Runtime Analysis

node runtime.js results of **extraLargeArray:**

```
Wade@LAPTOP-E2UQ95U9 MINGW64 ~/Desktop/BaseCamp/w7-DevOps&ComputerScience/5Fri/cs
$ node runtime.js
Results for the extraLargeArray
insert 1.1224652 s
append 3.4074 ms
```

node runtime.js results of **largeArray:**

```
Wade@LAPTOP-E2UQ95U9 MINGW64 ~/Desktop/BaseCamp/w7-DevOps&ComputerScience/5Fri/cs
$ node runtime.js
Results for the largeArray
insert 9.3791 ms
append 612.5 µs
```

node runtime.js results of **mediumArray:**

```
Wade@LAPTOP-E2UQ95U9 MINGW64 ~/Desktop/BaseCamp/w7-DevOps&ComputerScience/5Fri/cs
$ node runtime.js
Results for the mediumArray
insert 199.5 µs
append 154.4 µs
```

node runtime.js results of **smallArray:**

```
Wade@LAPTOP-E2UQ95U9 MINGW64 ~/Desktop/BaseCamp/w7-DevOps&ComputerScience/5Fri/cs
$ node runtime.js
Results for the smallArray
insert 54.5 µs
append 108.8 µs
```

node runtime.js results of **tinyArray:**

```
Wade@LAPTOP-E2UQ95U9 MINGW64 ~/Desktop/BaseCamp/w7-DevOps&ComputerScience/5Fri/cs
$ node runtime.js
Results for the tinyArray
insert 40.3 µs
append 95.6 µs
```

The append function scales better. It is a constant type - O(1), this is because it is always adding to the last index of the array, where as the insert function is a linear - O(n) type. The insert function is adding to the front of the array, changing the index of every item after it - the larger the array the more items are shifted in that array.