INTRODUCTION TO QUICK SORT

1. Quicksort is an algorithm used to quickly sort items within an array no matter how big the array is.

2. It is quite scalable and works relatively well for small and large data sets, and is easy to implement with little time complexity.

3. It does this through a divide-and-conquer method that divides a single large array into two smaller ones and then repeats this process for all created arrays until the sort is complete.

4. Quicksort is a comparison sort, meaning that it can sort items of any type for which a "less-than" relation (formally, a total order) is defined

5. In efficient implementations it is not a stable sort, meaning that the relative order of equal sort items is not preserved.

6. Quicksort can operate in-place on an array, requiring small additional amounts of memory to perform the sorting.

7. It is very similar to selection sort, except that it does not always choose worst-case partition.

8. Mathematical analysis of quicksort shows that, on average, the algorithm takes O(*n* log *n*) comparisons to sort *n* items. In the worst case, it makes O(*n*2) comparisons, though this behavior is rare.

The quicksort algorithm is performed as follows:

- A pivot point is chosen from the array.
- The array is reordered so that all values smaller than the pivot are moved before it and all values larger than the pivot are moved after it, with values equaling the pivot going either way. When this is done, the pivot is in its final position.
- The above step is repeated for each sub array of smaller values as well as done separately for the sub array with greater values.

This is repeated until the entire array is sorted.

HOW IT WORKS

The basic idea of Quicksort is to repeatedly divide the array into smaller pieces (these are called partitions), and to recursively sort those partitions. Quicksort divides the current partition by choosing an element - the pivot - finding which of the other elements are smaller or larger, sorting them into two different sub-partitions (one for the values smaller than the pivot, one for those larger than the pivot). For example, say we have the input (5,7,2,3,1,4,6). Here's what happens in the first partitioning run of a standard Quicksort algorithm (one that always chooses the middle element of the current partition as the pivot):

Data							Comments
5	7	2	3	1	4	6	• First of all, a pivot element is selected
3	7	2	5	1	4	6	• It is swapped with the first element in the partition (to get it out of the way!).
3	7	2	5	1	4	6	• The program searches from the left for an element that belongs to the right of the pivot element, and from the right for an element that belongs to the left
3	1	2	5	7	4	6	• Whenever it finds any two such elements, it swaps them.
3	1	2	5	7	4	6	• Eventually the search from the left and the search from the right reach the same place. If, at the place where the search stops, the element belongs on the left of the pivot element, it is swapped with it. That's not the case here!
2	1	3	5	7	4	6	• ...Otherwise, the pivot element is swapped with the element to the left of the spot where the search stopped. Quicksort implementations that skip this step may not terminate.
2	1	3	5	7	4	6	• We recursively call the Quicksort algorithm to sort the smaller of the two sub partitions (in this case, there are two elements on the left of the pivot, and four on the right, so we sort the left and then the right).
1	2	3	5	7	4	6	• And, lastly, we sort the larger partition...
1	2	3	4	5	6	7	