MergeSort Demo

Let's practice the merge sort algorithm. Merge sort is a recursive algorithm that continually splits a list in half. If the list is empty or has one item, it is sorted by definition (the base case). If the list has more than one item, split the list and recursively call a merge sort on each half. As the recursion unwinds, each half is continually **merged** (or zippermerged) with its other half to make a single, sorted, list.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Array 1** |  | 7 | 1 | 8 | 3 | 5 | 2 | 6 | 4 |
|  | 1st merge | 1 | 7 | 8 | 3 | 5 | 2 | 6 | 4 |
|  | 2nd merge | 1 | 7 | 3 | 8 | 5 | 2 | 6 | 4 |
|  | 3rd merge | 1 | 3 | 7 | 8 | 5 | 2 | 6 | 4 |
|  | 4th merge | 1 | 3 | 7 | 8 | 2 | 5 | 6 | 4 |
|  | 5th merge | 1 | 3 | 7 | 8 | 2 | 5 | 4 | 6 |
|  | 6th merge | 1 | 3 | 7 | 8 | 2 | 4 | 5 | 6 |
|  | 7th merge | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|  |  |  |  |  |  |  |  |  |  |
| **Array 2** |  | 3 | 1 | 4 | 1 | 5 | 9 | 2 | 6 |
|  | 1st merge | 1 | 3 | 4 | 1 | 5 | 9 | 2 | 6 |
|  | 2nd merge | 1 | 3 | 1 | 4 | 5 | 9 | 2 | 6 |
|  | 3rd merge | 1 | 1 | 3 | 4 | 5 | 9 | 2 | 6 |
|  | 4th merge | 1 | 1 | 3 | 4 | 5 | 9 | 2 | 6 |
|  | 5th merge | 1 | 1 | 3 | 4 | 5 | 9 | 2 | 6 |
|  | 6th merge | 1 | 1 | 3 | 4 | 2 | 5 | 6 | 9 |
|  | 7th merge | 1 | 1 | 2 | 3 | 4 | 5 | 6 | 9 |
|  |  |  |  |  |  |  |  |  |  |
| **Array 3** |  | 5 | 3 | 1 | 7 | 2 | 8 | 4 | 6 |
|  | 1st merge | 3 | 5 | 1 | 7 | 2 | 8 | 4 | 6 |
|  | 2nd merge | 3 | 5 | 1 | 7 | 2 | 8 | 4 | 6 |
|  | 3rd merge | 1 | 3 | 5 | 7 | 2 | 8 | 4 | 6 |
|  | 4th merge | 1 | 3 | 5 | 7 | 2 | 8 | 4 | 6 |
|  | 5th merge | 1 | 3 | 5 | 7 | 2 | 8 | 4 | 6 |
|  | 6th merge | 1 | 3 | 5 | 7 | 2 | 4 | 6 | 8 |
|  | 7th merge | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|  |  |  |  |  |  |  |  |  |  |
| **Array 4** |  | 2 | 5 | 8 | 1 | 6 | 7 | 4 | 3 |
|  | 1st merge | 2 | 5 | 8 | 1 | 6 | 7 | 4 | 3 |
|  | 2nd merge | 2 | 5 | 1 | 8 | 6 | 7 | 4 | 3 |
|  | 3rd merge | 1 | 2 | 5 | 8 | 6 | 7 | 4 | 3 |
|  | 4th merge | 1 | 2 | 5 | 8 | 6 | 7 | 4 | 3 |
|  | 5th merge | 1 | 2 | 5 | 8 | 6 | 7 | 3 | 4 |
|  | 6th merge | 1 | 2 | 5 | 8 | 3 | 4 | 6 | 7 |
|  | 7th merge | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|  |  |  |  |  |  |  |  |  |  |
| **Array 5**  **descending** |  | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 |
|  | 1st merge | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 |
|  | 2nd merge | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 |
|  | 3rd merge | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 |
|  | 4th merge | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 |
|  | 5th merge | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 |
|  | 6th merge | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 |
|  | 7th merge | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 |
|  |  |  |  |  |  |  |  |  |  |
| **Array 6**  **ascending** |  | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|  | 1st merge | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|  | 2nd merge | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|  | 3rd merge | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|  | 4th merge | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|  | 5th merge | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|  | 6th merge | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|  | 7th merge | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |

To analyze the Big-O of the Mergesort, note that for any given list of length *n*, there will be exactly *log n* divisions. Each of the *n* items needs to run the zippermerge method, *O(n).* The result is *O(n log n)*. Note that the order of the data neither increases nor decreases the Big-O calculations. Mergesort usually requires additional memory space.