Merge Sort

Array = { 8, 3, 4, 12, 5, 6 }

Sorting using merge sort.

Step 1) Divide this array in two parts:

8, 3, 4 , 12, 5, 6

↓ ↓

Sort the 1st half — Sort the 2nd half

And Merge Both the halves.

(recursive call)

1st : 3, 4, 8 — 2nd : 5, 6, 12

After sorting → Merge

[3, 4, 5, 6, 8, 12]

Step 2) Sort Both parts sorted via recursion

Step 3) Merge them.

Explaination:

If, arr 1 = [3, 5, 9]
arr 2 = [4, 6, 8]

to sort it, start checking from very first indices.

create an array of size (arr1 + arr2)

Check the indices value and put the values accordingly.

(*) if, suppose arr1 = [3, 5, 9, 19, 32] and arr2 = [4, 6, 8]

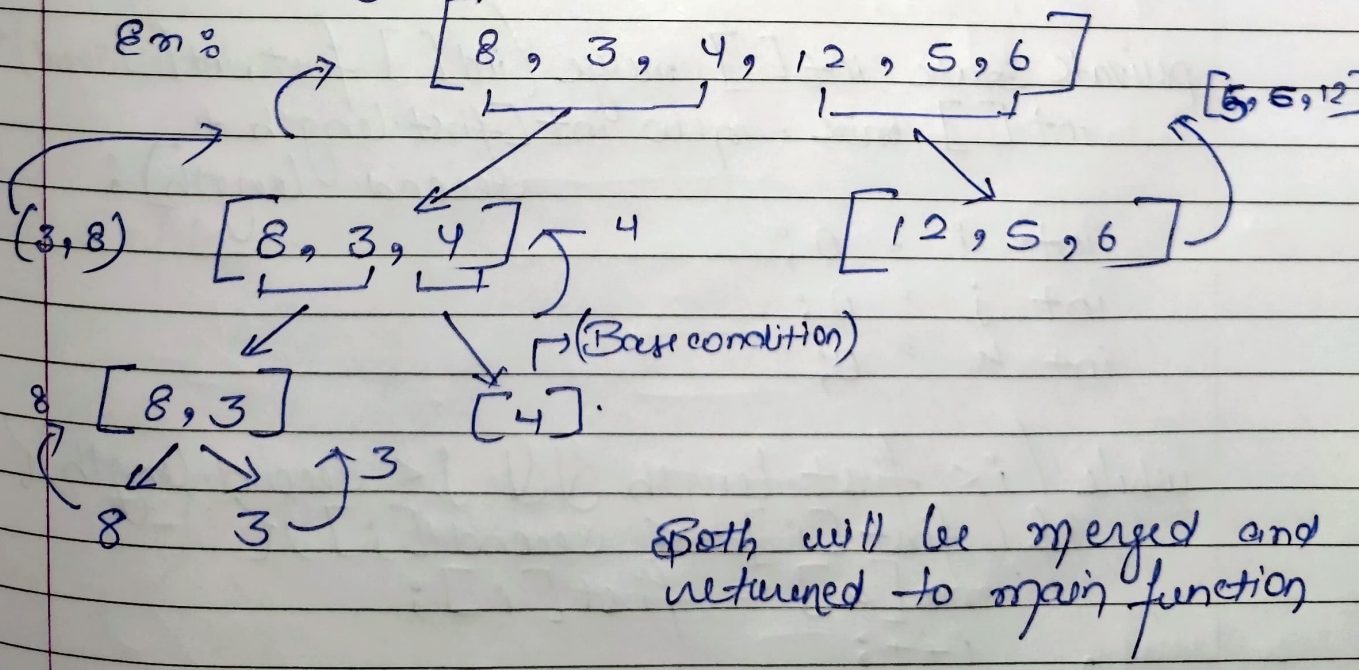
Remaining

arr Size [arr1+arr2] = [3, 4, 5, 6, 8]

put at the back of new array

[Recursive - Explanation]

(*) Recursion will sort the left and right part the array.



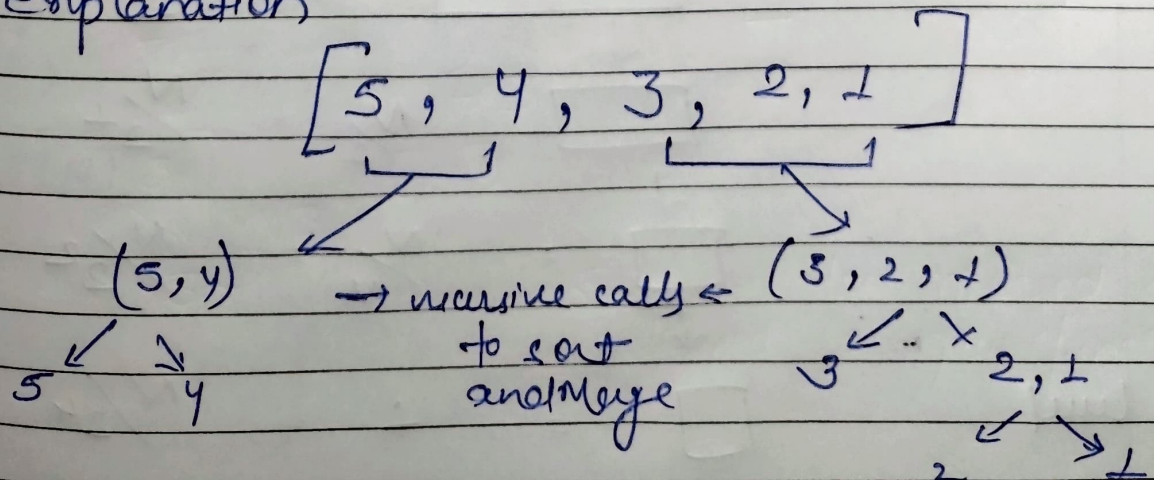
Both will be merged and returned to main function

Merge Sort Code :

```
public class MergeSort{
    public static void main(String[] args){
        int[] arr = {5, 4, 3, 2, 1};
        int[] ans = mergeSort(arr);
        System.out.println(Arrays.toString(arr));
    }

    static int[] mergeSort(int[] arr){
        if(arr.length == 1){
            return arr;
        }
        int mid = arr.length/2;

        int[] left = mergeSort(Arrays.copyOfRange(arr, 0, mid));
        int[] right = mergeSort(Arrays.copyOfRange(arr, mid,
                                arr.length));
        return merge(left, right);
    }

    private static int[] merge(int[] first, int[] second){
        int[] mix = new int[first.length +
                            second.length];
        int i = 0;
        int j = 0;
        int k = 0;

        while(i < first.length && j < second.length){
            if(first[i] < second[j]){
                mix[k] = first[i];
                i++;
            }else{
```

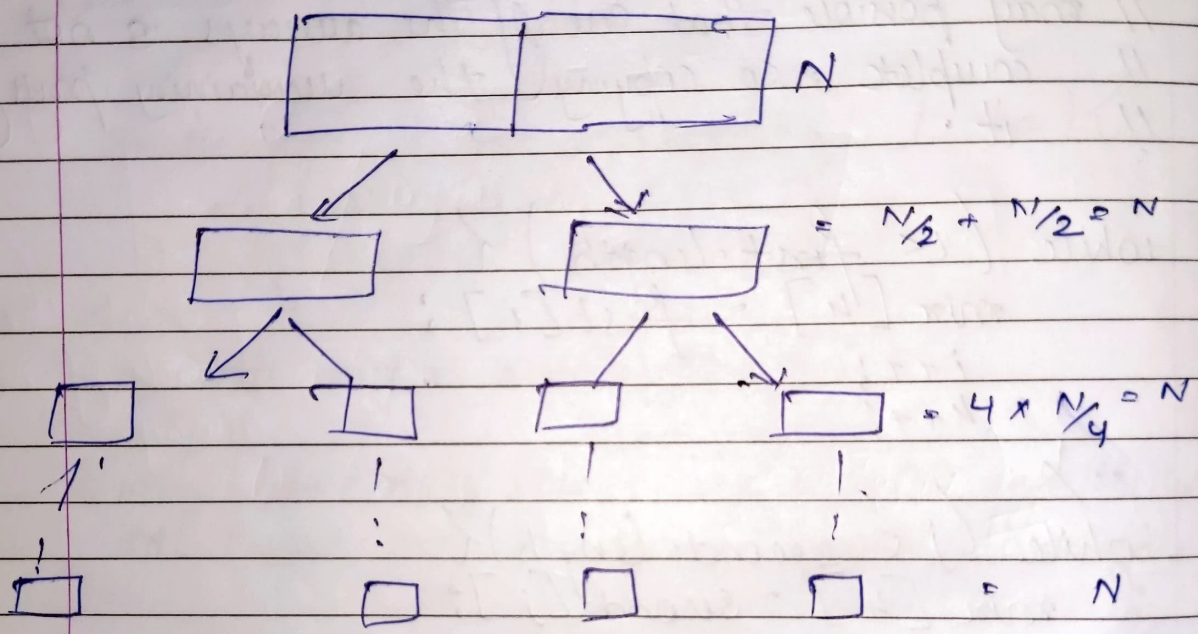
```
        mix[k] = second[j];
        j++;
    }
    k++;
}

// may possible that one of the arrays is not
//  complete, so copying the remaining parts of
//  it:

while (i< first.length) {
    mix[k] = first[i];
    i++;
    k++;
}
while (j < second.length){
    mix[k] = second[j];
    j++;
    k++;
}
return mix;
}
```

→ Explanation



It will return arrays of small indices which will merge to get a sorted one.

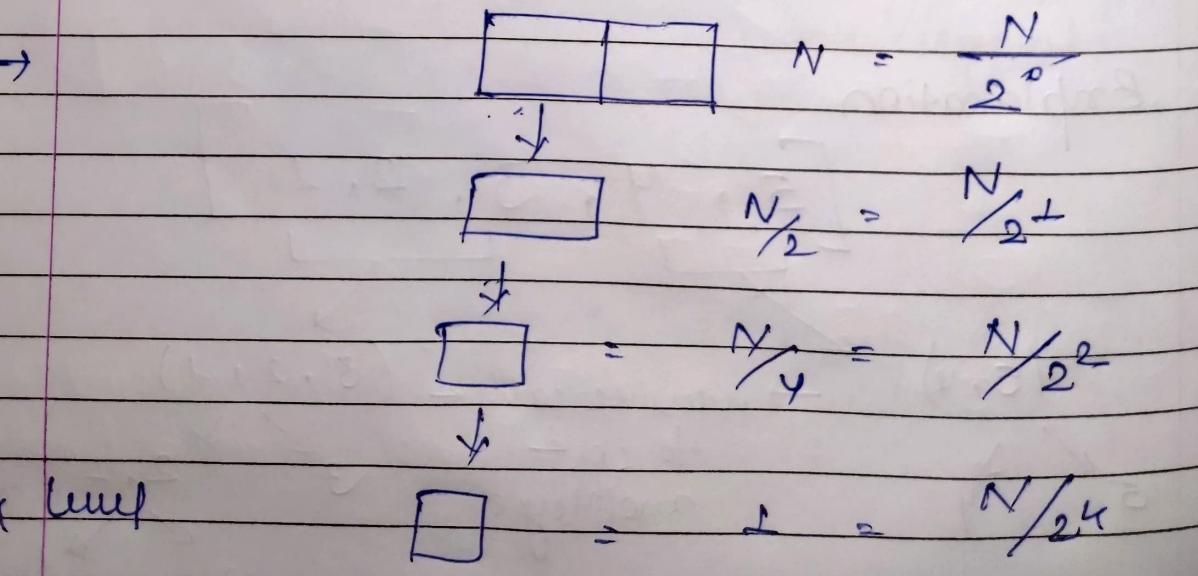
Time Complexity:



⇒ At every level, N elements are being merged.

Total Combinations = N

→



$$\Rightarrow \frac{1}{} = \frac{N}{2^k}$$

$$\Rightarrow 2^k = N$$

$$\Rightarrow k \log 2 = \log N$$

$$\Rightarrow [k = \log_2 N]$$

(*) So, ∴ Complexity = $O(N + \log N)$

(*) Space Complexity will be maximum height of recursive tree.

Auxiliary Space = $O(N)$

→ Space Complexity using Akra-Bazzi formula :

$$\Rightarrow T(N) = T(N/2) + T(N/2) + (N-1)$$

$$= 2T(N/2) + (N-1)$$

$$\because, \quad 2 \times \frac{1}{2^p} = 1 \quad [p = 1]$$

$$\Rightarrow T(N) = x + x \int_1^x \frac{u-1}{u^2}$$

$$= \int \frac{du}{u} - \int \frac{du}{u^2}$$

$$\Rightarrow \log u - \int u^{-2}\, du$$

$$\Rightarrow \log u + u^{-1}$$

$$= \left[\log u + \frac{1}{u}\right]_1^x$$

$$= \log x + \frac{1}{x} - 1$$

$$= x + x\left[\log x + \frac{1}{x} - 1\right]$$

$$= x + x\log x + 1 - x$$

$$= O(x \log x) = O(N \log N)$$