Runtime Assessment

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| array: | extraLargeArray | largeArray | mediumArray | smallArray | tinyArray |
| insert: | 1.0888752 s | insert 9.2384 ms | insert 189.2 μs | insert 105.8 μs | insert 38.3 μs |
| append: | 2.6948 ms | append 531.7 μs | append 182 μs | append 112.8 μs | append 142.6 μs |

Starting with extraLargeArray, insert function takes significantly longer than the append function, but as we shrink the size of the array being processed this changes to where the insert function starts to become faster than the append function. The append function scales significantly better because as we increase the arrays the time it takes to run through its function is significantly less than the insert function. I tested this with an extraExtraLargeArray and insert took 2.17 min to finish, where append only took 19.9114 ms. This is a significant difference between the two.

After testing this on python tutor and understanding how unshift vs push works I now understand the reason these two methods scale differently. This is because the append method pushes the even numbers into an array at the end of the array. The insert method uses unshift, which takes the even numbers and shoves it into the first element (array[0]) of the array and then all the other numbers have to shift to the right, for ten numbers this shift is small, but the process scales exponentially as we use larger numbers.