How long does it take to double every number in a given array?

| Arrays | doublerInsert Result | Comp | doublerAppend Result |
| --- | --- | --- | --- |
| extraLargeArray | 772.629625 ms | > | 2.875584 ms |
| largeArray | 6.48075 ms | > | 419.25 μs |
| mediumArray | 144.042 μs | > | 81.458 μs |
| smallArray | 37.541 μs | < | 50.667 μs |
| tinyArray | 23.167 μs | < | 45.458 μs |

As expected there was a correlation between the size of the array and how long it took to run. Large arrays took significantly longer to run, when compared to smaller arrays. I had expected the doublerInsert function to be slower in all accounts, since it uses unshift() and it requires each number to move down one space when the new one is added to the front, however this appeared to only be the case with larger arrays. Smaller arrays returned results faster using the douberInsert function than the doublerAppend function.

With this difference aside, I would say that the doublerAppend function is better for scaling purposes, as the push() method tends to have a time complexity of O(1), and the unshift() method tends to have a time complexity of O(n). However, it should be noted that this may not apply to space complexity. While doing some research, I learned that the push() method has the potential of having a space complexity of O(n) because of its dynamic allocation of memory. When you create an array, without specifying the size you need, it will create an array of the default value. Until the default size is full, the compiler has to create a new contiguous block of memory (which is twice the size of the default memory) and copy the already existing elements to a newly allocated memory.