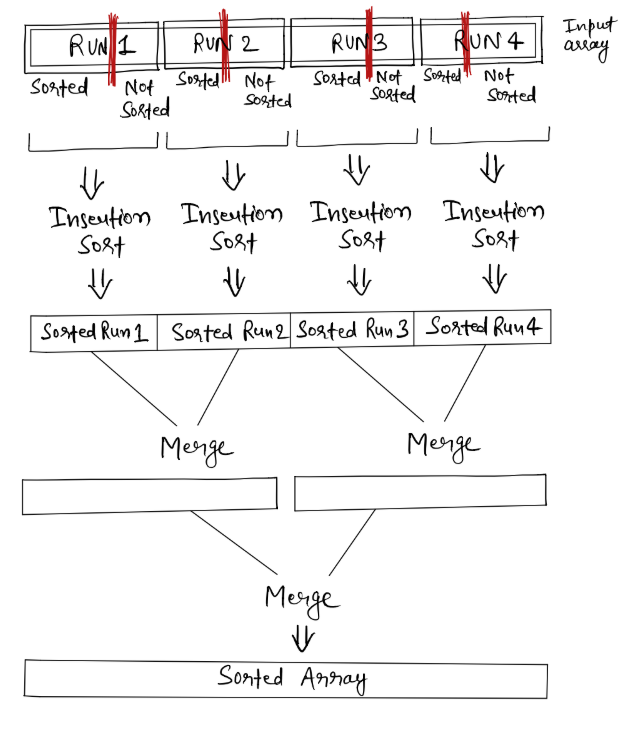
**New Algorithm Report**

*“Timsort”*

**Description**

Timsort is a sorting algorithm created by Tim Peters initially for the Python programming language in 2001. Its big O notation is O(n log n). Timsort is essentially a combination of insertion sort and merge sort depending on the length of the list it is given to sort. So as it begins to analyze said list, it performs just insertion sort when the array contains less than 64 elements. But if an array contains 64 elements or more it executes a combination of insertion sort and merge sort to sort that list. How that works is- the algorithm searches for increasing and decreasing parts in the list and every decreasing part is then reversed by using insertion sort. Each of these individual parts are called a **run**. The **size of a run** is considered **32**. Eventually, we are left with a bunch of sorted runs from the list each of which are then merged using the combine function used in merge sort. Now we have one big sorted list. The motive of Timsort is derived from the fact that the merge function performs well when the size of subarrays are in powers of 2 and that insertion sort performs well with small arrays (<https://skerritt.blog/timsort/> , <https://www.geeksforgeeks.org/timsort/> ).

**Example**

Here is a list containing only a few elements (way less than 64):

38,47,5,44,3

So accordingly a simple insertion sort is performed:

* **38**, 47, 5, 44, 3
* **38, 47**, 5, 44, 3
* **5, 38, 47**, 44, 3
* **5, 38, 44, 47**, 3
* **3, 5, 38, 44, 47**

But when a part (a **run)** of a long list is in the decreasing trend, it is revered:

[3, 2, 1, 9, 17, 34]

[**1, 2, 3,** 9, 17, 34]

If not decreasing the run would remain the same.

To zoom out and see how these individual runs are then sorted to finally have the large list sorted in its entirety, would look something like in the diagram above (<https://njha-collab.github.io/blogs/understanding-tim-sort>):