⇒ Merge Sort

- Let T[1--n] be an array of n elements, our problem is to sort these elements into ascending order.
- The obvious Divide-Conquer approach to this problem consists of separating the array T into two parts whose sizes are as nearly equal as possible, sorting these parts by recursive calls and then merging the solution for each part being careful, to preserve the order.
- To do this, we need an efficient algorithm for merging two sorted arrays U and V into a single sorted array T whose length is the sum of the length of U and V

i.e. Length T = Length of U + Length of V

- This sorting algorithm illustrates well all the facts of Divide-and Conquer.
- Merge sort separate the instances ~~if two~~ into two subinstances half the size, solve each of these recursively, and then combines the two sorted half-arrays to obtain the solution of original instance.

Algorithm

1) Initialize an array [0--n-1]
2) low = 0
   high = n-1
   m = (low + high)/2
3) Now A[0-mid] <-> left sub-list
   mid+1 = high <-> right sub-list
4) n1 <- size of array a
   n2 <- size of array b
5) i <- 1 (pointing 1st element of array a)

6) $j \leftarrow 1$ (pointing 1st element of array b)
7) for $k \leftarrow 1 - n_1 + n_2$
8) if $a_i < b_j$
$c_k \leftarrow a_i$
$i++$
else
$c_k \leftarrow b_j$
b_j++
9) Return
10) Stop

Analysis

Let $T(n)$ be the total list to sort an array

$$T(n) = T(n/2) + T(n/2) + C_n$$

C_n → Basic Operation

$T(n/2)$ → Time required to sort left sublist

$T(n/2)$ → Time required to sort right sublist

$$T(n) = 2T\left(\frac{n}{2}\right) + C_n$$

Applying Master's theorem we get,

$a = 2 \quad b = 2 \quad d = 1$

$b^d = 2^1 = 2$

$\therefore a = b^d$

$$\boxed{T(n) = \Theta(n \log n)}$$

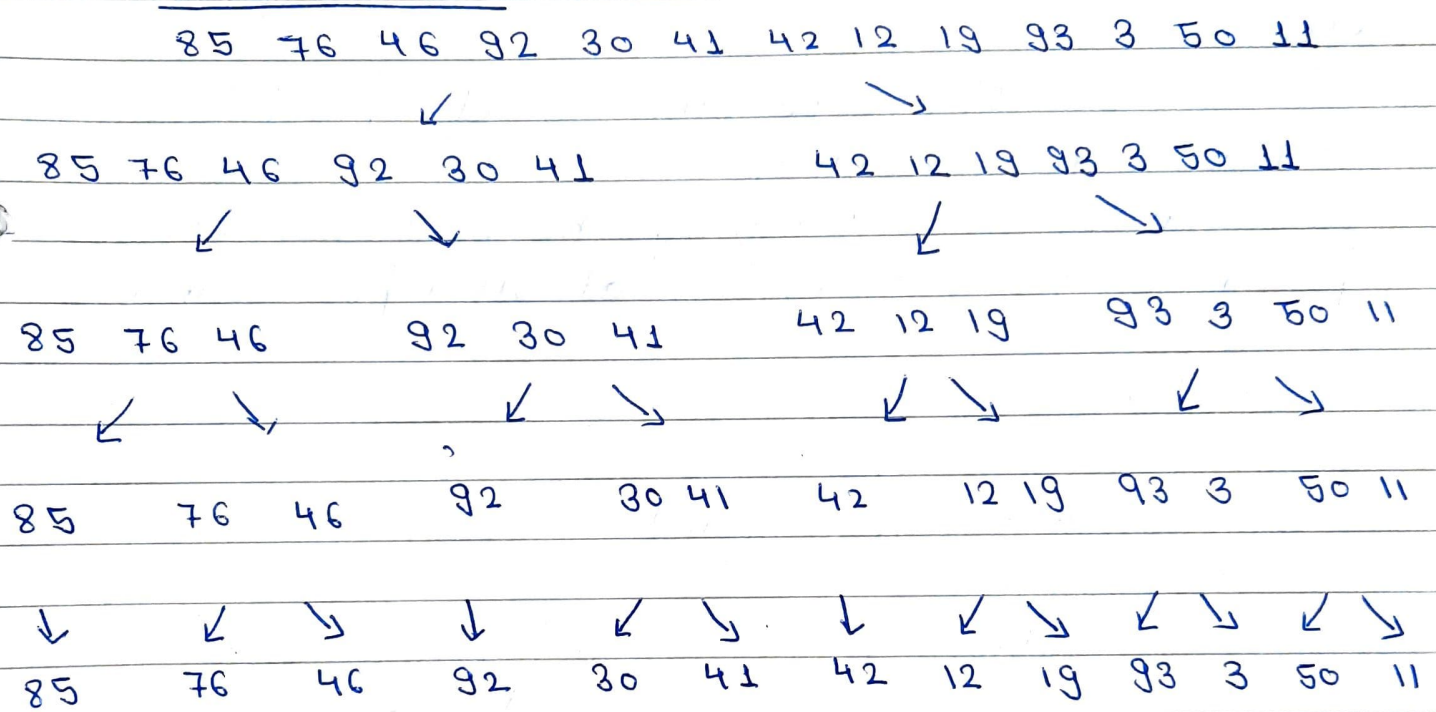
Example

Suppose the array A contains 13 elements as follow:

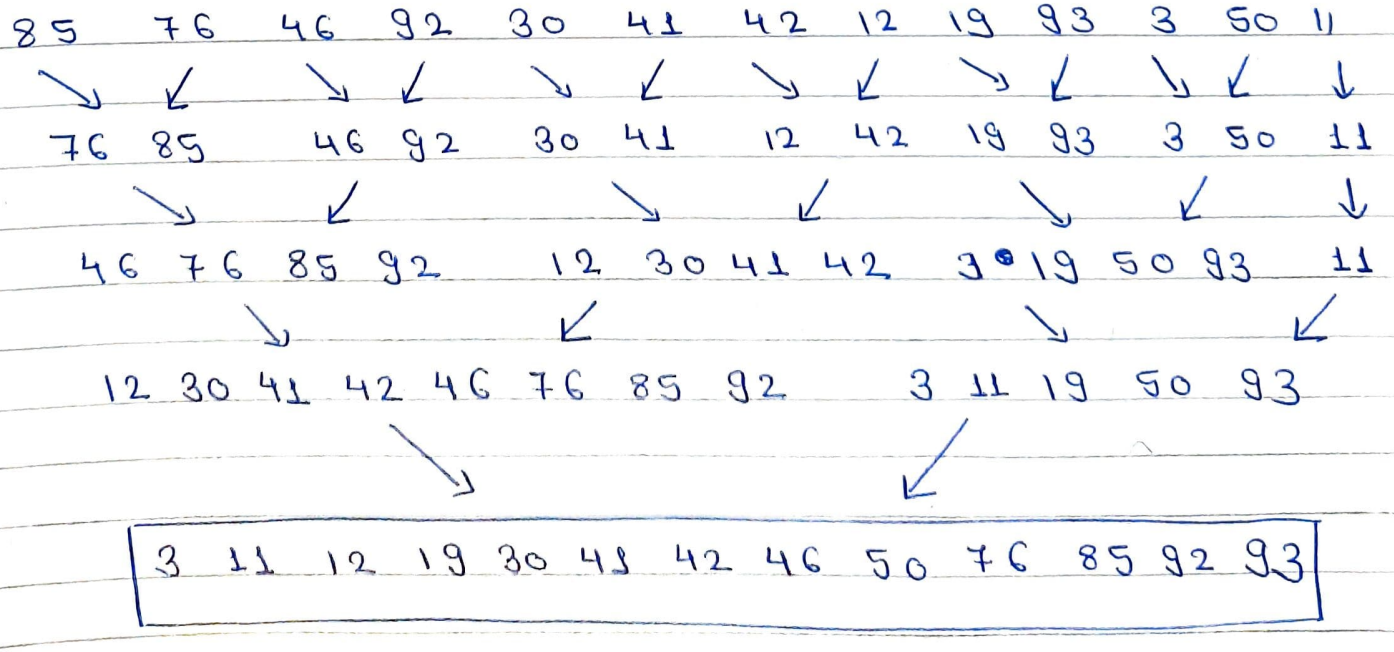
$A = \langle 85, 76, 46, 92, 30, 41, 42, 12, 19, 93, 3, 50, 11 \rangle$

Solution

→ **Divide Phase**



→ **Merge Phase**



Quick Sort

- In Quick Sort, we divide the list of elements into lists by choosing a partioning element called pivot element.
- One list contains all elements less than or equal to the partioning element and the other list contains all elements greater than the pivot element
- These two lists are recursively partitioned in the same manner.
- We then go on combining the sorted lists to produce the sorted list of all input elements.

Algorithm

```
1) QuickSort (A, low, high)
2) if (low < high)
3) Pivot-location <— Partition (A, low, high)
4) QuickSort (A, low, Pivot-location -1)
5) QuickSort (A, Pivot-location +1, high)
6) Partition (A, low, high)
7) Pivot <— A[low]
8) Left side <— low
9) for i <— low+1 to high
10) if (A[i] < Pivot) then
11) left side <— left side +1
12) Swap (A[i], A[left side])
13) Swap (A[low], A[left side])
```

Example

A = ⟨3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5, 8, 9⟩

Step-1 The array is pivoted about its first element i.e, Pivot (P) = 3

3 1 4 1 5 9 2 6 5 3 5 8 9
P

Step-2 Find first element greater then Pivot (make i) and find element not larger than Pivot from end and make j

3 1 4 1 5 9 2 6 5 3 5 8 9
P i j

Step-3 Swap these i & j and start again

3 1 3 1 5 9 2 6 5 4 5 8 9
P

3 1 3 1 5 9 2 6 5 4 5 8 9
P i j

Apply Swapping

3 1 3 1 2 9 5 6 5 4 5 8 9
P j i

The pointers have crossed i.e j on left of i
Then, in this situation swap Pivot with j

2 1 3 1 3 9 5 6 5 4 5 8 9
P

Now, Pivoting Process is complete

Step-4 Recursively sort sub-arrays on each side of Pivot.

Sub-array 1 : 2 1 3 1

Sub-array 2 : 9 5 6 5 4 5 8 9

First apply Quick sort for sub-array 1

2 1 3 1
P i j

2 1 1 3
j i

The pointers have crossed
Swap pivot with overline

| 1 1 2 3 | Sorted sub-array 1

Now apply Quick Sort for sub-array-2

9 5 6 5 4 5 8 9
P j i

Pointers have swapped, interchange Pivot and j

8 5 6 5 4 5 9 9
Sub array 3 P Sub array 4

8 5 6 5 4 5
P i

Swap i and P

5 5 6 5 4 8
Sub array 5 P

5 5 6 5 4
P i j

5 5 4 5 6
P j i

Swap P with j

4 5 5 5 6
P

Now combine all sub-arrays

1	1	2	3	3	4	5	5	5	6	8	9	9
				P								

Sorted array

Analysis

Let $T(n)$ be the total list to sort an array

$T(n) = T(n/2) + T(n/2) + Cn$ → Basic operation

- $T(n/2)$: Time required to sort left Sublist
- $T(n/2)$: Time required to sort right Sub-list

$$T(n) = 2T\left(\frac{n}{2}\right) + Cn$$

Applying Master's Theorem

$a = 2 \quad b = 2 \quad d = 1$

$b^d = 2^1 = 2$

~~$b^d = x$~~

$\therefore a = b^d$

$$\boxed{T(n) = \Theta(n \log n)}$$