Project Report

Evaluation:

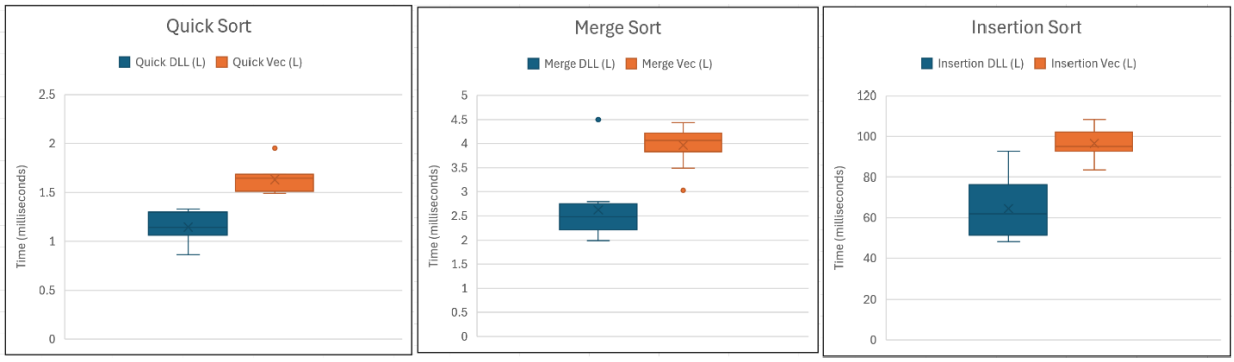
Each sort was done on a vector and a doubly linked list of 3 different sets of data, one ‘small’ (100 numbers), one ‘medium’ (1000 numbers), one ‘large’ (10000 numbers).

For each run, each dataset is a random block of numbers from the 4 lines in evaluation.txt corresponding to the dataset. (4-digit for small, 5-digit for medium, 6-digit for large)

Here is the data for 10 successful runs of the evaluation:

Run #	Insertion DLL (S)	Insertion Vec (S)	Insertion DLL (M)	Insertion Vec (M)	Insertion DLL (L)	Insertion Vec (L)	Merge DLL (S)	Merge Vec (S)	Merge DLL (M)	Merge Vec (M)	Merge DLL (L)	Merge Vec (L)	Quick DLL (S)	Quick Vec (S)	Quick DLL (M)	Quick Vec (M)	Quick DLL (L)	Quick Vec (L)
1	0.01	0.01	0.63	1.16	76.93	95.21	0.02	0.03	0.17	0.41	2.54	4.22	0.01	0.01	0.1	0.14	1.04	1.95
2	0.02	0.02	0.74	1.39	71.97	92.44	0.02	0.03	0.21	0.42	2.46	4.18	0.01	0.01	0.09	0.14	1.12	1.52
3	0.01	0.02	0.62	1.23	48.31	94.84	0.01	0.04	0.22	0.47	2.11	4.12	0.01	0.01	0.08	0.15	1.33	1.5
4	0.02	0.02	0.79	1.67	92.59	105.69	0.01	0.05	0.19	0.42	2.51	4.43	0.01	0.01	0.09	0.14	1.16	1.55
5	0.02	0.02	0.88	1.37	54.14	108.4	0.02	0.04	0.19	0.54	4.5	3.99	0.01	0.01	0.09	0.13	1.29	1.68
6	0.01	0.02	0.63	1	65.36	100.87	0.01	0.05	0.34	0.52	2.74	4.22	0.01	0.01	0.1	0.13	1.16	1.68
7	0.01	0.02	0.61	1.15	76.43	83.45	0.01	0.03	0.14	0.31	1.99	3.03	0	0.01	0.06	0.09	0.86	1.62
8	0.01	0.02	1.09	1.68	58.74	96.15	0.01	0.03	0.17	0.37	2.25	3.49	0.01	0.01	0.1	0.16	1.33	1.66
9	0.01	0.02	0.8	1.32	48.67	92.7	0.01	0.02	0.13	0.27	2.39	4.02	0	0.01	0.06	0.12	1.07	1.49
10	0.01	0.02	0.57	1.17	52.28	94.77	0.01	0.04	0.22	0.74	2.8	3.94	0.01	0.01	0.08	0.12	1.08	1.66

Here are boxplots showing differences in time between vectors and doubly linked lists for each sort. This is not a comprehensive representation of all the data (only considers large dataset for each sort), but at the timers accuracy (to the 1/100 millisecond) does not allow for a visible varied distribution of times for the sorting of the smaller datasets, so we use the larger for specifically comparing vectors to doubly linked lists.



Recommendations:

Insertion sort’s worst-case time complexity is $O(n^2)$, so its running time grows quadratically as the dataset size increases. This is evident in the provided data: the jump in execution time from medium to large inputs is substantially greater than from small to medium. Despite this, the doubly linked-list implementation still outperforms the vector version—likely because inserting nodes in a linked list avoids the costly element shifts required by an array.

By comparison, merge sort runs in $O(n \log n)$ on average, yielding much more consistent scaling as dataset size grows. Quick sort can degrade to $O(n^2)$ in the worst case but typically achieves $O(n \log n)$ performance; here, too, the linked-list implementation narrows the gap versus the vector version.

Overall, while any method may suffice for small inputs, for larger datasets the best performance comes from merge sort or quick sort applied to a doubly linked list. Reserve insertion sort for small or nearly sorted collections.

Project Design Decisions:

Initial planning occurred during a brief in-person meeting to establish objectives, with subsequent coordination handled via text. Key modifications included adding a copy constructor to the DoublyLinkedList class to maintain data integrity during testing and revising the ingest process to randomly select 25% of each dataset for evaluation. This randomization ensured varied data subsets across runs, preventing bias from repeated identical inputs. The evaluation logic was simplified to execute one sort per dataset size per algorithm, reducing redundancy while maintaining comparative accuracy.