Johnny Li

Project 2: Sorting Analysis

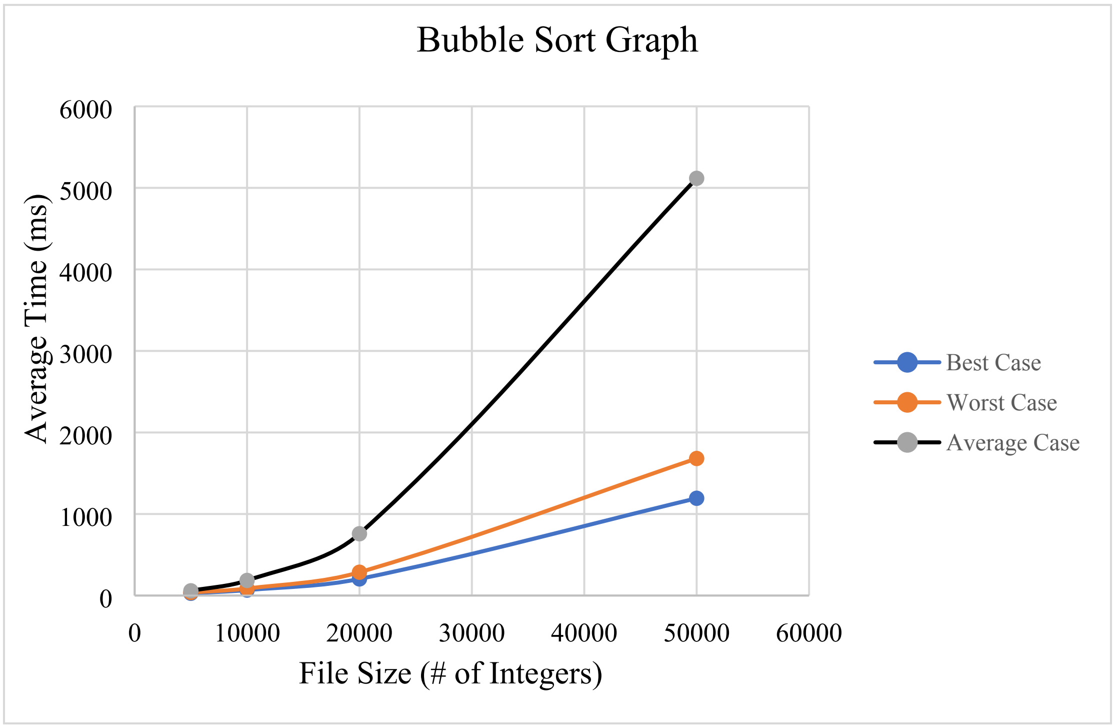
COP 3530 – Data Structure

Sorting Analysis

Sorting Method: Bubble Sort

Bubble Sort is a sorting algorithm that works by repeatedly swapping the adjacent elements if they are in the incorrect order.

Bubble Sort Table														
	Best Case					Worst Case					Average Case			
File Size (# of Integers)	5k	10k	20k	50k		5k	10k	20k	50k		5k	10k	20k	50k
Average Time (ms)	28	67	204	1193		37	86	286	1681		58	185	758	5117



From the data obtained, the relative runtime of the test cases was somewhat as anticipated. The list of integers in ascending order (best case) was sorted the fastest, followed by the list of integers in descending order (worst case), and lastly the list of integers in random order (average case). As seen in the graph, the best case performed the fastest due to the "presorted nature" of the list of integers where the sorting algorithm only needs to check the value in the list, no sorting was required thus saving time. It was unexpected that the worst case was to perform better than the average case, especially by such large margins. For the worst case, which should have taken the longest because it is in reverse order of the ideal sort, the sorting algorithm would have to swap every integer resulting in the maximum number of swaps to sort the given list. Yet, the worst case was the second fastest as the average case took a considerably long time to sort its list. For the average case, not all the integers were out of order, it did not need the maximum number of swaps, therefore less time was needed. This unanticipated outcome of the average case is most likely due to branch prediction where the program tries to guess which way a branch will go before this is known definitively, therefore, causing additional processes being done which results in taking much more time to sort the list.

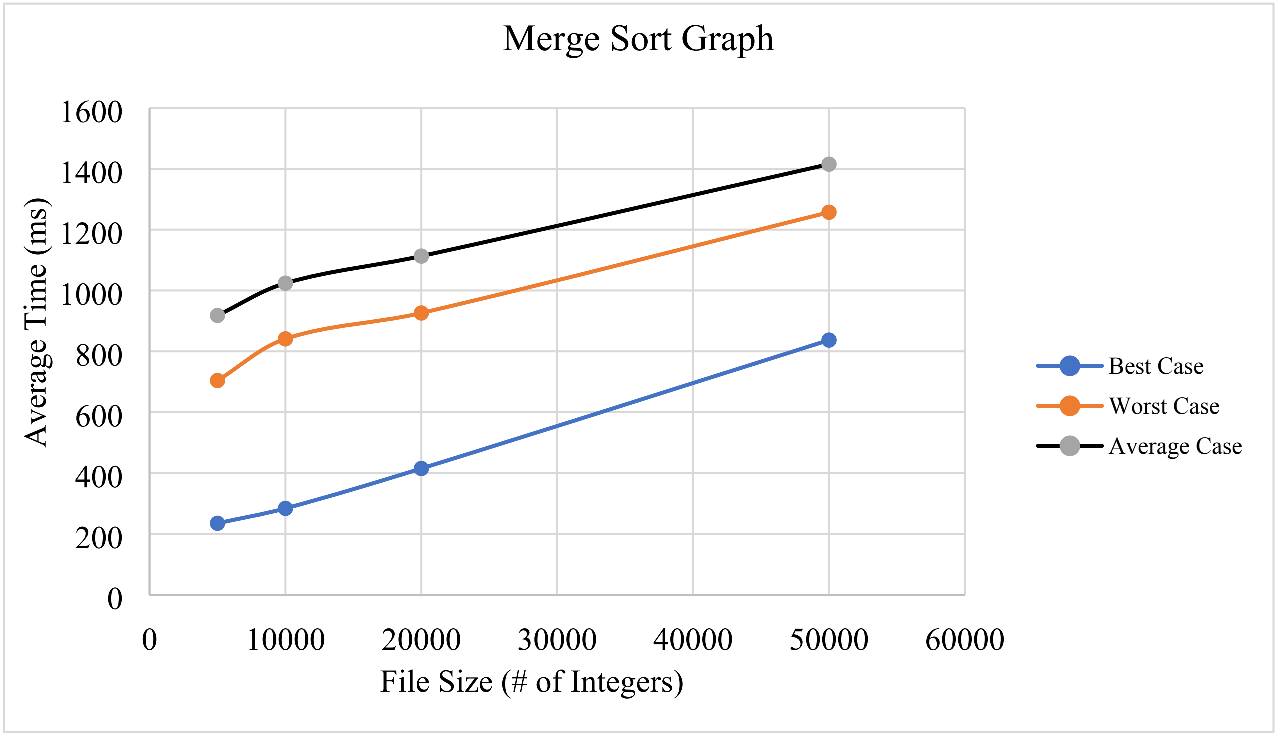
Taken from the graph, it is clear that with an increase of file size there is an increase in the average time taken to sort the files across all cases. However, the relative amount of average time increased in relation to the increase of file size differ for each case. With the worst and best cases, the average time taken to sort seems to be proportional to the smaller file size but scales off in the larger file sizes. With the average case, the average time taken to sort the list grows exponentially with the increasing file size, this is not normal and is most likely due to branch prediction.

The predicted computational complexity of bubble sort is O(n) for the best case and O(n^2) for the average and worst case. The given graph indicates a timing complexity of O(n^2) across all cases where for the average case it is most likely a stretched quadratic function such as O($\frac{1}{2}n^2$). The variation of the complexity in the best case is likely due to the small sample as with the given graph it seems to curve but if the function continues in several more file sizes it would eventually straighten out to be a linear function.

Sorting Method: Merge Sort

Merge Sort is a divide and conquer algorithm that divides integer list in two halves, calls itself for the two halves and then merges the two sorted halves.

Merge Sort Table														
	Best Case					Worst Case					Average Case			
File Size (# of Integers)	5k	10k	20k	50k		5k	10k	20k	50k		5k	10k	20k	50k
Average Time (ms)	235	284	415	837		704	841	926	1257		918	1024	1113	1415



From the data obtained, the relative runtime of the test cases is also somewhat as anticipated. The list of integers in ascending order (best case) was sorted the fastest, followed by the list of integers in descending order (worst case), and lastly the list of integers in random order (average case). As seen in the graph, the best case performed the fastest mainly due to the "presorted nature" of the list of integers where the sorting algorithm only needs to divide the list, check the value in the list, and merge the halves together where no sorting was required thus saving time. Again, it was unexpected that the worst case was to perform better than the average case, but the margins were not as large as they were in bubble sort. For the worst case, which should have taken the longest because it is in reverse order of the ideal sort, the sorting algorithm would have to divide the list to its maximum number of parts, sort through the parts, and merge the sorted parts together, resulting in more time to sort the given list. Yet, the worst case was the second fastest as the average case took a considerably long time to sort its list. As for the average case, not all the integers were out of order, it did not need to divide the list to its maximum number of parts thus reducing the number processes for all subsequent parts, therefore less time was needed to sort. Again, this unanticipated outcome of the average case is most likely due to with branch prediction where it still occurs at each divided half, causes additional processes being done which results in taking much more time to sort the list.

Taken from the graph, it is clear that with an increase of file size there is an increase in the average time taken to sort the files across all cases. However, the relative amount of average time increased in relation to the increase of file size is similar for each case while also having a less extensive growth than that of bubble sort. In all cases, the average time taken to sort seems to be proportional to slight derivations.

The predicted computational complexity of bubble sort is O(nlogn) for all cases. The given graph indicates a timing complexity of O(nlogn) for the average and worst cases but not the best case. The variation of the complexity in the best case, being more linear, is likely due to the small sample as with the given graph it seems to be straight but if the function continues in several more file sizes it would eventually curve out to be a logarithmic function.

In comparison to bubble sort, being O(n^2), merge sort, being O(nlogn), is hypothetically supposed to be faster and more effective. However, the graphs reveal that merge sort takes longer to sort with small file sizes and is significantly faster at sorting large file sizes when compared to bubble sort. This fault at the beginning of the merge sorting time might be the result of the merge function being implemented in a way that makes it slower than it needs to be. The deviations of the relative amount of average time increased in relation to the increase of file size is also significantly reduced in merge sort as compared to bubble sort where the growth rate for merge sort is even and stable while for bubble sort the growth rate is dramatic for large file size. Both sorting algorithms obtained the same runtime results where the best case was the fastest, followed by the worst case, and lastly the average case.