Merge Sort

Merge Sort follows the rule of Divide and Conquer to sort a given set of numbers/elements, recursively, hence consuming less time.

Merge sort is similar to the quick sort algorithm as it uses the divide and conquer approach to sort the elements. It is one of the most popular and efficient sorting algorithm. It divides the given list into two equal halves, calls itself for the two halves and then merges the two sorted halves. We have to define the merge() function to perform the merging.

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

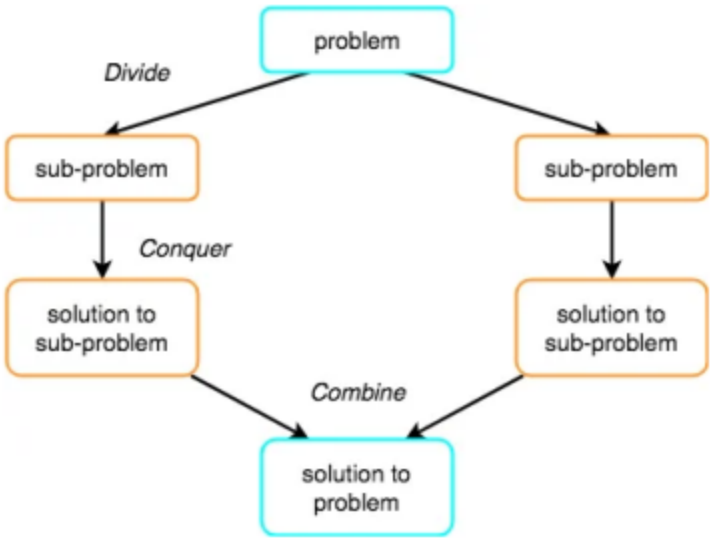
In Merge Sort, the given unsorted array with n elements, is divided into n subarrays, each having one element, because a single element is always sorted in itself. Then, it repeatedly merges these subarrays, to produce new sorted subarrays, and in the end, one complete sorted array is produced.

The concept of Divide and Conquer involves three steps:

1. Divide the problem into multiple small problems.

2. Conquer the subproblems by solving them. The idea is to break down the problem into atomic subproblems, where they are actually solved.

3. Combine the solutions of the subproblems to find the solution of the actual problem.



• **How Merge Sort Works?**

As we have already discussed that merge sort utilizes divide-and-conquer rule to break the problem into sub-problems, the problem in this case being, sorting a given array.

In merge sort, we break the given array midway, for example if the original array had 6 elements, then merge sort will break it down into two subarrays with 3 elements each.

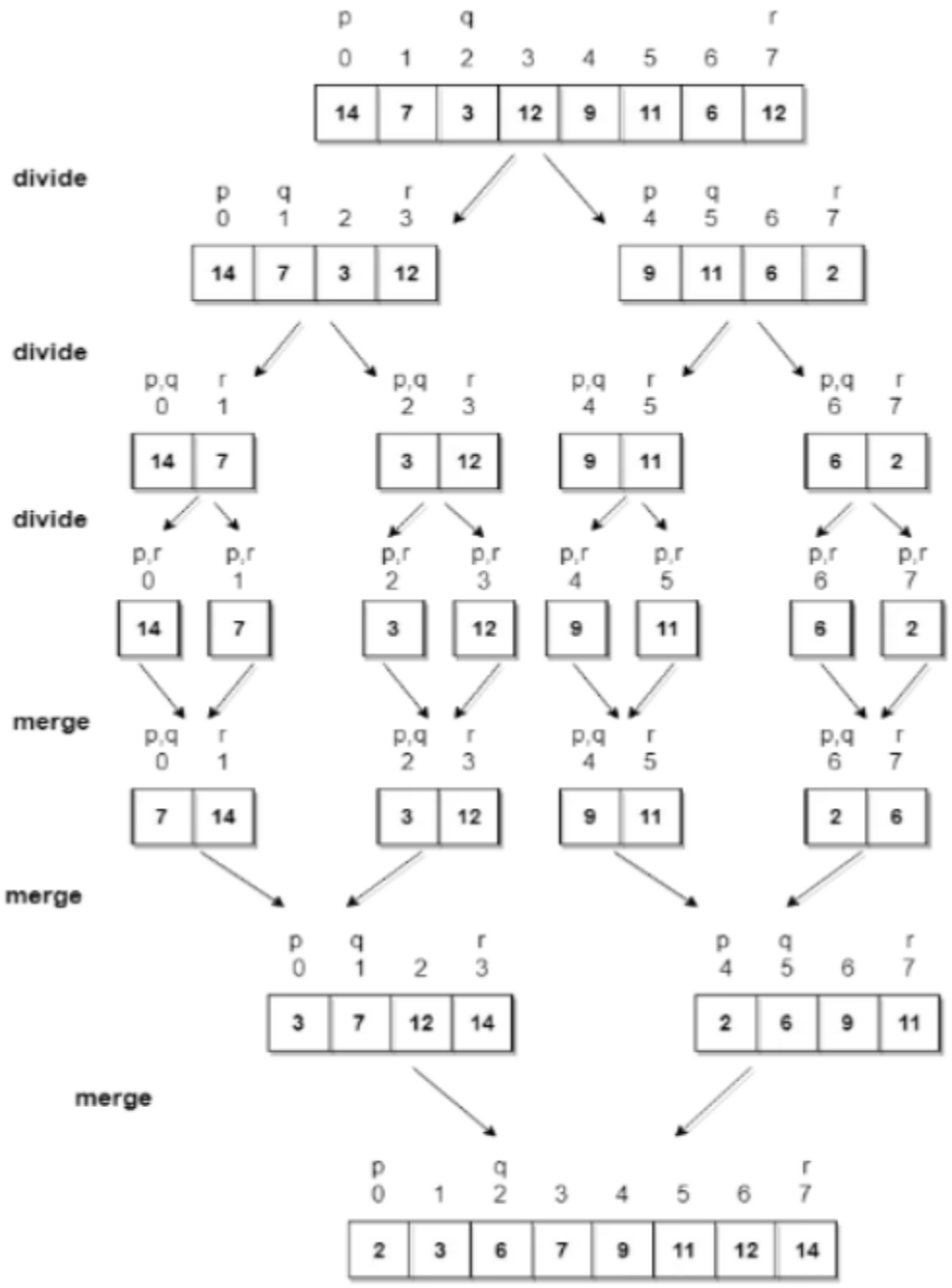
But breaking the orignal array into 2 smaller subarrays is not helping us in sorting the array.

So we will break these subarrays into even smaller subarrays, until we have multiple subarrays with single element in them. Now, the idea here is that an array with a single element is already sorted, so once we break the original array into subarrays which has only a single element, we have successfully broken down our problem into base problems.

And then we have to merge all these sorted subarrays, step by step to form one single sorted array.

Let's consider an array with values {14, 7, 3, 12, 9, 11, 6, 12}

Below, we have a pictorial representation of how merge sort will sort the given array.



In merge sort we follow the following steps:

1. We take a variable p and store the starting index of our array in this. And we take another variable r and store the last index of array in it.

2. Then we find the middle of the array using the formula (p + r)/2 and mark the middle index as q, and break the array into two subarrays, from p to q and from q + 1 to r index.

3. Then we divide these 2 subarrays again, just like we divided our main array and this continues.

4. Once we have divided the main array into subarrays with single elements, then we start merging the subarrays.

- **Implementation of Merge Sort**

```
#include <iostream>

class Merge {
public:
   void merge(int a[], int beg, int mid, int end) {
      int n1 = mid - beg + 1;
      int n2 = end - mid;

      int LeftArray[n1];
      int RightArray[n2];

      for (int i = 0; i < n1; i++)
         LeftArray[i] = a[beg + i];

      for (int j = 0; j < n2; j++)
         RightArray[j] = a[mid + 1 + j];
```

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

        while (i < n1 && j < n2) {
            if (LeftArray[i] <= RightArray[j])
                a[k++] = LeftArray[i++];
            else
                a[k++] = RightArray[j++];
        }

        while (i < n1)
            a[k++] = LeftArray[i++];

        while (j < n2)
            a[k++] = RightArray[j++];
    }

    void mergeSort(int a[], int beg, int end) {
        if (beg < end) {
            int mid = (beg + end) / 2;
            mergeSort(a, beg, mid);
            mergeSort(a, mid + 1, end);
            merge(a, beg, mid, end);
        }
    }

    void printArray(int a[], int n) {
        for (int i = 0; i < n; i++)
            std::cout << a[i] << " ";
    }
};

int main() {
    int a[] = { 11, 30, 24, 7, 31, 16, 39, 41 };
```

```
    int n = sizeof(a) / sizeof(a[0]);
    Merge m1;
    std::cout << "Before sorting array elements are - \n";
    m1.printArray(a, n);
    m1.mergeSort(a, 0, n - 1);
    std::cout << "\nAfter sorting array elements are - \n";
    m1.printArray(a, n);
    std::cout << std::endl;
    return 0;
}
```