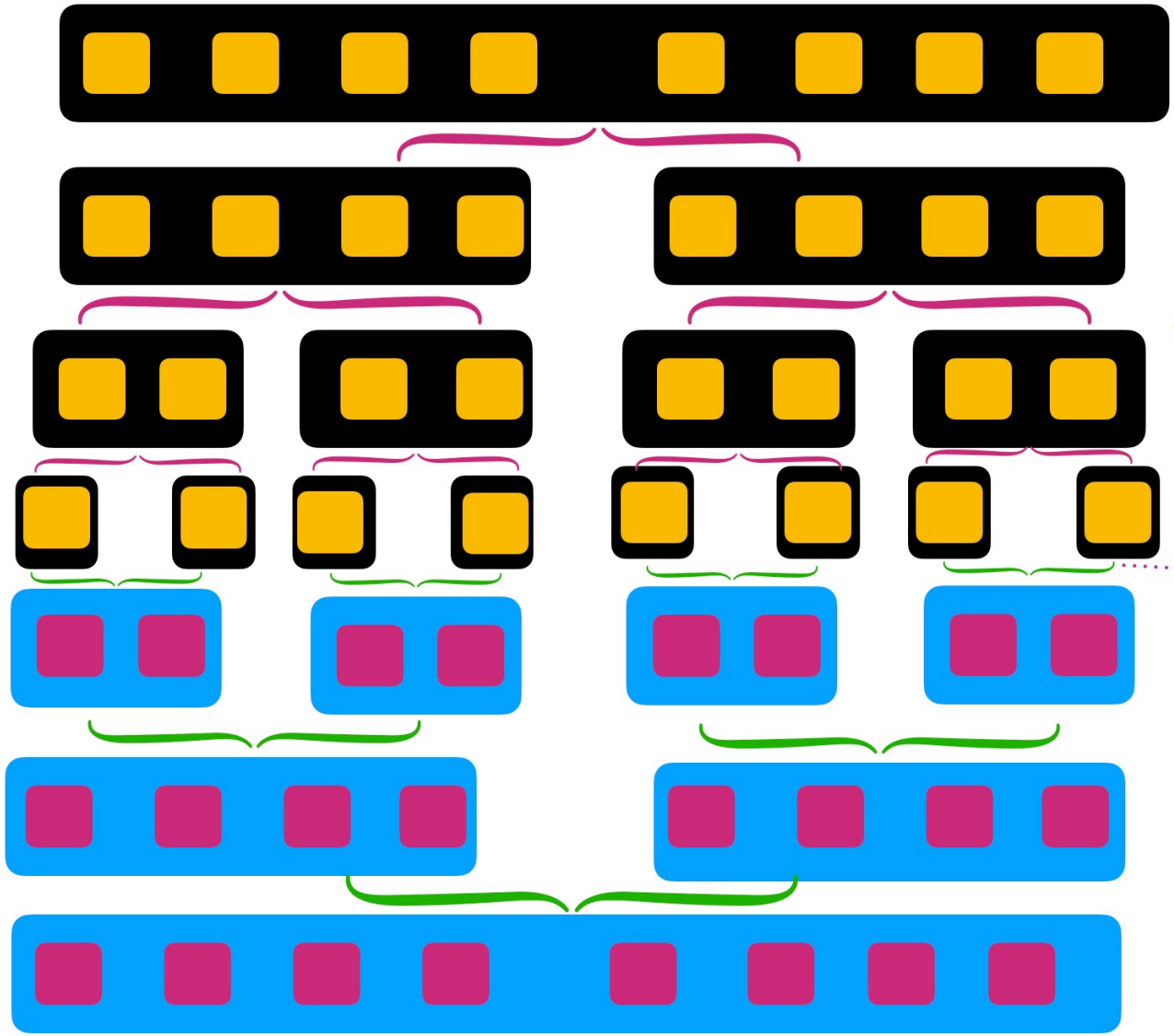
Merge Sort uses divide and conquer pattern.

Merge sort divides the problem into possible small problems then applies sorting recursively.

Divide => divides source collection into possible n/2 sub problems recursively.

Conquer=> Applies the sorting at subproblem level (compare, swap & merge) then repeats recursively.



Divide

Divide=> Break up the problem into smallest possible sub problems.

Compare, Swap & Merge

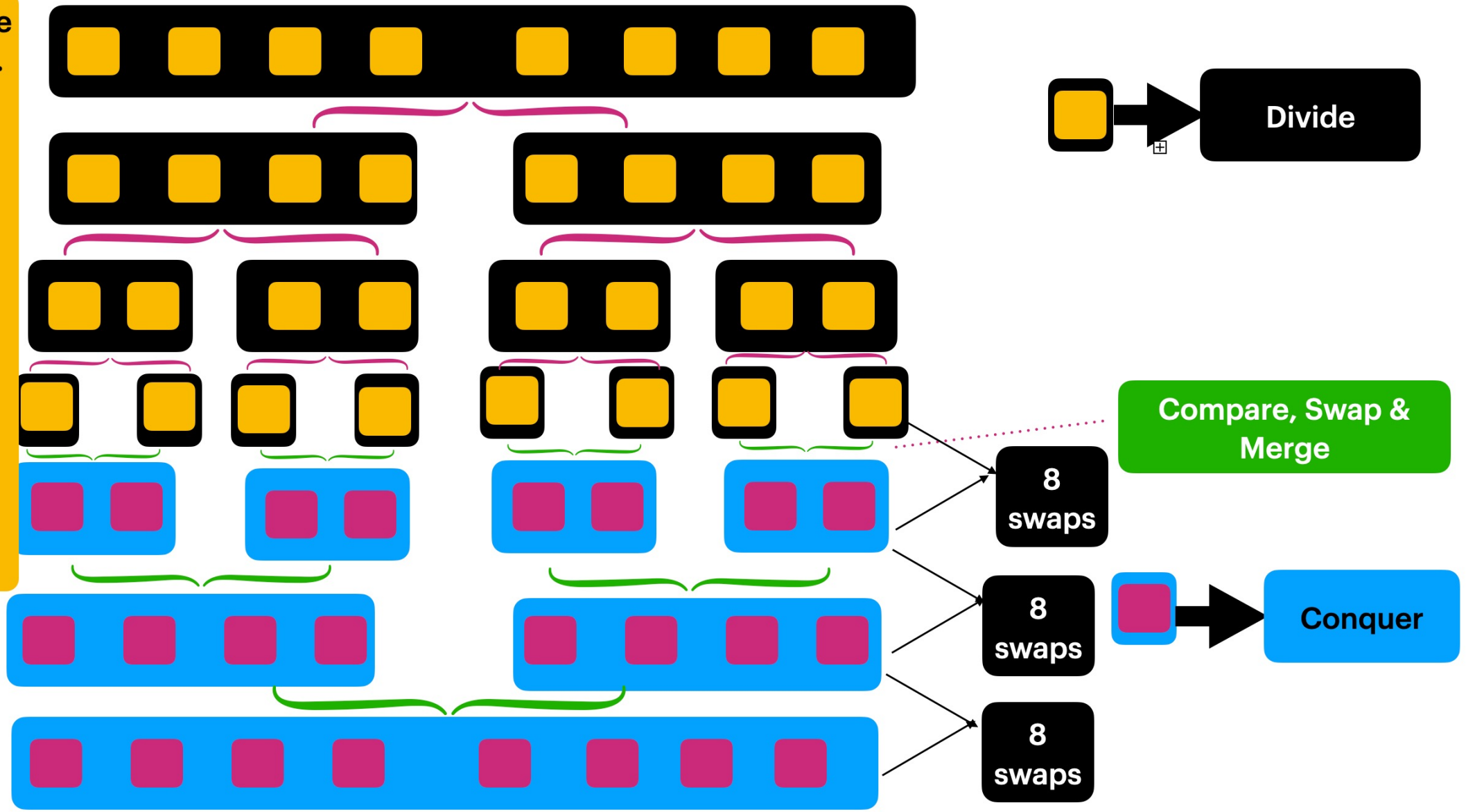
Conquer

Conquer=> Figure out the solution for the smallest sub problem, then apply the same technique to solve larger problems recursively .

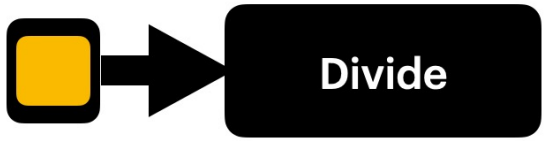
Let's figure out the Time Complexity.

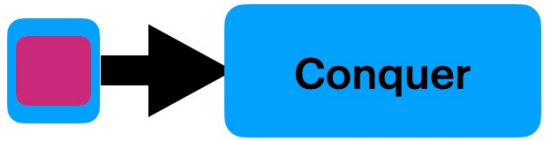
For a merge sort the complex operations happens while swapping (Conquer logic).

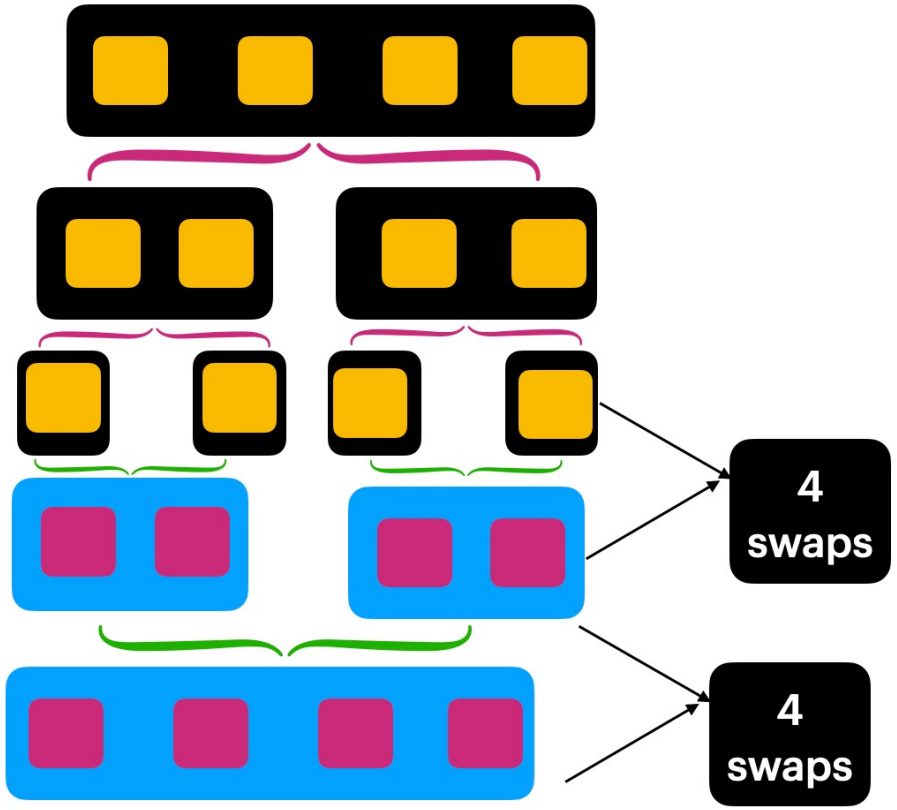
Let's divide solution for time complexity by considering it.

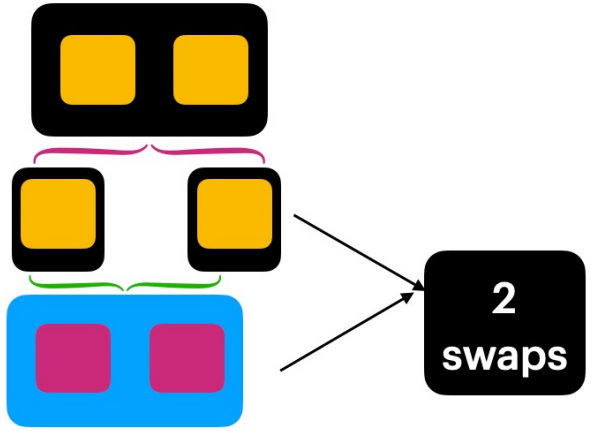


Merge Sort taken 24 swaps If the size of the array is 8.
SizeOf(8) => 8swaps in 3steps = 8*3 = 24 swaps









For SizeOf(2) => 2*1 = 2 swaps

Merge Sort taken 8 swaps If the size of the array is 4.
SizeOf(4) => 4swaps in 2steps = 4*2 = 8 swaps

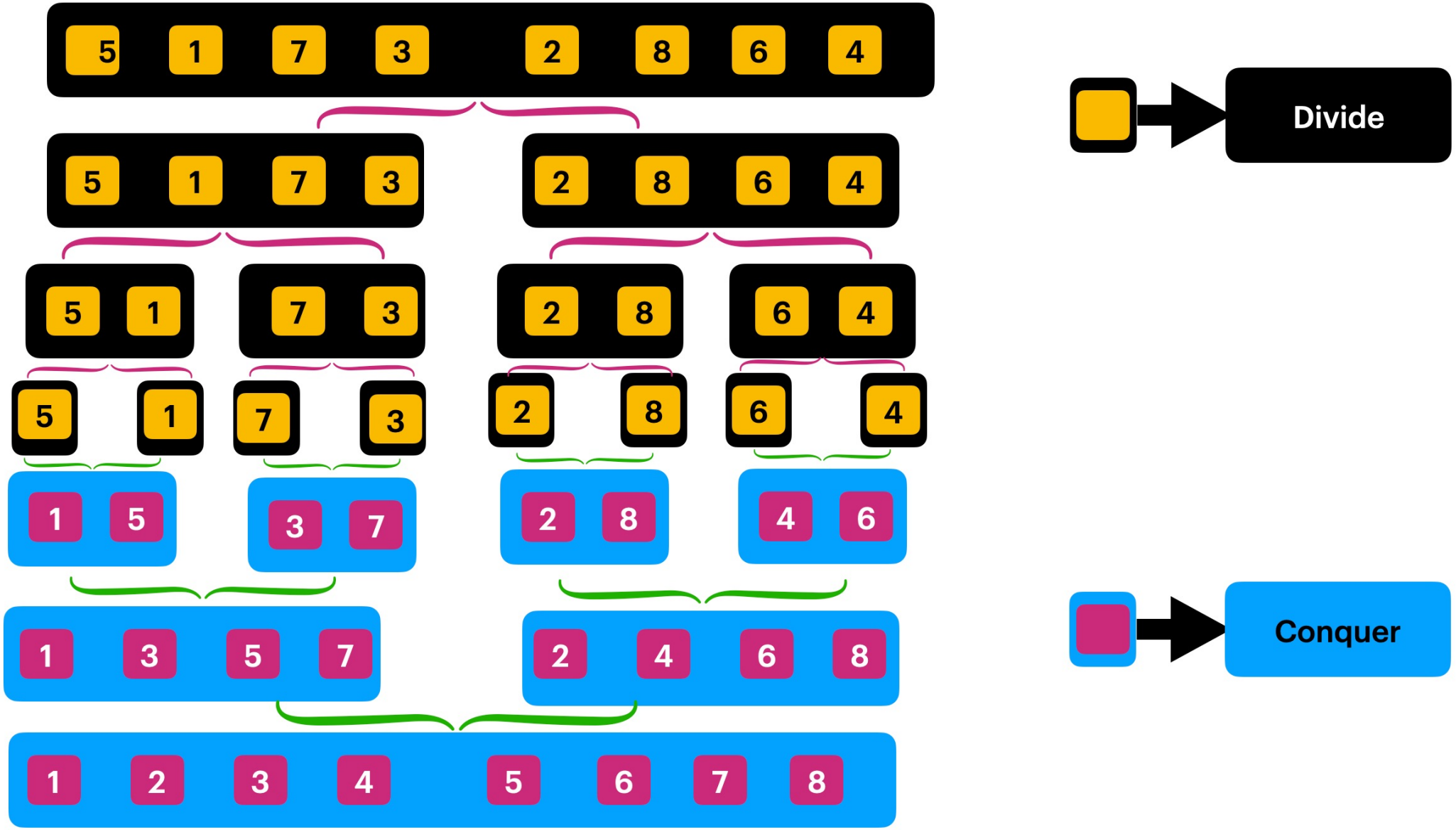
MergeSort :

For Size Of (2) = 2 => 2 * 1 = 2 * log2^1
For Size Of (4) = 8 => 4 * 2 = 4 * log2^2
For Size Of (8) = 24 => 8 * 3 = 8 * log2^3
Finally for a Merge Sort we can derive a time complexity as nlogn.

Time Complexity = O(nlogn)
Space Complexity = O(n)
Recursive / Non Recursive = Recursive
Stability = Stable

{4,1,4,3}
{4,1} {4,3}
{4} {1} {4} {3}
{1,4} {3,4}
{1,3,4,4}

Internal /Eternal Sort = Can be used for both.
Comparison Sort = Yes
Swap = O(nlögn)


5 1 7 3 2 8 6 4
5 1 7 3
2 8 6 4
5 1
7 3
2 8
6 4
5
1
7
3
2
8
6
4
1 5
3 7
2 8
4 6
1 3 5 7
2 4 6 8
1 2 3 4 5 6 7 8
Divide
Conquer