QuickSort Demo

Let's practice the QuickSort algorithm. The first item is the **pivot**. The goal of the partition process is to move items that are on the "wrong side" with respect to the pivot value while also converging on the split point.

Increment the **left** marker until it locates a value that is **greater than** the pivot value. Decrement the **right** marker until it finds a value that is **less than** the pivot value. Swap these two items. Continue incrementing the left and decrementing the right and swapping out-of-order pairs.

Stop at the point where the **right marker becomes less than** the left marker. The position of the right marker is now where the pivot value belongs. Swap the contents of that position (sometimes called the "split point") with the pivot value. The pivot never changes again.

Notice that all the items to the left of the split point are less than the pivot value, and all the items to the right of the split point are greater than the pivot value. The list is then split left and right of the split point and the quick sort is invoked recursively on each side.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Array 1** | pivot is 4 | 4 | 5 | 3 | 8 | 7 | 2 | 6 | 1 |
|  | pivot is 2 | 2 | 1 | 3 | 4 | 7 | 8 | 6 | 5 |
|  | pivot is 7 | 1 | 2 | 3 | 4 | 7 | 8 | 6 | 5 |
|  | pivot is 6 | 1 | 2 | 3 | 4 | 5 | 7 | 6 | 8 |
|  | in order |  |  |  | 4 |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
| **Array 2** | pivot is 8 | 8 | 3 | 2 | 5 | 1 | 4 | 7 | 6 |
|  | pivot is 6 |  |  |  |  |  |  |  |  |
|  | pivot is 4 |  |  |  |  |  |  |  |  |
|  | pivot is 1 |  |  |  |  |  |  |  |  |
|  | pivot is 3 |  |  |  |  |  |  |  |  |
|  | in order |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
| **Array 3** | pivot is 5 | 5 | 3 | 1 | 7 | 2 | 8 | 4 | 6 |
|  | pivot is \_\_\_ |  |  |  |  |  |  |  |  |
|  | pivot is 3 |  |  |  |  |  |  |  |  |
|  | pivot is \_\_\_ |  |  |  |  |  |  |  |  |
|  | pivot is 6 |  |  |  |  |  |  |  |  |
|  | in order |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
| **Array 4** | pivot is 2 | 2 | 5 | 8 | 1 | 6 | 7 | 4 | 3 |
|  | pivot is \_\_\_ |  |  |  |  |  |  |  |  |
|  | pivot is \_\_\_ |  |  |  |  |  |  |  |  |
|  | pivot is 5 |  |  |  |  |  |  |  |  |
|  | pivot is \_\_\_ |  |  |  |  |  |  |  |  |
|  | in order |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
| **Array 5**  **descending** | pivot is 8 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 |
|  | pivot is 1 | 1 | 7 | 6 | 5 | 4 | 3 | 2 | 8 |
|  | pivot is 7 |  |  |  |  |  |  |  |  |
|  | pivot is 2 |  |  |  |  |  |  |  |  |
|  | pivot is 6 |  |  |  |  |  |  |  |  |
|  | pivot is 3 |  |  |  |  |  |  |  |  |
|  | pivot is 5 |  |  |  |  |  |  |  |  |
|  | in order |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
| **Array 6**  **ascending** | pivot is 1 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|  | pivot is \_\_\_ |  |  |  |  |  |  |  |  |
|  | pivot is \_\_\_ |  |  |  |  |  |  |  |  |
|  | pivot is \_\_\_ |  |  |  |  |  |  |  |  |
|  | pivot is \_\_\_ |  |  |  |  |  |  |  |  |
|  | pivot is \_\_\_ |  |  |  |  |  |  |  |  |
|  | pivot is \_\_\_ |  |  |  |  |  |  |  |  |
|  | pivot is \_\_\_ |  |  |  |  |  |  |  |  |
|  | in order |  |  |  |  |  |  |  |  |

To analyze the Big-O of the Quicksort, note that for a list of length *n*, if the partition always occurs in the middle of the list, there will be exactly *log n* divisions. In order to find the split point, each of the *n* items needs to be compared to the pivot value. The result is *O(n log n)*.

Unfortunately, depending on the data, the split points may be at either end instead of in the middle. In this case, sorting a list of *n* items divides into sorting a list of 0 items and a list of *n−1* items. Then sorting a list of *n−1* divides into a list of size 0 and a list of size *n−2*, and so on. The result is an *O(n2)* sort in the worst case, a situation often called a "bad pivot".