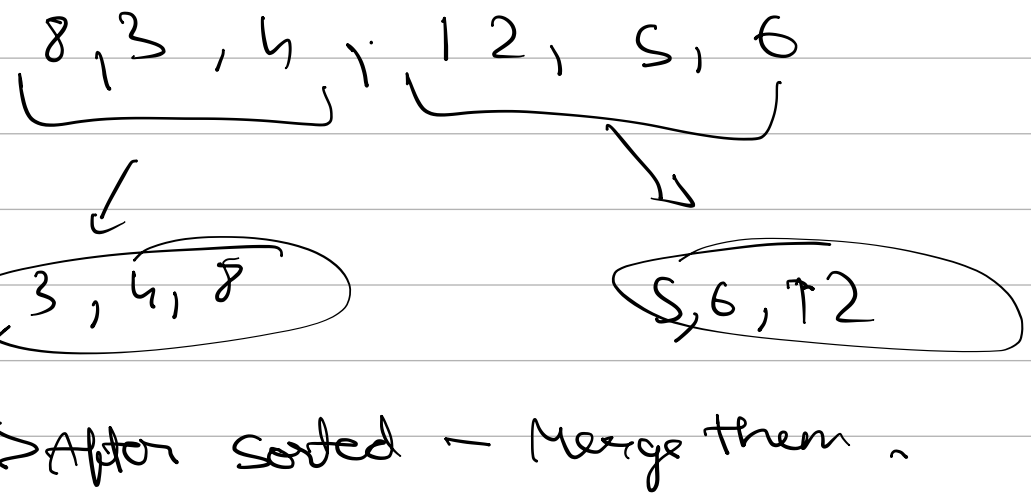
Merge Sort → Divide and Conquer.



After sorted — Merge them.

Merge comparison:-

$3 < 5$

[3]

$4 < 5$

[3, 4]

$5 < 8$

[3, 4, 5]

$6 < 8$

[3, 4, 5, 6]

$8 < 12$

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

Steps:- ① Divide Array to 2 parts, get them sorted via recursion.

② Merge the sorted parts.

② Step Explanation:- How to merge. ?

arr1 = [3,5,4]
arr2 = [4,6,8].

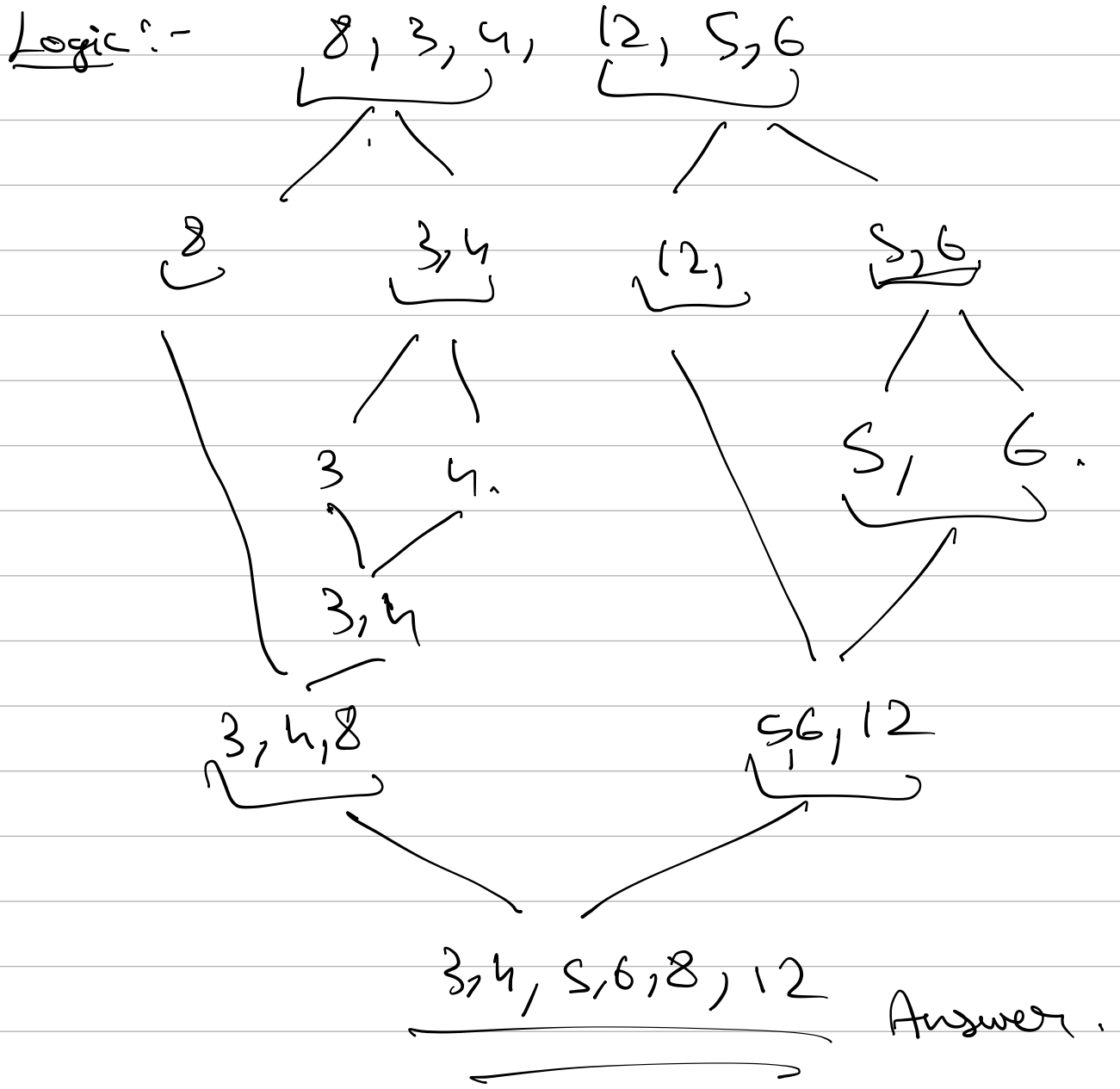
Create new array = size (arr1 + arr2).

arr 1 = [3,5,4]
i

arr 2 = [4,6,8]
j

*) Compare i and j,

```
if(arr1[i] < arr2[j])
newArray[k++] = arr1[i++];
else.
new arr[k++] = arr2[j++];
while (i<arr1.length
&& j<arr2.length)
```




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

```java
static int[] mergeSort(int[] arr) {
    if (arr.length == 1) {
        return arr;
    }

    int mid = arr.length / 2;

    int[] left = mergeSort(Arrays.copyOfRange(arr, from: 0, mid));
    int[] right = mergeSort(Arrays.copyOfRange(arr, mid, arr.length));

    return merge(left, right);
}
```

J-> base condition when arr.length=1

It passes a subarray from 0 to mid index.

```java
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++;
    }
```

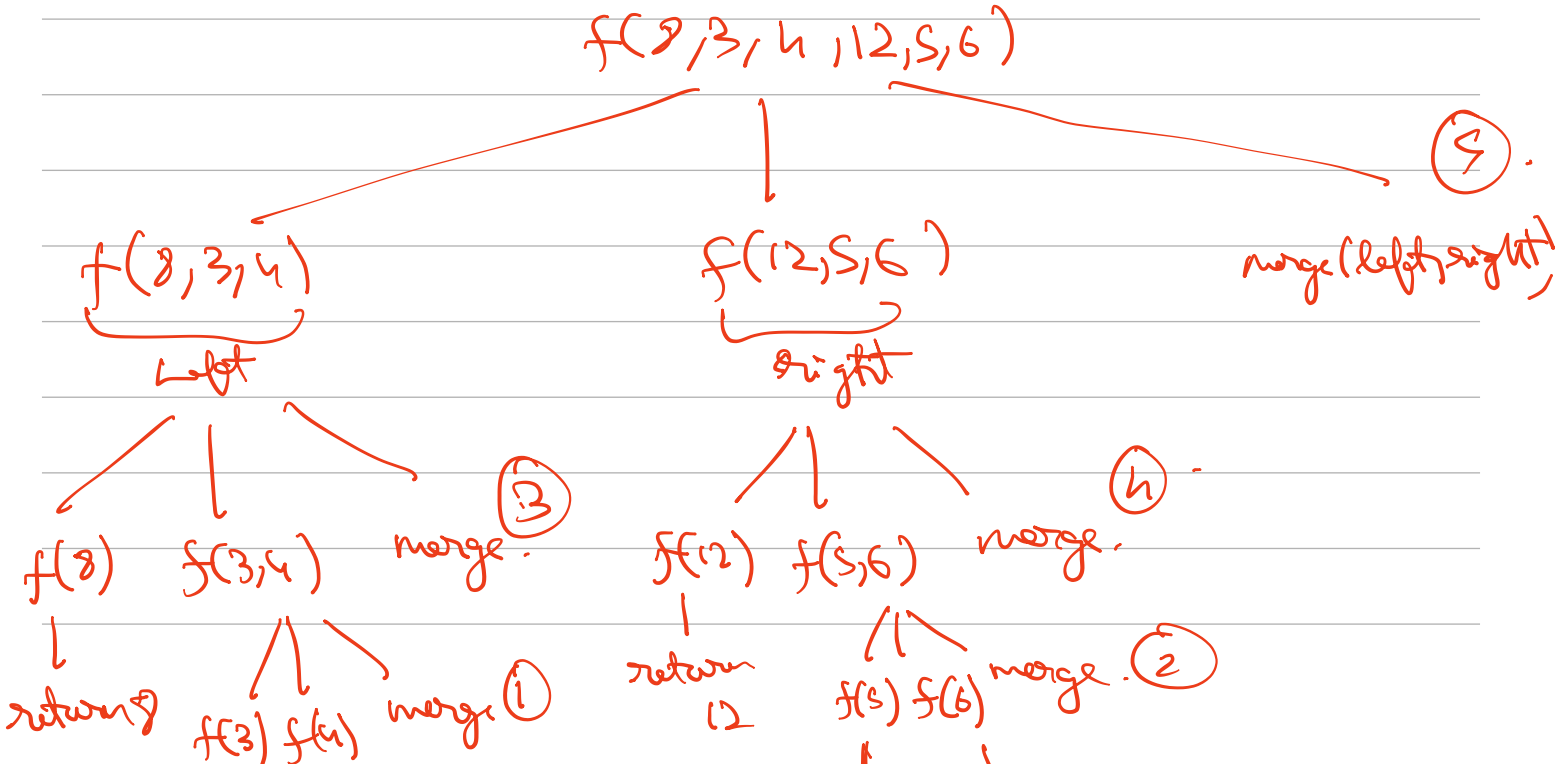
for merging

```java
    // it may be possible that one of the arrays is not complete
    while (i < first.length) {
        mix[k] = first[i];
        i++;
        k++;
    }

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

    return mix;
}
}
```

Recursion tree:-



Steps of Merging from above tree, go from bottom to top. (Left to right).

i) → [3,4] → merge. ① [5,6] → merge. ②

※ Here merge function merges in ascending order

ii) → [8], [3,4] → merge ③.

[3,4,8].

[12], [5,6] → merge ④.

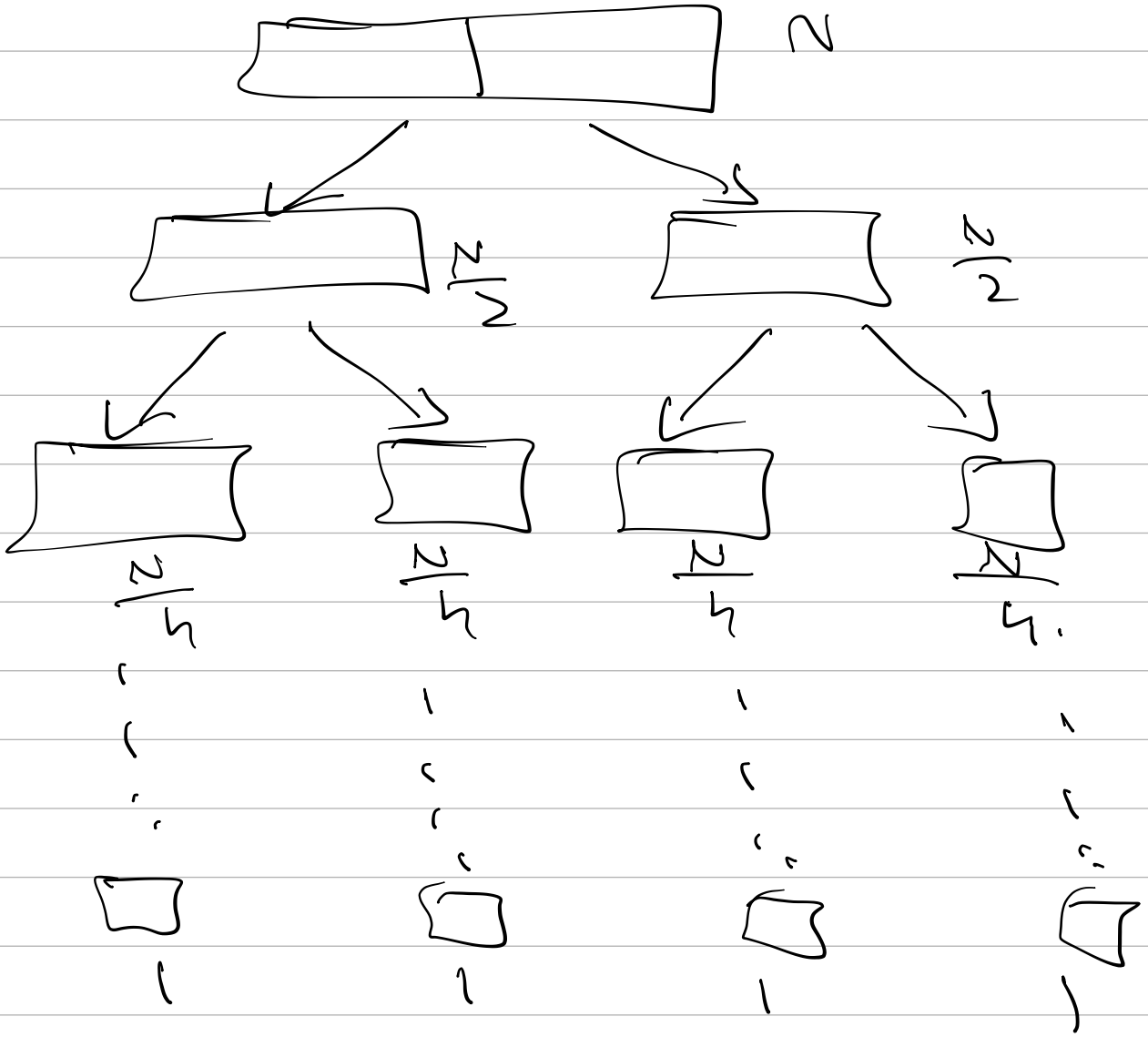
[5,6,12].

iii). [3,4,8], [5,6,12] → merge ⑤.

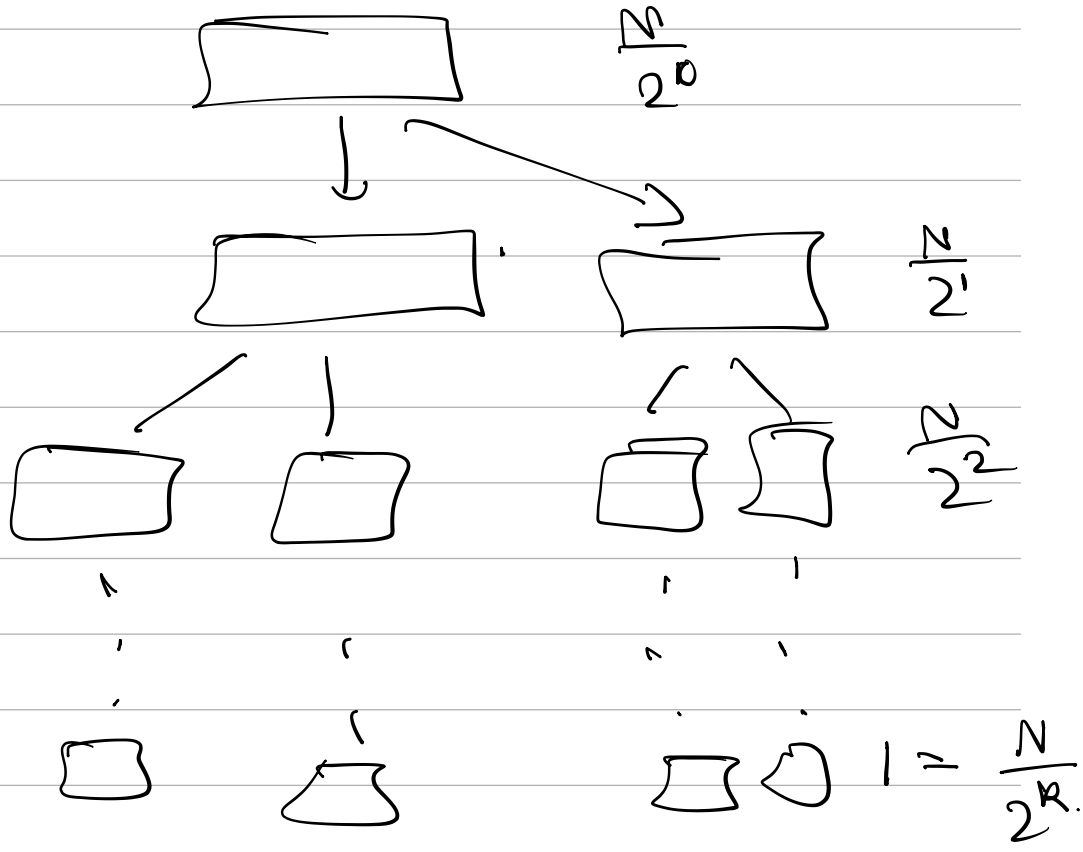
[3, 4, 5, 6, 8, 12] → answer

Time complexity

Graphical approach:-



At every level N, elements are being merged.



$\Rightarrow \quad N = 2^R$

R → total levels.

$\log N = R \log 2.$

$$\log_2 N = R$$

↓

At each level $\log_2 N$ complexity.

So, total N levels are there, so

Time complexity $= N \log_2 N$

Space complexity

'mix' array size of N.

$O(N)$. → Auxiliary memory is used as 'mix' array is temporary array created during merging phase.

RECURRENCE RELATION

$$T(N) = T\left(\frac{N}{2}\right) + T\left(\frac{N}{2}\right) + (N-1)$$

(N-1) → steps for merging

$$= 2T\left(\frac{N}{2}\right) + (N-1).$$

USING AKRA BAAZI FORMULA

Explained in space and Time complexity Notes.

$$\frac{2 \times 1}{2^p} = 1$$

$$\Rightarrow \boxed{p = 1}$$

$$T(N) = x + x \int_1^x \frac{(u-1)}{u^{p+1}} \rightarrow g(u) \ (\text{i.e } N-1)$$

$$\approx x + x \int_{1}^{x} \frac{(u-1)}{u^2}$$

$$\approx x + x \int_{1}^{x} \frac{1}{u} - \frac{1}{u^2}$$

$$\approx x + x \left[\log u - \left(\frac{u^{-2+1}}{-2+1} \right) \right]_{1}^{x}$$

$$\approx x + x \left[\log u + \frac{1}{u} \right]_{1}^{x}$$

$$\approx x + x \left(\log x + \frac{1}{x} - \log 1 - \frac{1}{1} \right)$$

$$\approx x + x \left(\log x + \frac{1}{x} - 1 \right)$$

$$\approx \cancel{x} + x \log x + 1 - \cancel{x}$$

Time complexity $\approx x \log x$

$\Rightarrow N \log N$

In place merge sort

We observed, in above techniques, we were splitting array.

In-place Merge sort, we do not split the array physically, instead we just pass the index values of splitting portion.

Logic:-

s m e

5, 4, 3, 2, 1

0 1 2 3 4 → index

s → start

m → mid

e → end

fun(arr, s, e).

mergesort function → function(arr, 0, 4)

f(arr, 0, 1)
s (m-1)

f(arr, 2, 4)
m e

Merge function Logic :-

fun(arr, s, m, e)

i = s → beginning of first half

j = m → beginning of second half

k = 0.

new array = Size (e - s)

↓

end - start

This will have full size of array.

Code :-

```java
public class MergeSort {
    public static void main(String[] args) {
        int[] arr = {5, 4, 3, 2, 1};
        mergeSortInPlace(arr, s: 0, e: arr.length - 1);
        System.out.println(Arrays.toString(arr));
    }
```

```java
    static void mergeSortInPlace(int[] arr, int s, int e) {
        if (e - s == 1) {
            return;
        }

        int mid = (s + e) / 2;

        mergeSortInPlace(arr, s, mid);
        mergeSortInPlace(arr, mid, e);

        mergeInPlace(arr, s, mid, e);
    }
```

→ Here this condition will return the function when arr = [s, e] → 2 elements will be there.

```java
private static int[] mergeInPlace(int[] arr, int s, int m, int e) {
    int[] mix = new int[e - s];

    int i = s;
    int j = m;
    int k = 0;

    while (i < m && j < e) {
        if (arr[i] < arr[j]) {
            mix[k] = arr[i];
            i++;
        } else {
            mix[k] = arr[j];
            j++;
        }
        k++;
    }
```

```java
    // It may be possible that one of the arrays is not complete
    // copy the remaining elements
    while (i < m) {
        mix[k] = arr[i];
        i++;
        k++;
    }

    while (j < e) {
        mix[k] = arr[j];
        j++;
        k++;
    }

    for (int l = 0; l < mix.length; l++) {
        arr[s+l] = mix[l];
    }
}
}
```

This is necessary for copying merged subarrays to original array (arr).

s → ensures that, the element from 'mix' array is copied to correct location to main array 'arr'.

★). Time Complexity is same $O(n \log n)$

★). Space complexity is reduced to

$O(1)$ → as no auxiliary array is created in in-place mergesort