Chase Strickler

CSC 320-B

26 January 2018

Performance Analysis of Insertion Sort

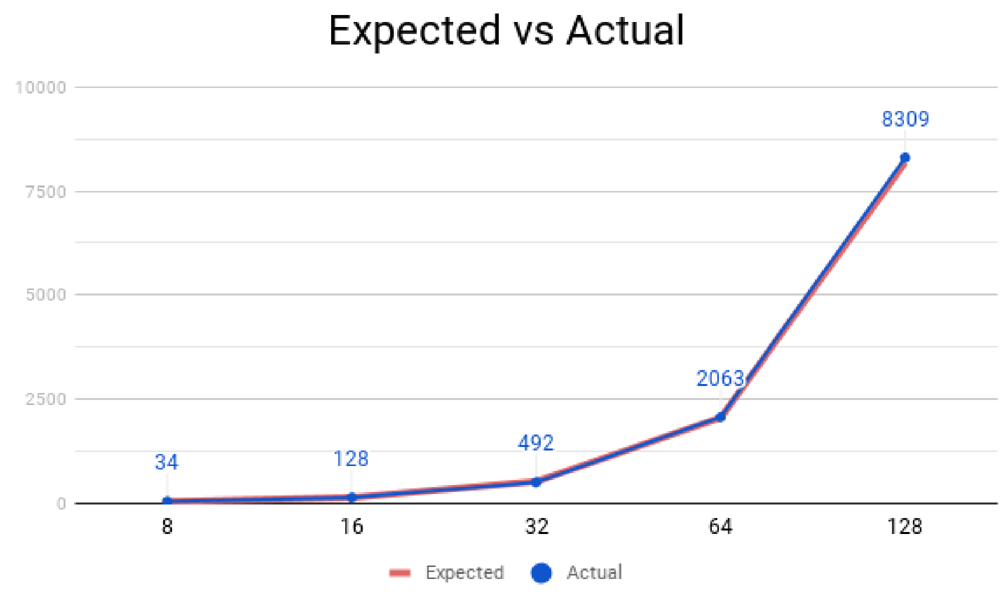
When it comes to sorting algorithms, insertion sort is one of the easiest to implement as it is incredibly basic. It doesn't use recursion, operates in place, and can be completed within one function call. With that being said, insertion sort can be slow on large input sizes, with the best case performance of O(n), and the worst case, as well as average case, performance of O(n^2). Using Java, I have set up an experiment to test the speed of insertion sort by counting the number of swaps and comparisons on randomly filled arrays of various sizes.

The first step in conducting my experiment was to insert counters within the sorting function. I created two variables: swaps, which increments when I swap occurs, and comp, which increments for each comparison. I placed the comp variable within the outer loop, as well as the inner while loop. As for swaps, I placed it only in the inner loop. After the sorting completed, I then added swaps and comp to a new variable, count, which was then returned.

Next, I wanted to verify the correctness of my insertion sort algorithm using the counters. To do this, I tested the algorithm on two arrays of size ten, with one in sorted order, and the other in reverse sorted order. This acted as a test of correctness, as the expected return values were ten and one hundred, respectively. After verifying the correctness of my program, I moved on to the actual simulation. Here, I used four different sized array lengths: 10, 100, 1,000, 10,000. Within each array length, I generated ten arrays which were then sorted using insertion sort. I then calculated the average number of swaps and comparisons completed across all arrays of equal length. I've included a table below with my results.

Size of Array	Average Swaps & Comparisons	Expected = $O(n^2)$
10	56	100
100	5,072	10,000
1,000	500,006	1,000,000
10,000	50,122,088	100,000,000

At first, I was troubled that the averages were so far from the expected $O(n^2)$. In fact, it seemed as though the averages were roughly $(\frac{1}{2} * n^2)$. I began searching for an error in my code, but then remembered that there is some constant associated with n^2, in this case, ½. I decided to generate new data, where the array size is some power of two, and graph both the expected and actual functions. The referenced graph is below, accompanied by a table of the data.



Size of Array	Average Swaps & Comparisons	Expected = $\frac{n^2}{2}$
8	34	32
16	128	128
32	492	512
64	2063	2048
128	8309	8192

The graph proved my hypothesis: the constant associated with the average case performance $O(n^2)$ was ½, and my program was working properly. The constant made quite a bit of difference here, as the input size was quite small. Additionally, the graph helped visualize the fact that insertion sort is efficient for small input sizes, but becomes ineffective rather quickly when the input size increases.