Programming Assignment 3

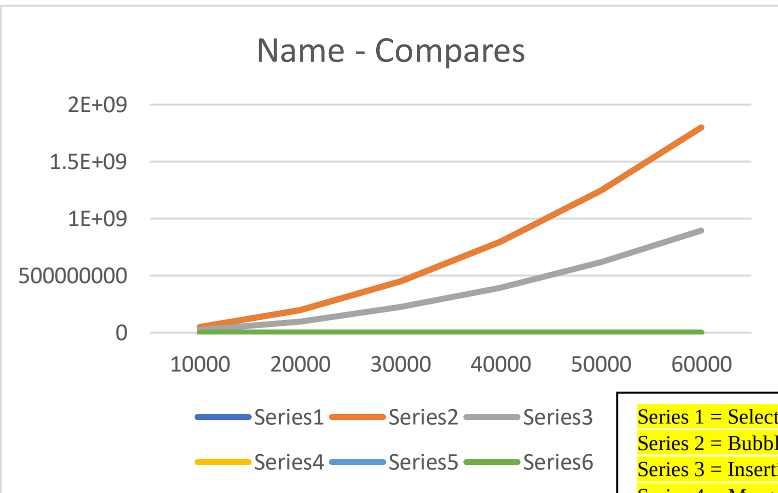
Analysis of sorting algorithms

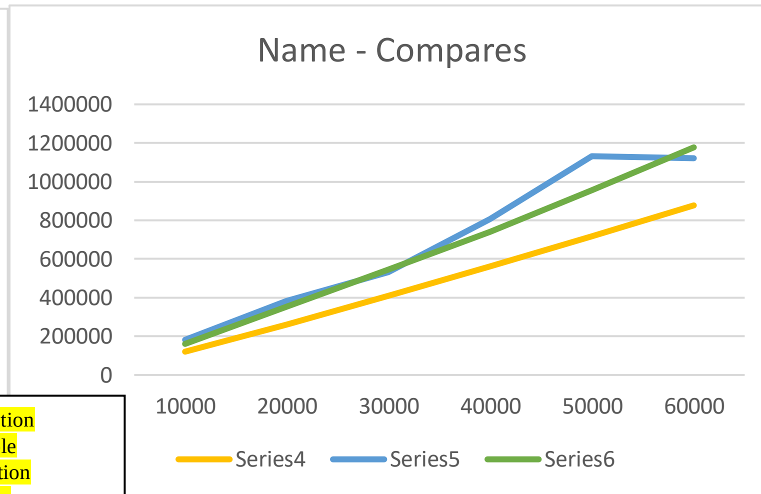
Joseph Hodson
November 2, 2020
COP 3502

Note: X values are the size of arrays, Y values are respective totals of comparisons, copies, and times.*

Criteria 1: Name

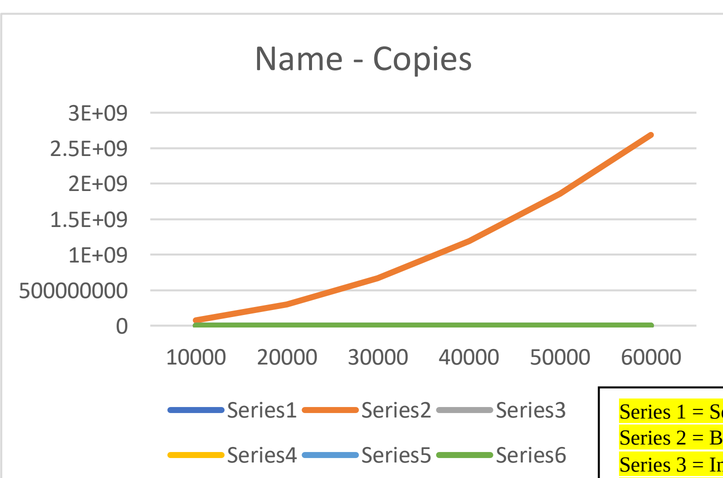
	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
1	DataSize	Selection	Selection	Selection	BubbleSo	BubbleSo	BubbleSo	Insertion	Insertion	Insertion	MergeSor	MergeSor	MergeSor	Merge_In	Merge_In	Merge_In	QuickSort	QuickSort	QuickSortTime
2	10000	49995000	30000	0.640625	49995000	75194508	1.0625	25074826	19998	0.375	120467	267232	0	181721	210000	0.015625	160775	293109	0
3	20000	2E+08	60000	2.9375	2E+08	2.99E+08	5.5	99597394	39998	1.921875	260947	574464	0.015625	383449	460000	0.015625	353921	543168	0.015625
4	30000	4.5E+08	90000	6.8125	4.5E+08	6.7E+08	11.20313	2.23E+08	59998	4.1875	408511	894464	0.03125	531013	750000	0.03125	545004	976713	0.03125
5	40000	8E+08	120000	12.09375	8E+08	1.19E+09	20.07813	3.97E+08	79998	8.046875	561890	1228928	0.015625	806902	1000000	0.03125	741106	1391037	0.03125
6	50000	1.25E+09	150000	20.07813	1.25E+09	1.86E+09	33.79688	6.2E+08	99998	13.375	718339	1568928	0.046875	1131595	1250000	0.03125	955319	1564701	0.046875
7	60000	1.8E+09	180000	32.10938	1.8E+09	2.69E+09	53.3125	8.96E+08	119998	22.39063	877244	1908928	0.03125	1122112	1620000	0.046875	1177496	2037585	0.046875

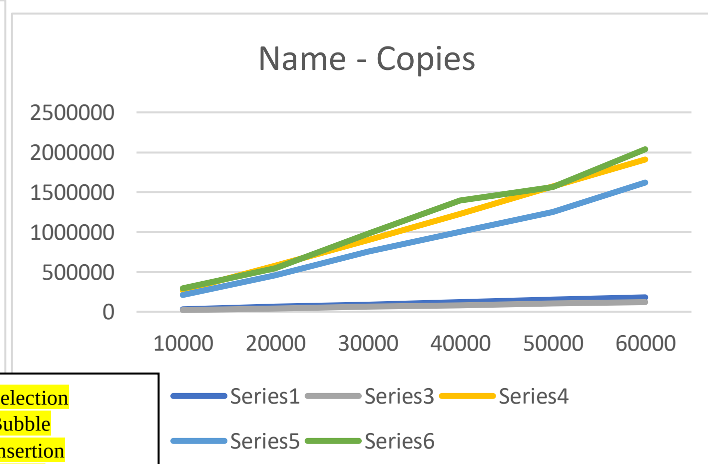




Series 1 = Selection
Series 2 = Bubble
Series 3 = Insertion
Series 4 = Merge
Series 5 = Insertion Merge
Series 6 = Quick

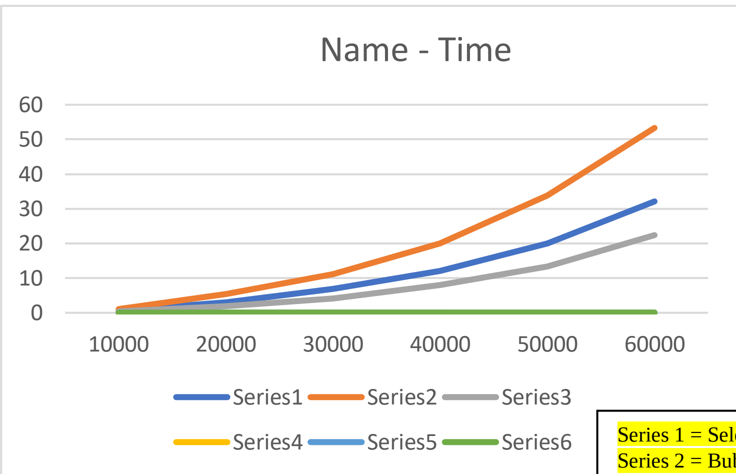
Seen above, it's clear that the O(n^2) algorithms use the most comparisons to sort the arrays. Both bubble and selection use the same amount, and are overlapped in the first figure. Merge, in this instance, uses the least amount of comparisons but is comparable to both quick and insertion merge sorts.

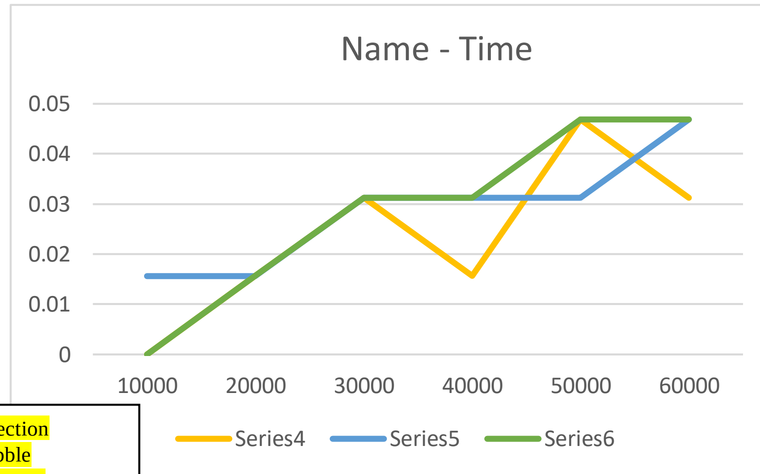




Series 1 = Selection
Series 2 = Bubble
Series 3 = Insertion
Series 4 = Merge
Series 5 = Insertion Merge
Series 6 = Quick

Seen above, it's clear that the bubble sort uses the most copies by far to sort the arrays. The divide and conquer algorithms, that being merge, merge insertion and quick sort all use roughly the same amount of copies. Selection and Insertion sort, however, use the least amount.





Series 1 = Selection
Series 2 = Bubble
Series 3 = Insertion
Series 4 = Merge
Series 5 = Insertion Merge
Series 6 = Quick

Seen above, it's evident that bubble sort requires the most time to sort the different files' arrays. We've noted with the copies and compares that bubble sort has proven to be rather inefficient as well. To me, this concludes that bubble sort, for criteria 3, is the least efficient sorting algorithm. Insertion and selection both require a lot of time when compared to our O(n*logn) algorithms as well. All 3, in which are relatively efficient and take little to no time to sort large quantities of elements in arrays.

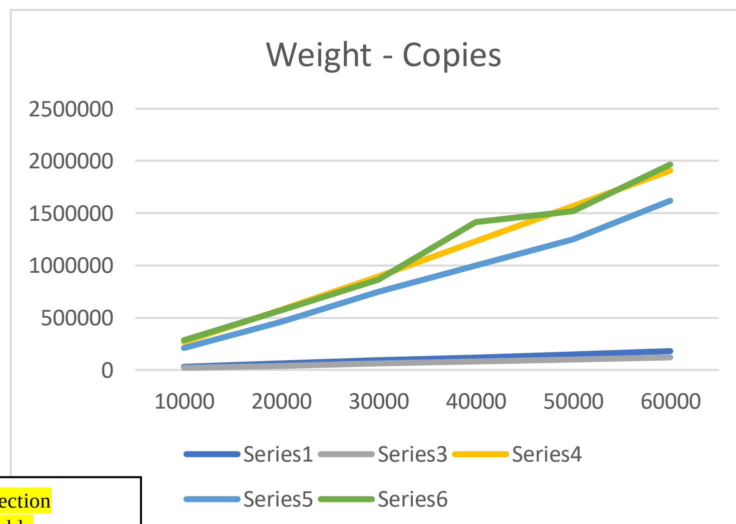
Criteria 2: Weight

	A	B	C	D	E	F	G	H	I	J
1	DataSize	Selection	Selection	Selection	BubbleSo	BubbleSo	BubbleSo	Insertion	Insertion	Insertion
2	10000	49995000	30000	0.328125	49995000	75574053	0.875	25201344	19998	0.203125
3	20000	2E+08	60000	1.390625	2E+08	2.99E+08	3.328125	99771084	39998	0.78125
4	30000	4.5E+08	90000	3.078125	4.5E+08	6.75E+08	8.0625	2.25E+08	59998	1.828125
5	40000	8E+08	120000	5.625	8E+08	1.19E+09	14.90625	3.98E+08	79998	3.28125
6	50000	1.25E+09	150000	9.875	1.25E+09	1.87E+09	25.26563	6.23E+08	99998	5.640625
7	60000	1.8E+09	180000	15.17188	1.8E+09	2.69E+09	37.54688	8.96E+08	119998	8.171875
8										

	K	L	M	N	O	P	Q	R	S	T
1	MergeSor	MergeSor	MergeSor	Merge_In	Merge_In	Merge_In	QuickSort	QuickSort	QuickSortTime	
2	118975	267232	0	180285	210000	0	163970	284808	0	
3	255684	574464	0	378229	460000	0.015625	334639	572310	0	
4	397914	894464	0.03125	520330	750000	0.015625	517858	863313	0.015625	
5	544110	1228928	0.015625	789185	1000000	0.015625	764314	1413201	0.015625	
6	692331	1568928	0.03125	1105853	1250000	0.015625	948956	1520049	0.015625	
7	842281	1908928	0.03125	1087105	1620000	0.03125	1118916	1966914	0.03125	
8										

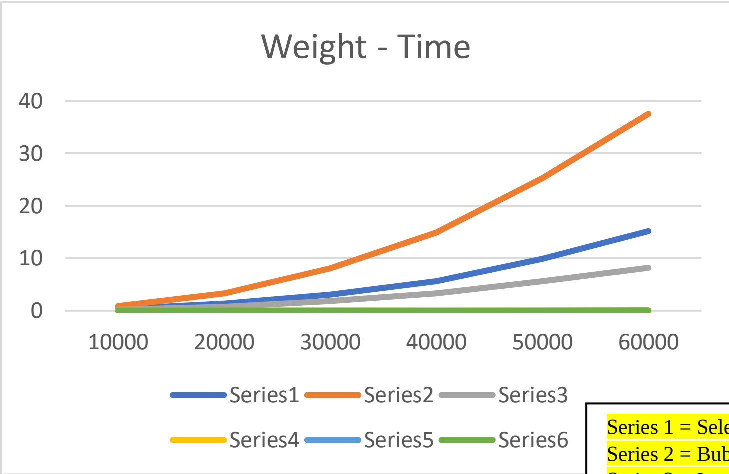
Series 1 = Selection
Series 2 = Bubble
Series 3 = Insertion
Series 4 = Merge
Series 5 = Insertion Merge
Series 6 = Quick

Seen above, it's clear that the O(n^2) algorithms use the most comparisons to sort the arrays. Both bubble and selection use the same amount, and are overlapped in the first figure. Merge, in this instance, uses the least amount of comparisons but is comparable to both quick and insertion merge sorts.



Series 1 = Selection
Series 2 = Bubble
Series 3 = Insertion
Series 4 = Merge
Series 5 = Insertion Merge
Series 6 = Quick

Seen above, it's clear that the bubble sort uses the most copies by far to sort the arrays. The divide and conquer algorithms, that being merge, merge insertion and quick sort all use roughly the same amount of copies. Selection and Insertion sort, however, use the least amount.

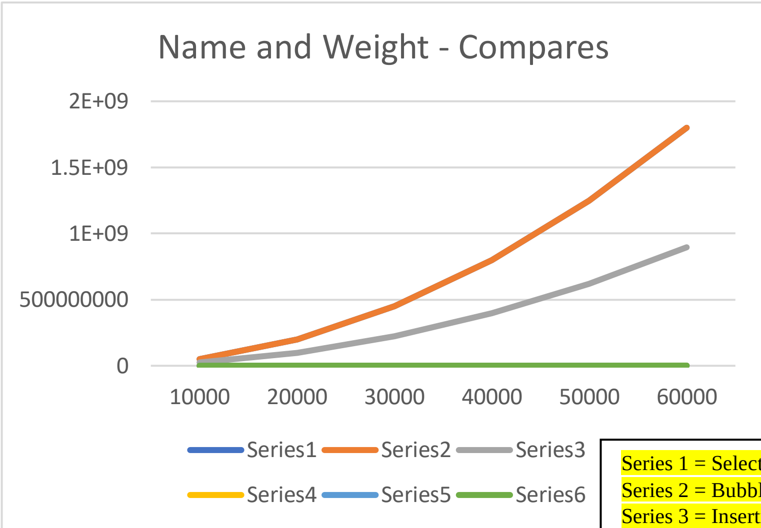


Series 1 = Selection
Series 2 = Bubble
Series 3 = Insertion
Series 4 = Merge
Series 5 = Insertion Merge
Series 6 = Quick

Seen above, it’s evident that bubble sort requires the most time to sort the different files’ arrays. We’ve noted with the copies and compares that bubble sort has proven to be rather inefficient as well. To me, this concludes that bubble sort, for criteria 3, is the least efficient sorting algorithm. Insertion and selection both require a lot of time when compared to our O(n*logn) algorithms as well. All 3, in which are relatively efficient and take little to no time to sort large quantities of elements in arrays.

Criteria 3: Name and Weight

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
1	DataSize	Selection!	Selection!	Selection!	BubbleSo	BubbleSo	BubbleSo	Insertion!	Insertion!	Insertion!	MergeSor	MergeSor	MergeSor	Merge_In:	Merge_In:	Merge_In:	QuickSort	QuickSort	QuickSortTime	
2	10000	49995000	30000	0.6875	49995000	75194523	1.171875	25074831	19998	0.453125	120472	267232	0	181724	210000	0	149794	280899	0.015625	
3	20000	2E+08	60000	2.96875	2E+08	2.99E+08	4.921875	99597399	39998	1.765625	260952	574464	0.015625	383452	460000	0.015625	337420	585183	0	
4	30000	4.5E+08	90000	6.734375	4.5E+08	6.7E+08	11.17188	2.23E+08	59998	4.3125	408515	894464	0.015625	531017	750000	0.015625	548914	937755	0.015625	
5	40000	8E+08	120000	12.01563	8E+08	1.19E+09	20.14063	3.97E+08	79998	7.71875	561895	1228928	0.03125	806905	1000000	0.03125	892890	1530000	0.03125	
6	50000	1.25E+09	150000	20.90625	1.25E+09	1.86E+09	34.0625	6.2E+08	99998	13.75	718343	1568928	0.03125	1131597	1250000	0.03125	966187	1595025	0.046875	
7	60000	1.8E+09	180000	37.28125	1.8E+09	2.69E+09	64.90625	8.96E+08	119998	24.6875	877248	1908928	0.03125	1122116	1620000	0.046875	1152912	1842126	0.046875	
8																				
9																				

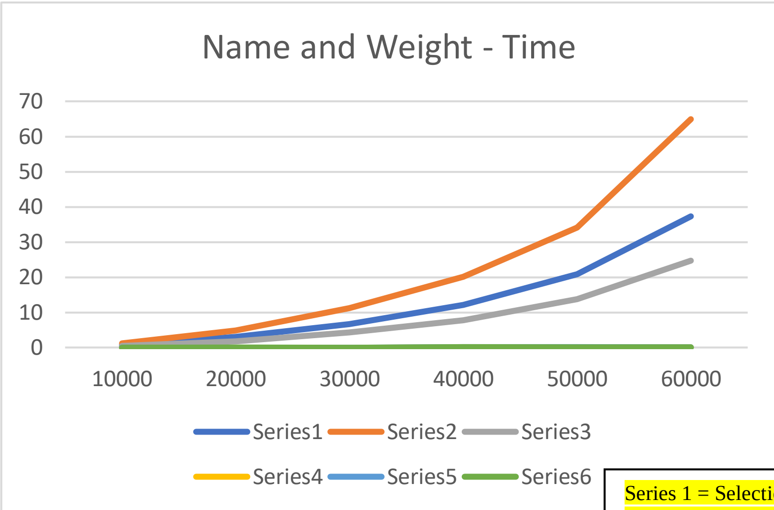


Series 1 = Selection
Series 2 = Bubble
Series 3 = Insertion
Series 4 = Merge
Series 5 = Insertion Merge
Series 6 = Quick

Seen above, it's clear that the O(n^2) algorithms use the most comparisons to sort the arrays. Both bubble and selection use the same amount, and are overlapped in the first figure. Merge, in this instance, uses the least amount of comparisons but is comparable to both quick and insertion merge sorts.

Series 1 = Selection
Series 2 = Bubble
Series 3 = Insertion
Series 4 = Merge
Series 5 = Insertion Merge
Series 6 = Quick

Seen above, it's clear that the bubble sort uses the most copies by far to sort the arrays. The divide and conquer algorithms, that being merge, merge insertion and quick sort all use roughly the same amount of copies. Selection and Insertion sort, however, use the least amount.



Series 1 = Selection
Series 2 = Bubble
Series 3 = Insertion
Series 4 = Merge
Series 5 = Insertion Merge
Series 6 = Quick

Seen above, it's evident that bubble sort requires the most time to sort the different files' arrays. We've noted with the copies and compares that bubble sort has proven to be rather inefficient as well. To me, this concludes that bubble sort, for criteria 3, is the least efficient sorting algorithm. Insertion and selection both require a lot of time when compared to our O(n*logn) algorithms as well. All 3, in which are relatively efficient and take little to no time to sort large quantities of elements in arrays.

Summary:

It is clear that the least efficient algorithms are the O(n^2) algorithms. In particular, the bubble sort takes the most time to sort an array, and it is tied for the most amount of comparisons made when sorting an algorithm. Our most efficient sorting technique appears to be the quick sort. The quick sort runs relatively much faster than the rest of the algorithms but is comparable to both the merge and modified merge sort. We know that quick sort's efficiency solely depends on the pivot in which is chosen prior to sorting. However, I have learned through this assignment that to maximize the quick sort algorithm we must aim to choose the median element value, since the partition will push smaller elements to the left of the pivot and larger items will therefore stay on the right side. Merge sort is generally very efficient as well but since it is a non in-place sorting algorithm we must allocate an array of the same size to position the proper elements with respect to the sort. All in all, divide and conquer sorting algorithms are generally much more efficient, but may be slightly more complex to understand due to the use of recursion.