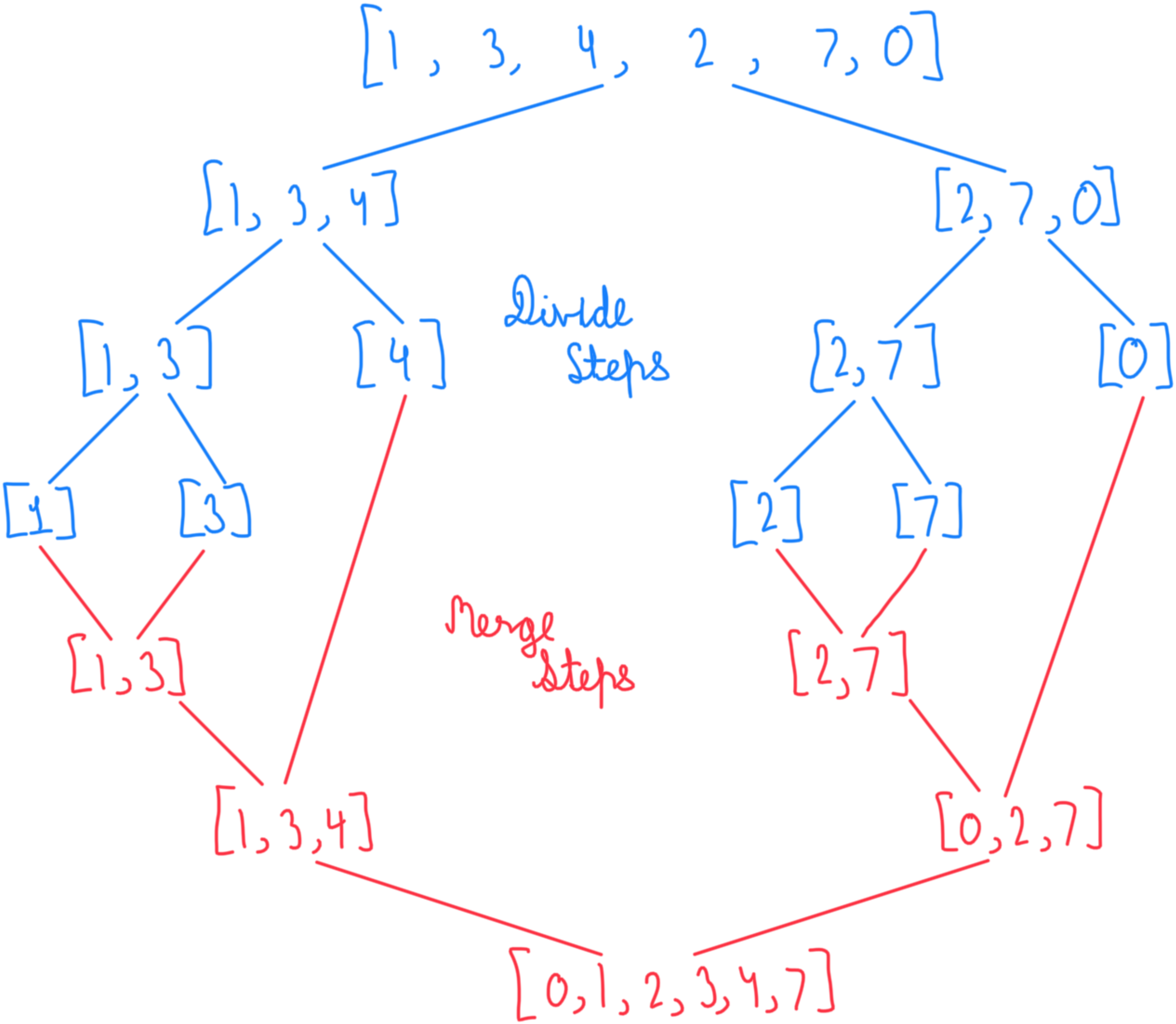
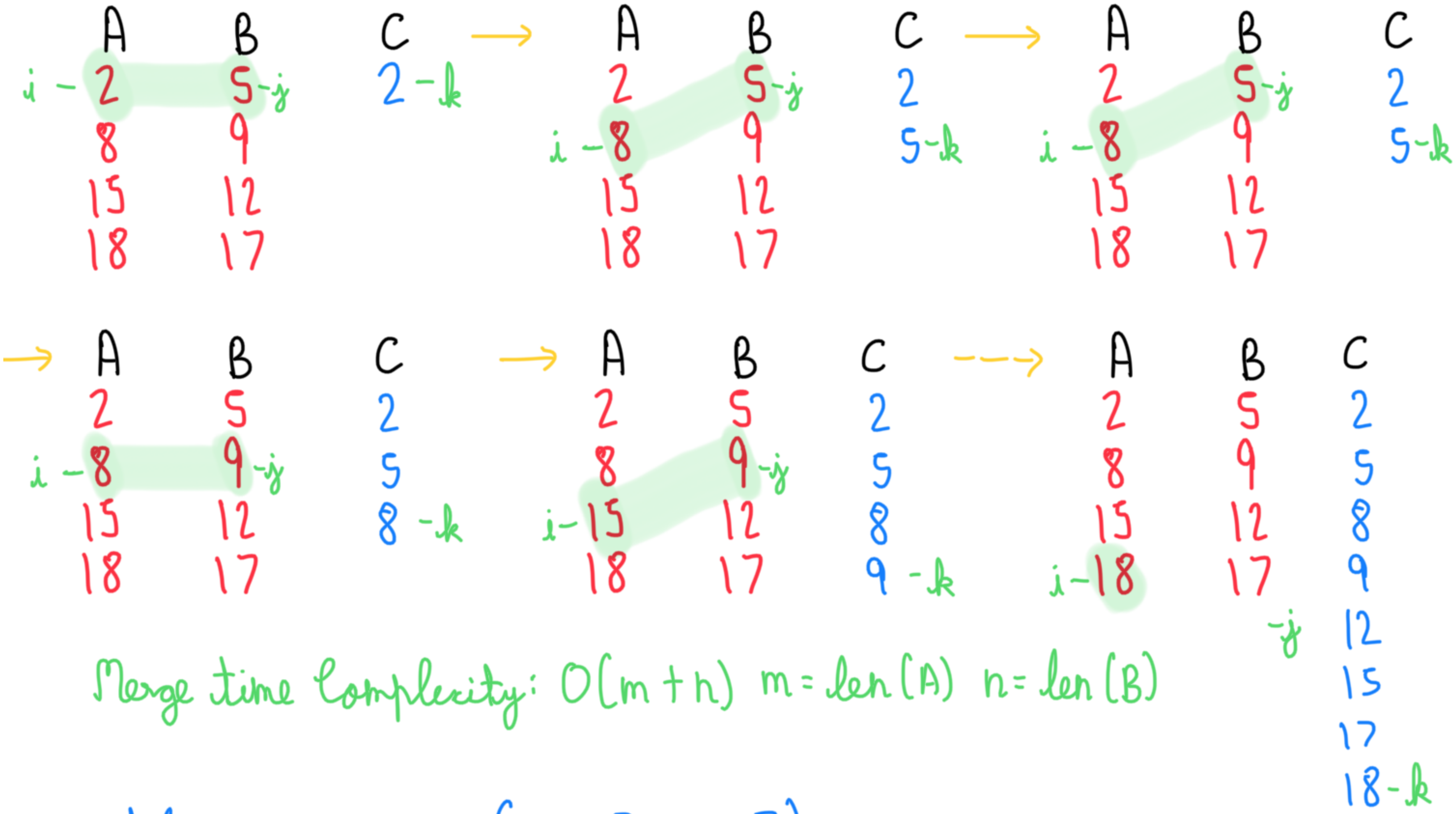
4 Merge Sort


[1, 3, 4, 2, 7, 0]
[1, 3, 4]
[2, 7, 0]
Divide Steps
[1, 3]
[4]
[2, 7]
[0]
[1]
[3]
[2]
[7]
Merge Steps
[1, 3]
[2, 7]
[1, 3, 4]
[0, 2, 7]
[0, 1, 2, 3, 4, 7]

Merge Algorithm of 2 sorted lists:



Merge time Complexity: O(m + n) m = len(A) n = len(B)

```
def merge_sort(arr[lo...hi]):

    if (len(arr) <= 1):  }b
        return arr
    mid = $\lfloor (lo+hi)/2 \rfloor$ —— $c$

    A = merge_sort(arr[lo... mid]) — $T(n/2)$
    B = merge_sort(arr[mid+1...hi]) — $T(n/2)$
    return merge(A, B) —— $O(n)$

def merge(A[i... $n_1$], B[j... $n_2$]):

    merged = []

    while i <= $n_1$ and j <= $n_2$:

        if A[i] <= B[j]
            merged.append(A[i])
            i += 1
        else:
            merged.append(B[j])
            j += 1

    return merged + A[i:] + B[j:]
```

Time:

$$T(n) = \begin{cases} b & \text{, if } n=1 \\ 2T(n/2) + n + c & \text{, if } n>1 \end{cases} \qquad T(n) = O(n \log n)$$

Best/Worst-Case: $O(n \log n)$

Space: merge list + call stack = $n + \log n = O(n)$

In-place: No, aux space is not $O(1)$ and algo. is recursive.

Stable: Yes, the merge step makes sure that if item in left sub-array is <= the item in right sub-array, add the item from left sub-array to the merged sorted list. This maintains the relative order of elements with the same value.

Online: No, the merge sort need the entire input array as it needs to divide it in equal parts.

Code:

```
def mergeSort(arr):
    if len(arr) <= 1:
        return arr

    mid = len(arr)//2

    A = mergeSort(arr[:mid])
    B = mergeSort(arr[mid:])
    return merge(A, B )



def merge(A, B):
    i, j = 0, 0
    merged = []

    while i < len(A) and j < len(B):
        if A[ i ] <= B[ j ]:
            merged.append(A[ i ])
            i+=1
        else:
            merged.append(B[ j ])
            j+=1

    return merged + A[i:] + B[j:]
```