

	Mean Run Times (ms)								
				D					
	A	B	C	50	500	1000	2000	5000	E
<b>Bubble</b>	2870.892	532.295	3805.208	0.079	6.068	25.978	109.927	707.228	679.602
<b>Insertion</b>	1410.634	132.929	2754.732	0.045	3.067	13.326	55.034	344.849	328.231
<b>Selection</b>	1166.091	288.555	1283.630	0.048	3.021	11.896	46.621	286.691	274.029
<b>Quick</b>	9.538	11.357	1977.770	0.027	0.306	0.649	1.411	3.926	3.868
<b>Merge</b>	13.375	5.291	10.843	0.054	0.546	1.106	2.314	6.297	5.890

**3a.** For experiments A,D,E I found that quicksort to perform the best since it has a time complexity of  $O(n \log n)$  compared to bubble, insertion, and selection which have  $O(n^2)$ . It outperforms mergesort because partitioning in place is faster than merging using auxiliary arrays. It loses to mergesort in experiment C because a reverse list is quicksort's worst case.

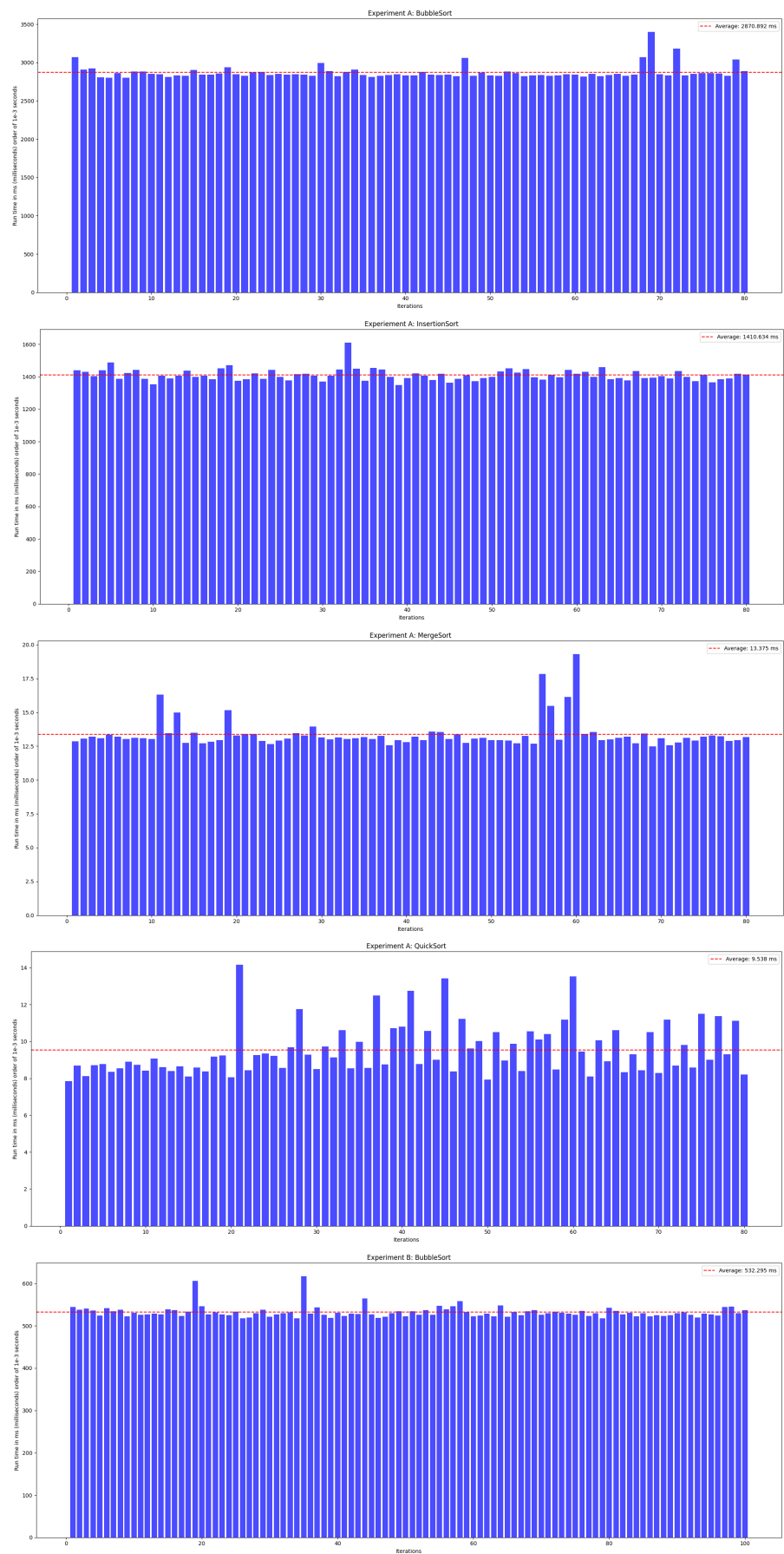
**3b.** For all experiments bubble sort had the worst performance because it performs  $n^2$  comparisons and  $n^2$  swaps, brute force checking and swapping will result in the worst performance out of all of these since it often does unnecessary checks.

**3c.** Mergesort outperforms selection sort in every single experiment because selection sort has a time complexity of  $O(n^2)$  quadratic time stemming from the fact that it does  $((n - 1) * n) / 2$  comparisons. Merge sort has a much lower time complexity of  $O(n \log n)$  linearithmic time since halving a list until singletons is much faster.

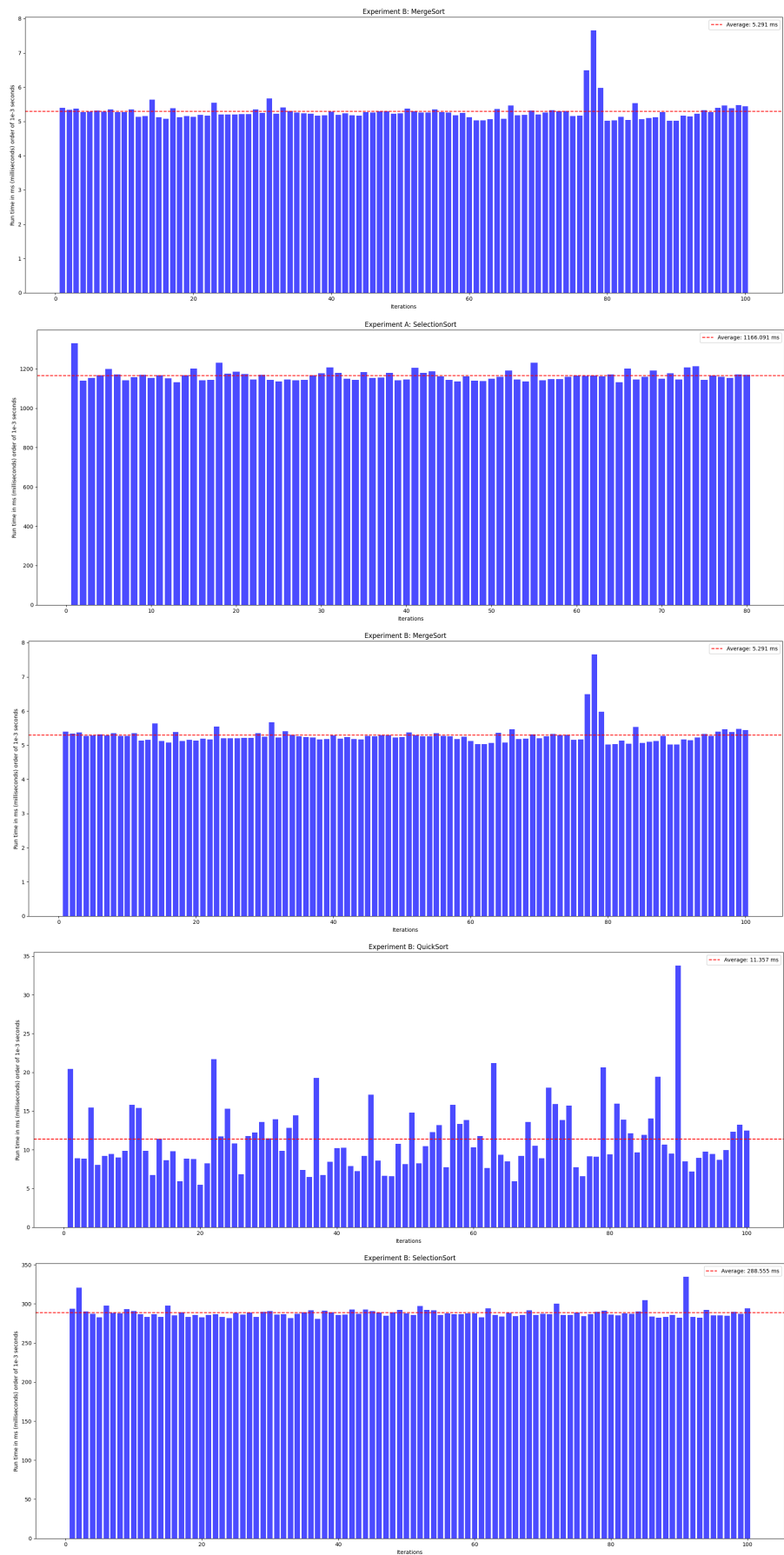
**3d.** My experiments did not cover such scenarios but the only time insertion sort would beat merge sort is when the list is nearly sorted, more sorted than the 75% near sorted list I had. This happens because insertion sort will only do around  $n$  number of comparisons and a few swaps giving it a linear run time  $O(n)$  which is faster than merge sort's linearithmic  $O(n \log n)$  run time.

**3e.** Experiment A represents the average case of quicksort because usually we shuffle the list before performing quicksort to avoid the worst case where we partition around the first/last element and the list is in reversed order. So the randomness of experiment A best reflects the shuffle that is performed before quicksort which closely represents the average case.

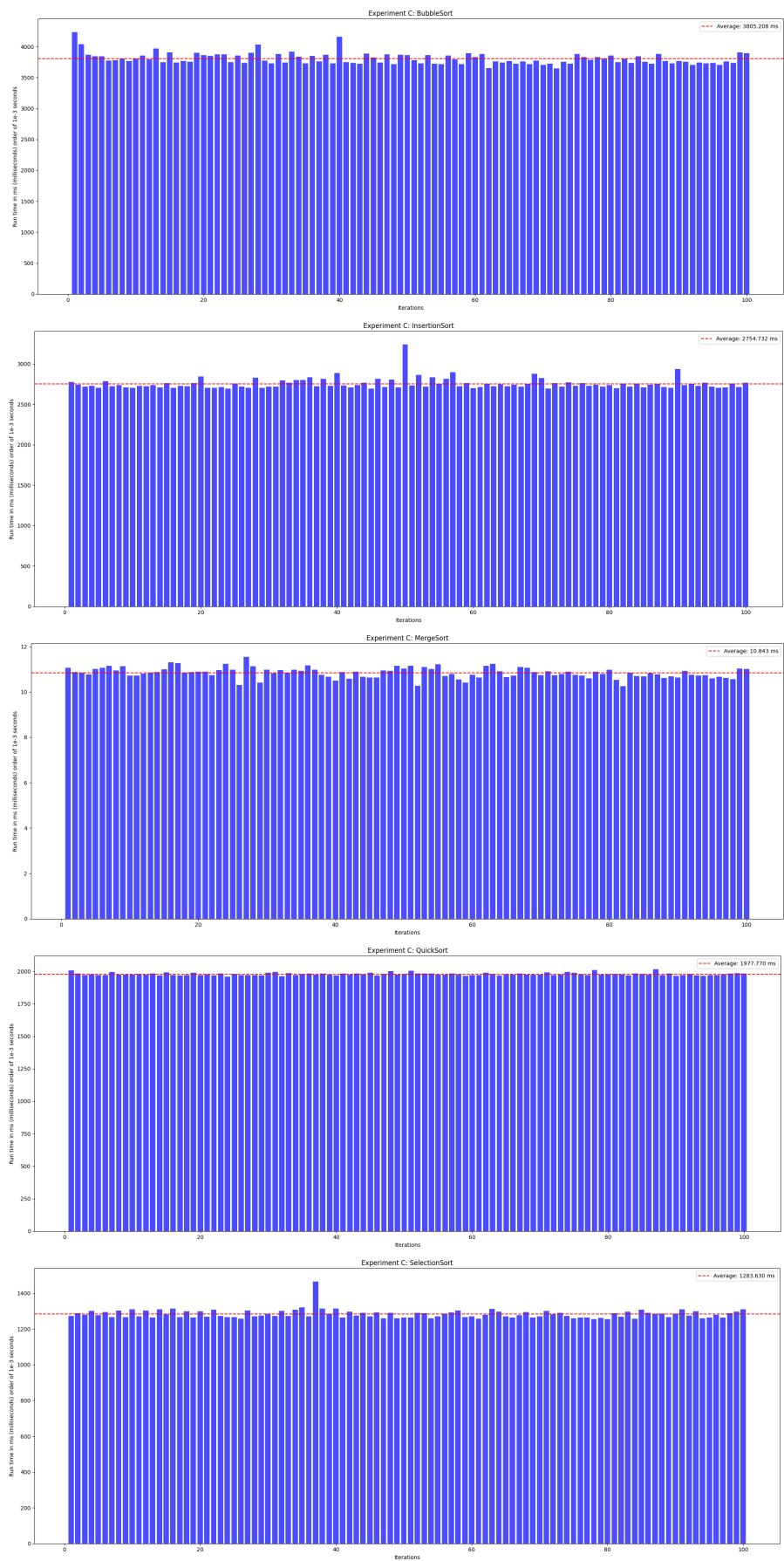
Appendix: Experiment A



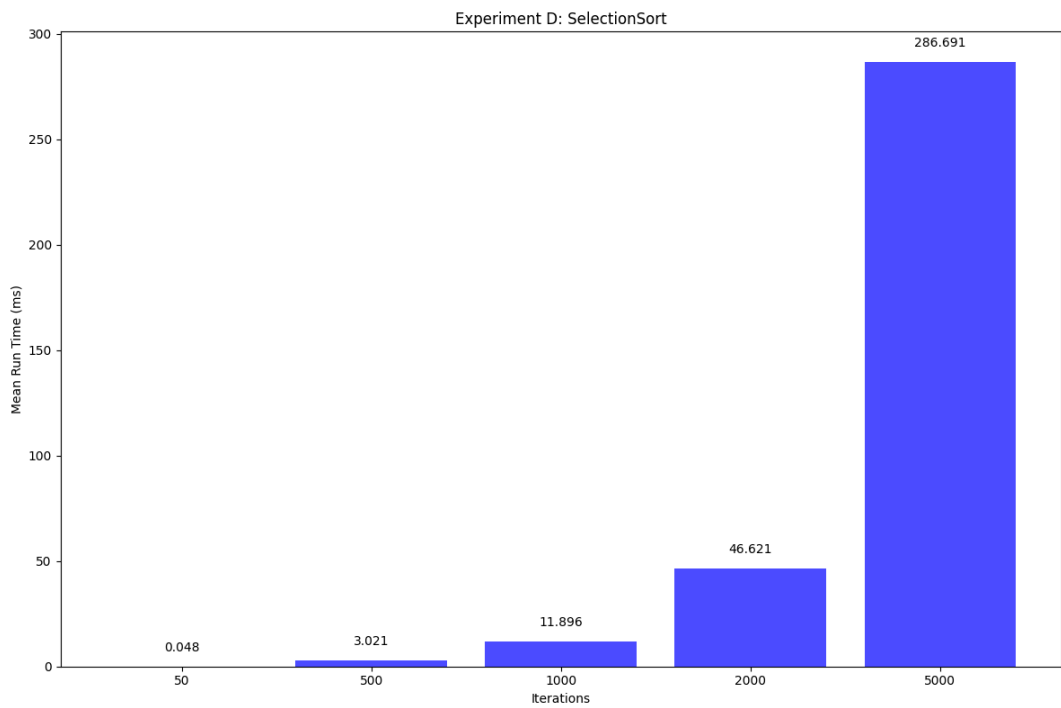
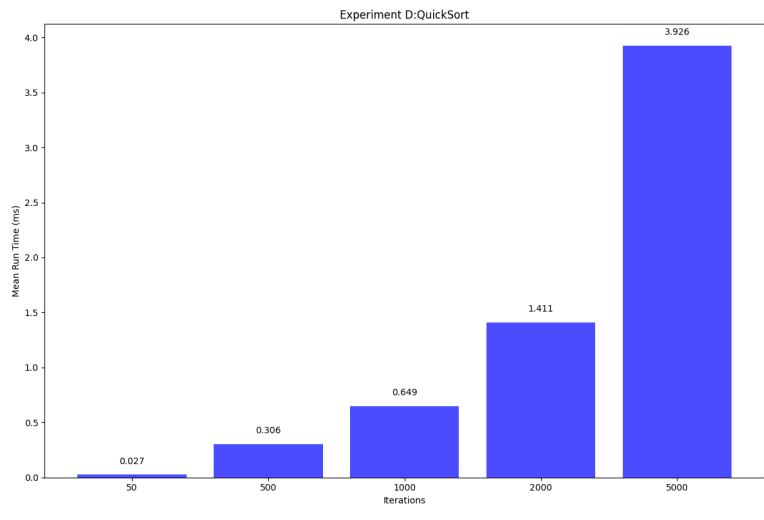
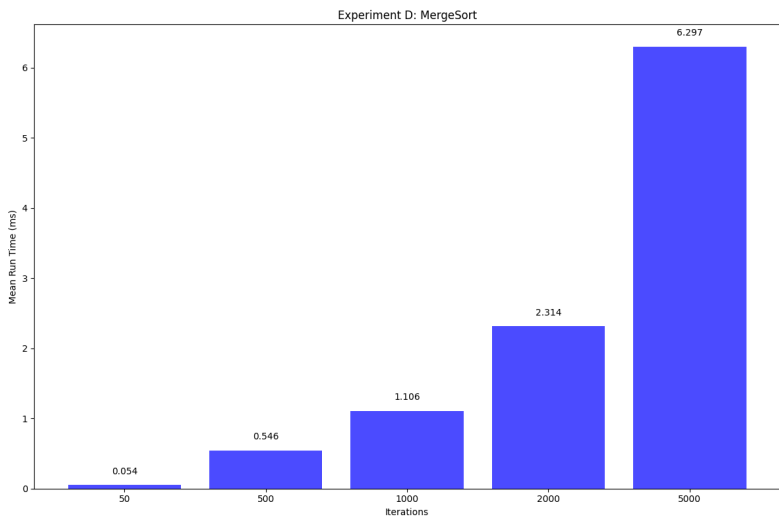
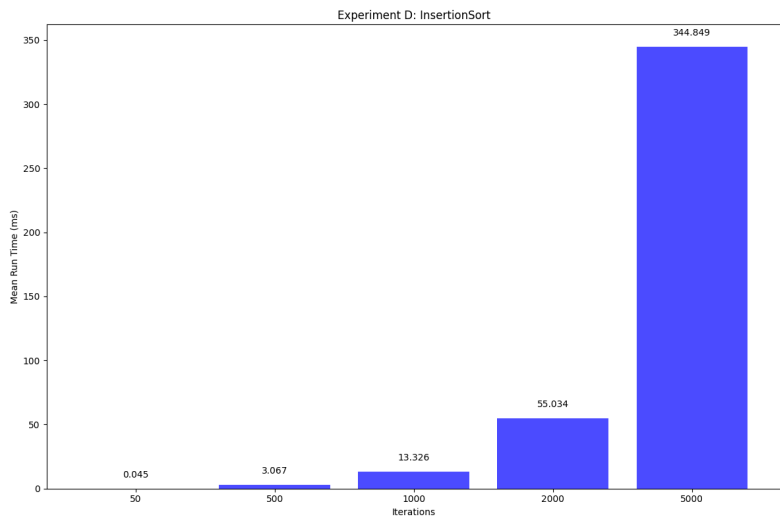
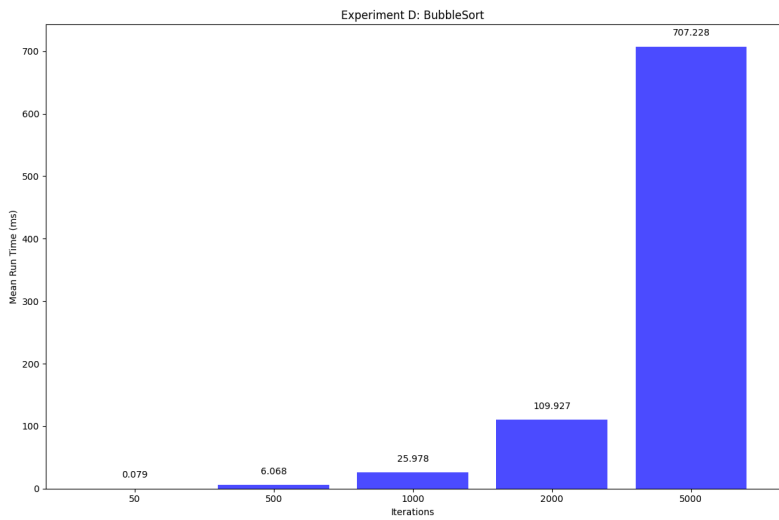
Appendix: Experiment B



# Appendix: Experiment C



# Appendix: Experiment D



Appendix: Experiment E

