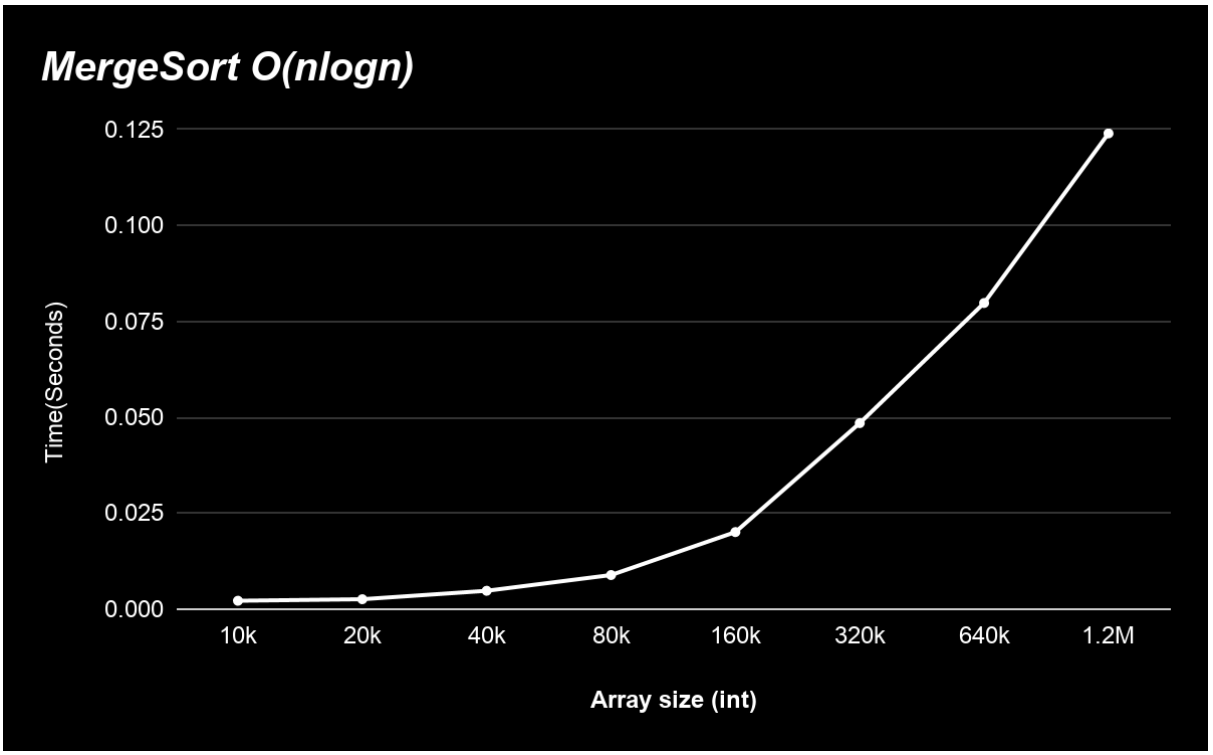
MergeSort Analysis:

Normal MergeSort:

Array Size	Time(S)
10k	**0.0022**
20k	**0.0026**
40k	**0.0048**
80k	**0.0089**
160k	**0.0201**
320k	**0.0485**
640k	**0.0798**
1.2M	**0.124**



As seen from the graph above we can assume that merge sort runs with a linear/linearithmic giving us a result of ~ nlogn. However to prove this we need to take a closer look at the code of mergeSort.

```
split(leftArray,rightArray,array,mid,hi);

MergeSort(leftArray,mid);
MergeSort(rightArray,hi-mid);
```

MergeSort relies on the principle of divide and conquer where the problem is broken down into sub-problems which can then be calculated easier.
Since mergeSort splits its array into subArrays each call we can assume that the total amount of splits is bounded by O(logn)

```
while(i < mid && j < hi) {
    //increments based on largest value in each subarray
    if(leftArray[i] <= rightArray[j]) {
        a[k++] = leftArray[i++];
    }
    else {
        a[k++] = rightArray[j++];
    }
}
```

However we must also combine these 2 arrays back into 1 array meaning we have to do n comparisons in order to fill the array.
As such we get a time complexity of O(nLogn).

Comparing Enhanced normal and insertion sort:

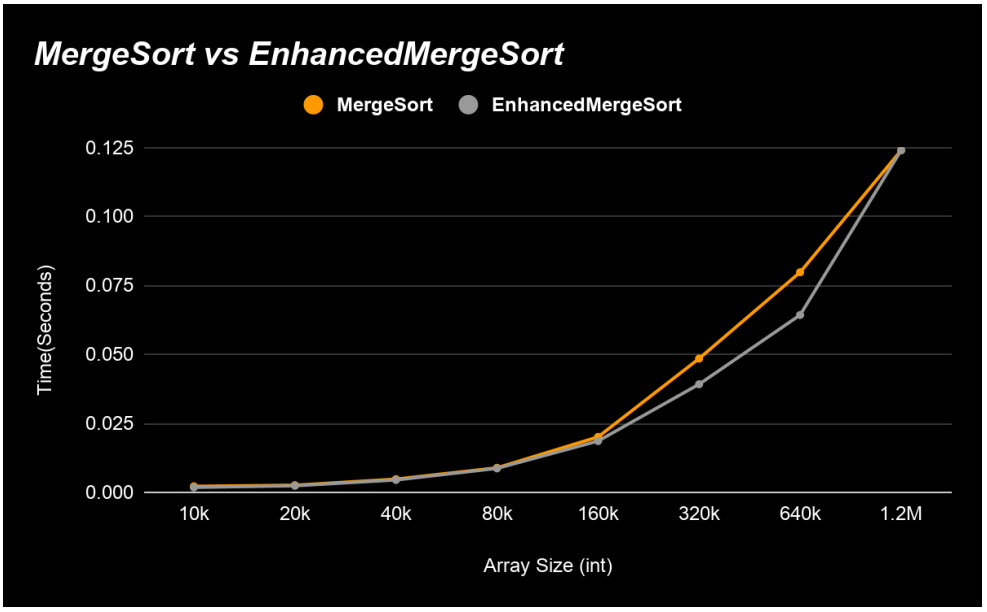
Array Size	MergeSort	EnhancedSort	InsertionSort
10k	**0.0022**	**0.0018**	**0.014**
20k	**0.0026**	**0.0024**	**0.055**
40k	**0.0048**	**0.0045**	**0.219**
80k	**0.0089**	**0.0087**	**0.866**
160k	**0.0201**	**0.0186**	**1.902**
320k	**0.0485**	**0.0392**	**7.456**
640k	**0.0798**	**0.0643**	**72.647**
1.2M	**0.124**	**0.124**	**unknown**

Enhanced mergeSort makes 2 main improvements the first being a cutoff for insertion sort,Since mergeSort has overhead for creating and splitting arrays,subarrays of length 10 and less are calculated using insertion sort

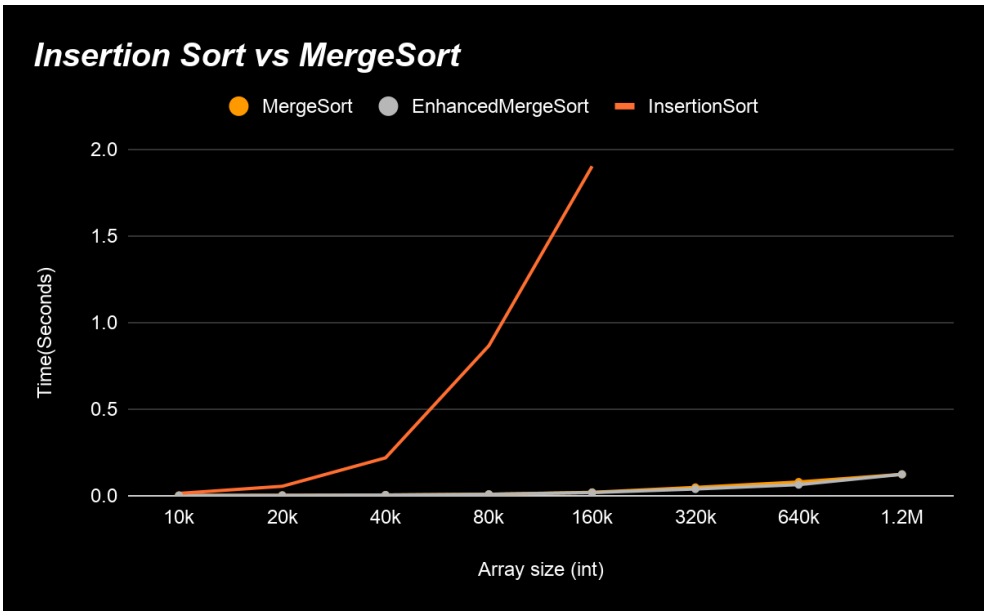
```
if(array.length <= CUTOFF) {
    insertionSort(array,0,array.length);
}
```

We also employ the fact that if the last element of the left array is less than the leftmost element of the right array then we have no need for running the merge step.

```
if(leftArray[mid-1] > rightArray[0]) {
merge(array,leftArray,rightArray,mid,hi-mid);
}
```



As seen from the graph there isn't a drastic improvement, as in some cases neither of the improvements will be used to their full extent,however we can see there are marginal improvements.



Finally we by comparing insertion sort O(n^2) with mergeSort o(nlogn) we see the disparity between linear and quadratic growth functions as insertion sort is over 20 times slower than MergeSort for 80k ints than mergeSort at 1.2Million ints.