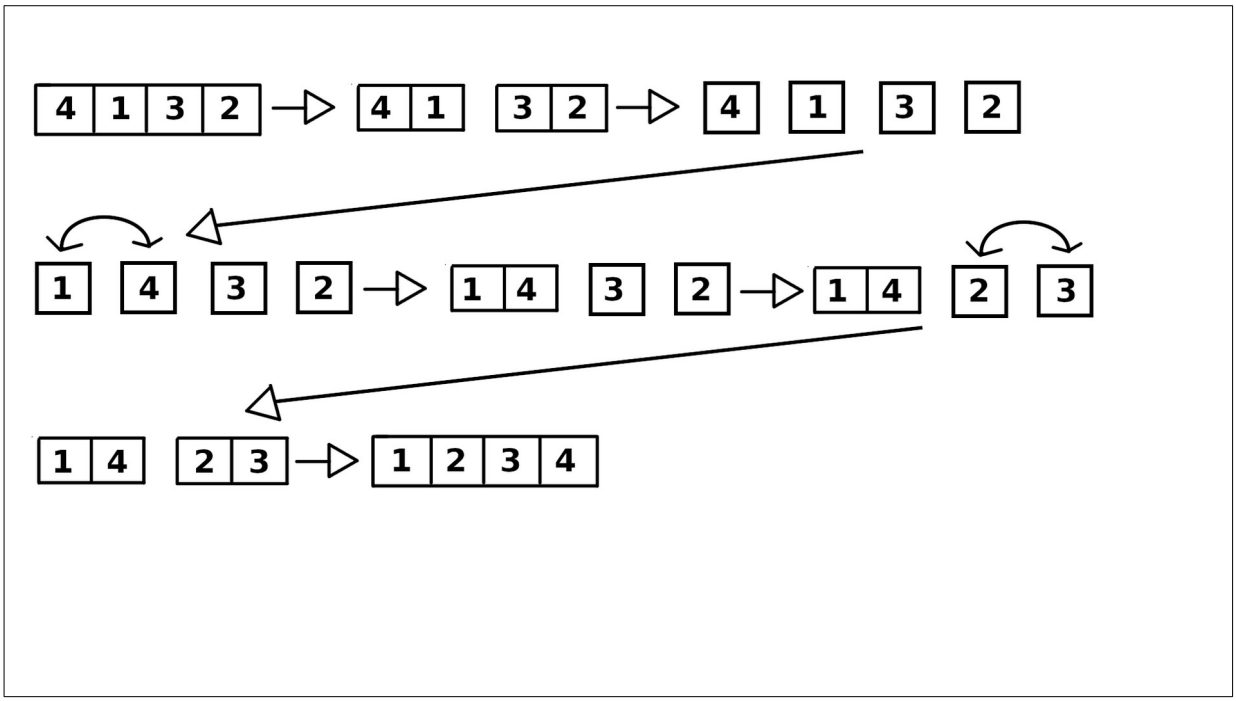
Background

Sorting is one of the most fundamental tasks which humans carry out. Taking a group or list of items and putting them in a specific order is simple in concept, but an incredibly important skill to have. Due to this importance, sorting algorithms are taught quite early on in computer science education. Quite a few different algorithms exist, each of them achieving roughly the same goal through very different methods. The algorithm on which we will focus is called the merge sort. It is efficient and uses an intuitive method that most people may have already used themselves without ever realizing it.

Divide and Conquer

The way that the merge sort algorithm works is by breaking down the large problem (sorting the entire list) into smaller problems of the same type (sorting a part of the list). Eventually, the problem is broken down into small enough pieces (a single item from the list) that it can be solved directly, and then that solution can be used to solve the next step. This method for breaking down a problem encapsulates a category of algorithms called “divide and conquer”. Merge sort is just one of many algorithms of this type.

Example



Steps:

1. The list is broken down into the smallest pieces possible, individual elements. These single elements can each be considered a sorted list of length one. This is pictured in the top row.
2. Compare the first two elements, and swap if necessary. Merge the elements into one list.
3. Compare the second set of elements, swap if necessary. Merge the elements into one list. There are now two separate sorted lists.
4. Compare each element of the two lists one by one, placing them into the final sorted list in the correct order. This completes the process.

Variations

There are several variations of merge sort that exist. They are all equally efficient and just make small changes to how the algorithm is implemented.

- Top-down (depth first) is the default variation. It breaks the data into smaller chunks until length one, then merges them back together in the same order (half by half) that it was broken down in.
- Bottom-up (breadth first) is a simple variation. It breaks the data into smaller chunks until length one, then merges each smallest chunk with the next before merging the larger sorted chunks the process creates.
- In-place is a variation that removes the requirement for auxiliary space in memory, usually used to create the new sorted lists. Requiring no auxiliary space makes the algorithm “in-place”. This lesson will not explain how the inner-workings of this variation.

Stability

Merge sort is a stable sorting algorithm. This means that when there are repeated elements in the list that is being sorted, they will appear in the sorted list in the same order as they were originally. For example, when sorting a set of playing cards as such {5 ♥ , 9 ♠ , 5 ♣ , 2 ♦ }, the cards of rank 5 must then appear in the sorted set in the exact same order as in the original: hearts then clubs. The final sorted set would be {2 ♦ , 5 ♥ , 5 ♣ , 9 ♠ }.

Time Complexity

The efficiency of algorithms is generally measured in time complexity, which is a way of understanding how many operations must be carried depending on the size of the input (usually denoted by “n”). The more efficient the algorithm, the better it performs with massive inputs. These measures of performance divide algorithms into different classifications. Some of these categories are constant time, linear time, logarithmic time, polynomial time, and exponential time. These classifications’ names correspond to the functions that appear in the time complexity equations of the algorithms which they contain. In the case of merge sort, its divide and conquer nature makes it very efficient, landing it in the logarithmic time classification (technically *linearithmic* time). Its complexity, even in the worst case, is defined as O(n log n), where n is the size of the input. The letter “O” here stands for “order”.