**Day 29**

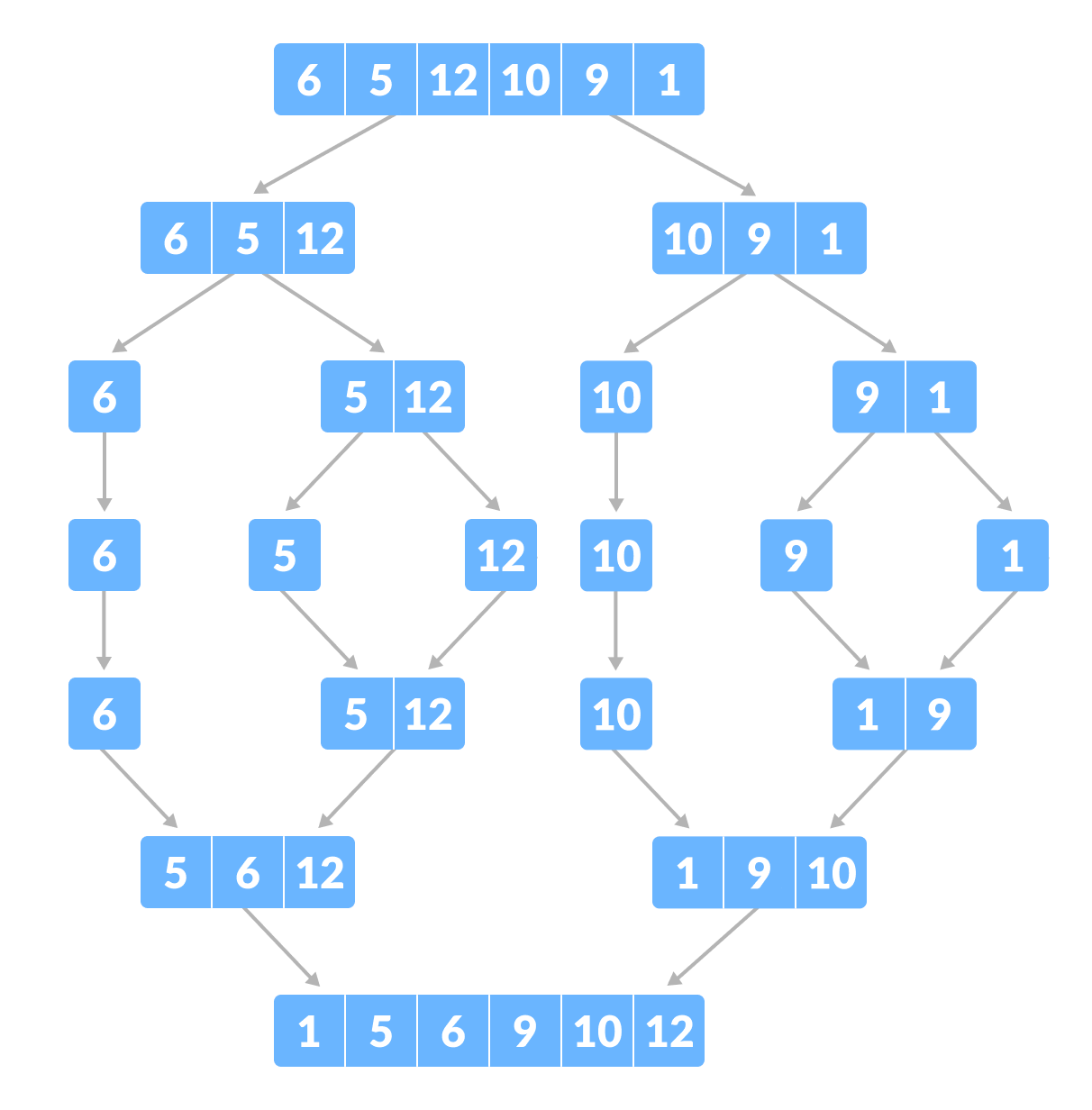
**Merge Sort**

■ Merge Sort is one of the most popular sorting algorithms that is based on the principle of Divide and Conquer Algorithm.

■ Here, a problem is divided into multiple sub-problems. Each sub-problem is solved

individually.

■ Finally, sub-problems are combined to form the final solution.



**Divide and Conquer Strategy**

■ Using the Divide and Conquer technique, we divide a problem into subproblems.

■ When the solution to each subproblem is ready, we 'combine' the results from the subproblems to solve the main problem.

■ Suppose we had to sort an array A.

■ A subproblem would be to sort a sub-section of this array starting at index p and ending at index r, denoted as A[p..r].

■ Divide

If q is the half-way point between p and r, then we can split the subarray A[p..r] into two arrays A[p..q] and A[q+1, r].

■ Conquer

In the conquer step, we try to sort both the subarrays A[p..q] and A[q+1, r].

If we haven't yet reached the base case, we again divide both these subarrays and try to

sort them.

Combine

■ When the conquer step reaches the base step and we get two sorted subarrays A[p..q] and A[q+1, r] for array A[p..r], we combine the results by creating a sorted array A[p..r] from two sorted subarrays A[p..q] and A[q+1, r].

**MergeSort Algorithm**

■ The MergeSort function repeatedly divides the array into two halves until we reach a stage where we try to perform MergeSort on a subarray of size 1 i.e. p == r.

■ After that, the merge function comes into play and combines the sorted arrays into

larger arrays until the whole array is merged.

MergeSort(A, p, r): if p > r

return

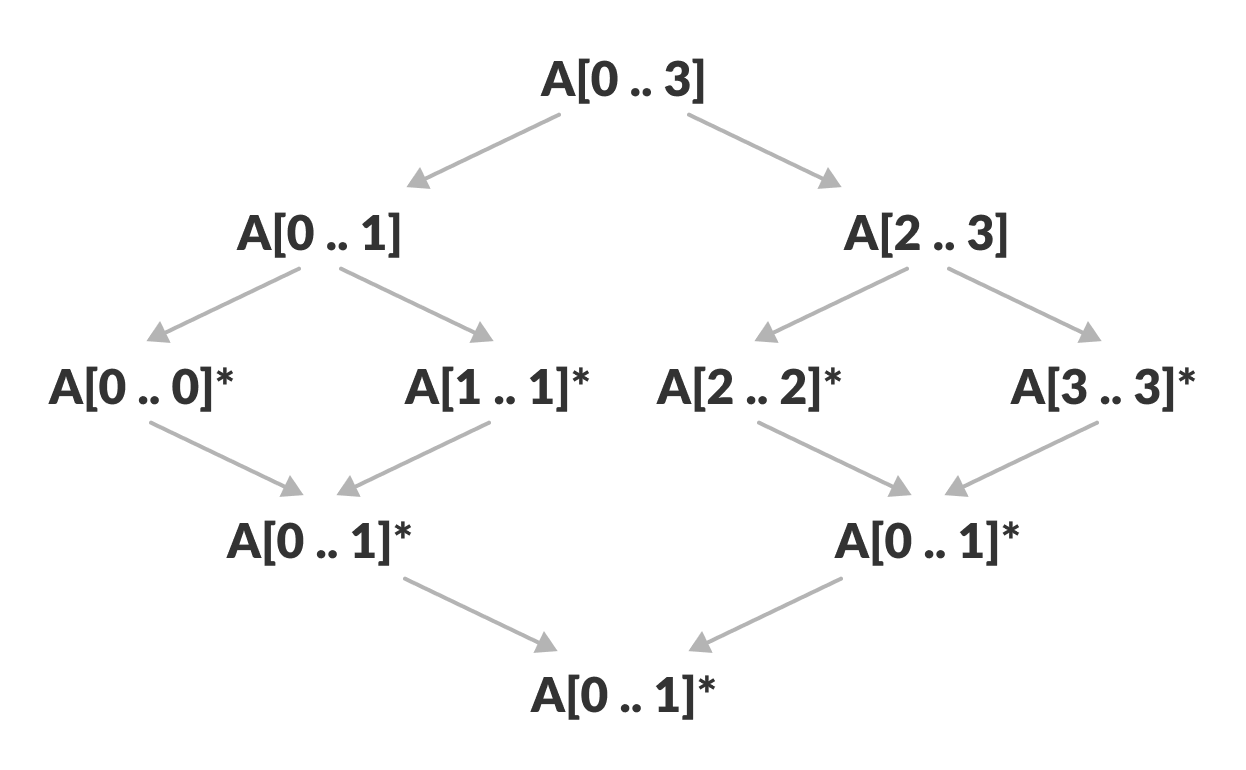
q = (p+r)/2 mergeSort(A, p, q) mergeSort(A, q+1, r) merge(A, p, q, r)

■ To sort an entire array, we need to call MergeSort(A, 0, length(A)-1).

■ As shown in the image below, the merge sort algorithm recursively divides the array

into halves until we reach the base case of array with 1 element.

■ After that, the merge function picks up the sorted sub-arrays and merges them to gradually sort the entire array.



**The merge Step of Merge Sort**

■ Every recursive algorithm is dependent on a base case and the ability to combine the results from base cases.

■ Merge sort is no different.

■ The most important part of the merge sort algorithm is, you guessed it, merge step.

■ The merge step is the solution to the simple problem of merging two sorted lists(arrays) to build one large sorted list(array).

■ The algorithm maintains three pointers, one for each of the two arrays and one for

maintaining the current index of the final sorted array.

Have we reached the end of any of the arrays?

No:

Compare current elements of both arrays

Copy smaller element into sorted array

Move pointer of element containing smaller element Yes:

Copy all remaining elements of non-empty array

M**erge Sort Complexity**

Time Complexity

■ Best Case Complexity: O(n\*log n)

■ Worst Case Complexity: O(n\*log n)

■ Average Case Complexity: O(n\*log n)

Space Complexity

■ The space complexity of merge sort is O(n).

**Merge Sort Applications**

• Inversion count problem

• External sorting

• E-commerce applications