NWEN 303 Concurrent Programming (2015)
Project 2: MPI

I have chosen Insertion sort, Shell sort and merge sort. I choose them because they are simple to implement and insertion sort and shell sort has the similar logic. So when implementing the parallel version should be similar. Also for merge sort it works by keep splitting the array into half until there is only one value left for each array, so all the arrays are sorted. then it will recursively sorting the two subarrays back up until the whole array is sorted. It should help a lot when implementing parallel version i can just spit the array into different processes instead of arrays, and merge them back up to sort the array.

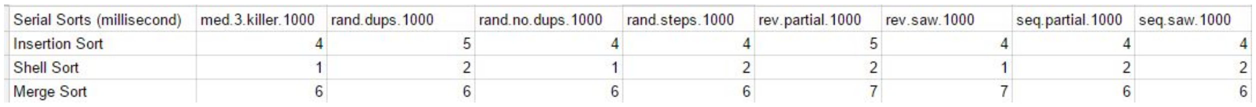
I used the stopwatch class from Assignment 1 to calculate the time for my programs.

I tested my algorithm by outputting the result, the sorting algorithm all works. But the file was too big so i decided to remove the data result and just output the time it takes to sort.

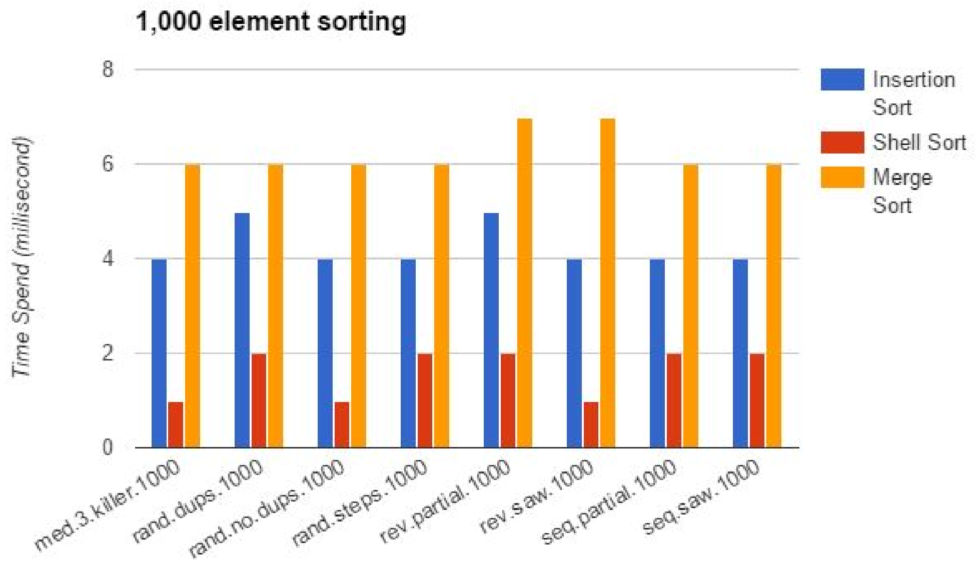
Serial Sort

while doing merge sort I started to implement it using arraylist but i found that it takes too long to sort even longer than insertion and selection sort which means it wrong because merge sort should be faster than any of those two sort. Then i change the arraylist to list instead. I found that after I change the arraylist to list the speed has greatly increased.

The image below shows the test result of the three sorting algorithms:

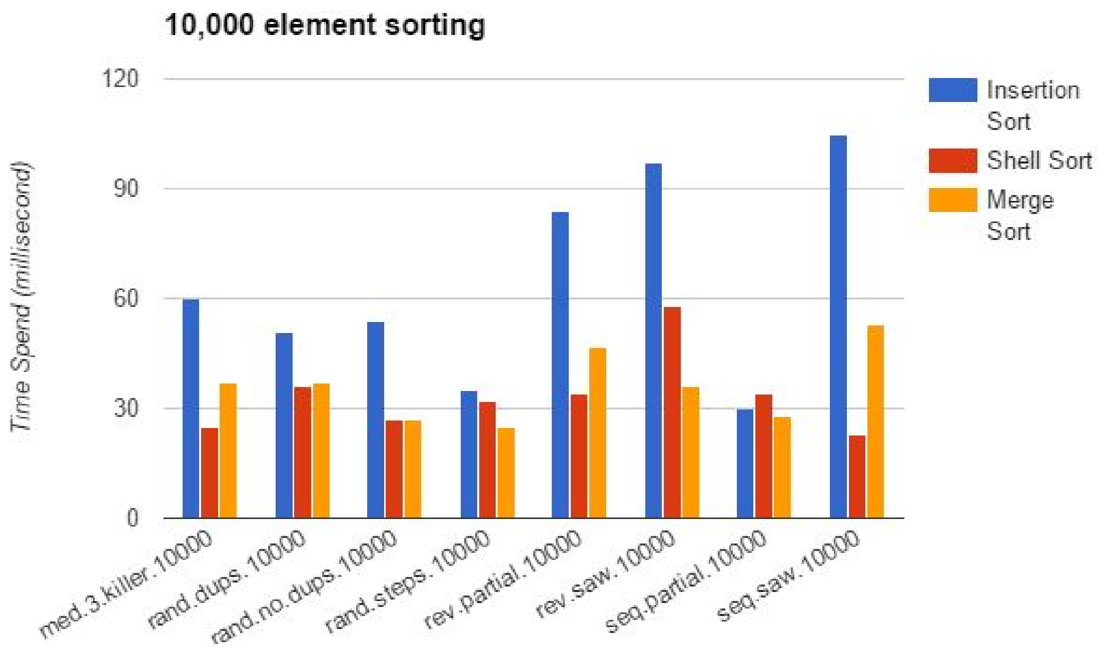


Serial Sorts (millisecond)	med.3.killer.1000	rand.dups.1000	rand.no.dups.1000	rand.steps.1000	rev.partial.1000	rev.saw.1000	seq.partial.1000	seq.saw.1000
Insertion Sort	4	5	4	4	5	4	4	4
Shell Sort	1	2	1	2	2	1	2	2
Merge Sort	6	6	6	6	7	7	6	6



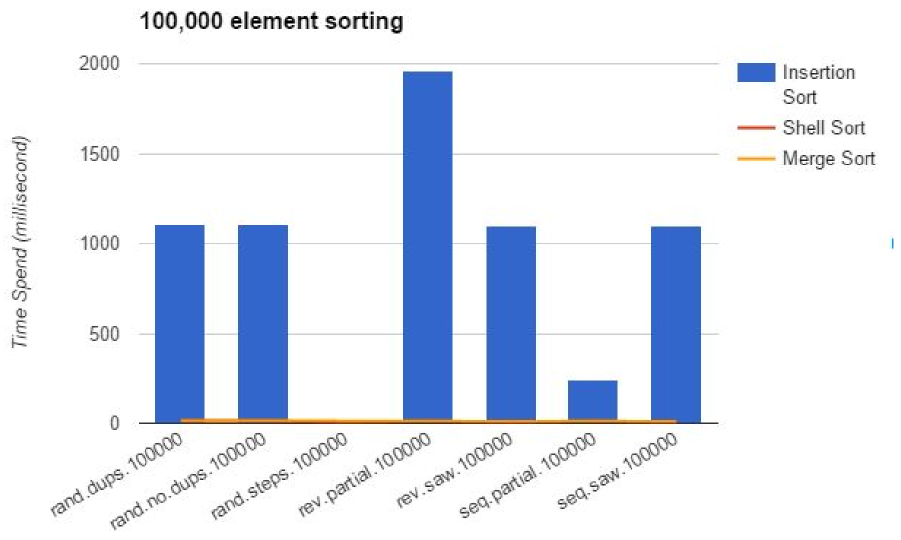
the bar chart above shows that when sorting small datas(1000 items) seems to be slower than insertion and shell sort is because merge sort have to take time to split the array and merge them back up. its seems merge sort are consistent the time difference is between 1 millisecond. For insertion sort it has to swap the value every time the value is not sorted its cost is most depending on how much swap it needs to sort the array. For shell sort it don't have to do as much swap as insertion, because shellsort first search for values with a gap between them then if the value is not sorted it will swap them, it will keep doing this until the end of the array then it will use insertion sort to sort the array at the end so it don’t have to do as much swap as insertion sort.

Serial Sorts (millisecond)	med.3.killer.10000	rand.dups.10000	rand.no.dups.10000	rand.steps.10000	rev.partial.10000	rev.saw.10000	seq.partial.10000	seq.saw.10000
Insertion Sort	60	51	54	35	84	97	30	105
Shell Sort	25	36	27	32	34	58	34	23
Merge Sort	37	37	27	25	47	36	28	53



The bar chart above shows that when the sorting data gets larger merge sort and shell sort start to scale better and insertion sort start to take longer to sort for the data that are almost reverse. For the data that are partially sorted it still have a good performance. Same with shell sort but shellsort only sort with insertion sort at the last step so it has better performance than insertion sort over all.

Serial Sorts (millisecond)	rand.dups.100000	rand.no.dups.100000	rand.steps.100000	rev.partial.100000	rev.saw.100000	seq.partial.100000	seq.saw.100000
Insertion Sort	1107	1109	6	1960	1099	241	1096
Shell Sort	17	16	8	14	7	14	7
Merge Sort	17	17	15	14	12	14	12



the bar chart above shows that when the sorting data is large, merge sort and shell scale really well we cannot see the merge sort and shell sort above is because it's too small to see. For insertion sort it just very bad when handling datas that are almost reverse or the small values are right at the end of the array. As you can see the "rand.steps" file its performance is very good for insertion sort that because the file is almost sorted so it doesn't have to do much swaps to sort the file.

Parallel Sort

Insertion sort have the best cost when the file is sorted. In the parallel version I split the data into many chunks and pass it into different processes to run it, after all the processes have sort the data then combine them back to a list at this point the array will be mostly sorted so i do one final sort to get the result.

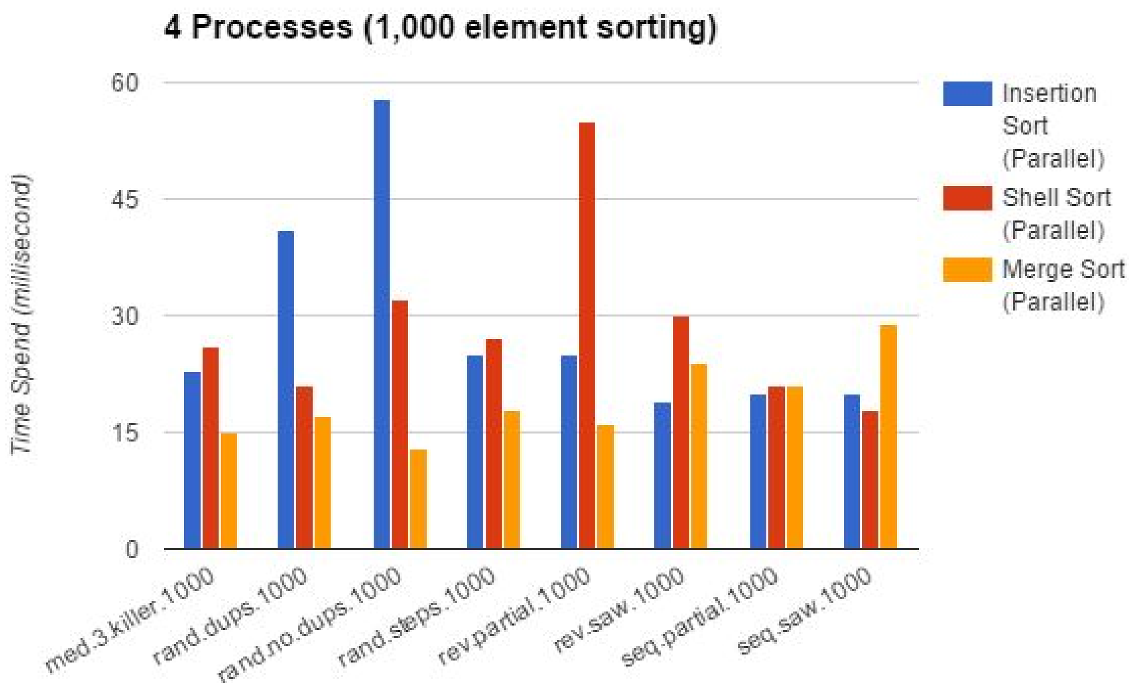
In the parallel version of shellsort it works by splitting the data into many chunks and pass each chunk into different processes to call the shellsort method to sort the arrays, after all the other processes has finished sorting the chunks it then gather them back into one array and use insertion sort to do one final sort.

For merge sort it works by splitting the data into many chunks and pass each chunk into different processes and call the sort method to sort each array, after the chunks has been sorted it will gather all the chunks back together and do one final sort.

To time the sorting I use the maximum time of the processes took to sort the subarrays and add the time it takes for processes 0(rank 0) to do the sorting. I did this because processes 0 have to wait for all the other processes to finish sorting before processes 0 can sort the final array and other processes will run in parallel so i just take the maximum time out of other processes.

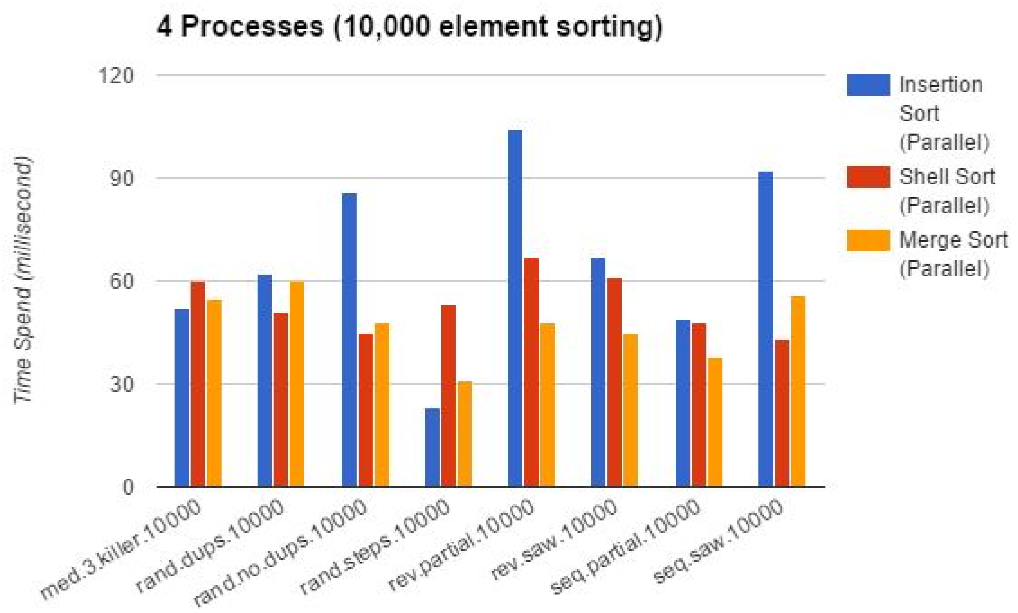
Bar charts below shows the performance of the parallel sorting algorithms:

4 Processess	med.3.killer.1000	rand.dups.1000	rand.no.dups.1000	rand.steps.1000	rev.partial.1000	rev.saw.1000	seq.partial.1000	seq.saw.1000
Insertion Sort (Parallel)	23	41	58	25	25	19	20	20
Shell Sort (Parallel)	26	21	32	27	55	30	21	18
Merge Sort (Parallel)	15	17	13	18	16	24	21	29



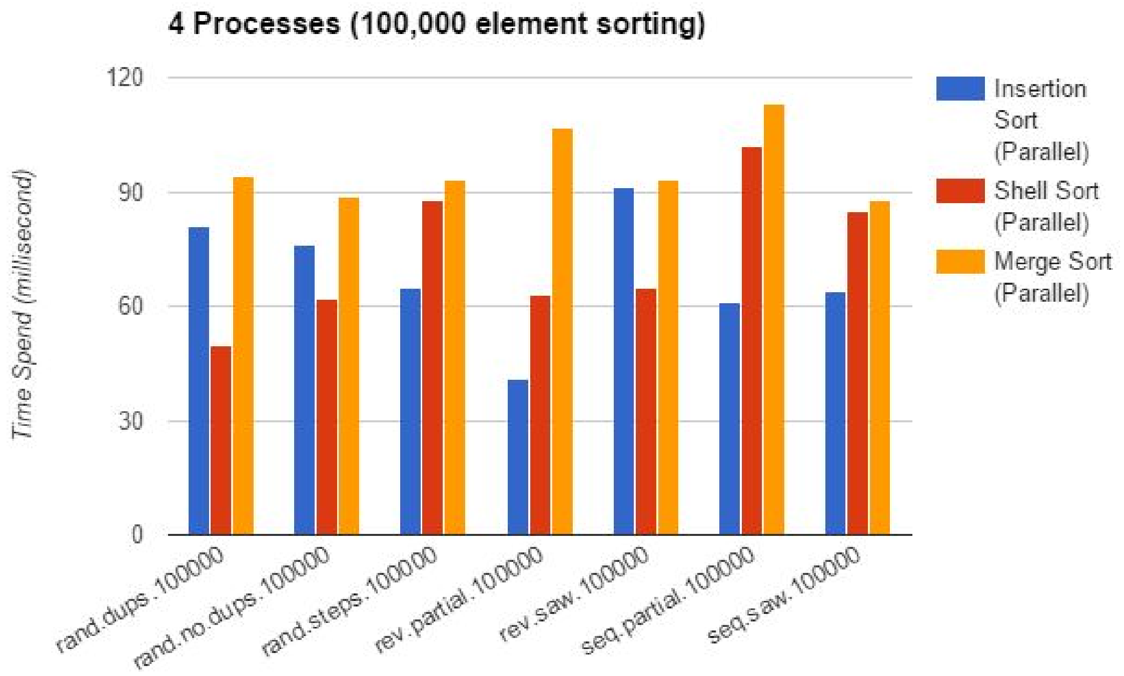
The bar chart above shows that when sorting 1,000 element with 4 processes merge Sort seems to have the best performance overall. Shell sort and insertion sort scale based on sorting data.

4 Processess	med.3.killer.10000	rand.dups.10000	rand.no.dups.10000	rand.steps.10000	rev.partial.10000	rev.saw.10000	seq.partial.10000	seq.saw.10000
Insertion Sort (Parallel)	52	62	86	23	104	67	49	92
Shell Sort (Parallel)	60	51	45	53	67	61	48	43
Merge Sort (Parallel)	55	60	48	31	48	45	38	56



The bar chart above shows that as the data get larger, shell sort seems to scale better, and merge sort start to taking longer to sort .

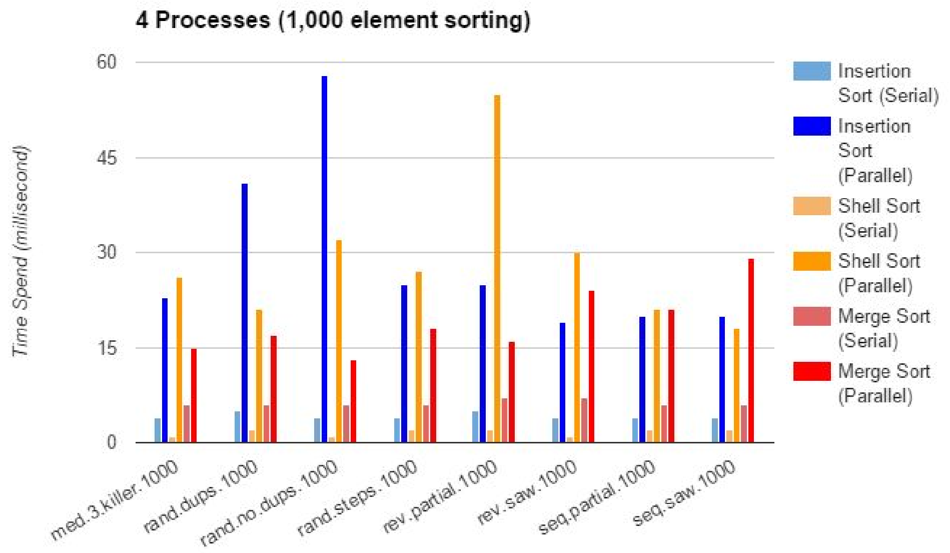
4 Processess	rand.dups.100000	rand.no.dups.100000	rand.steps.100000	rev.partial.100000	rev.saw.100000	seq.partial.100000	seq.saw.100000
Insertion Sort (Parallel)	81	76	65	41	91	61	64
Shell Sort (Parallel)	50	62	88	63	65	102	85
Merge Sort (Parallel)	94	89	93	107	93	113	88



The bar chart above shows that merge Sort is taking longer to sort while shell sort and insertion start to get better performance.But over all parallel insertion sort has the most speed up.

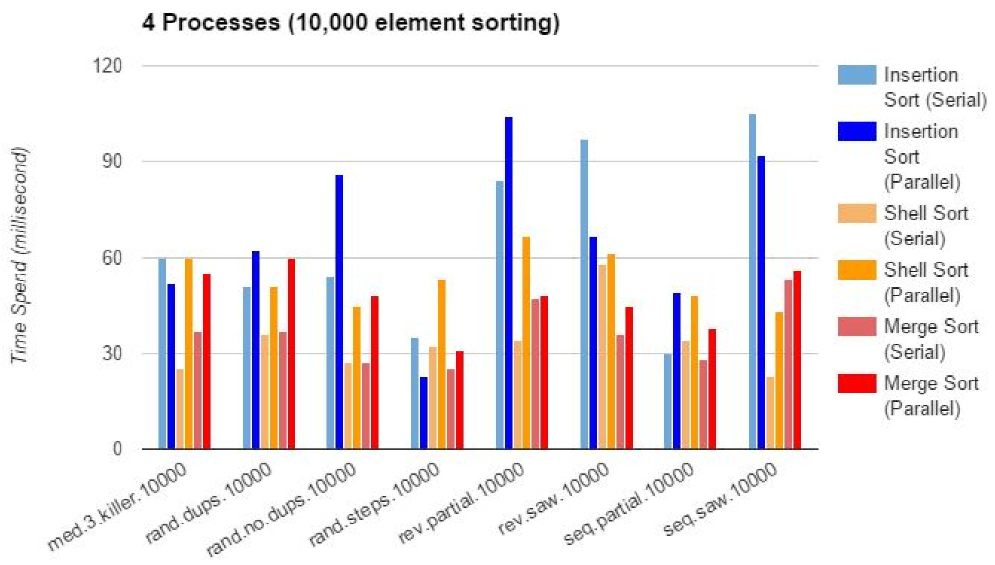
Parallel Sort VS Serial Sort

4 Processess	med.3.killer.1000	rand.dups.1000	rand.no.dups.1000	rand.steps.1000	rev.partial.1000	rev.saw.1000	seq.partial.1000	seq.saw.1000
Insertion Sort (Serial)	4	5	4	4	5	4	4	4
Insertion Sort (Parallel)	23	41	58	25	25	19	20	20
Shell Sort (Serial)	1	2	1	2	2	1	2	2
Shell Sort (Parallel)	26	21	32	27	55	30	21	18
Merge Sort (Serial)	6	6	6	6	7	7	6	6
Merge Sort (Parallel)	15	17	13	18	16	24	21	29



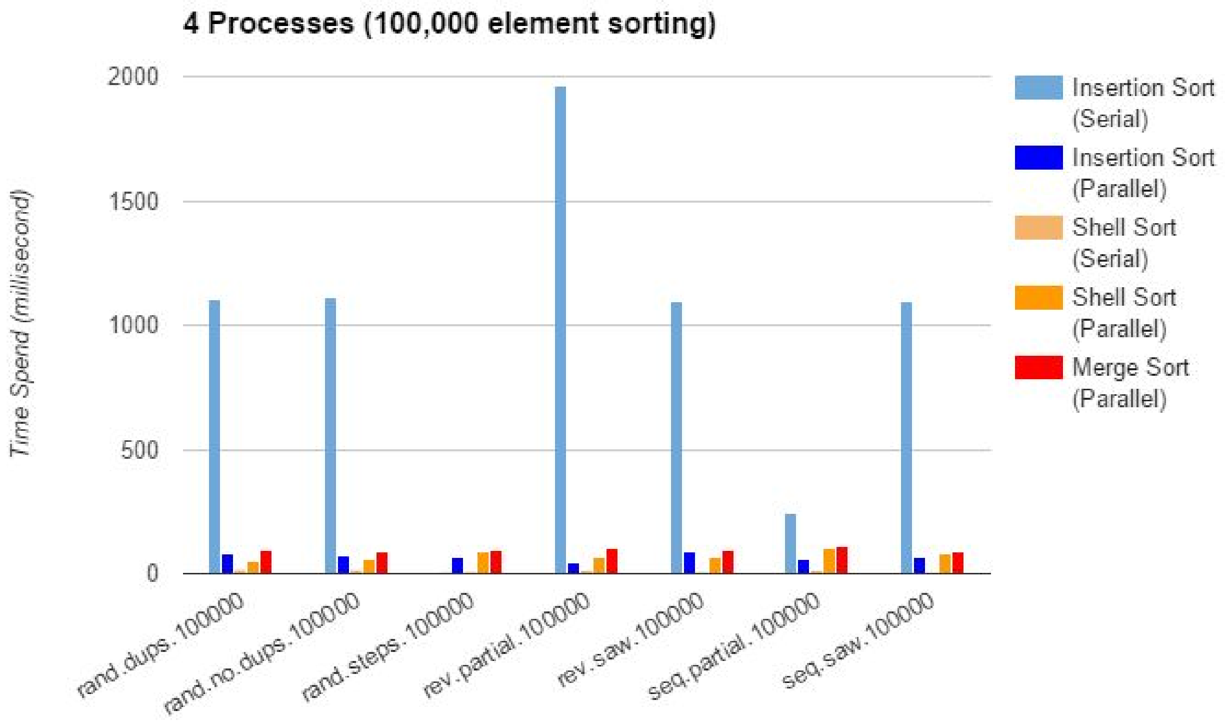
The bar chart above shows the time it takes for the three sorting algorithm to sort 1000 elements with 4 processes all three parallel sort seems to take longer than the serial version when sorting small data set, is because the execution time on the grid factors in network latency, bandwidth and chunk processing time.

4 Processes	med.3.killer.10000	rand.dups.10000	rand.no.dups.10000	rand.steps.10000	rev.partial.10000	rev.saw.10000	seq.partial.10000	seq.saw.10000
Insertion Sort (Serial)	60	51	54	35	84	97	30	105
Insertion Sort (Parallel)	52	62	86	23	104	67	49	92
Shell Sort (Serial)	25	36	27	32	34	58	34	23
Shell Sort (Parallel)	60	51	45	53	67	61	48	43
Merge Sort (Serial)	37	37	27	25	47	36	28	53
Merge Sort (Parallel)	55	60	48	31	48	45	38	56



The bar chart above shows the time it takes for the three sorting algorithm to sort 10,000 elements with 4 processes. the three parallel sorting algorithm starting to scale better while the data get bigger. And the serial sorting algorithm is starting to take longer to sort.

4 Processess	rand.dups.100000	rand.no.dups.100000	rand.steps.100000	rev.partial.100000	rev.saw.100000	seq.partial.100000	seq.saw.100000
Insertion Sort (Serial)	1107	1109	6	1960	1099	241	1096
Insertion Sort (Parallel)	81	76	65	41	91	61	64
Shell Sort (Serial)	17	16	8	14	7	14	7
Shell Sort (Parallel)	50	62	88	63	65	102	85
Merge Sort (Serial)	17	17	15	14	12	14	12
Merge Sort (Parallel)	94	89	93	107	93	113	88



The bar chart above shows the time it takes for the three sorting algorithm to sort 100,000 elements with 4 processes. The serial version of insertion sort perform really bad if the data is large and not almost sorted. But for”rand.steps” file insertion sort seems to have the best performance is because the file is almost sorted. Insertion sort has a cost of O(1) if the file is sorted that why insertion is the fastest in this situation. But overall the parallel version of insertion sort seems to have the most speed up.