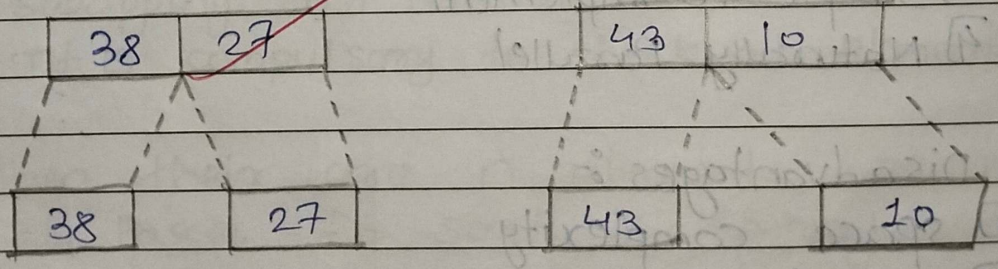
• Example :-

Let's sort the array or list [38, 27, 43, 10] using Merge sort.

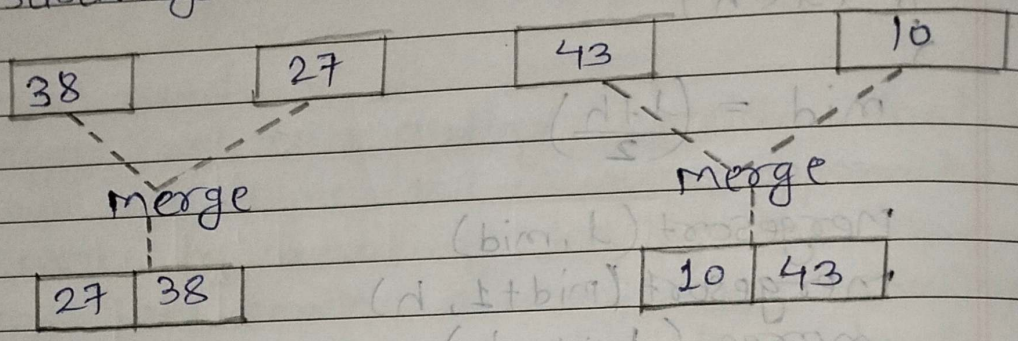
Step 1 :- Splitting the Array into two equal halves



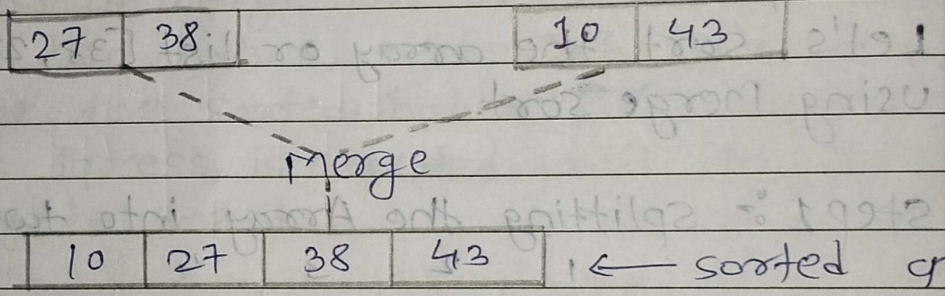
Step 2 :- Splitting the subarrays into two halves



step3 :- merging unit length cells into sorted subarrays



step4 :- merging sorted subarrays into the sorted array.



- Time complexity :-
  $O(n \log n)$

- Advantages :-
1] stability
2] Guaranteed worst-case Performance :- $O(n \log n)$
3] simple to implement
4] Naturally Parallel

- Disadvantages :-
1] space complexity
2] Not in-place
3] slower than quick sort in general.

• Application :-
1) Sorting large datasets
2) External sorting
3) Inversion counting
4) It is a preferred algorithm for sorting Linked List.

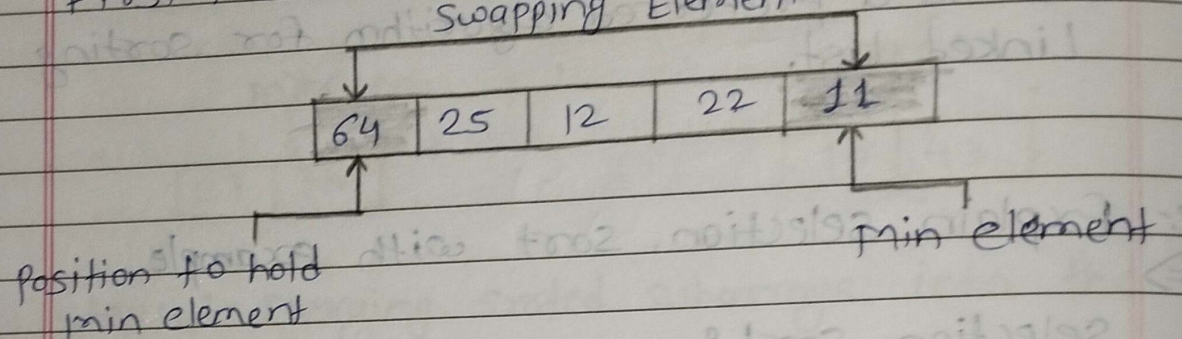
Que- Explain selection sort with example.
⇒

Selection sort :-
- Selection sort is a simple and efficient sorting algorithm that works by repeatedly selecting the smallest (or largest) element from the unsorted portion of the list and moving it to the sorted portion of the list.
- The algorithm repeatedly selects the smallest element from the unsorted portion of the list and swaps it with the first element of the unsorted part.
- This process is repeated for the remaining unsorted portion until the entire list is sorted.
- Selection sort is generally used when -
  - A small array is to be sorted.
  - Swapping cost doesn't matter
  - It is compulsory to check all elements.

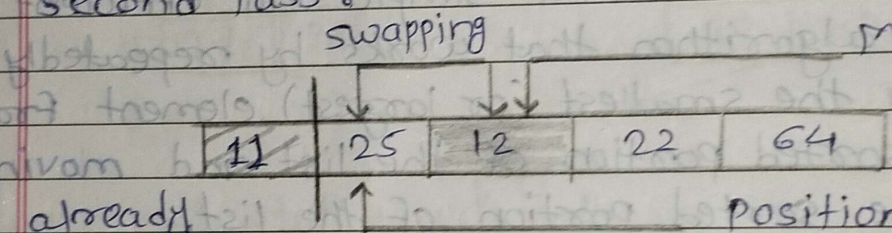
- When their are n elements we need n-1 Passes to sort the array.

• Example :-
Let consider arr[] = {64, 25, 12, 22, 11}

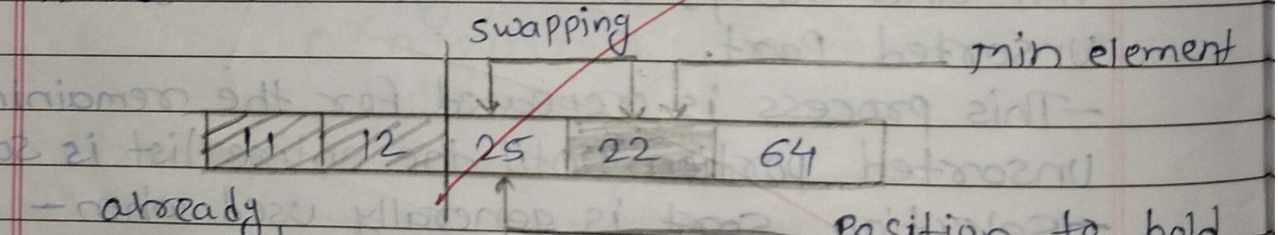
First Pass :-



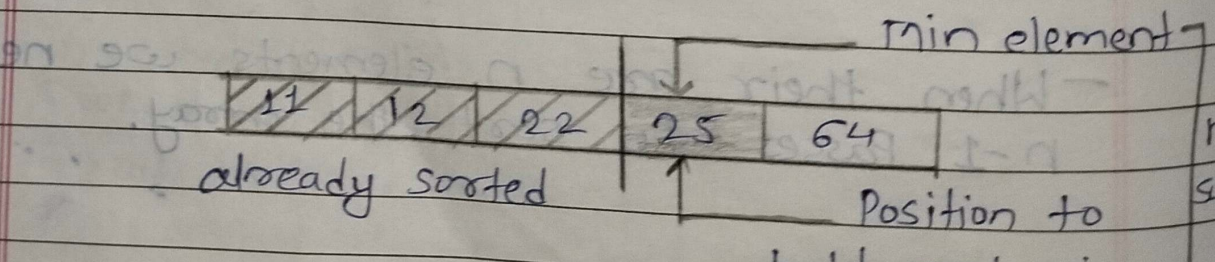
second Pass :-



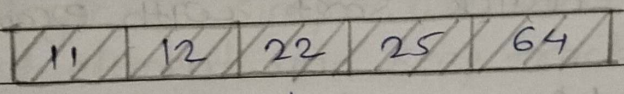
Third Pass :-



Fourth Pass :-



Fifth Pass :-



sorted array

- Advantages :-
1) Simple and easy to understand
2) Works well with small datasets.

- Disadvantages :-
1) Selection Sort has a time complexity of $O(n^2)$ in the worst and average case.
2) Does not work well on large datasets.
3) Does not preserve the relative order of items with equals keys which means it is not stable.

- Complexity :-
Time complexity :- $O(N^2)$

- Application :-
1) Small Data sets
2) Partially sorted Data
3) Choosing smallest / Largest Element
4) Manual sorting and ordering.

que- Explain Quick Sort with example.

⇒

Quick Sort :-

- Quick sort is a sorting algorithm based on the Divide and conquer algorithm that picks an element as a Pivot and partitions the given array around the picked Pivot by placing the Pivot in its correct position in the sorted array.

Quick Sort Working :-

- The key process in quick sort is a Partition().
- The target of partitions is to place the Pivot (any element can be chosen to be a pivot) at its correct position in the sorted array and Put all smaller elements to the left of the Pivot, and all greater elements to the right of the Pivot.
- Partition is done recursively on each side of the Pivot after the Pivot is placed in its correct position and this finally sorts the array.

• Choice of Pivot :-

- There are many different choices for picking Pivots :-

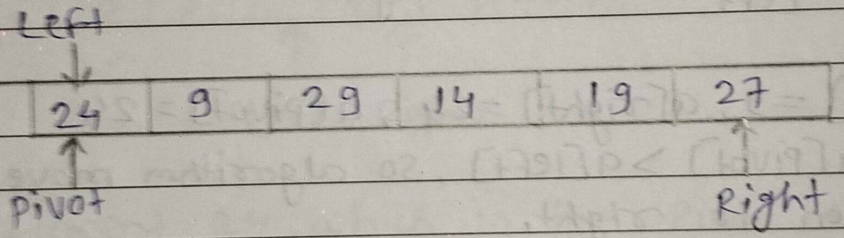
1) Always Pick the first element as a pivot.
2) Always Pick the last element as a Pivot
3) Pick a random element as a Pivot.
4) Pick the middle as the pivot.

• Partition Algorithm :-
- The logic is simple, we start from the leftmost element and keep track of the index of smaller (or equal) elements as i.
- While traversing, if we find a smaller element, we swap the current element with arr[i].
- otherwise, we ignore the current element.

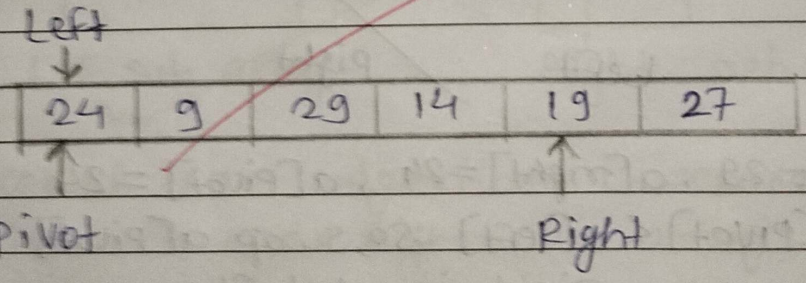
• Example :-
Let the elements of array are -

24	9	29	14	19	27

a[Left] = 24, a[right] = 27, a[Pivot] = 24

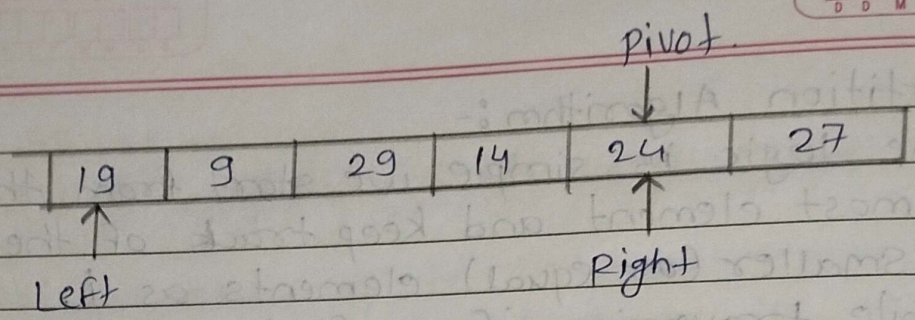


a[Pivot] < a[right], so algorithm moves forward one position towards left.

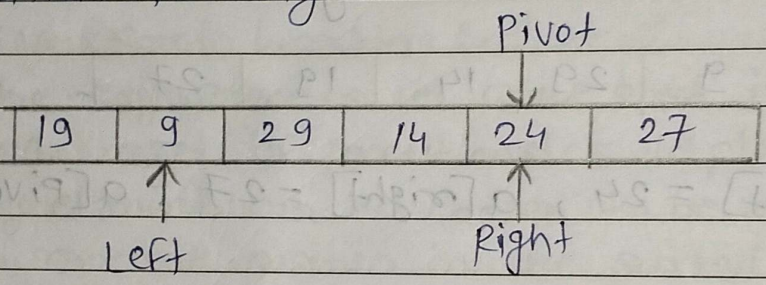


a[left] = 24, a[right] = 19, a[Pivot] = 24

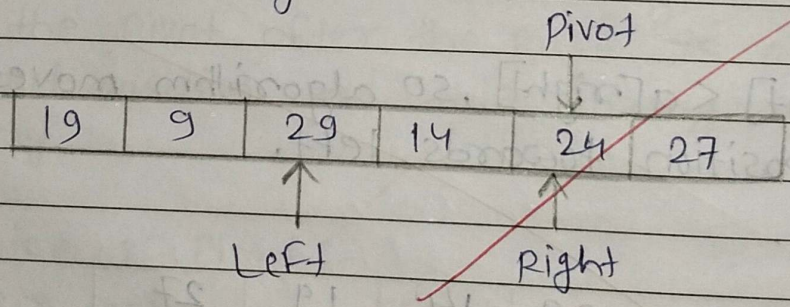
a[Pivot] > a[right] so, algorithm will swap a[Pivot] with a[right] and pivot moves to right.



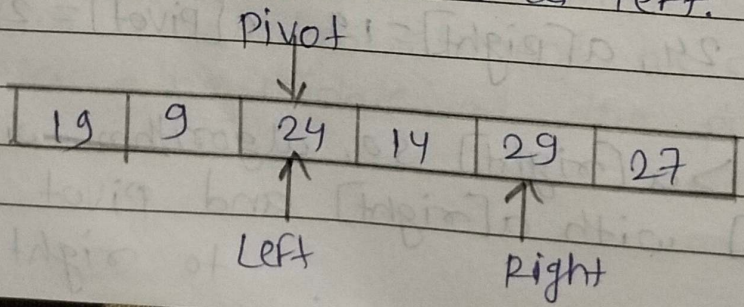
a[left] = 19, a[right] = 24, a[pivot] = 24
- since, pivot is at right so algorithm starts from left and moves to right.
- As a[pivot] > a[left], so algorithm moves one position to right.



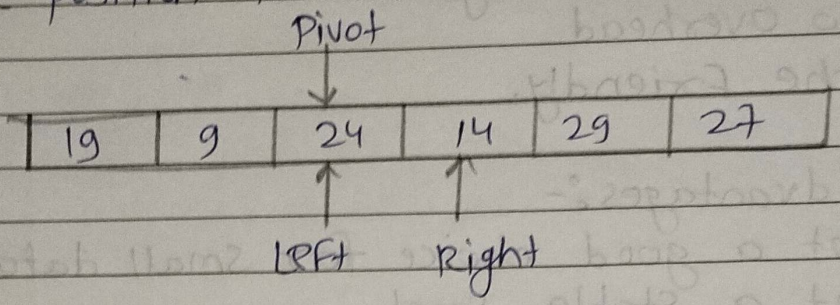
a[left] = 9, a[right] = 24, a[pivot] = 24
- As a[pivot] > a[left], so algorithm moves one position to right.



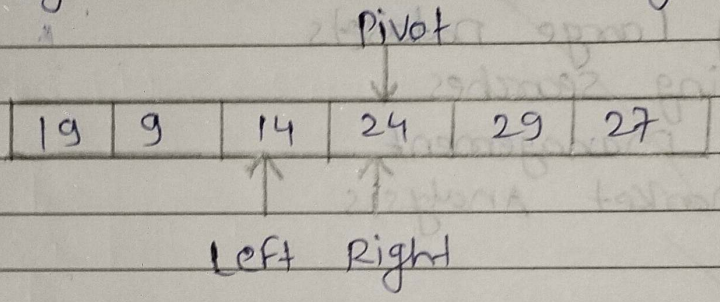
a[left] = 29, a[right] = 24, a[pivot] = 24
- As a[pivot] < a[left] so, swap a[pivot] and a[left] now pivot is at left.



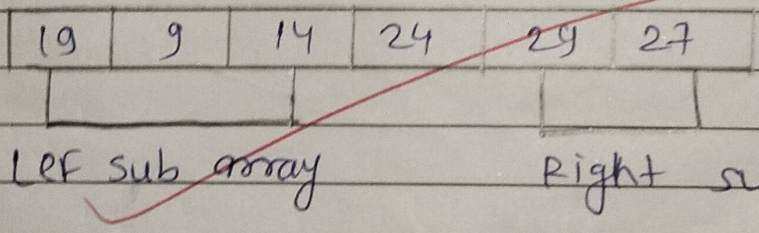
a[left] = 24, a[right] = 29, a[Pivot] = 24
- As a[Pivot] < a[right] so, algorithm moves one position to left.



a[Pivot] = 24, a[left] = 24, a[right] = 14
- as a[Pivot] > a[right] so, swap a[Pivot] and a[right], now Pivot is at right.



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



- Quick sort algorithm is separately applied to the left and Right sub-arrays.
- After sorting gets done, the array will be-

9	14	19	24	27	29

Time complexity :- Best - $O(n \log n)$
Worst - $O(n^2)$

- Advantages :-
1) makes it easier to solve problems
2) efficient on large data sets
3) Low overhead
4) Cache Friendly.

- Disadvantages :-
1) not a good choice for small data sets.
2) not a stable sort
3) worst-case time complexity of $O(N^2)$.

- Application :-
1) sorting large Datasets
2) optimizing searches
3) memory management
4) stock Market Analysis