Insertion sort is a sorting algorithm that works a little like sorting a hand of cards when playing poker or bridge. It is a simple algorithm in that it takes an unsorted array and iterates one element from the list and compares it to the other elements. When working left to right it examines each item and compares it to the items on its left and after that it inserts the item in the correct position in the array. The run-time complexity for insertion sort is O(n) for the best-case scenario, $O(n^2)$ for the worst-case scenario, and has an average-case scenario time complexity of $O(n^2)$.

Quick sort is a divide and conquer algorithm, which makes it very different from the much simpler insertion sort algorithm. The main point here is to divide a bigger problem into smaller subproblems until it is easier to solve. These solutions are thereafter combined to solve the original problem. The run-time complexity for quicksort is $O(n^2)$ in worst case, and $O(n * log(n))$in the best and average case.

The algorithm begins with choosing a pivot element(which is mostly the last element of an array or a random element and then stores the elements that are less than the pivot in a left subarray. The greater elements are stored in a right subarray. Then, the algorithm calls its own function recursively on both the left and right subarray.

Heapsort works around an ordered binary tree, with max heaps always being bigger than the so-called child heap. The algorithm begins with creating a max heap from an unsorted array. It later removes the largest item from the max heap, and as a third step it places the item in a sorted partition.

When studying the three algorithms, one should take in mind not only the time complexity of these algorithms but also its usefulness and coding characteristics. That is, it is also good to take in consideration how an algorithm works in the bigger picture. If time complexity is the only factor of interest, one would suggest that the least favourable option of these three would be insertion sort, since its average performance and worst-case performance is quadratic $O(n^2)$. However, there are some other factors that would suggest that the insertion sort could be a favourable algorithm to use. Depending on the task at hand, different algorithms could turn out to be the better choice, even though it would not look that way at first glance. For example, if one would deal with arrays or lists that are already sorted beforehand, the insertion sort would have the upper hand, since in that case the time complexity would be O(n), and it is an easier algorithm to implement. When working with smaller numbers of n, insertion sort could also prove to be the better algorithm. The additional memory space needed for insertion sort is also low, rather a constant of $O(1)$, which would come in handy when working with very limited memory space.

However, when working with larger numbers of n, the average(and worst) time complexity makes both quick sort and heap sort more favourable than the simpler insertion sort. The question is which one of these two algorithms to use when managing very large numbers of n. This question is probably up to debate, but there are some factors and differences between them that one should have in mind. The main difference between quicksort and heapsort is the worst-case running time. Heapsort is the only one of these algorithms that has the same worst-case, average-case and best-case scenario. They both have an $O(n * log(n))$. This can be compared to the worst-case scenario of quicksort, which is the same as insertion sort, namely $O(n^2)$. This being said, quicksort is usually faster than heapsort as it does not have the need to swap elements like heapsort does. Even though you work with an already listed array, heapsort will still swap all of the elements. In the best case scenario for quicksort, almost no swaps are made, however the comparison between elements would still be the same as with heapsort. The time saved on not swapping any elements is what makes quicksort the faster algorithm in the best-case and average-case scenario.

When it comes to space complexity, heapsort and insertion sort have a complexity of O(1) since they only work with one array, and this could come in handy when working with very limited memory spaces compared to the quicksort, which has a space complexity of $O(n)$.