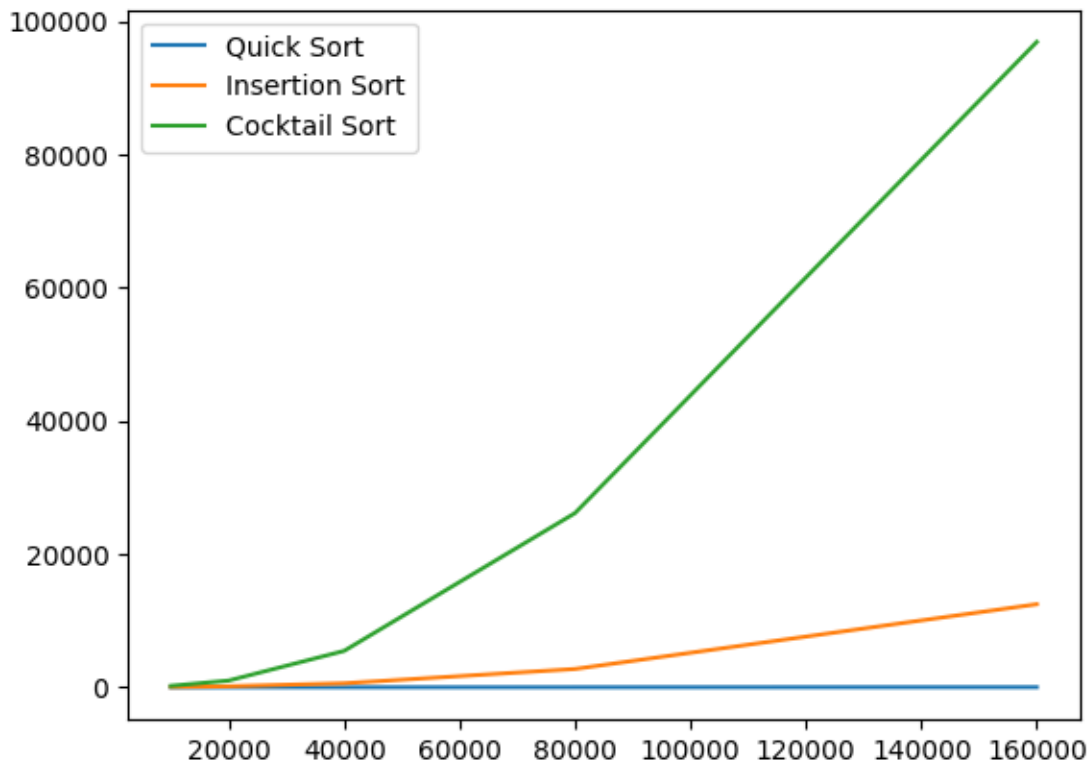
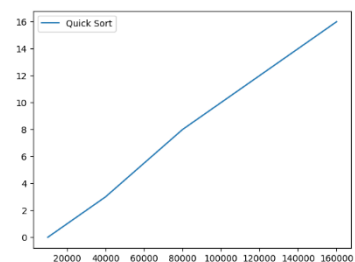
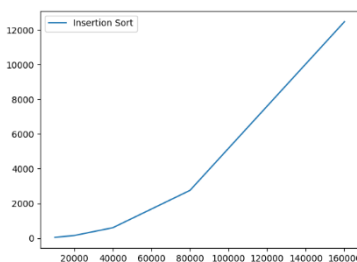
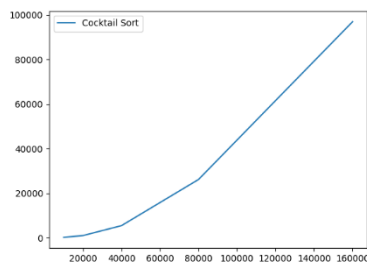


PA5 Part 2 Runtime Analysis

Insertion Sort vs Cocktail Sort vs Quicksort



We can see that in terms of performance, Cocktail sort was the slowest followed by Insertion sort and then by Quick Sort. Due to the skewed graph it is hard to tell the time complexity of the different sorts therefore here are the sorts individually.



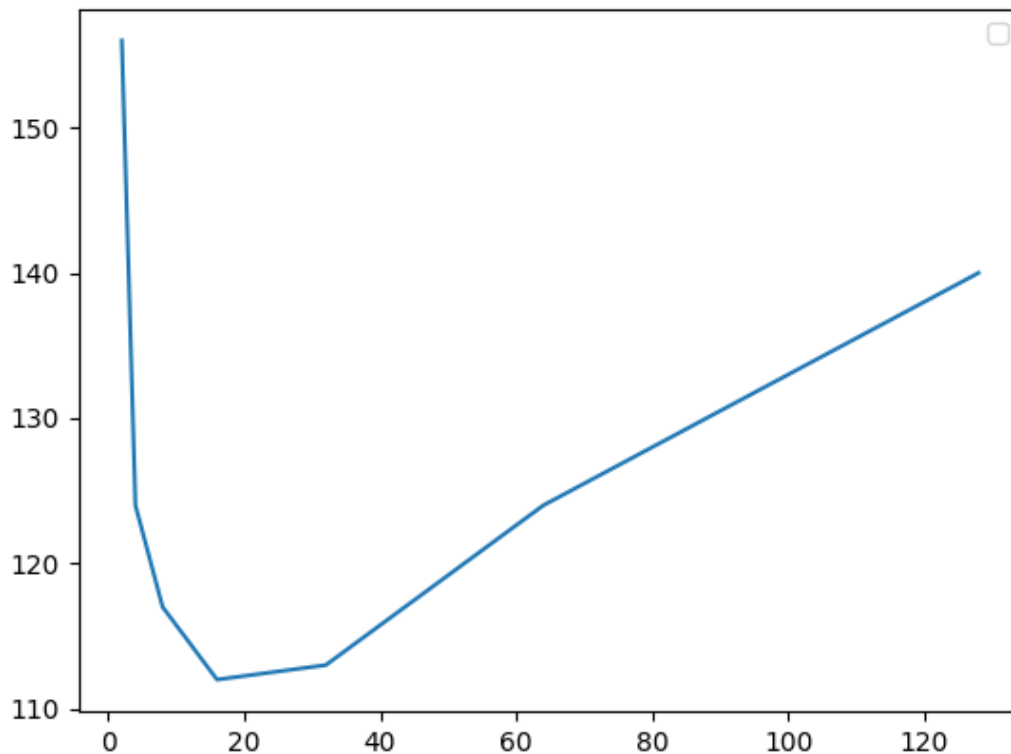
Looking at the graphs we can see that Cocktail sort has a time complexity of $O(n^2)$, while insertion sort has a time complexity of $O(n^2)$ and quick sort has a time complexity of $O(n \log n)$. However, although both insertion sort and Cocktail sort have similar time complexities, my reason as to why Cocktail

performs a lot more poorly in terms of runtime than insertion sort, is due to the number of operations that Cocktail sort has compared to insertion sort. As Cocktail sort swaps each pair where the element on the right is smaller than the element on the left. While insertion sort starts of sorted and inserts the element at it is sorting at the iteration to the right position immediately, therefore there isn't a lot of extra unneeded swaps. Therefore, causing the discrepancy in the runtime.

Data got for each sort:

```
insertion_sort_time = [44, 149, 598, 2752, 12477] #10
quick_sort_time = [0, 1, 3, 8, 16] #1000
cocktail_sort_time = [215, 1009, 5464, 26163, 96938] #5
no_data = [10000, 20000, 40000, 80000, 160000]
```

Modified Quicksort with different cutoff values



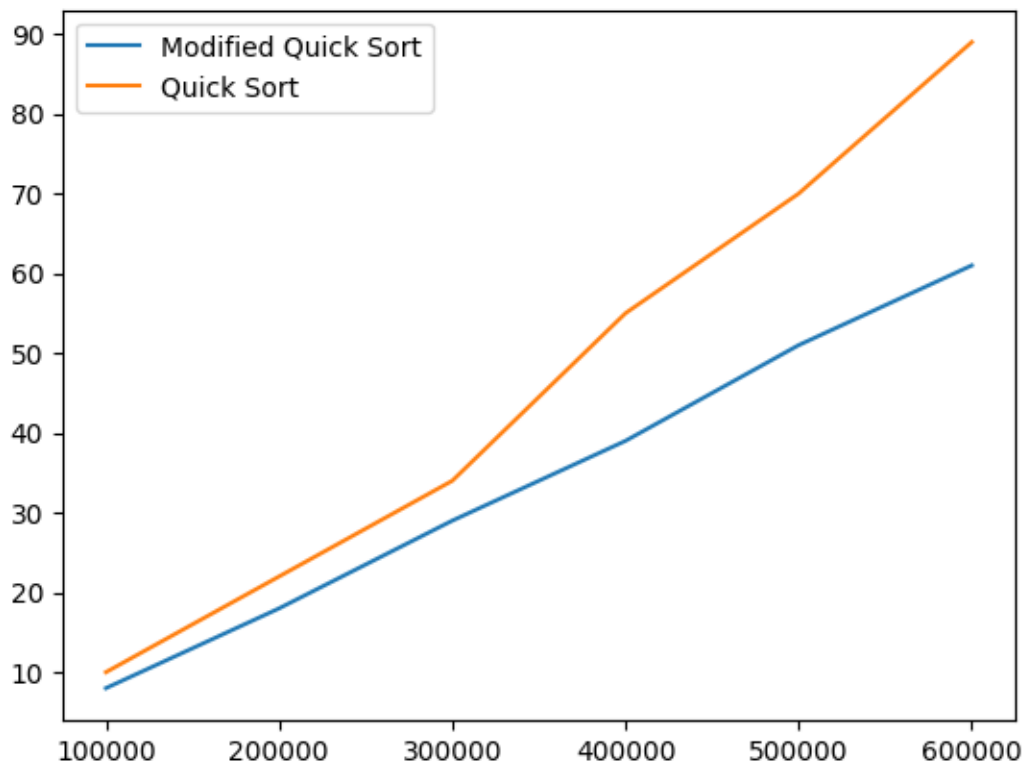
Looking at the graphs we see a initial steep decline in terms of time taken to sort the data, then after hitting a approximate minimum, it starts a steady incline. Which tells us that the modified sort has

a best cutoff values in approximately the 16-32 range where switching to an insertion sort greatly benefits the sorts.

Data collected and used for the graph:

```
modified_quick_sort_times = [156, 124, 117, 112, 113, 124, 140]  
cutoffs = [2, 4, 8, 16, 32, 64, 128]
```

Traditional Quicksort vs Modified Quicksort



For the cutoff values of the Modified quicksort, I used 20 as the cutoff as that seems as close to the minimum point achieved earlier. Both sorts achieve approximately a time complexity of $O(n \log n)$ as a large majority of the sorts that happen in the modified quicksort happens in a quicksort form. Overall however, we can see that the modified quicksort performed better than the traditional quicksort for all the data test that we had and as the data set got larger, the gap in terms of speed also started to increase which could mean that as the dataset gets large, the performance of the Modified quicksort will progressive get better than a traditional quicksort.

Data used and collected for the Runtime analysis:

```
quick_sort_time2 = [10, 22, 34, 55, 70, 89]  
modified_quick_sort_times2 = [8, 18, 29, 39, 51, 61]  
no_data = [100000, 200000, 300000, 400000, 500000, 600000]
```