**What are the differences between the three algorithms? For each algorithm, mention a situation where it has an advantage over the other two**

--

Insertion sort divides the array into two subarrays, one where the elements are sorted and one where they are unsorted. Through the sorting process elements of the unsorted part are moved into the sorted part. The element to be added is compared against each element of the sorted array until the unsorted element is smaller than the element it is compared to. At this point the element is inserted into the corresponding position. Insertion sort is best implemented on problems where n is small. For the best case scenario where the array is already sorted the sorting process is fast. As n grows insertion sort becomes inefficient and a less attractive option than other sorting algorithms. For insertion sort the best case is O(n) and the worst case is $O(n^2)$.

Quick Sort divides the array around a value q, where everything to the left of q is smaller than q and everything to the right is larger. The parts are then sorted recursively through reconstructing sorted subparts. The computationally heavy part is the Partition that finds the q. Quick Sort is in most cases fast, having a time complexity of Θ(nlogn) in best and average case. In the worst case the pivot element is always found in the first or last position of the array resulting in a maximum depth of the recursive tree. This results in a time complexity of Θ(n^2). This occurs when the array is either already sorted or reverse sorted. Since quicksort, in comparison to insertion and heap sort, doesn't implement unnecessary element swaps on an already sorted or partially sorted array, therefore quicksort's implementation of best case is preferred to other algorithms. However, in the worst case, as we can see by time complexity values, quicksort is inferior to heapsort.

Heapsort first restructures the array to be like a well indexed max heap, with the biggest value in the first position treated as a root. A max heap is a complete binary tree, with each node having a parent (except for the root) that are larger than the node and up to two children that are smaller than the node. As the tree should be complete all positions depth <= height - 1 must be filled. The last depth is filled from left to right. In a well indexed max heap each internel can have a left and right child in position $2^i$ and $2^i+1$ in the array respectively. As the biggest value is in the first position it is then switched with the last position and taken out of the tree (but not out of the array). By then restoring the max heap property then the biggest element in the tree is found and moved to the last position in the tree and the position is removed from the tree. This repeats until the tree is only one node as the array then is sorted. Heap sort is fast in all cases, with a time complexity of O(nlogn), and is therefore consistently good and guarantees a fast response time. However it is harder to implement than the other two algorithms and is therefore mostly used in cases where responstimes are critical, such as for big datasets.