**Assessment-CS**

Step 1: Runtime Analysis

Results for the extraLargeArray

doublerAppend 2.334917ms

doublerInsert 776.946208 ms

|  |  |  |
| --- | --- | --- |
|  | Append uses .push | Insert uses .unshift |
| tinyArray | 84.459us | 23.041us |
| smallArray | 73.834us | 32.916us |
| mediumArray | 141.208us | 170.625us |
| largeArray | 534.083us | 8.3445ms |
| extraLargeArray | 3.522958ms | 779.007708ms |

All functions seem to scale better with push because no matter how much data, it will execute in constant time. All functions seem to get slower with unshift because of the amount of data that is sent in. Unshift is trying to achieve more steps at a time than push. Unshift is adding more work to get the same results done. Like adding 2+2+2+2 instead of just multiplying 2x4 both get the same result. Push is multiplying and unshift is adding.

Push is always going to run faster. All it does is add an element and gives it an index that’s one greater than the index of the last element in the array. So no matter the size it will be O(1). Unshift adds new elements to the beginning of the array. Unshift overwrites the original array. Unshift is slower than push because it also needs to unshift all the elements to the left once the first element is added. Unshift has a linear time complexity and is O(n).

Push of course in this example is going to scale much faster than unshift which is slower. You can tell by how the numbers react as they are increasing in size of the arrays. Readability is more important. It is more favorable to use push which is O(1) giving steady input no matter the size of the array. Push(turtle) runs at the same rate the entire race. And Unshift(rabbit) runs fast at first and then slows down through out the race.