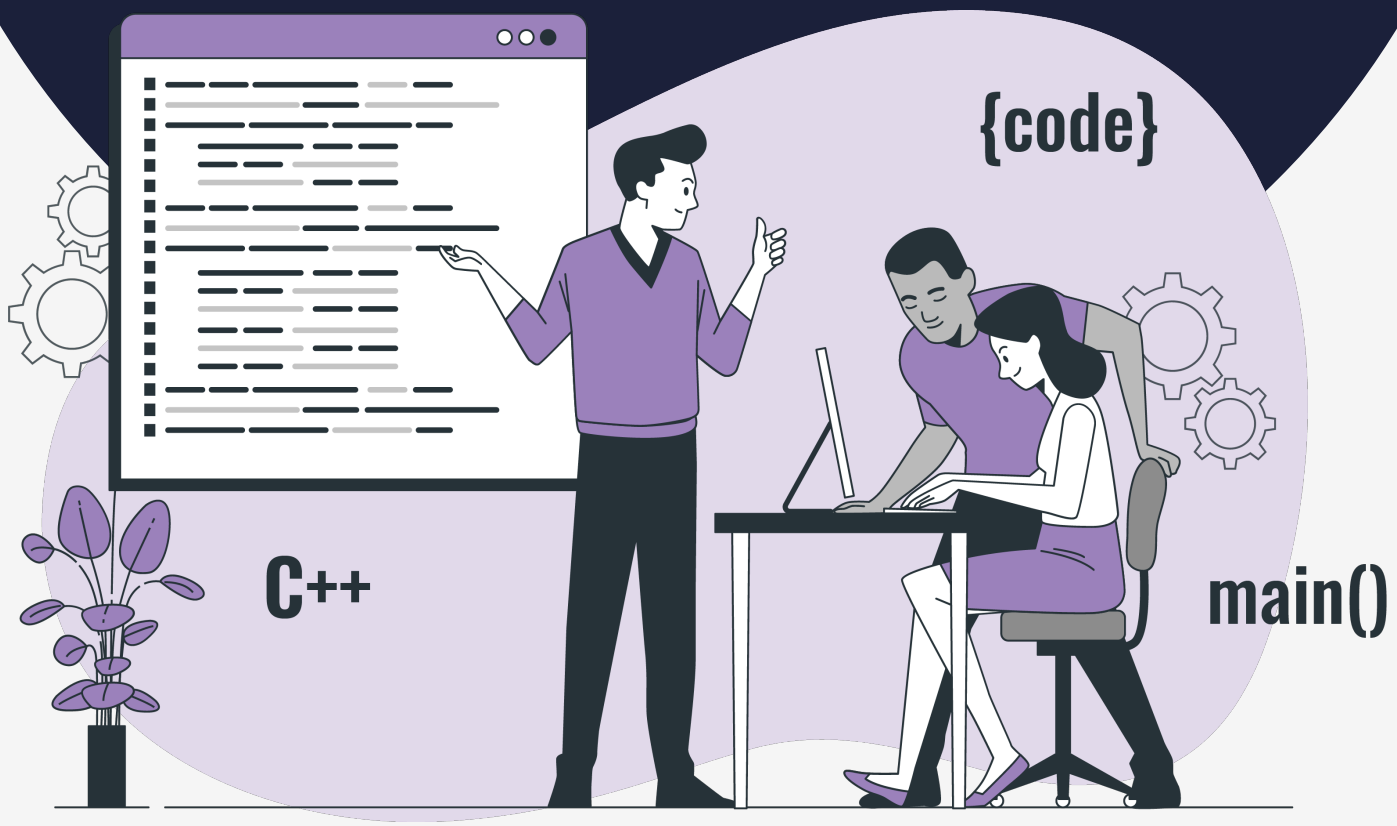
Lesson:

Merge Sort



Pre-Requisites

- Functions
- Loops
- Arrays
- Recursion

List of Concepts Involved

- Merge Sort Algorithm
- Merge Sort Time Complexity
- Merge Sort Space Complexity
- Worst case scenario in Merge Sort
- How to optimize the Merge sort in the case of nearly sorted arrays?

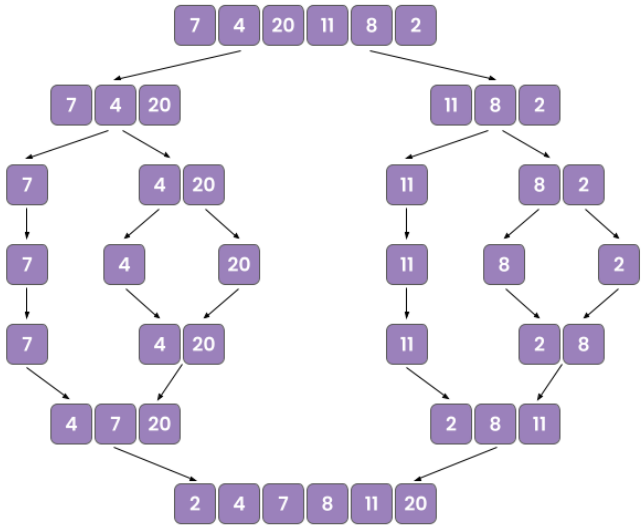
As we are now familiar with the concept of Bubble sort , Selection sort and Insertion sort from the previous lessons, we will now expand our knowledge further to learn merge sorting algorithm.

Topic: Merge Sort Algorithm

This algorithm is a sorting algorithm that is based on the **Divide and Conquer** paradigm. In this algorithm, the array is initially divided into two equal halves and then they are merged in a sorted manner.. We have to define the function to perform the merging.

The sub-arrays are divided again and again into halves until the array cannot be divided further. Then we combine the pair of one element array into two-element array, sorting them in the process. The sorted two-element pairs are merged into the four-element array, and so on until we get the sorted array.

Here, a problem is divided into multiple sub-problems. Each sub-problem is solved individually. Finally, sub-problems are merged to form the final solution.



Now, let's see the algorithm of merge sort.

To understand the working of the merge sort algorithm, let's take an unsorted array. It will be easier to understand the merge sort via an example.

Let the elements of array are -

According to the merge sort algorithm, the first array is divided into two equal halves. Merge sort keeps dividing the array into equal parts until it cannot be further divided(That means, the size of the array is reduced to 1).

As there are eight elements in the given array, it is divided into two arrays of size 4.

Now, again divide these two arrays into halves. As they are of size 4, divide them into new arrays of size 2.

Now, again divide these arrays to get the atomic value that cannot be further divided.

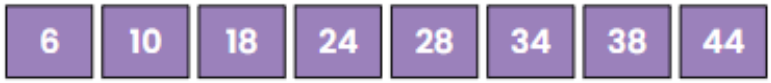
Now, combine them in the same manner they were broken by taking into consideration the sorted order.

In combining, first compare the elements of each array and then combine them into another array in sorted order.

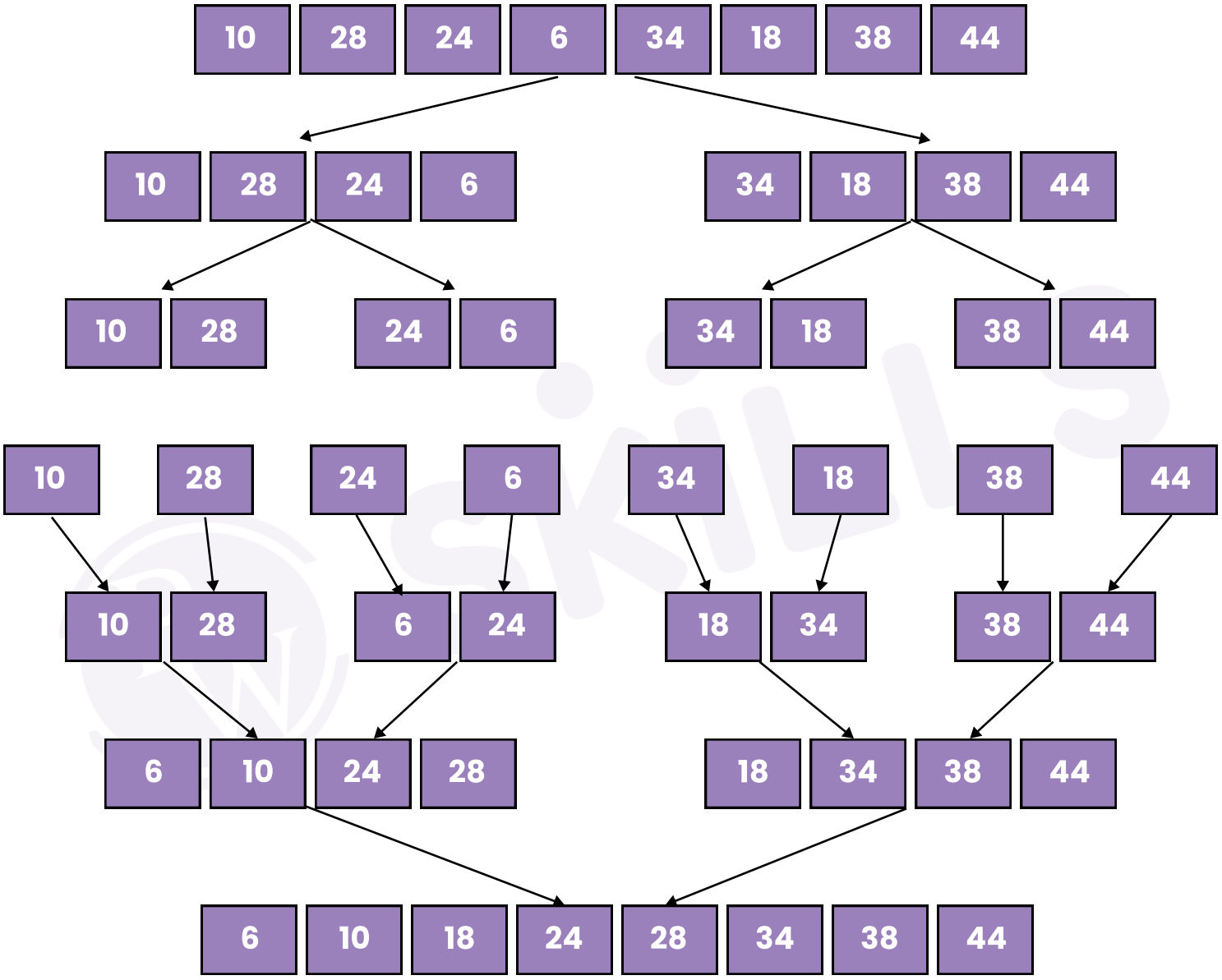
So, first compare 10 and 28, both are in sorted positions. Then compare 24 and 6, and in the list of two values, put 6 first followed by 24. Then compare 34 and 18, sort them and put 18 first followed by 34. After that, compare 38 and 44, and place them sequentially.

In the next iteration of combining, now compare the arrays with two data values and merge them into an array of found values in sorted order.

Now, there is a final merging of the arrays. After the final merging of above arrays, the array will look like -



Now, the array is completely sorted.



Algorithm:

step 1: start

step 2: declare an array and left, right, mid variable

step 3: perform merge function.
if left > right
return
mid= (left+right)/2
mergesort(array, left, mid)
mergesort(array, mid+1, right)
merge(array, left, mid, right)

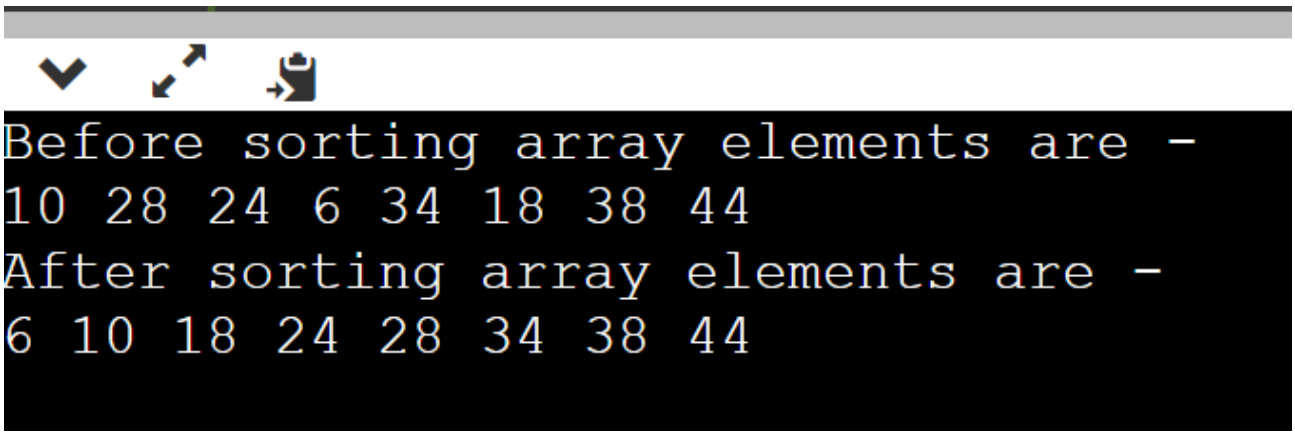
step 4: Stop
Follow the steps below the solve the problem:
MergeSort(arr[], l, r)

If r > l

- Find the middle point to divide the array into two halves:
  - middle mid = l + (r – l)/2
- Call mergeSort for first half:
  - Call mergeSort(arr, l, mid)
- Call mergeSort for second half:
  - Call mergeSort(arr, mid + 1, r)
- Merge the two halves sorted in steps 2 and 3:
  - Call merge(arr, l, mid, r)

Code link for above approach:

https://pastebin.com/Ws71DNBn



Topic - Merge Sort Time Complexity:

Now, let us calculate time complexity with the steps. our very own first part was to divide the input into two halves until no further division is possible, which comprised us of a logarithmic time complexity ie. log(N) where N is the number of elements in the array as there will be log(N) divisions.

Our second part was to merge back the array into a single array, so if we observe it in all the number of elements to be merged N, and to merge back we use a simple loop which runs over all the N elements giving a time complexity of O(N).

finally, total time complexity will be - step -1 + step-2
$T(n) = 2T(n/2) + O(n)$

The solution of the above recurrence can be written as O(nLogn).

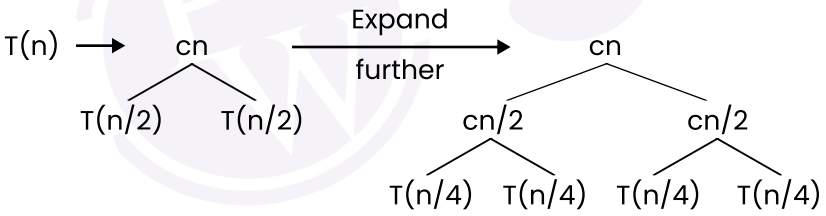
The array of size N is divided Logn times, and so basically there can be at max Logn level and each level total number of elements are N so total time for merging all the parts at a particular level will be N and as there are Logn levels so total time complexity will be O(NlogN .

The time complexity of MergeSort is O(n*Log n) in all the 3 cases (worst, average and best) as the mergesort always divides the array into two halves and takes linear time to merge two halves.

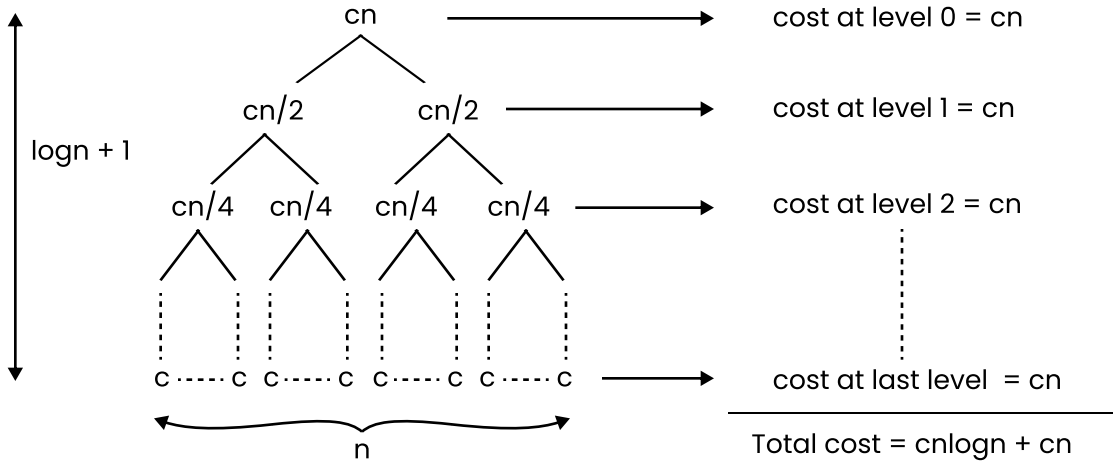
Recurrence relation of the merge sort

$$T(n) = \begin{cases} c, \text{ if } n = 1 \\ 2T(n/2)+cn, \text{ if } n>1 \end{cases}$$

Let's draw the recurrison tree

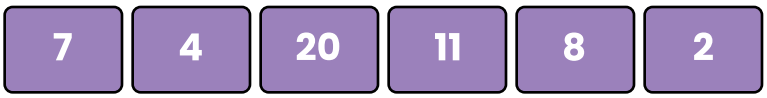


So on..... here is the complete recurrison tree diagram



Topic - Merge Sort Space Complexity:

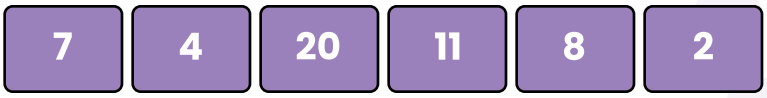
Let us take this example again.



Once we call merge sort for the entire array (N = 6), the array is divided into two parts, (each of size N / 2 = 3).

However, we must note that function calls are **not** running in parallel. Everytime during the **divide phase**, we are making a single function call, first with the left part and then waiting for its return (with sorted array in place) to call for the right part.

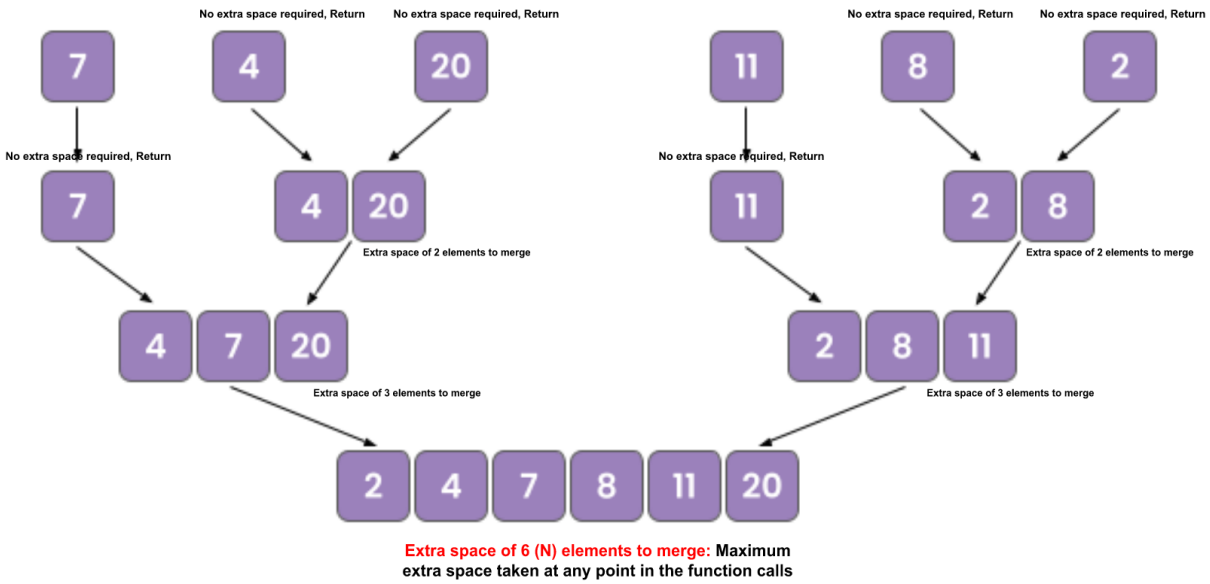
The same happens for the next function calls, till we get a single element, in which case we return.



So, the first time we encounter a single element in a function call, it would be as follows:

At any point, in the call stack, we would be having a maximum of **O(logN)** function calls.

Now, once we are returned from the left and the right call, the **merge phase** for that function call would begin. So, if the size of the subarray for that function call is N with its left and right part of size N/2, we would be creating a new array of size N to do the merge. So, to merge the arrays: The extra space that we will be needing is as follows:



Also note that once we return from any function, the extra space we take to merge the two sorted subarrays is also freed. So, we need not sum up the extra space taken in each function call since maximum space taken at any point is something we are concerned about.

Hence, during the divide phase, the maximum space we take in the recursion call stack is **O(logN)** and while merging, out of all the function calls, the maximum space we take is **O(N)** when we merge two subarrays of the **entire array**. Since **O(N)** is dominating, we would consider **O(N)** to be the space complexity of the merge sort.

Apart from the intuition, the space complexity can be found as:

Let S(N) denote the amount of extra space required for a function sorting array of size N. We need to calculate the recursion call stack space and the extra space we use for merging at each function call. In general terms for sorting an array of size N:

S(1) = O(logN) (recursive stack space)
S(2) = O(logN - 1 (recursive stack space) + 2 (Number of elements))
S(4) = O(logN - 2 (recursive stack space) + 4 (Number of elements))
S(8) = O(logN - 3 (recursive stack space) + 8 (Number of elements))
...
So, for sorting an array of N elements, S(N) = O(N (Number of elements)), since after returning from left and right, the recursion stack would contain just a single function call, which would be the function call for the original array.

Topic - Is Merge Sort Stable?

Yes, Merge Sort is a stable sorting algorithm which means that the elements with same value in an array maintain their original positions with respect to each other.

Topic : Applications of merge sort

- More efficient and works fast in case of large Data sets.
- It is the best Sorting technique used for sorting Linked Lists.(will be explained in coming lectures)

Topic : Drawbacks of Merge Sort:

- Slower compared to the other sort algorithms for smaller tasks.
- This algorithm requires an additional memory space of 0(n) for the temporary array.
- It goes through the whole process even if the array is sorted.

Upcoming Class Teasers

- Quick sort