# UNIT-4

## Analaysis of Algorithms:—

Analaysis of algorithms is the determination of the amount of time and space resources required to execute it.
Usually the efficiency or running time of an algorithm is stated as a function relating the input length to the number of steps, known as space Complexity.
Analaysis of algorithm is the process of analzing the problem-solving capability of the algorithm in term of time and size required (the size of memory for storage while implementation. However the main concern of analaysis algorithm is the required time or performance. Generally we perform the following types of analyses:—

Worst Case— The maximum no. of steps taken on any instance of size a.

Best Case The minimum no. of steps taken on any instance of size a.

~~Worst Case~~— Average Case— An average no. of Steps taken on any instance size of a.

To solve a problem, we need to consider a time as well as space Complexity as the program may run on a system where memory is limited but adequate space is available or may be vice-versa.

Complexity with big O notation.

→ O(1) - Constant Complexity public void Some function (int [] input array)
{ System.out.print("n = " + Input Array.length);
}

→ O(n) - Linear Complexity
public void Some function (int [] Input Array)
{
for (int Item : Input Array)
{ System.out.println (item);
}
}

→ O(n²) - Quadratic Complexity
public void Some function (int [] Input Array)
{
for (int firstItem : Input Array)

```
{
for (int SecoundItem : InputArray)
{
for int[] orderedpair = new int[]
{ firstItem , SecoundItem };
System.Out.println(Arrays.toString
(orderedpair));
}
}
}
```

→ O(log n) - Logarithm Complexity public void SomeFunction(int[] InputArray)

```
{
for (int i=1; i<=InputArray.length;
i = i*2)
{ System.out.println(Item[i]);
}
}
```

→ O(n³) - Cubic Complexity : Similar to O(n²), but Consider that example with an added nested loop.

→ O(n log n) - Linearithmic Complexity : The merge sort worst case Complexity is o (n log n)

→ O(2n) - Exponential Complexity : The algorithm takes twice as long for every new element added So even small Increases in n dramatically Increase the running time.

# Searching :-

Searching in data structure refers to the process of finding the required information from a collection of Items Stored as elements in the Computer memory. These sets of Items are in different forms, Such as an array, linked list graph, or tree.

Another way to define Searching in the data structure is by locating the desired element of specific characteristics in a collection of items.

Types of Searching.

* linear Search
* binary Search

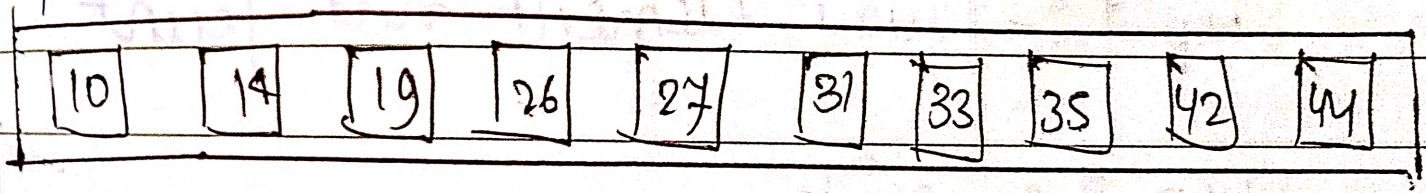
* Linear Search :- Linear Search is a very basic and simple Search algorithm. In Linear Search, we Search an element or value in given array by traversing the array from the starting till the desired element or value is found.

It Compares the element to be Searched with all the elements present in the array and when the element is matched Successfully, it returns the index of the

element in the array, else it return -1.

Linear search is applied on ~~unsorted~~ unsorted or unordered lists, when there are fewer elements in a list

for example Linear Search




=33

## ⇒ Binary Search

Binary search is used with sorted array or list. In binary search, we follow the following steps :-

1. We start by comparing the element to be searched with the element in the middle of the list / array
2. If we get a match, we return the index of the middle element.

3 If we do not get a match, we check whether the element to be Searched is less or greater than in value than the middle element.

4. If the element / number to be Searched is greater in value than the middle number then we pick the elements on the right side of the middle element (as the list / array is Sorted, hence on the right, we will have all the numbers greater than the middle number), and start again from the Step 1.

5 If the element / number to be Searched is lesser in value than the middle ~~element~~ number, then we pick the elements on the left side of the middle element, and start again from the Step 1.

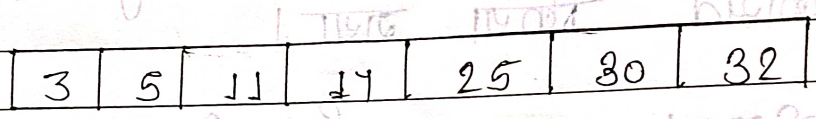
Binary Search is useful when there are large number of elements in an array and they are Sorted. So a necessary Condition for Binary

Search to work is that the list/ array should be sorted.

Example :-

इसको हम भलीभांति उदाहरण के द्वारा समझ सकते हैं।

माना हमारे पास ति० ति० Array List है।




हमें दिया गया element 5 है जिसे हमने List में ढूंढना है।

Step 1 :- सबसे पहले हम दिये गये element 5 की तुलना middle element 17 से करते हैं।

| 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 3 | 5 | 11 | 17 | 25 | 30 | 32 |

दोनों एकसमान नहीं है और 5 जो है वह 17 से छोटा है।

तो हम list के बाएँ वाले भाग (छोटे वाले भाग) में ही search करेंगे।

| 0 | 1 | 2 |
|---|---|---|
| 3 | 5 | 11 |

Step 2 :- किये गये element 5 को middle elemen 5 के साथ compare करेंगे।

| 0 | 1 | 2 |
|---|---|---|
| 3 | 5 | 11 |

दोनों एकसमान है तो हम तुलना करना बन्द कर देंगे। और index 1 return करेंगे।

## * Difference Between Linear Search and Binary Search :*

| | Linear Search | Binary Search |
|---|---|---|
| Input | Unsorted or sorted list or array | Sorted list or array |
| Method | Sequantially checks each element until a match is found | Divides the search interval in half with each iteration |
| Time complexity | O(n) | O(log n) |
| Efficiency | Slow for large datasets | Fast for large database |
| Data requirement | Can be used on unsorted data | Requires sorted data |
| Implementation | Simple to implement | More complex to implement |
| Advantages | Can be used on unsorted data | Fast for large database |
| Disadvantages | Slow for large database | Requires Sorted Data |

# What is a Sorting Algorithm?

A Sorting algorithm is just a series of orders or instructions. In this, an array is an input, on which the Sorting algorithm performs operations to give out a sorted array

There are two different Categories in Sorting.

Internal Sorting :- If the input data is such that it can be adjusted in the main memory at once, it is called Internal Sorting.

External Sorting :- If the input data is such that it cannot be adjusted in the memory entirely at once, it needs to be stored in a hard disk, floppy disk, or any other storage device. This is called external Sorting.

Some common Internal Sorting algorithms include :

- Bubble Sort
- Insertion Sort
- Quick Sort
- Heap Sort
- Radix Sort
- Selection Sort.

else

E = mid - 1

6. Return Loc.

# Insertion Sort.

Insertion sorts works by taking elements from the list one by one and inserting them in their current position into a new sorted list. Insertion sort consists of N-1 passes, where N is the number of elements to be sorted. The $i^{th}$ pass of Insertion Sort will insert the $i^{th}$ element A[i] into its rightful place among A[1], A[2] .... A[i-1]. After doing this insertion the records occupying A[1] .... A[i] are in sorted order.

Insertion sort procedure

Void Insertion_Sort (int a[], int n)

```
{
    int i, j, temp;
    for (i=0; i<n; i++)
    {
        temp = a[i];
        for (j=i; j>0 && a[j-1]>temp; j--)
        {
            a[j] = a[j-1];
        }
        a[j] = temp;
    }
}
```

Example

Consider an unsorted array as follows

| 20 | 10 | 60 | 40 | 30 | 15 |
|---|---|---|---|---|---|

# Passes of Insertion Sort.

| original | 20 | 10 | 60 | 40 | 30 | 15 | Position moved |
|---|---|---|---|---|---|---|---|
| After i=1 | 10 | 20 | 60 | 40 | 30 | 15 | 1 |
| After i=2 | 10 | 20 | 60 | 40 | 30 | 15 | 0 |
| After i=3 | 10 | 20 | 40 | 60 | 30 | 15 | 1 |
| After i=4 | 10 | 20 | 30 | 40 | 60 | 15 | 2 |
| After i=5 | 10 | 15 | 20 | 30 | 40 | 60 | 4 |
| Sorted array | 10 | 15 | 20 | 30 | 40 | 60 | |

## What is Bubble Sort?

Bubble Sort is one of the simplest Sorting algorithms. It is also known as Sorting by exchange. It is a Complexity Comparison-based algorithm. Its time complexity is of the order $O(n^2)$ where $n$ is the number of elements.

### Working of Bubble Sort

In bubble sort, we compare adjacent elements and whenever we get a pair that is out of order, we swap them.

For $n$ elements, there are $n-1$ iterations or passes. In every iteration, the largest element reaches its highest iteration, the largest element reaches its highest item.

The steps followed by the bubble sort algorithm are:

Start comparing each element with its adjacent element from the starting index

2. if the current and the next element are out of order, swap them.

3. repeat step 2 for all the elements of the array/list.

4. repeat step 1, 2 and 3 until we have reached the final sorted state of the array.

Bubble Sort - Pseudo - Code

The pseudo-code for the bubble Sort algorithm is:

```
Procedure Bubble_Sort (int array)
for (pass = 1; pass <= n; pass++)
for (index = 0; index <= n - 1 - pass;
        index ++)
if (array [index] > array [index + 1])
        Swap (array [index],
```

array [index + 1])

end if
end for
end for
end Procedure.

## Complexity of Bubble Sort

### Time Complexity:

Time complexity in bubble sort is mainly due to two operations:

1. Comparisons
2. Swaps

In bubble sort, we compare adjacent elements. Thus the number of comparision will be:

| Iteration | Number of Comparisons |
|---|---|
| 1 | (n-1) |
| 2 | (n-2) |
| 3 | (n-3) |
| . | . |
| . | . |
| n | 1 |

| | |
|---|---|
| $n-2$ | 2 |
| $n-1$ | 1 |

Thus, the total number of comparisons are

$$(n-1)+(n-2)+(n-3)+\ldots\ldots+2+1$$
$$= n(n-1)/2$$
$$= O(n^2)$$

Example :-

7 11 9 2 17 4

pass 1

| | | | |
|---|---|---|---|
| 7 ↖ no change | 7 | 7 | 7 |
| 11 ↙ | 11 ↖ Swap | 9 | 9 |
| 9 | 9 ↙ | 11 ↖ Swap | 2 |
| 2 | 2 | 2 ↙ | 11 ↖ no change |
| 17 | 17 | 17 | 17 |
| 4 | 4 | 4 | 4 |

7
9
2
11
17 ↔ Swap
4

7
9
2
11
4
17

pass 2

7 ↔ no change
9
2
11
4
17

7
9 ↔ Swap
2
11
4
17

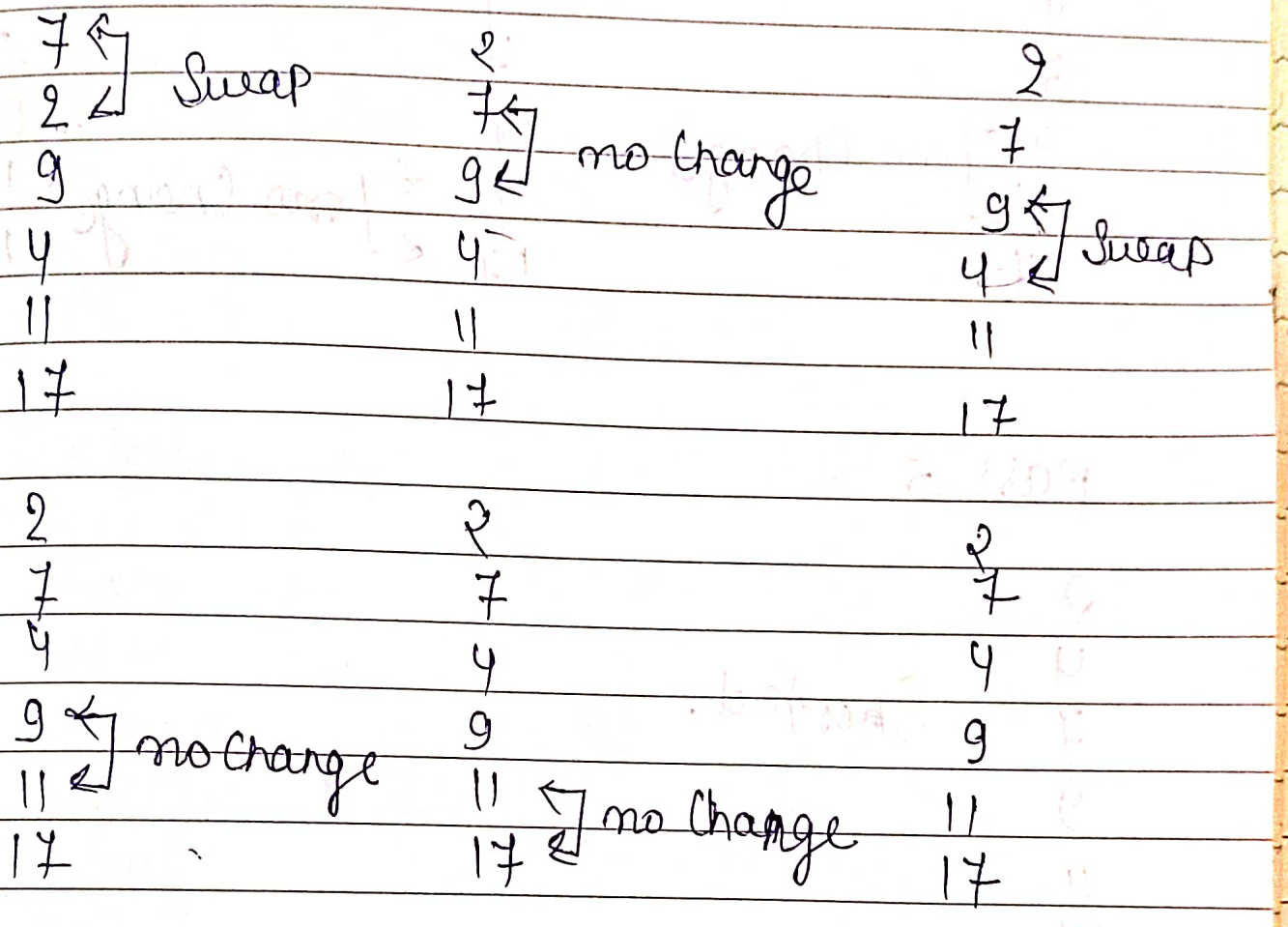
7
2
9 ↔ no chang
11
4
17

7
2
9
11 ↔ Swap
4
17

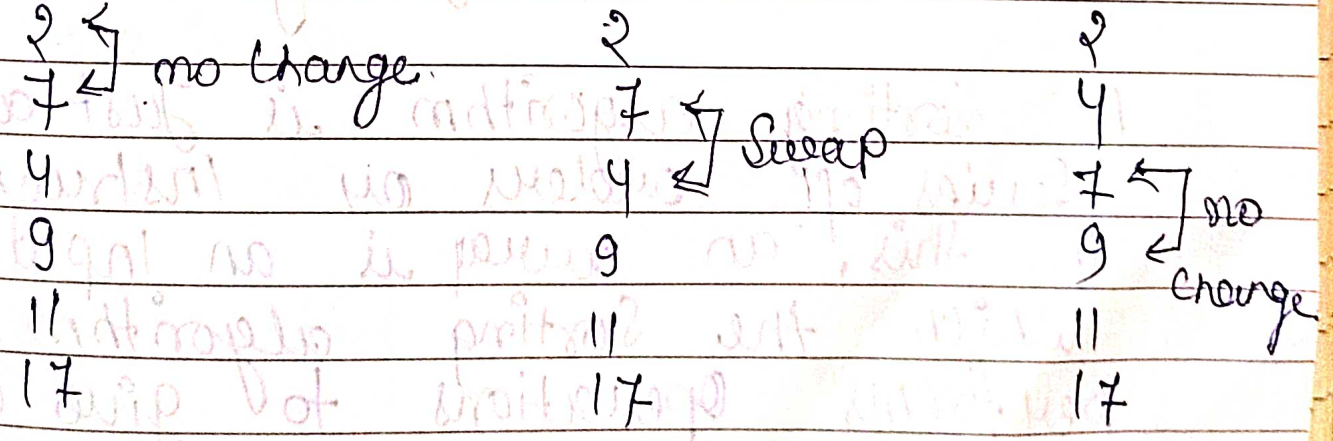
7
2
9
4
11 ↔ no change
17

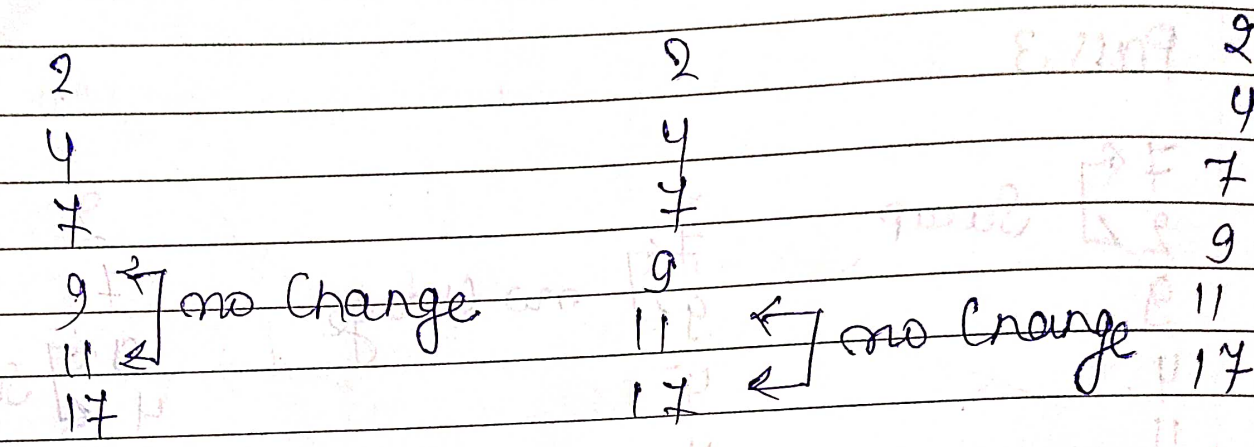
7
2
9
4
11
17

Pass 3


7
2
Swap
9
4
11
17
2
7
9
no change
4
11
17
2
7
9
4
Swap
11
17
2
7
4
9
11
17
no change
2
7
4
9
11
17
no change
2
7
4
9
11
17

pass 4


2
7
no change
4
9
11
17
2
7
4
Swap
9
11
17
2
4
7
9
no change
11
17




pass 5

2
4
7
9
11
17

Sorted.

⇒ Selection Sort :- The Selection Sort algorithm is a simple, yet effective Sorting algorithm. A Selection-based Sorting algorithm is described as an in-place Comparison-based algorithm that divides the list into two parts, the sorted part on the left and the unsorted part on the right. Initially the sorted Section is empty, and the unsorted Section Contains the entire list. When Sorting a small list, Selection Sort can be used.
In the Selection Sort, the cost of Swapping is irrelevant, and all element must be checked. The Cost of writing to memory matters in Selection Sort, just as it does in flash memory (the number of writes/swap is $O(n)$ as opposed to $O(n^2)$ in bubble Sort.)

Algorithm and pseudocode of a Selection Sort Algorithm.

Algorithm of the Selection Sort Algorithm

The Selection Sort algorithm is as follows

Step 1:- Set Min to location 0 in ~~Sft~~ Step 1.

Step 2:- Look for the smallest element on the list.

Step 3:- Replace the value at location Min with a different value.

Step 4:- Increase Min to point to the next element

Step 5:- Continue until the list is sorted.

Swap array [minimum] and array[i]

end if
end for

if index of minimum != i then //
Swap the minimum element
with the current element.

Swap array [minimum] and array[i]

end if
end for

end function.

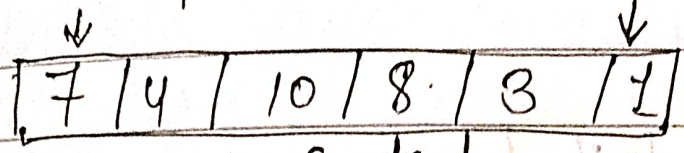
→

The Complexity of Selection Sort Algorithm
the time complexity of the
Selection Sort algorithm is:

The Selection Sort algorithm is made
up of two nested loops

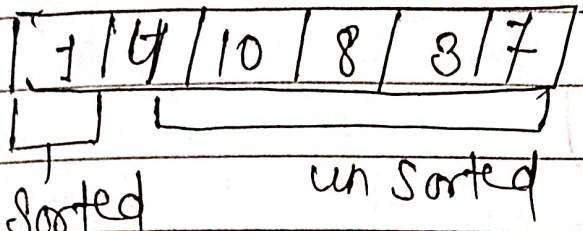
It has an O(n2) time complexity due
to the two nested loops.

Example

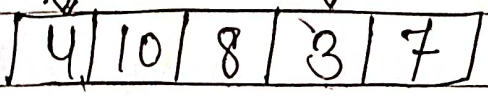


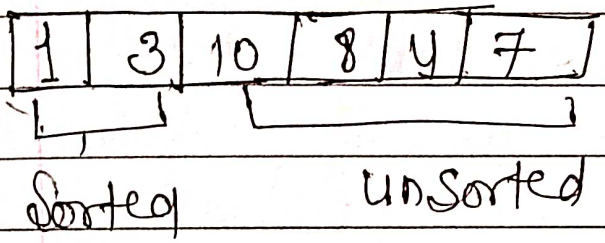

unSorted array

Pass 1

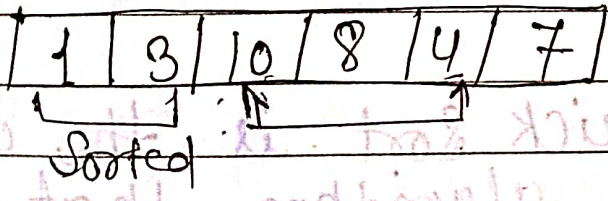


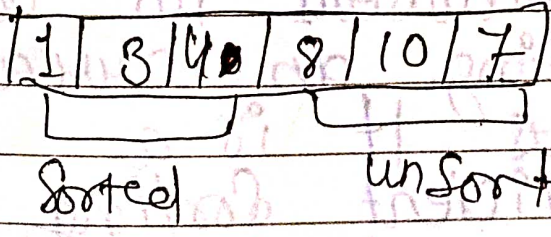

pass 2



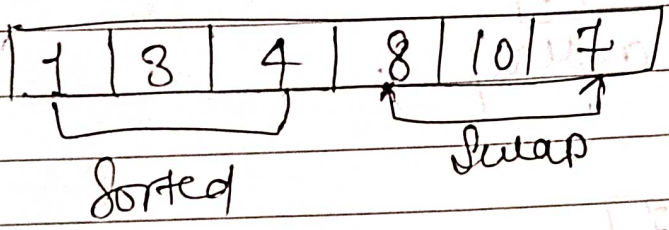



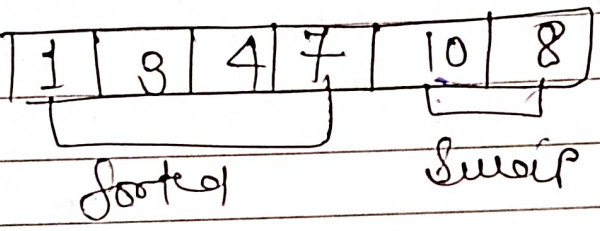

pass 3



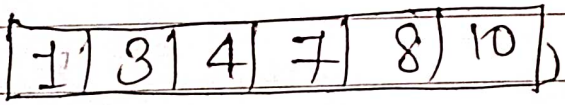



Pass 4




pass 5




Sorted.

⇒ Quick Sort :- Quick Sort is the widely used Sorting algorithm that makes n log n comparisons in average case for Sorting an array of n elements. It is a fastest and highly efficient Sorting algorithm. This algorithm follows the divide and Conquer approach

Divide and Conquer is a technique of breaking down the algorithms into subproblems, then solving the subproblems, and combining the result back together to solve the original problem.

Divide :- In Divide, first pick a pivot element. After that, partition or rearrange the array into two sub-arrays such that each element in the left sub-array is less than or equal to the pivot element and each element in the right sub-array is larger than the pivot element.

Conquer :- Recursively, sort two sub arrays with Quick Sort.

Combine :- Combine the already sorted array.

Quick Sort picks an element as pivot, and then it partitions

the given array around the picked pivot element. In Quick Sort, a large array is divided into two arrays in which one holds values that are smaller than the specified value (pivot) and another array holds the values that are greater than the pivot

After that, left and right Sub-array are also partitioned using the same approach. It will continue until the single element remains in the Sub-array.

Example :-

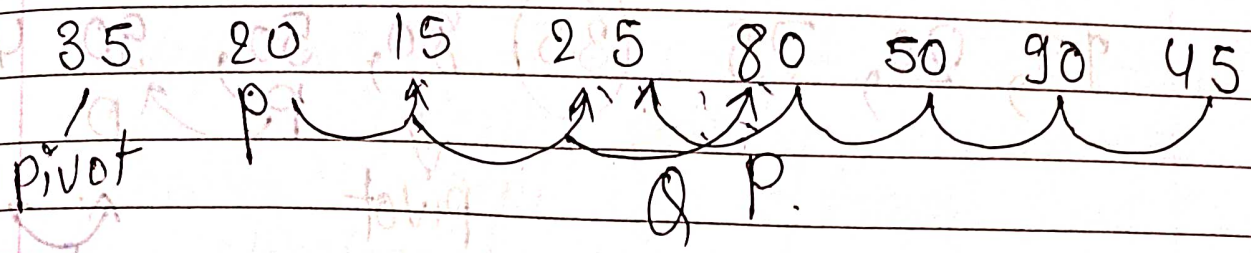
| 35 | 50 | 15 | 25 | 80 | 20 | 90 | 45 |
|---|---|---|---|---|---|---|---|

Pivot

→ Pass 1.

35 50 15 25 80 20 90 45 +∞

p q




when p and Q are intersect each Other pivot element are ~~in~~ goes in middle while left part of pivat element is smalley then pivat element & right part is ~~b~~ largey then pivot element.

25, 20, 15, 35, 80, 50, 90, 45

Pass 1 Completed.

→ pass 2

25, 20, 15, 35, 80, 50, 90, 45

Pivot p q Pivot p q

when pivot value is largey then all values of left side so p goes to infinity and p & q are intersect then values are swap

15, 20, 25 (35) 80, 50, 90 45 +∞

pivot p q

p q

swap

15, 20, 25 (35) 80, 50, 45, 90 +∞

pivot p q

q p

when p and q are intersect each other pivot value swap to Q

35, ~~45~~

15, 20, 25 (35) ~~45~~, 50, 80, 90

left half right half

Combine both left half and right half

| 15 | 20 | 25 | 35 | 45 | 50 | 80 | 90 |
|---|---|---|---|---|---|---|---|

Complete Sorted.

⇒ Heap Sort :- Heap Sort is a popular and efficient Sorting algorithm. The concept of heap sort is to eliminate the elements one by one from the heap part of the list, and then insert them into the sorted part of the list.

Heap Sort is the In-place sorting algorithm.

Algorithm.

```
Heap Sort (arr)
Build max heap (arr)
for i = length (arr) to 2
Swap arr [1] with arr [i]
heap_size [arr] = heap_size [arr] - 1
max heapify (arr, 1)
END.
```

```
Build max heap (arr)
Build max heap (arr)
  heap_Size(arr) = length (arr)
    for i = length (arr) /2 to 1
  max heapify (arr, i)
End.
max Heapify (arr, i)
max heapify (arr, i)
L = left (i)
R = Right (i)
if L ? heap_Size [arr] and arr[L] >
  arr [i]
largest = L
else
largest = i
```

If R? heap_Size [arr] and arr[R] > arr [largest]

largest = R

If largest != i

Swap arr [i] with arr [largest]

max heapify (arr, largest)

END.

Working of heap Sort algorithm Now, let's See the working of the HeapSort Algorithm.

In heap Sort, basically, there are two phases involved in the Sorting of elements. By using the heap Sort algorithm, they are as follows:-

The first Step includes the Creation of a heap by adjusting the element of the array.

After the creation of heap, now remove the root element of the heap repeatedly by shifting it to the end of the array and then store the heap structure with the remaining elements.

Time Complexity

| Case | Time Complexity |
|---|---|
| Best Case | $O(n \log n)$ |
| Average case | $O(n \log n)$ |
| Worst Case | $O(n \log n)$ |

Example :-

| 10 | 15 | 5 | 8 | 25 |
|---|---|---|---|---|

arr[0] arr[4]

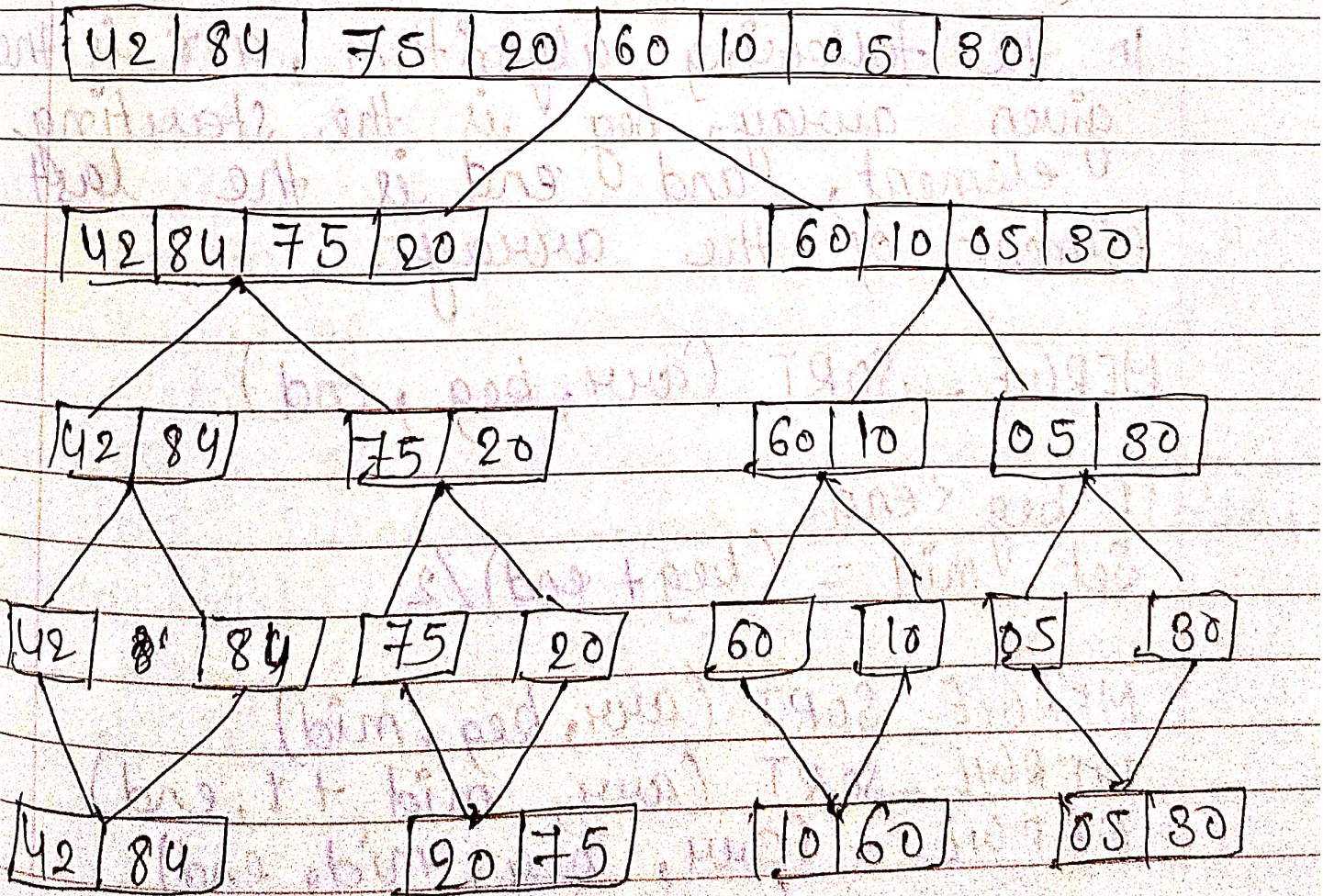
[External Sort]

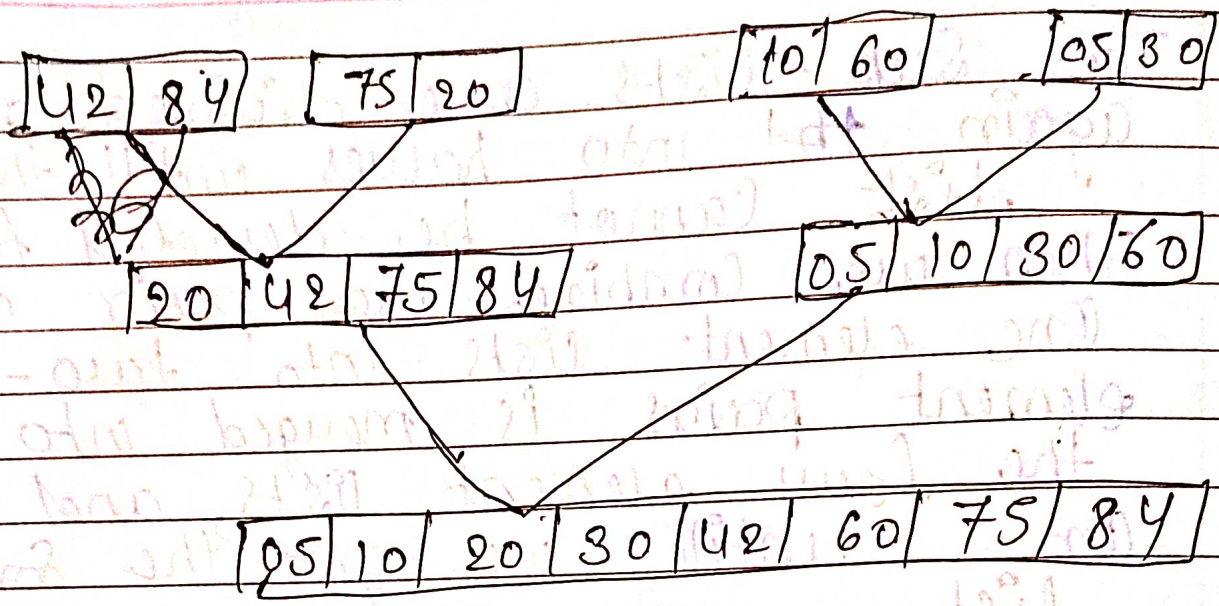
⇒ Merge Sort :- Merge Sort is the sorting technique that follows the divide and conquer approach. This article will be very helpful and interesting to Student as they might face merge sort as a question in their examinations. In coding or technical interviews for software engineers sorting algorithms are widely asked. So, it is important to discuss the topic.

merge Sort is similar to the quick Sort algorithm as it uses the divide and conquer approach to sort the elements. It is one of the popular and efficient sorting algorithm. It divides the given list into two equal halves calls it self for the two halves and then merges the two sorted halves. we have to define the merge() function to perform the merging.

The Sub-lists are divided again again into halves untill the list cannot be devided fauther Then we combine the pair of One element lists into two-element pairs is merged into the four element lists, and so on untill we get the sorted list.

merge Sort (Divide, Conquer & Combined)






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 for if
```

END MERGE_SORT