We consider each sort on a 0-indexed array A of n elements.

1. Selection Sort

   On the i^{th} iteration (beginning with 0), we pass through the array from A_i to A_{n-1} to find the smallest value. Thus, the number of operations is given by

   $$(n-1)+(n-2)+\ldots+2+1=\frac{n(n-1)}{2}$$

   A selection sort is therefore $O(n^2)$.

2. Insertion Sort

   On the i^{th} iteration (beginning with 0), we place A_{i+1} in the correct position relative to the first $i+1$ elements. In the **worst case**, the array will be in reverse-order, and elements A_0 through A_i must be shifted one to the right, with A_{i+1} being placed at the beginning of the array. That is a total of

   $$1+2+\ldots+(n-2)+(n-1)=\frac{n(n-1)}{2}$$

   operations. An insertion sort is therefore $O(n^2)$ **in the worst case**.

   In the **best case**, the array is already sorted and we only need to pass through the array once, in which case the sort is $O(n)$.

3. Merge Sort

   The number of recursive calls to the merge sort will be $\lceil \log(n) \rceil$. At each step, the number of operations in merging the sublists will be proportional to the number of elements. Thus, a merge sort is $O(n \log(n))$.

4. Quick Sort

   For each level the number of operations will be $O(n)$. In the **worst case**, the pivot will be the least or greatest value of each array, and thus there will be n levels, giving an $O(n^2)$ performance. In the **average** and **best case**, the number of levels will be $O(\log(n))$, and the performance is therefore $O(n \log(n))$.

In summary:

Name	**Best**	**Average**	**Worst**
Selection Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$
Insertion Sort	$O(n)$	$O(n^2)$	$O(n^2)$
Merge Sort	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$
Quick Sort	$O(n \log n)$	$O(n \log n)$	$O(n^2)$