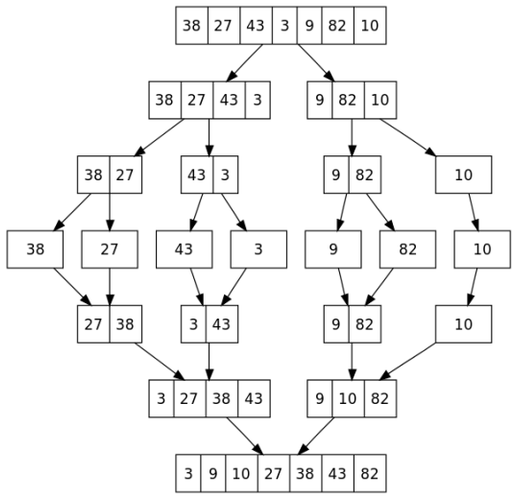
Merge Sort

6 5 3 1 8 7 2 4

Merge Sort


38 27 43 3 9 82 10
38 27 43 3
9 82 10
38 27
43 3
9 82
10
38
27
43
3
9
82
10
27 38
3 43
9 82
10
3 27 38 43
9 10 82
3 9 10 27 38 43 82

Merge Sort (iterative)

1. Divide array of n elements into n arrays of 1 element
2. Merge neighboring arrays in sorted order
3. Repeat 2 until there's only one array

Merge Sort (recursive)

1. If array is one element, good job it's sorted!
2. Otherwise, split the array and merge sort each half
3. Merge combined halves into sorted whole

Big O

	Bubble Sort	Merge Sort
Time	$O(n^2)$	$O(n \cdot \log n)$
Space	$O(1)$	$O(n)$

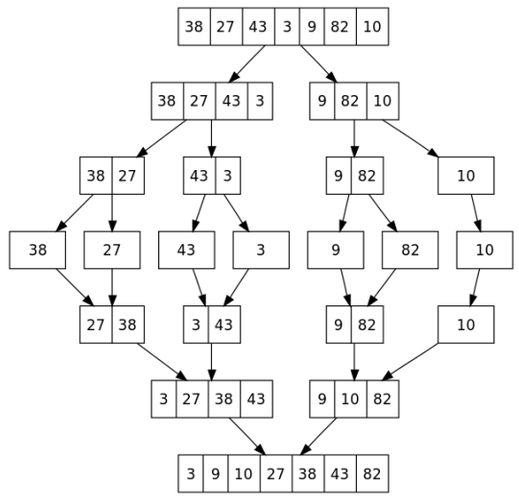
Why is merge sort faster?

Merge Sort Speedup

- Splitting a list into two sublists is a linear time operation
- Combining two lists that are each already sorted into one list that is sorted is a linear time operation
- There are $\log_2(n)$ steps needed to go from n lists of one item each to one list of n items (and vice-versa)

O(n) ops to split or merge

O(log n) times we split/merge



O(n) * O(log n) =
O(n · log n)

Intuition?

- Divide and conquer: can efficiently handle subtasks, and also efficiently combine sorted lists.
- Reduce the possible comparisons dramatically – only have to compare certain pairs of elements (avoiding vast majority of possible pairs).