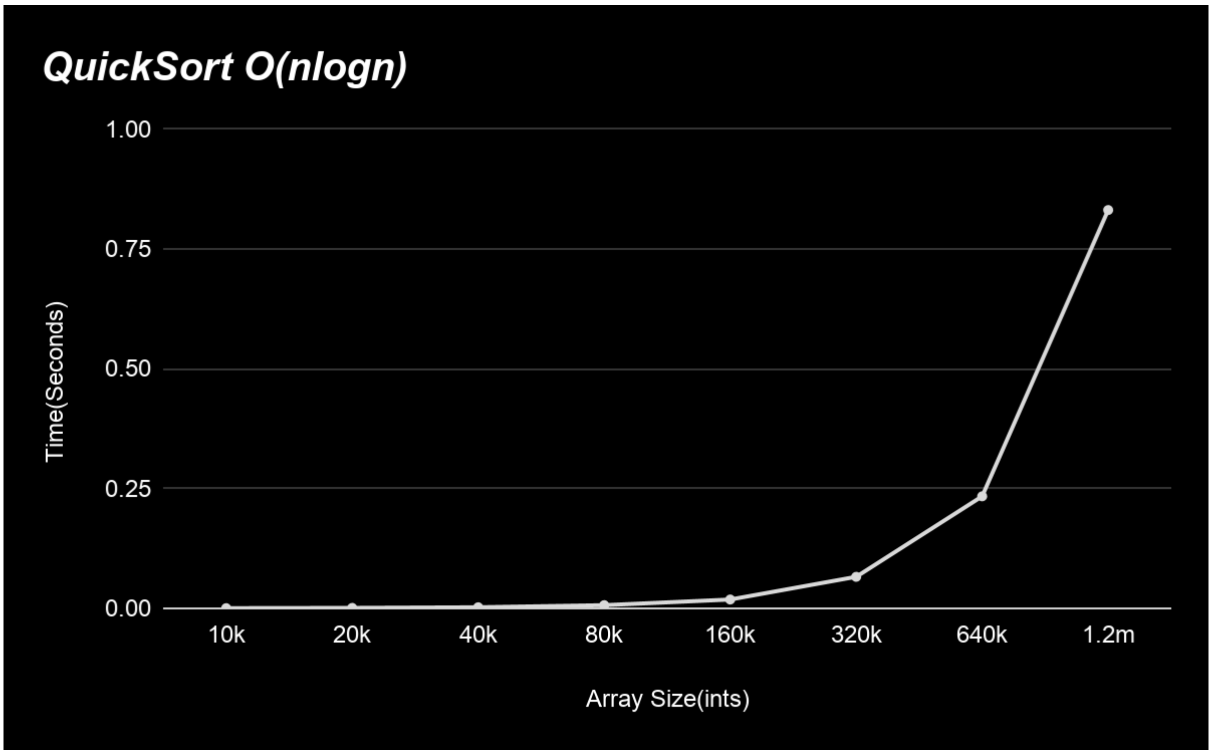
QuickSort Analysis:

Normal QuickSort:

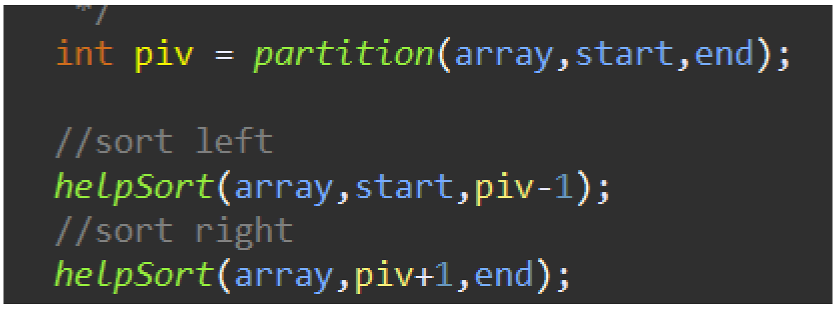
	QuickSort
10k	0.000478
20k	0.00102
40k	0.00234
80k	0.006714
160k	0.0185
320k	0.0659
640k	0.234
1.2m	0.83166



As seen from the graph above we can assume that Quick 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.

On closer inspection of QuickSort we can see that the actual worst case time complexity is O(n^2) however this is almost entirely never the case. What is the reason for this?

Well we need to figure out the average case growth function first, similar to mergeSort QuickSort uses divide and conquer to split the array into sub-problems



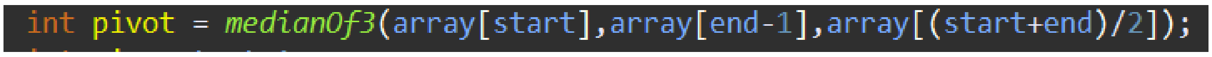
```
    */
    int piv = partition(array,start,end);

    //sort left
    helpSort(array,start,piv-1);
    //sort right
    helpSort(array,piv+1,end);
```

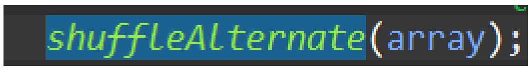
We see that if we assume the partition is relatively close to the centre of the array, then we execute logn steps to split the array, the partition function also runs at least n-1 comparisons giving us a time complexity of O(nlogn)

However if we think of the case where the partition is at one extreme side of the array at each partition then we will have n partitions and n comparisons meaning O(n^2) worst case time complexity.

Enhanced QuickSort:



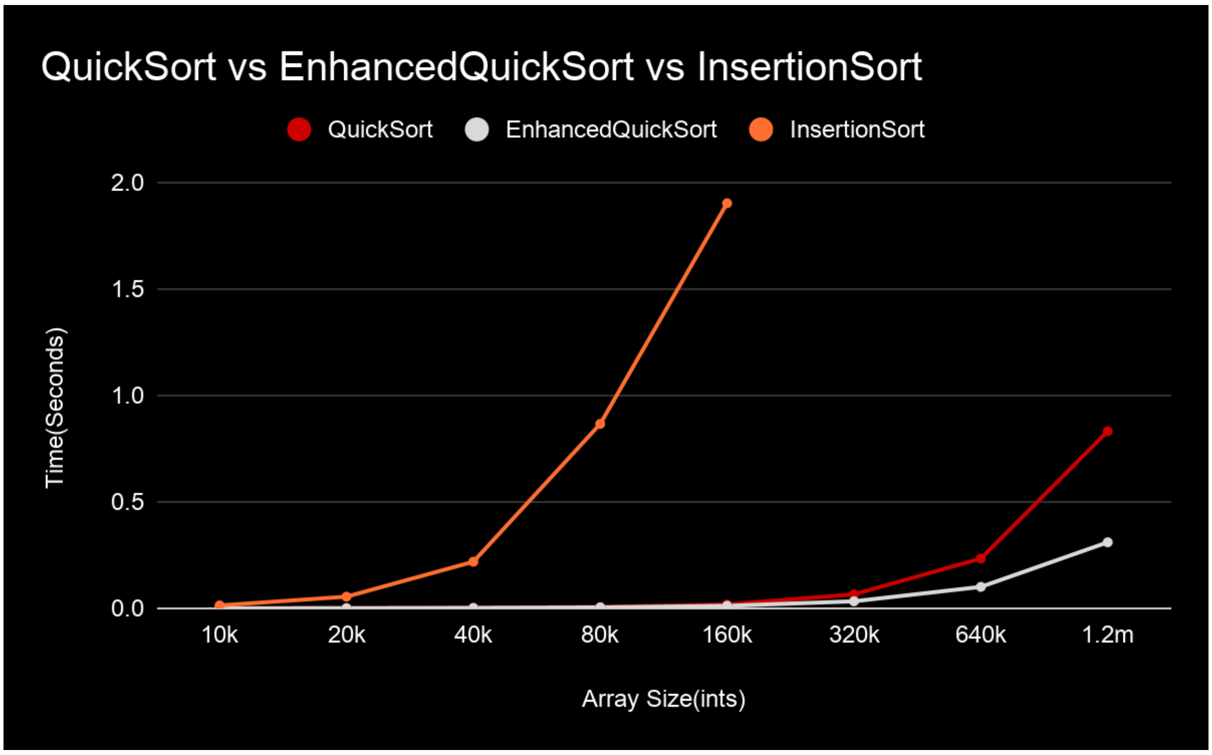
```
int pivot = medianOf3(array[start],array[end-1],array[(start+end)/2]);
```



```
shuffleAlternate(array);
```

Enhanced QuickSort uses a shuffle and a median of 3 to ensure that this case never happens. It also uses insertion sort similar to mergeSort to help with smaller arrays less than 10 elements.

	QuickSort	EnhancedQuickSort	InsertionSort
10k	0.000478	0.000446	0.014
20k	0.00102	0.0009	0.055
40k	0.00234	0.00212	0.219
80k	0.006714	0.00474	0.866
160k	0.0185	0.011918	1.902
320k	0.0659	0.03326	7.456
640k	0.234	0.101	72.647
1.2m	0.83166	0.310321	unknown



As seen, EnhancedQuickSort helps to cut down on uneven partitions giving quite a large difference in timing. This also shows that even if 2 algorithms have the same big O time Complexity, one still may be faster than the other.