**Step 1: Runtime Analysis**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Function** | **Extra-Large** | **Large** | **Medium** | **Small** | **Tiny** |
| **Array Length** | 100000 | 10000 | 1000 | 100 | 10 |
| **Insert (μs)** | 743315.6 | 6695.7 | 123.4 | 45.6 | 34.5 |
| **Append (μs)** | 2243.3 | 396.2 | 102.8 | 87.3 | 78.4 |

**Explanation:**

 The exercise provides 5 arrays of various lengths and 2 functions, the Insert Function and Append Function, both of which take an element from an input array, square the element, and add it to a new array. While both functions square the given element the same way, the Insert Function uses the unshift() array method to add a new number to the start of an array (changing the index of all elements), while the Append Function uses the push() array method to add a new number to the end of an array (without changing the index of previous elements).

Based on the result runtimes for arrays of various lengths (Tiny, Small, Medium, Large, Extra-Large), a pattern emerged. The Insert Function has a smaller runtime value than the Append Function for Small and Tiny arrays, but it has a much larger runtime value than the Append Function for Medium, Large, and Extra-Large arrays. When plotting the array input size vs. runtime (microseconds or "μs") for each function, the Append Function appears to have linear growth (orange line) and the Insert Function appears to have quadratic growth (blue line). In other words, the Append Function with a Constant Time Complexity of O(1) scales better than the Insert Function with Quadratic Time Complexity of O(n^2). Based on the graphs shown below, the Append Function scales better for Medium to Extra-Large arrays because the runtime growth rate of the Insert Function is so significant that it makes the runtime growth rate of the Append Function appear constant in comparison.

The number of iterations in a for-loop is dependent on the size of the input array (nums.length), making it have a linear time complexity of O(n). The push() array method has a constant time complexity of O(1) since it adds the new element to the end of the array, regardless of its length. Therefore, the Append Function has an overall time complexity of O(n).

Similarly, the for-loop inside the Insert function has a linear time complexity of O(n). However, the unshift() method within the loop also has a linear time complexity of O(n) because it adds the new squared element to the start of the array, changing the index of all previous elements in each iteration. Therefore, the Insert Function has an overall time complexity of O(n^2).





