MERGESORT

Mergesort is a popular sorting algorithm but also for sorting not only arrays sequentially stored elements. Its popularity is derived from the above feature and also from the fact that the worst case complexity is $O(n \log n)$.

Mergesort, like quicksort, is a naturally recursive algorithm. It crucially uses a procedure called "merge" which merges two sorted lists into a single sorted list in worst case $O(n)$ time where n is the sum of the numbers of elements in the two input lists.

Example for Merging:

Consider List 1 :

0	10	20	50

List 2 :

0	2	8	60

In order to merge these two lists, we maintain two variables p1 and p2 which point to the current elements in List 1 and List 2, respectively.

The merged list can be obtained by incrementing the two variables p1 and p2 alternately. In this case, the merged list would be

0	0	2	8	10	20	50	60

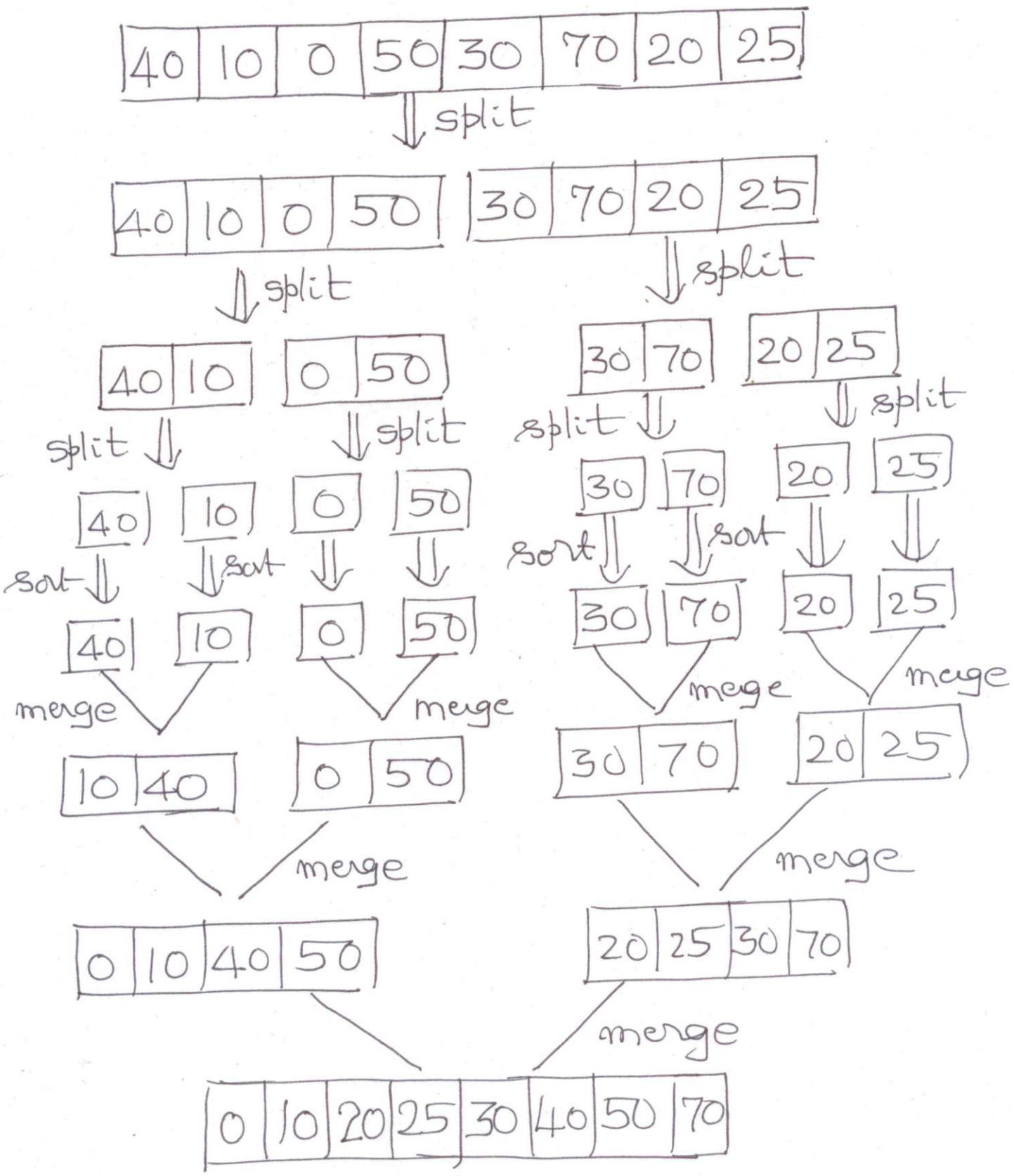
Assume that the function

merge (List 1, List 2)

returns a merged sorted list when two sorted lists List 1 and List 2 are input to it. Then the mergesort algorithm for sorting a list L with n elements can be written down as follows:

```
mergesort (L, n);
{ if (n = 1) return L;
  else
  {split L into two portions L1 and L2
   with lengths n1 and n2 respectively
   (with n1 approximately equal to n2);
   return (merge (mergesort (L1, n1),
                  mergesort (L2, n2));
  }
}
```

An Example of MergeSort



It is interesting to find out the exact sequence in which execution proceeds in the above picture. Since the splitting of each list at every stage is done so as to result in almost equal sized parts, the following recurrence relation captures the running time of mergesort algorithm:

$$T(n) \leq C_1 \quad (\text{for } n = 1)$$
$$\leq 2T\left(\frac{n}{2}\right) + C_2 n \quad (\text{for } n > 1)$$

The above recurrence relation leads to a computational complexity of

$$O(n \log n)$$

One of the attractive features of mergesort is its applicability to sequentially stored elements as well as randomly stored elements.