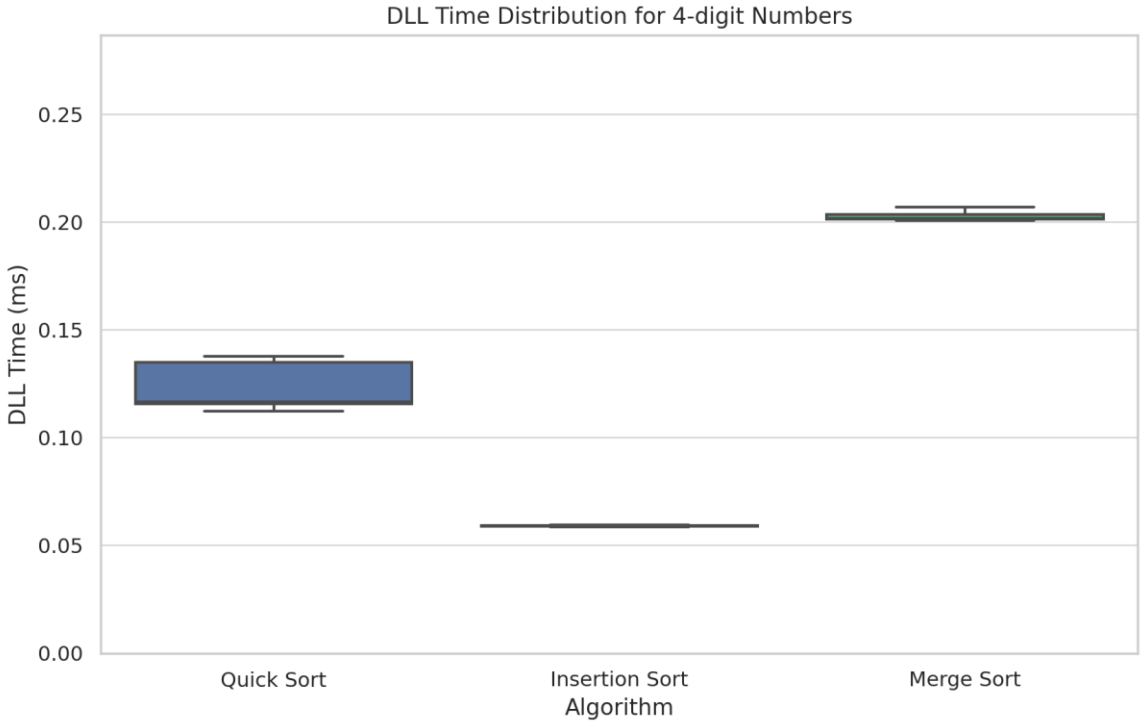
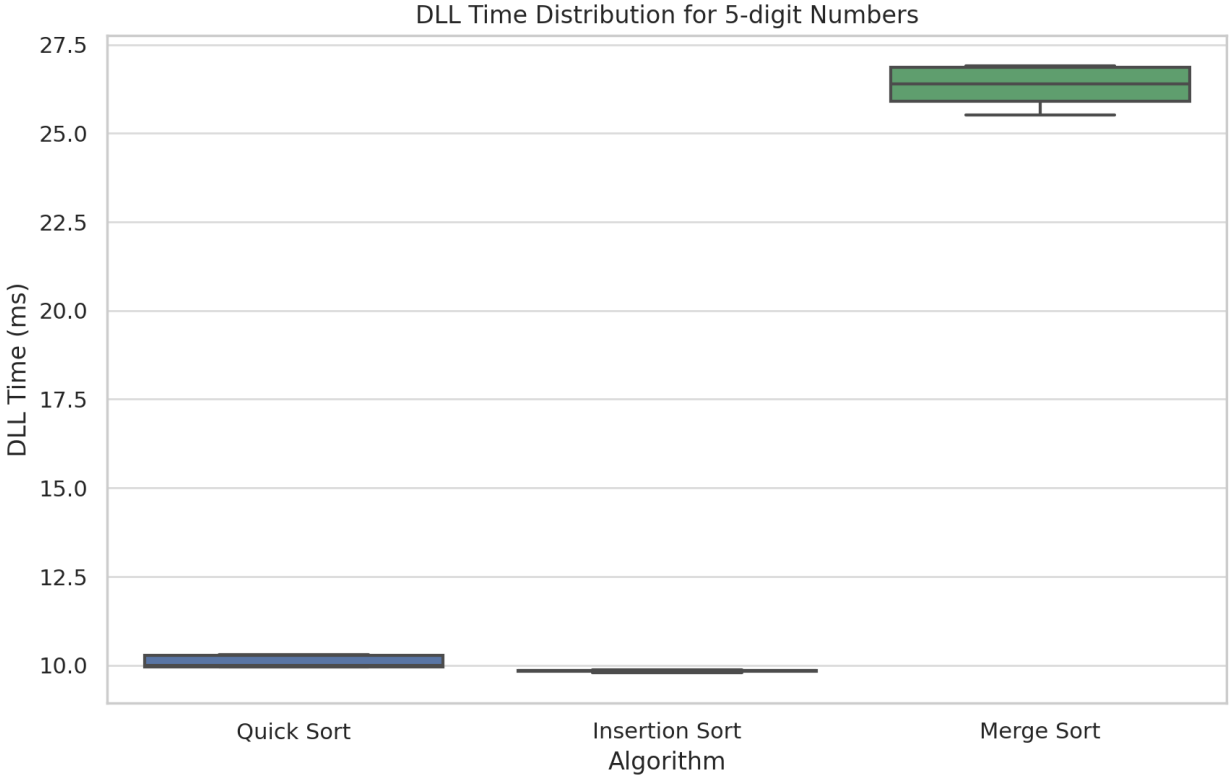
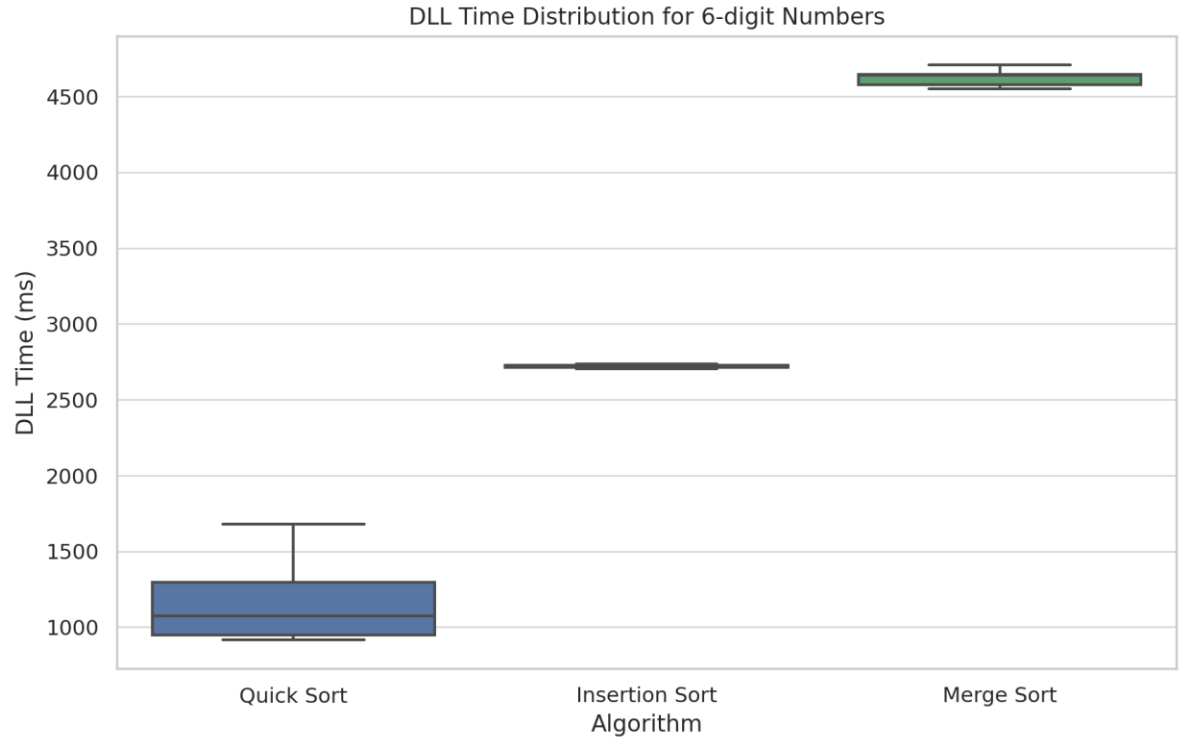
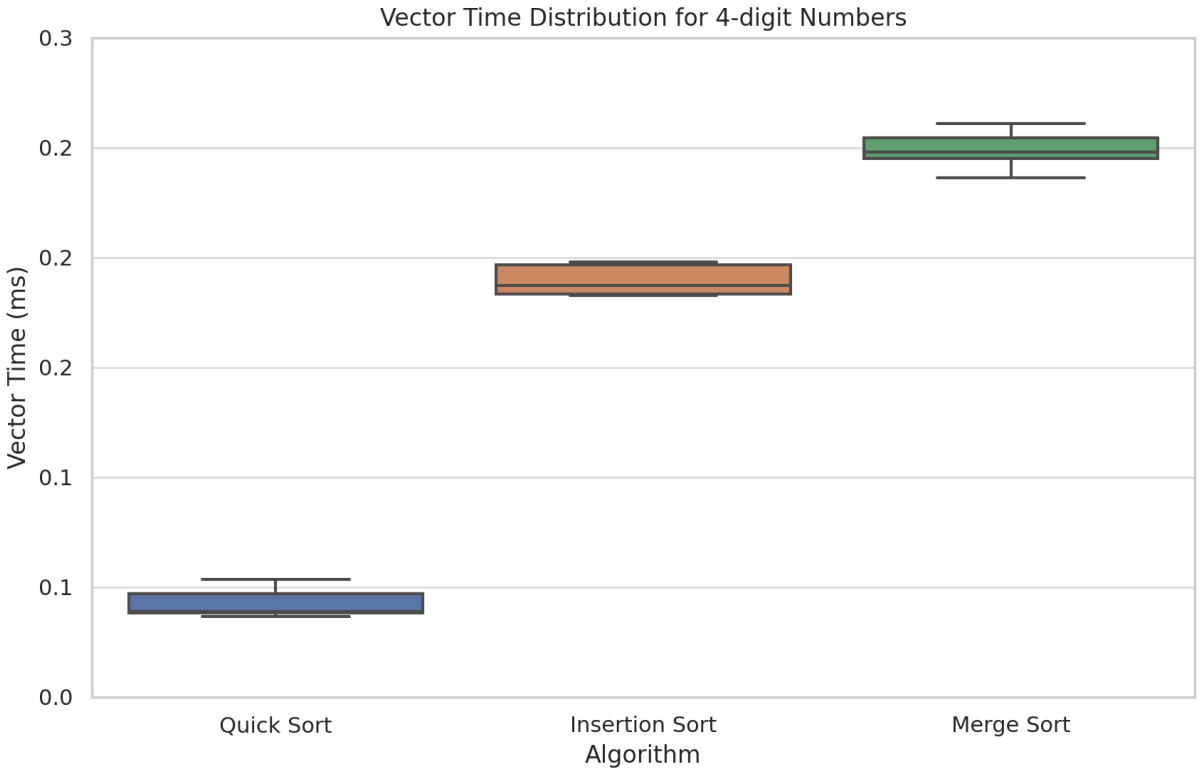
Project Report:

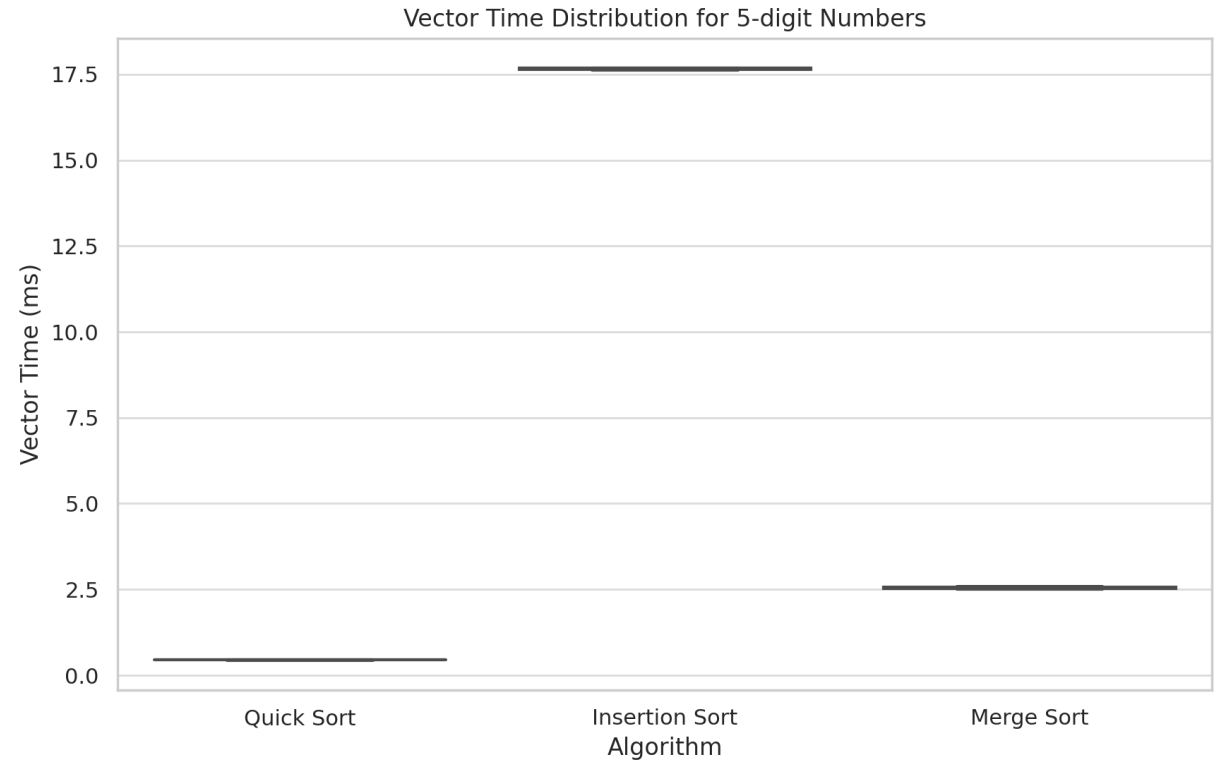
Evaluation:

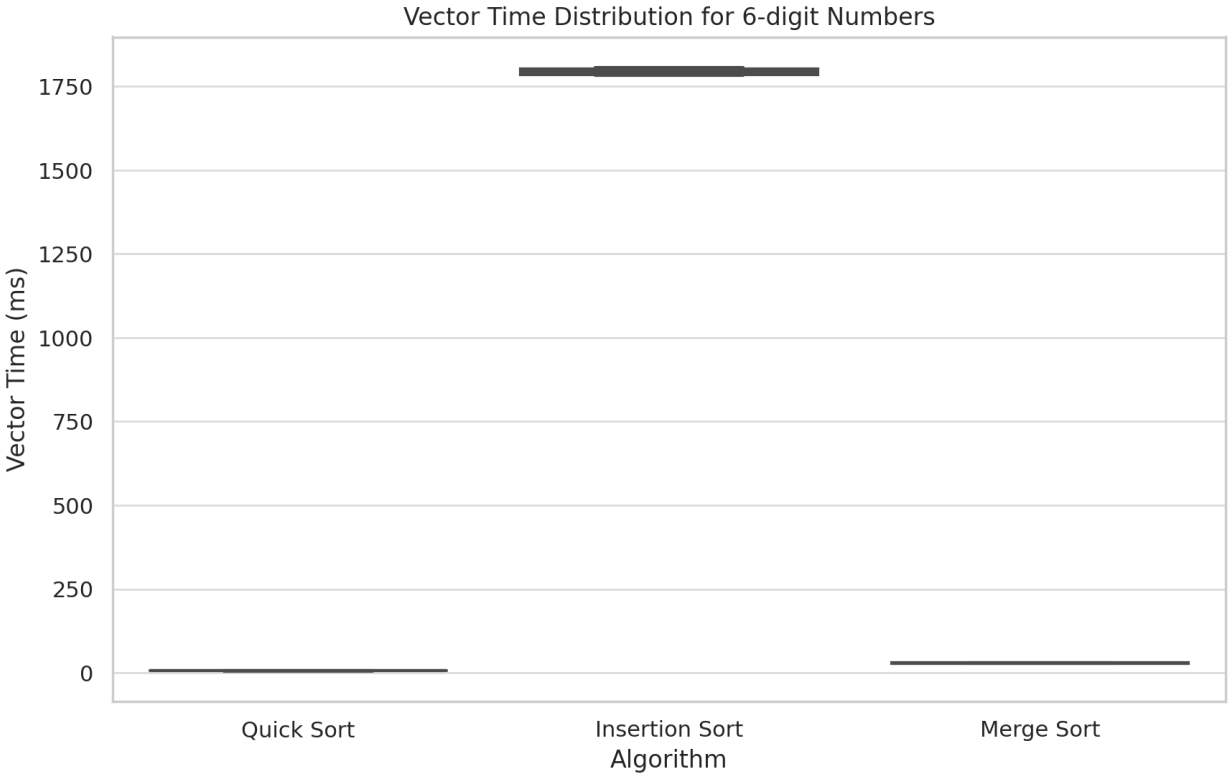





DLL Time Distribution for 6-digit Numbers
DLL Time (ms)
4500
4000
3500
3000
2500
2000
1500
1000
Quick Sort
Insertion Sort
Merge Sort
Algorithm


Vector Time Distribution for 4-digit Numbers
Vector Time (ms)
0.3
0.2
0.2
0.2
0.1
0.1
0.0
Quick Sort
Insertion Sort
Merge Sort
Algorithm


Vector Time Distribution for 5-digit Numbers
Vector Time (ms)
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
Quick Sort
Insertion Sort
Merge Sort
Algorithm


Vector Time Distribution for 6-digit Numbers
Vector Time (ms)
1750
1500
1250
1000
750
500
250
0
Quick Sort
Insertion Sort
Merge Sort
Algorithm

Recommendations:

From the values we obtained, the quick sort algorithm performed the best all-around for both the Doubly linked list and Vector version. The only time quick sort did not outperform the other algorithms was in the Doubly Linked List with four-digit and five-digit numbers. On the distribution of the four-digit numbers of the Doubly Linked list insertion sort consistently outperformed quick sort, by a significant margin between recorded median times. For the five-digit number distribution the mean of the insertion sort was very similar to quick sort, with a slightly better performance. A general recommendation for doubly linked list would be to use quick sort for six or more-digit numbers and use insertion sort for the smaller lists and avoid merge sort for Doubly Linked Lists. A general recommendation for vectors would be to use quick sort in every case. If we are unable to use quick sort, then for larger data sets usage of merge sort would be optimal over insertion sort for if the number of digits is greater than 4. Insertion sort should be avoided for vectors of five or more digits.

Project Design Decisions:

We all sit next to each other in class, so meeting with each other was relatively simple. Initially, we made a group chat so we could easily communicate with each other and this group chat was created on 3/18. All of us were pretty active in the group chat which made communication even easier. We decided to split up the work into our own sections with all of us sharing a testing file that we would each add on to as needed. Two of us would work on the insertion and quick sort, and the other two would work on merge sort. From there, we would further split up and separate into doing vector sorts vs doubly linked list sorts. So, each of us made our own specifications for the functions we would be creating on the document. After we had our sorting algorithms done, it was time to start thinking about the evaluation of these algorithms. When thinking about the evaluator class, it was pretty clear that three of the functions would follow a very similar format to each other and the only unique ones would really be evaluate and ingest. From this, we decided to have one person do the ingest and evaluate function, and we would all try to come up with how to do the comparison functions together. Once one of us had figured out one of the comparison functions, it was easily applicable to the rest of the comparison functions. Additionally, for the evaluations testing file, we only tested the three comparison functions by creating getters for the member variables and ensuring they were populated. We were told that we did not need to necessarily create a test for the evaluate or ingest function since those are not really needed. Throughout this project, we also tried to maintain somewhat regular communication between each other to make sure we all understood how far along we were. Overall, this project was definitely made easier when we had good communication.