3. Sorting Algorithms :: Insertion Sort :: Time Complexity

How efficient is insertion sort? Remember the steps to analyzing the worst case time complexity of an algorithm:

1. Identify one or more key operations. The key operation is the one that is integral to the behavior of the algorithm and for the worst case behavior, will be the one that is performed the most number of times.
2. Express the number of times the key operation is performed as a function $f(n)$ of the problem size.
3. Prove that $f(n)$ is $O(g(n))$ for some $g(n)$.

For insertion sort, the worst case behavior occurs during an ascending sort when the list is in reverse order, e.g., *list* = { 5, 4, 3, 2, 1 }. The key operation—the one that will occur the most times for a list such as this—will be the swap of the element at *pIndex* with the element at *pIndex* - 1.

To determine the number of swaps, let's examine an example:

i	**sublist being sorted**	**swaps**
1	{5, 4}	1
2	{4, 5, 3}	2
3	{3, 4, 5, 2}	3
4	{2, 3, 4, 5, 1}	4

After the fourth pass through the **for** ***i*** loop in *insertionSort*() the *list* will be sorted: {1, 2, 3, 4, 5} and the total number of swaps will be $1 + 2 + 3 + 4 = 10$.

3. Sorting Algorithms :: Insertion Sort :: Time Complexity (continued)

I hope the example helps you see that the number of swaps for a list of size n will be:

$$s(n) = 1 + 2 + 3 + \cdots (n - 1) = \sum_{i=1}^{n-1} i$$

There is a well-known formula for computing the sum of the first n positive integers:

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$

Therefore,

$$s(n) = \frac{n(n-1)}{2} = \frac{1}{2}n^2 - \frac{1}{2}n$$

Remember from *Data Structures and Algorithms : Section 7* that if $f(n)$ is *any* degree-p polynomial function $f(n) = c_n n^p + c_{n-1}n^{p-1} + c_{n-2}n^{p-2} + \ldots + c_1 n + c_0$, then $f(n)$ is $O(n^p)$. Therefore, $s(n)$ is $O(n^2)$ which means that the worst case time complexity of insertion sort is $O(n^2)$.