**Explanation of the pattern:**

The timing results will show that as the size of the array increases, the execution time for both functions (doublerAppend and doublerInsert) also increases. However, you will likely observe that the execution time for doublerInsert increases significantly compared to doublerAppend as the array size grows. Both functions have a linear time complexity of O(n), meaning their execution time scales linearly with the size of the input array. However, the doublerInsert function's use of unshift() inside the loop makes it less efficient compared to doublerAppend.

When using unshift(), each element added to the new array must be shifted to the right to make space for the new element at the beginning. This operation has a higher time complexity (O(n)) than the push() operation, which adds elements to the end of the array in constant time.

As a result, the doublerInsert function has worse performance for larger arrays because the time to shift elements increases significantly as the array grows, leading to a slower scaling behavior compared to the doublerAppend function.

In conclusion, the doublerAppend function scales better than the doublerInsert function as the array size increases due to its more efficient use of the push() method for appending elements to the new array.

**Extra Credit:**

The primary reason for the slowdown in the doublerInsert function is the use of the unshift() method inside the loop. The unshift() method adds elements to the beginning of an array, which forces all existing elements to shift one position to the right. This shifting operation takes additional time and results in an overall slower execution. The time complexity of unshift() is O(n) because, in the worst case, when inserting an element at the beginning of the array, all existing elements need to be moved one position to the right. This means that as the size of the array grows, the time taken for the unshift() operation grows linearly.

In contrast, the doublerAppend function uses the push() method to add elements to the end of the array. The push() method has a constant time complexity of O(1) because it simply adds elements to the end of the array without shifting any existing elements.

Due to the differences in the time complexity of unshift() and push() operations, as the size of the array increases, the doublerInsert function becomes significantly slower compared to the doublerAppend function. This performance difference becomes more noticeable with larger arrays, making the doublerInsert function less scalable and less efficient.

In summary, the use of the unshift() method inside the loop is the main reason for the slower performance of the doublerInsert function. For large arrays, the shifting operation in unshift() becomes increasingly time-consuming, leading to a significant slowdown compared to the doublerAppend function, which uses the more efficient push() method.