| | XL Array | Large Array | Medium Array | Small Array | Tiny Array |
|---|---|---|---|---|---|
| Append | 3.74384 ms | 691.227 µs | 171.332 µs | 110.646 µs | 101.841 µs |
| Insert | 1.58274 0674 s | 11.917341 ms | 190.078 µs | 54.868 µs | 47.075 µs |

The doublerAppend function takes the array, multiplies each item by 2, then pushes the new number to the end of a new array. The run time of a single push does not change as the length of the array increases. The doublerAppend function's run time is perfectly linear O(n).

The doublerInsert function uses the unshift method, which adds each modified number into the front of the array. Every time unshift occurs, every value in the array will increase their index value by one, then one index value will be added, and index 0 will change to the new number. Anything previously in index 0 gets moved to index 1, as with every following index in order. This is O^n run time.

You could compare these two functions to how people sit on a bench, assuming people will sit in a left-to-right order. The push method in the doublerAppend function places all new people on the right. It takes no time at all, because no one else has to move for a new person to sit down. In the doublerInsert function, the unshift method asks everyone currently sitting on the bench to move one seat to the right, so the new person can sit on the left. This creates an exponentially increasing time for the new person to sit down. The amount of people that would have to move every time a new person wants to sit down shows the inefficiency in this method. It takes longer to get all the same people on a bench using an unshift method over a push method, proving the push method scales better. (I worked at a theme park in the past and can't help but have flashbacks to loading people onto rides quickly so ride time efficiency would be high and wait time would not be affected)