|  | **doublerAppend** | **doublerInsert** |
| --- | --- | --- |
| **tinyArray** | 100.9 μs | 48.4 μs |
| **smallArray** | 107.3 μs | 62.3 μs |
| **mediumArray** | 152.1 μs | 201.2 μs |
| **largeArray** | 595.8 μs | 10.2168 ms |
| **extraLargeArray** | 3.2586 ms | 1.1481922 s |

I was very surprised to see these results. I knew inserting was slower, but these results show that it’s only slower for the arrays bigger than “smallArray”. It actually performed faster for tinyArray and smallArray.

Another, less surprising, result is that insert gets exponentially slower as the array size increases. Append might be the slightest bit slower for small arrays, but it scales much better for larger arrays. It was approximately 352,000 times faster given the extraLargeArray.

The reason insert is so slow is because of how arrays are stored in the computer’s RAM. They are stored as stacks in blocks of space in memory that the system allocates for them. Insert adds a new element to the first position of the array, which requires all existing elements in the array to shift over a position. Shifting every existing element in the array one space in memory over one by one is very inefficient. Instead, insert simply reallocates a new block of space in memory for a new array, adds the new element to the first position, then copies the rest of the array over from the old block of memory to the new block. Append is faster because all it has to do is add a new element to the end of the stack in memory. It only occasionally has to copy the array to a new, bigger block of memory when enough items are added that the old block of allocated memory runs out. Less allocating and copying means faster performance.