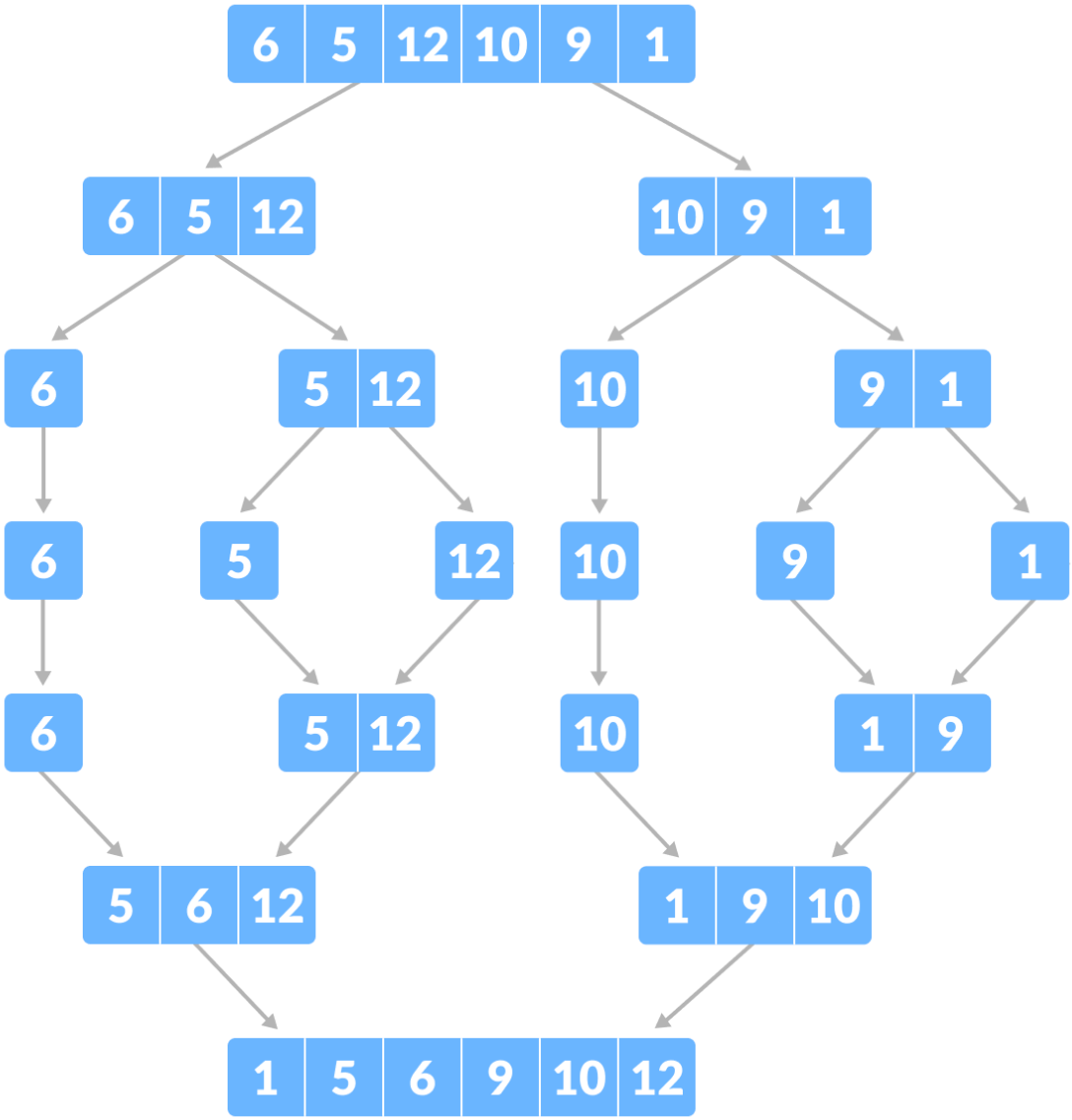
Merge Sort Algorithm

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.



Merge Sort example

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.

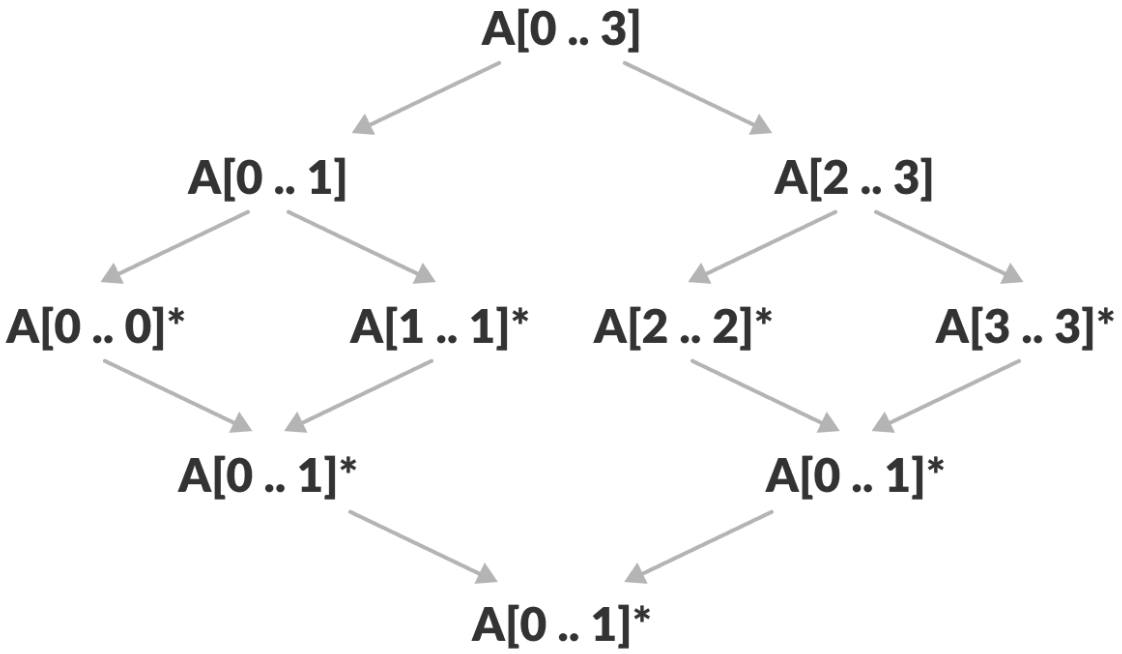
MergeSort Algorithm

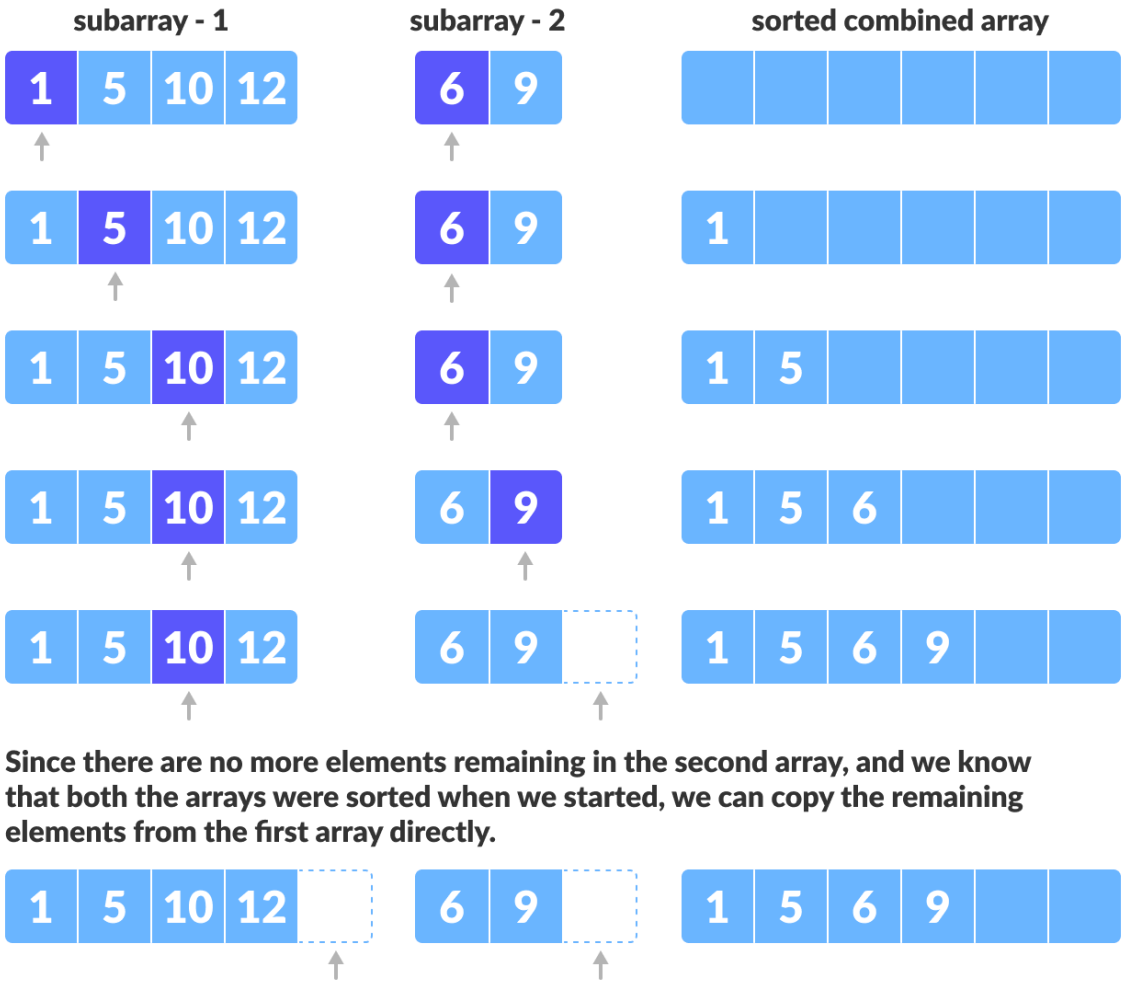
The MergeSort function repeatedly divides the array into two halves until we reach a stage where we try to perform MergeSort on a subarray of size 1 i.e. `p == r`.

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)`.


A[0 .. 3]
A[0 .. 1]
A[2 .. 3]
A[0 .. 0]*
A[1 .. 1]*
A[2 .. 2]*
A[3 .. 3]*
A[0 .. 1]*
A[0 .. 1]*
A[0 .. 1]*



Writing the Code for Merge Algorithm

Our task is to merge two subarrays `A[p..q]` and `A[q+1..r]` to create a sorted array `A[p..r]`.

So, the inputs to the function are A, p, q and r
The merge function works as follows:

1. Create copies of the subarrays `L <- A[p..q]` and `M <- A[q+1..r]`.

2. Create three pointers `i`, `j` and `k`
    a. `i` maintains current index of `L`, starting at 1
    b. `j` maintains current index of `M`, starting at 1
    c. `k` maintains the current index of `A[p..q]`, starting at `p`.

3. Until we reach the end of either `L` or `M`, pick the larger among the elements from `L` and `M` and place them in the correct position at `A[p..q]`

4. When we run out of elements in either `L` or `M`, pick up the remaining elements and put in `A[p..q]`

Merge( ) Function Explained Step-By-Step

A lot is happening in this function, so let's take an example to see how this would work.

As usual, a picture speaks a thousand words.

Merging two consecutive subarrays of array

The array **A[0..5]** contains two sorted subarrays **A[0..3]** and **A[4..5].** Let us see how the merge function will merge the two arrays.

```
void merge(int arr[], int p, int q, int r) {
// Here, p = 0, q = 4, r = 6 (size of array)
```

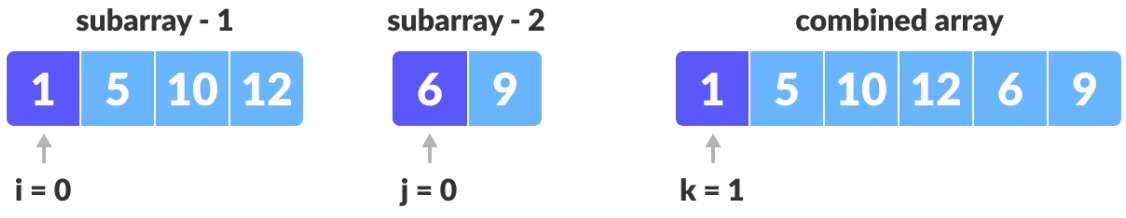
Step 1: Create duplicate copies of sub-arrays to be sorted

```
  // Create L ← A[p..q] and M ← A[q+1..r]
  int n1 = q - p + 1 = 3 - 0 + 1 = 4;
  int n2 = r - q = 5 - 3 = 2;

  int L[4], M[2];

  for (int i = 0; i < 4; i++)
     L[i] = arr[p + i];
     // L[0,1,2,3] = A[0,1,2,3] = [1,5,10,12]

  for (int j = 0; j < 2; j++)
     M[j] = arr[q + 1 + j];
     // M[0,1] = A[4,5] = [6,9]
```

Create copies of subarrays for merging

Step 2: Maintain current index of sub-arrays and main array

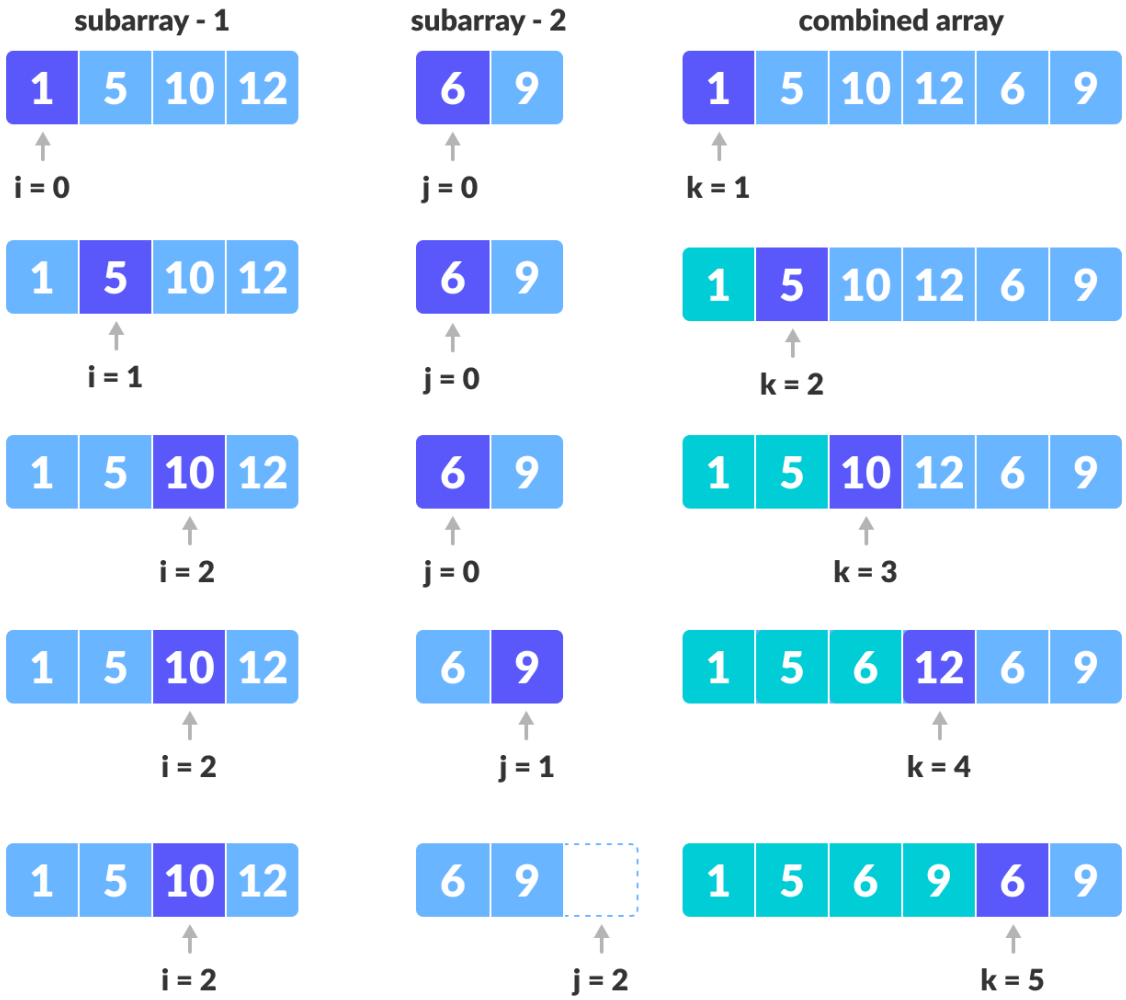
```
int i, j, k;
   i = 0;
   j = 0;
   k = p;
```



Maintain indices of copies of sub array and main array

Step 3: Until we reach the end of either L or M, pick larger among elements L and M and place them in the correct position at A[p..r]

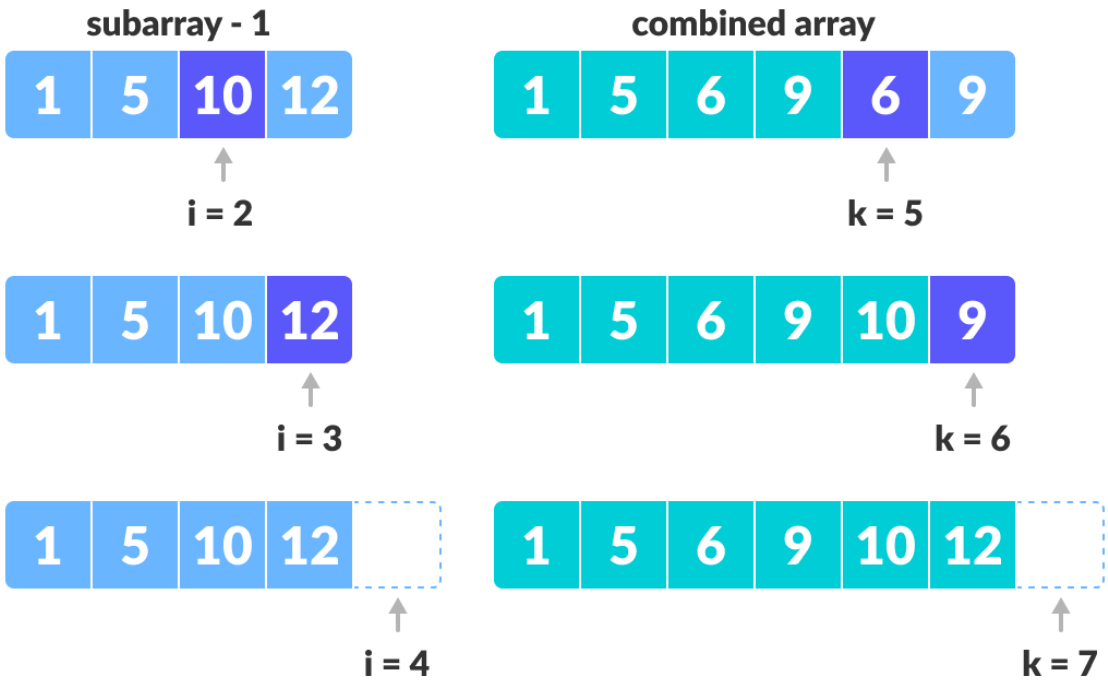
```
while (i < n1 && j < n2) {
    if (L[i] <= M[j]) {
        arr[k] = L[i]; i++;
    }
    else {
        arr[k] = M[j];
        j++;
    }
    k++;
}
```



Comparing individual elements of sorted subarrays until we reach end of one

Step 4: When we run out of elements in either L or M, pick up the remaining elements and put in A[p..r]

```
// We exited the earlier loop because j < n2 doesn't hold
   while (i < n1)
   {
      arr[k] = L[i];
      i++;
      k++;
   }
```



Copy the remaining elements from the first array to main subarray

```
   // We exited the earlier loop because i < n1 doesn't hold
   while (j < n2)
   {
      arr[k] = M[j];
      j++;
      k++;
   }
}
```

This step would have been needed if the size of M was greater than L.

At the end of the merge function, the subarray A[p..r] is sorted.