|  | .push runtimes | .unshift runtimes |
| --- | --- | --- |
| tinyArray | 0.13 ms | 0.06 ms |
| smallArray | 0.14 ms | 0.07 ms |
| mediumArray | 0.2 ms | 0.24 ms |
| largeArray | 0.71 ms | 12.3 ms |
| extraLargeArray | 4.1 ms | 1400.4 ms |

The functions had comparable runtimes with the tiny, small, and medium arrays, but the function using unshift fell behind with large arrays, and was so much slower with the extra large array that it’s not even comparable. The push method scaled significantly better than the unshift method.

I think the reason for this is because when an array, or stack, is created it is allocated space in memory. When you use the push method it simply adds that element to the end of the stack. With larger arrays, the allocated space in memory might not be enough and the system will have to create a new space in memory then copy the array over to continue. With the unshift method, it’s trying to add to the start of the stack, but that spot is already taken. So, instead it adds the new element to a new stack, and copies over the old stack, which is super inefficient with larger arrays. So essentially, the push method scales better because it only rarely has to reallocate memory and copy over, while the unshift method has to do it every time it adds an element to the array.