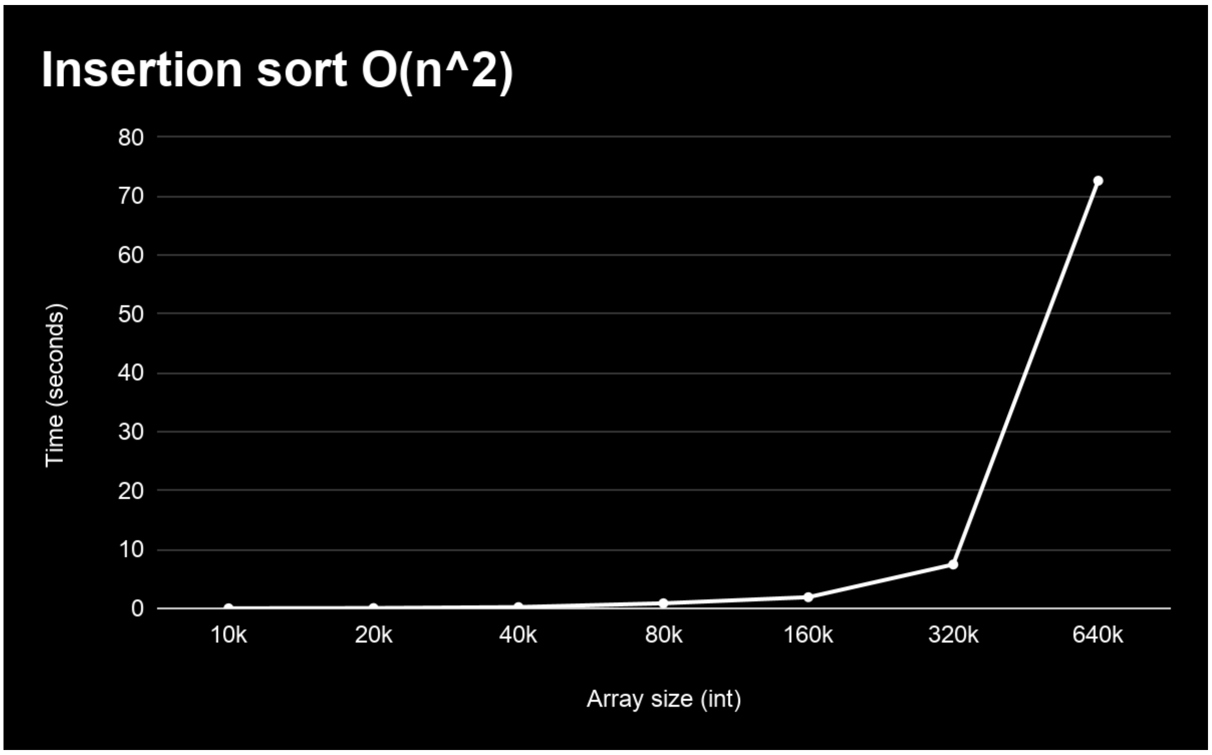
Elementary Sorting Analysis:

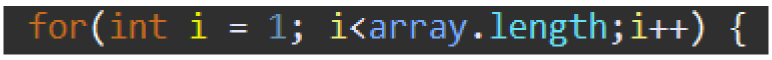
Insertion Sort:

Array size	Time(seconds)
10k	0.014
20k	0.055
40k	0.219
80k	0.866
160k	1.902
320k	7.456
640k	72.647

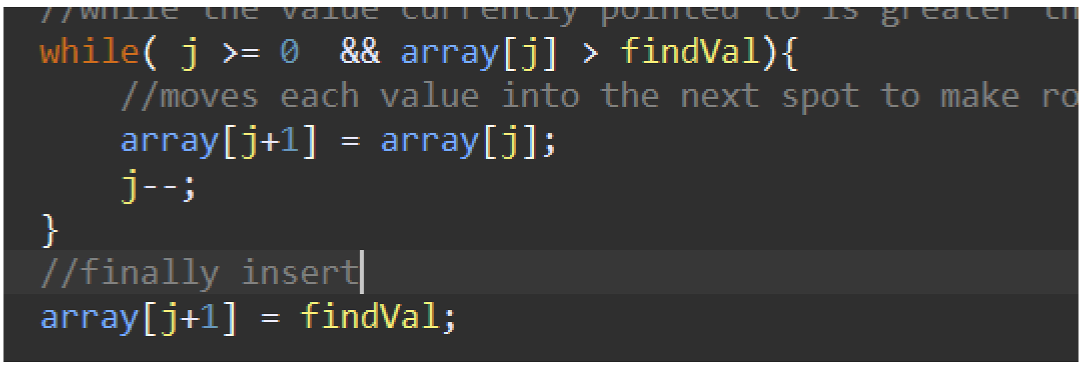
Graphing results:



From our results we can assume that insertion sort has a time complexity of -(n^2), However to show this we need to inspect how insertion sort is executed.



```
for(int i = 1; i<array.length;i++) {
```

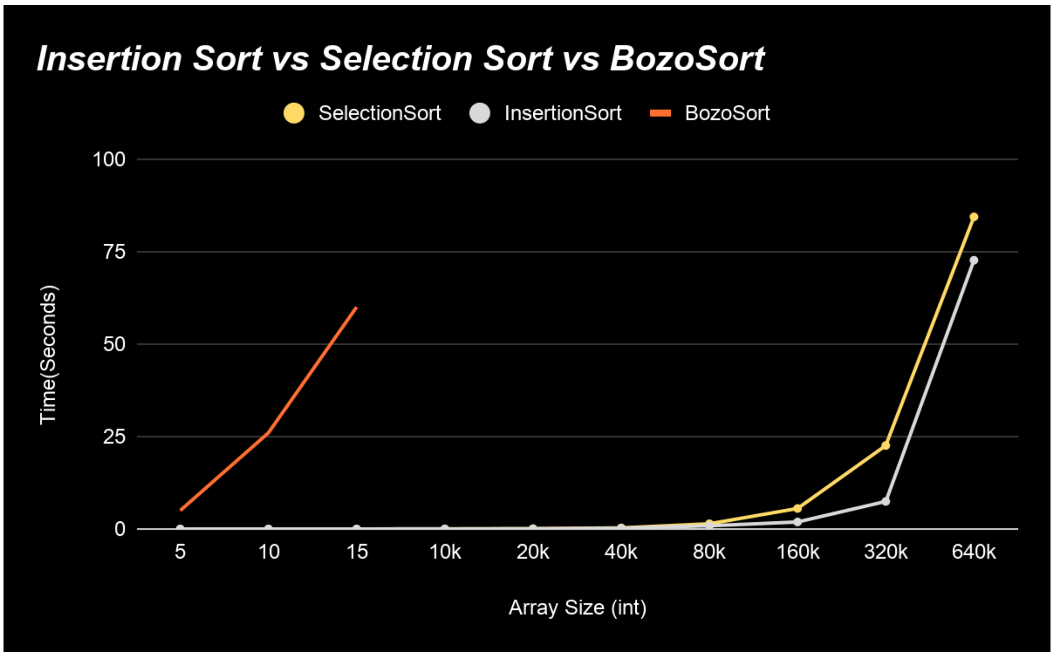


```
//while the value currently pointed to is greater th
while( j >= 0  && array[j] > findVal){
    //moves each value into the next spot to make ro
    array[j+1] = array[j];
    j--;
}
//finally insert
array[j+1] = findVal;
```

As seen in the code above insertion sort holds 1 invariant- that any value before i is in ascending order, as such everytime we find a mismatch, we have to execute up to i-1 comparisons to find the right space for the element. This leads to 2 cases
First case: Each time a comparison is made and the element is moved the whole to the back

Second case: The array is in sorted order and as such insertion sort only checks n elements, giving a time complexity of O(n) this is why insertion sort is useful for smaller nearly sorted arrays compared to something like merge or quick sort.

Array size	SelectionSort	InsertionSort	BozoSort
5	0.0000003	0.00000021	5
10	0.0000014	0.00000022	26
15	0.0000031	0.0000001	60
10k	0.026	0.014	
20k	0.087	0.055	
40k	0.309	0.219	
80k	1.415	0.866	
160k	5.554	1.902	
320k	22.57	7.456	
640k	84.37	72.647	



As seen selection sort runs considerably slower than insertion sort for the simple fact that selection sort will make more comparisons as it moves through the array, with insertion sort the number of comparisons are minimised due to its invariant. This also demonstrates that 2 algorithms with the same time complexity don't necessarily run at the same speed.

BozoSort is also incomparable to either of the sorts where in the first 15 integers it runs just as slow as insertion sort at 640k integers. BozoSort runs with a time complexity of O(n!)
As there are n! Possible options for the array.