Unit 7 Assessment

David Lay - 10/7/23

Step 1 - Runtime Analysis

Comparing time complexity for the following ‘doubler’ functions:

```
function doublerAppend(nums) {
  let new_nums = [];
  for (let i = 0; i < nums.length; i++) {
    let num = nums[i] * 2;
    new_nums.push(num);
  }
}

function doublerInsert(nums) {
  let new_nums = [];
  for (let i = 0; i < nums.length; i++) {
    let num = nums[i] * 2;
    new_nums.unshift(num);
  }
}
```

Findings over 2 iterations (each array size ran twice for each function)

Run	ArraySize	quantity	InsertRuntime (ms)	AppendRuntime (ms)	Append is Faster	Rate
1	tiny	10	0.149	0.376	False	0.40
2	tiny	10	0.162	0.377	False	0.43
3	small	100	0.173	0.340	False	0.51
4	small	100	0.137	0.249	False	0.55
5	medium	1000	1.053	0.513	True	2.05
6	medium	1000	0.541	0.424	True	1.27
7	large	10000	18.360	1.381	True	13.29
8	large	10000	24.837	1.823	True	13.62
9	extraLarge	100000	1745.429	7.200	True	242.42

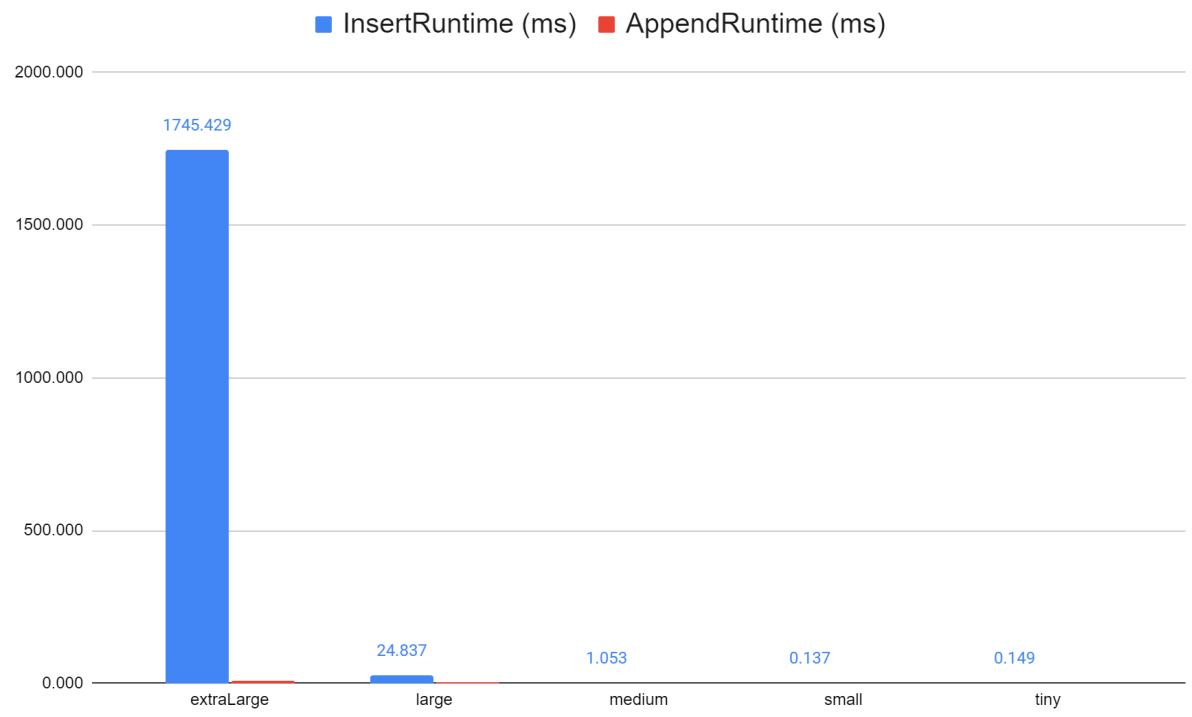
10	extraLarge	100000	1779.831	6.623	True	268.72

The Append function clearly scales better than the Insert function.
Observations:

- At a 'tiny' to '*small'* size the **Insert** function will run measurably faster, but the difference is negligible.
- Once the array size reaches medium, the The **Append** function begins to take the lead, but the difference is still negligible
- At the *'Large'* size the difference in scale becomes clear with *'ExtraLage'* making the advantage of the **Append** function obvious.
- The **Append** function will run about 250 times faster than the **Insert** function over an array of 100000 elements.
- At scale, the **Insert** function will begin to eat up seconds, which can be quite significant given today's computing capabilities.

Conclusion: Append will scale, running exponentially faster than Insert at large volume.

Graphic - runtime over volume, high to low.
This graph shows the runtime in **blue** for the **Insert** function and the **Append** in **red**. The runtimes for Append are barely visible when shown in proportion to the runtimes for Insert at large volume.



Reason: *Why is 'Append' better in this case?*

The doublerInsert function requires additional actions for each element in the array as it loops through because it shifts all existing elements to as it adds the new ones at the beginning.
The doublerAppend function simply adds a new element to the end of the existing array without needing to modify the elements already present.
Consequently, doublerAppend has a time complexity of *O(n)*, which is linear, while the doublerInsert function has a time complexity of *O(n^2)*, which is exponential. The exponential complexity occurs because, as the number of elements increases, the time required to process them increases by a factor of *n*; the Insert function requires that action is taken on all elements in the array as it loops over the array.