MERGE SORT

- Yeh ek RECURSIVE ALGORITHM hai!
- MERGE SORT is a DIVIDE AND CONQUER Algorithm!

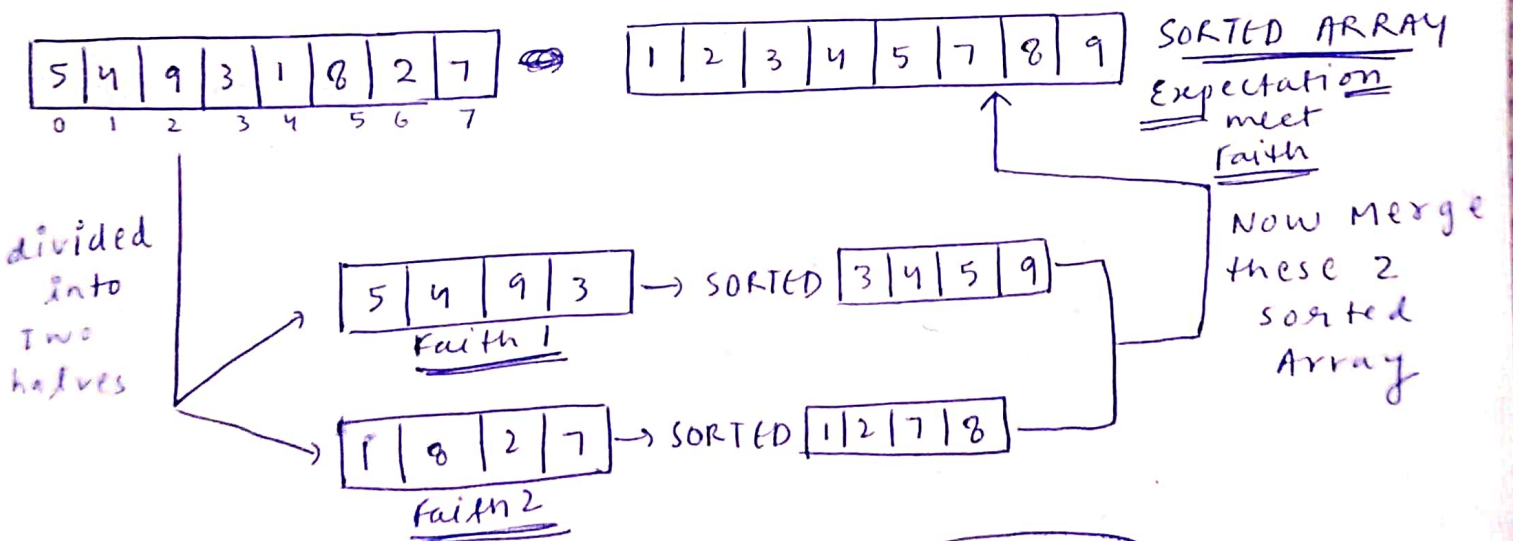
DIVIDE : Problem ko choti-choti problems me divide kiya jata hai!

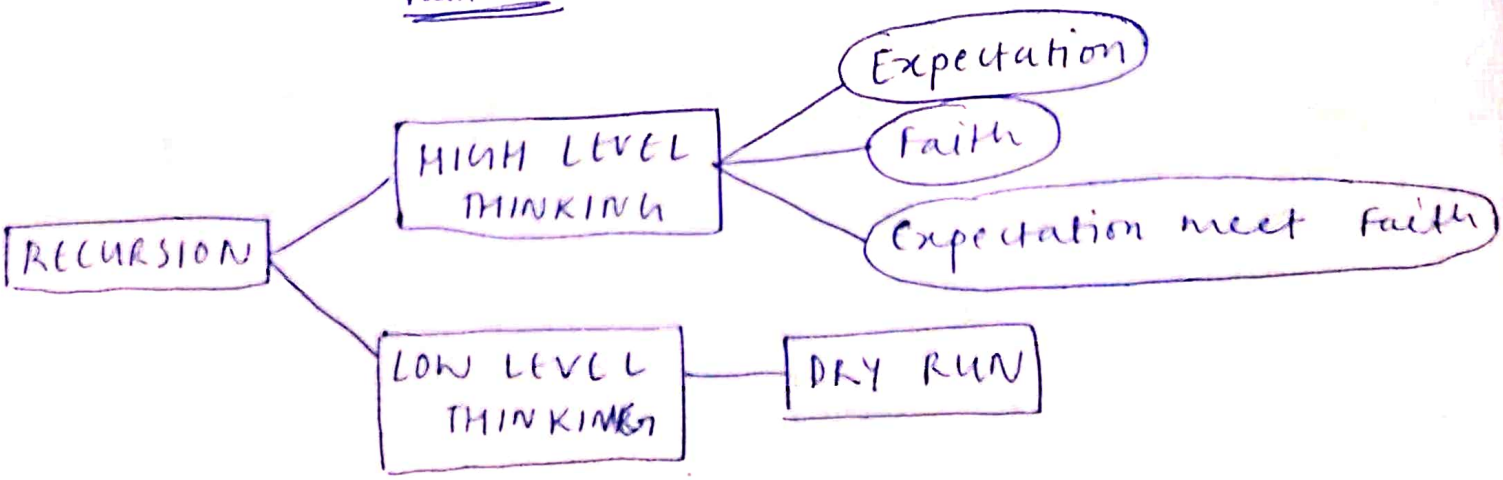
CONQUER : Matlab SOLVE krna choti sub-problems ko!

COMBINE : Jab choti problems solve ho jati hai, tab unke SOLUTIONS se Final Solution banega!

- Merge sort divide the input array into (2) halves, calls itself for the (2) halves and then merge the (2) sorted halves.

The merge() function is used to merge the (2) sorted halves.





Expectation

Hum expect krty hai humarey jo given array hai, Agar hum usse mergeSort function me pass krengy toh hume sort hokar miljayega as output.

Input Array

5 4 9 3 1 8 2 7

→ mergeSort (arr)

1 2 3 4 5 7 8 9

Faith

Hum Faith rakty hai ki agar mergeSort function hume purre array ko sort krke de sakta hai toh mergeSort function hume Array ke sub-arrays matlab Array ke Firsthalf aur Second half ko sort krke jarur dega!

Faith 1: 5493 → 3459

Faith 2: 1827 → 1287

Expectation Meets Faith

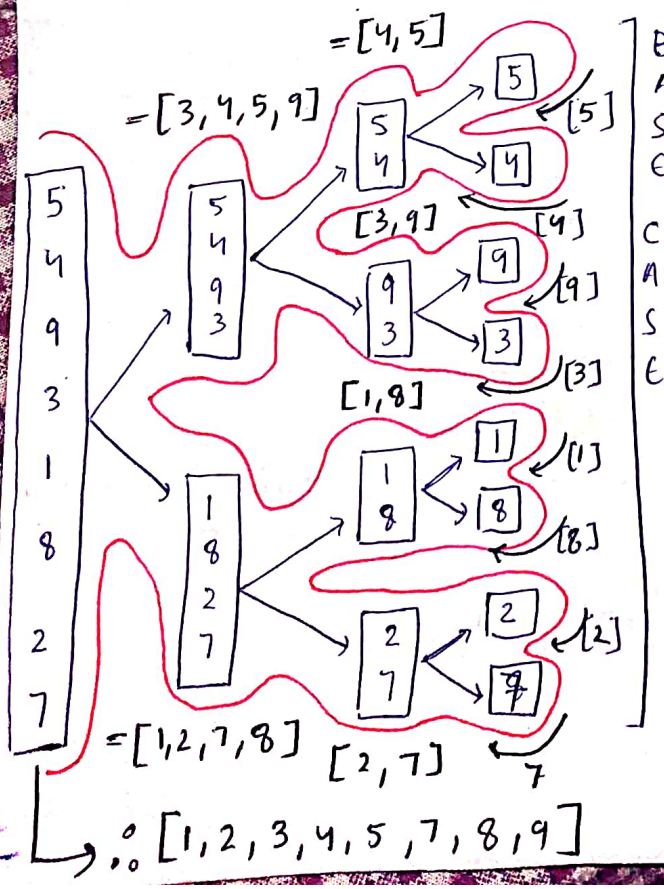
Hum Assume krtye hai ki Faith 1 aur Faith 2 purre ho jayega aur hume (2) sorted sub array mil jayega!

3459 1287

MERGE

Aab humara self work hoga ki dono sorted sub-arrays ko merge krdengy aur pura array combined & sorted banadengy!

1 2 3 4 5 7 8 9



BASE CASE → ① element left in the array so that array can't be divided into 1st and 2nd half so that, we can just create on array of size ①, put the element left & return it.

```
public static void main (String [] args){
Scanner s= new Scanner (System.in);
int n = s.nextInt();
int []arr = new int [n];
for(int i=0; i<arr.length ; i++)
{
   arr[i] = s.nextInt();
 }
arr = mergeSort (arr, 0, arr.length-1);
for( int val: arr)
{
 System.out.print (val+" ");
 }
 System.out.println (.);
 }
public static int [] mergeSort (int []arr, int low, int high)
{  if (low == high) {
      int baseArray = arr[0];
      baseArray [0] = arr [low];
      return baseArray;
    }
int mid = (low+high)/2;
int firsthalf = mergeSort (arr, low, high); ]T(n/2)
int Secondhalf= mergeSort (arr, mid+1, high); ]T(n/2]
int fullSortedArray = merge2SortedArrays (firsthalf, Secondhalf);
                                          ↳ T(n)
return fullSortedArray ;
}
public static int[] merge2SortedArrays (int []a, int [] b){
 int [] ans = new int (a.length + b.length);
 int i=0; int j=0; int K=0;
 while (i<a.length && j<b.length){
  if (a[i] <= b[j]) {
    ans[K] = a[i]; i++; K++;
  } else {
    ans[K] = b[j]; j++; K++;
  }
}
```

```
if (i == a.length)
{ while ( j < b.length) {
      ans[k] = b[j];
      j++;
      k++;
    }
}else {
   while ( j < a.length ) {
    ans[k] = a[i];
      i++;
      k++;
    }
  }
}
```

Merge Sort Algorithm DONE

∴ Merge Function Time complexity

$$T(n) = T\left(\frac{n}{2}\right) + T\left(\frac{n}{2}\right) + kn \quad -(1)$$

Evaluating Equation (1)

$$\therefore \quad \left(T(n) = T(n/2) + T(n/2) + kn\right) \times 2^0$$

$$\left(T(n) = kn + 2T(n/2)\right) \times 2^1$$

$$\left(T(n/2) = k(n/2) + 2T(n/4)\right) \times 2^2$$

$$\left(T(n/4) = k(n/4) + 2T(n/8)\right) \times 2^3$$

$$\vdots$$

$$\left(T(1) = c\right) \times 2^{x-1}$$

$$\Downarrow$$

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

$$2T(n/2) = 2k(n/2) + 4(T(n/4))$$

$$4T(n/4) = 4k(n/4) + 8(T(n/8))$$

$$\vdots$$

$$2^{x-1}T(1) = 2^{x-1}k\left(\frac{n}{2^{x-1}}\right)$$

x equations (Total)

$$\therefore T(n) = \underbrace{kn + kn + \cdots + kn}_{x \text{ times}}$$

$$\boxed{T(n) = kn(x)}$$

$$\therefore \quad [\underset{\downarrow}{n}, \frac{n}{2}, \frac{n}{4}, \ldots, \underbrace{1}_{x^{th} \text{ term}}]$$

a

$r = 1/2$

$\therefore n \to \frac{n}{2} \to \frac{n}{4}$

$$\therefore \quad ar^{x-1} = 1$$

$$\Downarrow$$

$$n\left(\frac{1}{2}\right)^{x-1} = 1$$

$$n\left(\frac{1}{2^{x-1}}\right) = 1$$

$$n = 2^{x-1}$$

Taking log both side,

$$\log_2 n = \log_2 2^{x-1}$$

$$\log_2 n = x - 1$$

$$\log_2 n + 1 = x \Rightarrow \boxed{x = \log n + 1}$$

$$\therefore T(n) = kn(x) = kn(\log n + 1)$$

$$T(n) = kn\log n + kn = \log n(kn) + kn$$

$$= k(n)(\log n) + k(n) = k[n\log n + n]$$

$$\therefore \boxed{T = k(n\log n) \propto n\log n = O(n\log n)}$$