MEI Report – 1st assignment

Pedro Alexandre G. Ribeiro, 2016218753

Fábio Gomes Cordeiro, 2016242575

Francisco Barão, 2016238992

Problem Statement

Our research question is “How are sorting algorithms affected by faults?”.

We first started out by thinking of what variables would be good to compare how the algorithms behave and we came up with number of swaps to re-organize the faulty array and the longest subarray provided by the professor divided by the number of elements so as to see a rough indicator of the percentage of the sorted array.

We then moved on to analyzing and thinking about how the dependent variables vary when we alter the independent variables.

After that we performed the experiments, draw the graphs where we took our conclusions.

Variables

The independent variables are the number of elements in the array, the probability of occurring an error and the range of numbers which will be ordered.

The dependent variables are the length of the longest non-decreasing string and the number of swaps needed to the sort the array that was resulted from the faulty algorithms.

For our experiments we chose number of swaps since we can analyze how scrambled the sorted arrays were after faults and the longest subarray since we can divide the size of this subarray by the number elements to get a rough estimate of the percentage of the sorted part of the array. Following that, we decided to vary each independent variable with some default environments. This variation led to gradually increasing each independent variable at a time whilst looking at the output. This way we can check the evolution of values in two separate environments.

Experiments

As to Cost/Benefit, for our experiments, we decided to create two environments for each independent variable so as not to exponentially increase the amount of information and graphs that we had to analyze. These two environments are designed to visualize how the dependent variables change with two distinct setups.

We ran each of our tests five times and calculated the median of each subset to get more accurate results.

We then calculated the median and standard Deviation.

Finally, we drew the graphs and the linear regression to better visualize and analyzed them.

Expected results

From an early view, we believed that if we increased the value range, the faults would be more significant which means the array would be less organized.

We also believed that if we have more elements for the same amount of error probability, there will be more chances of the fault occurring which leads to more swaps and smaller longest sorted subarray.

Last but not least, we were of the opinion that more error probability would also increase the number of swaps and make the longest sorted subarray smaller.

Environments

These two environments consist of dividing the range of each variable by three and using the values first and last values which allowed us to see how the variables behave in two different scenarios to better evaluate them.

For the value range we increased the range from 0.1 to 0.9 in 50 steps in the following environments:

	Environment 1	Environment 2
Number of elements	2500	7500
Error Probability	0.01	0.0033

Table 1 – Value range Environments

For the number of elements we increased the numbers from 200 to 10000 in 50 steps in the following environments:

	Environment 1	Environment 2
Value range	0.25	0.75
Error Probability	0.01	0.0033

Table 2 – Number of elements Environments

We decided to vary the error from 1/100 to 1/300 because if we gave too big or too small a values, they would be too predictable. We increased the probability from 0.002 to 0.04 in 50 steps in the following environments:

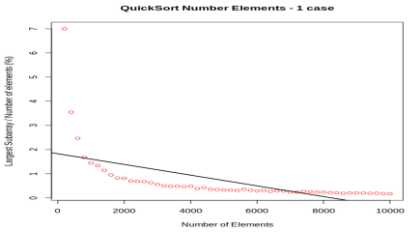
	Environment 1	Environment 2
Number of elements	2500	7500
Value range	0.25	0.75

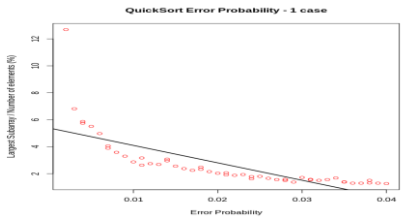
Table 3 – Error Probability Environments

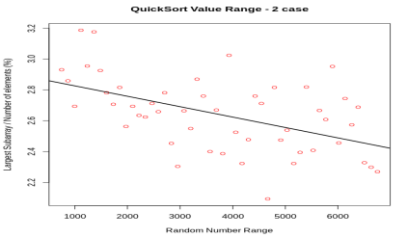
Graphs

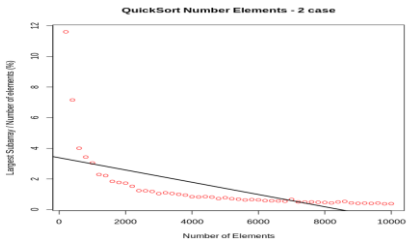
Variable - Longest Subarray

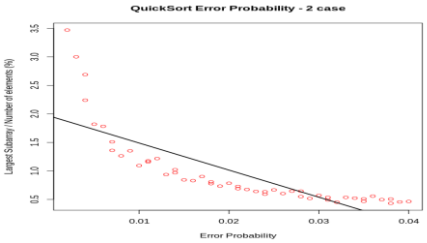
Quicksort



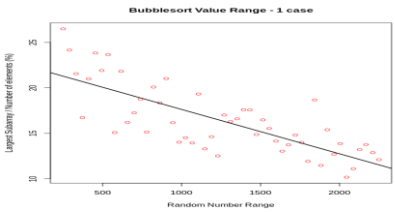


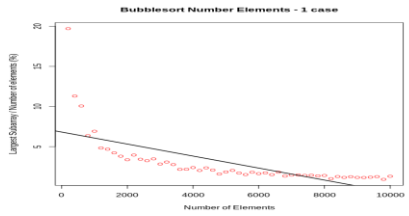


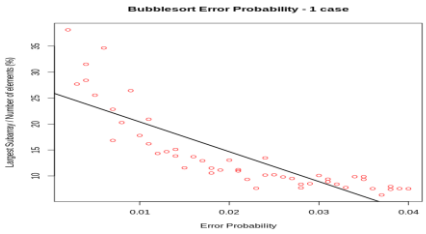


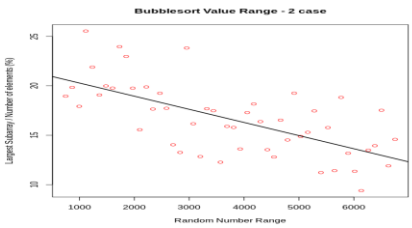


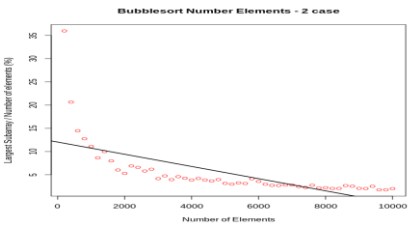
Bubblesort

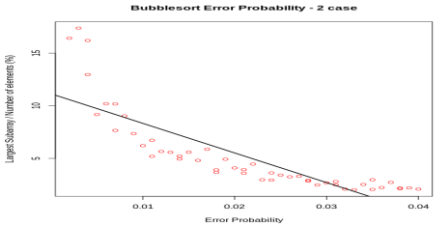




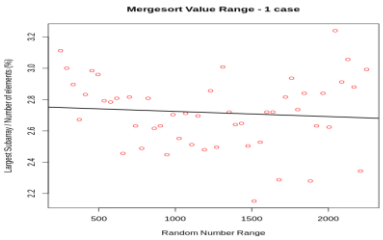


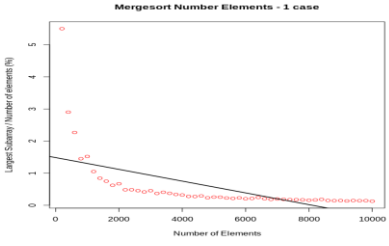


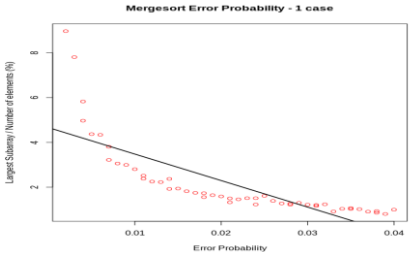


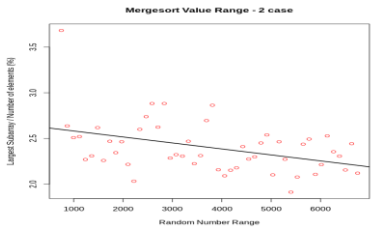


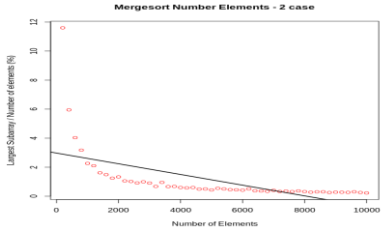
Mergesort

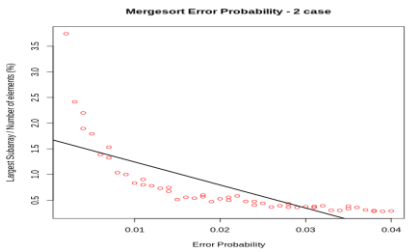




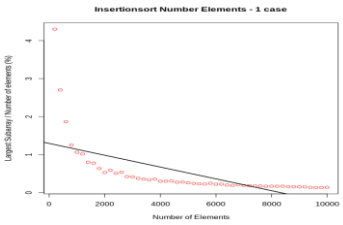


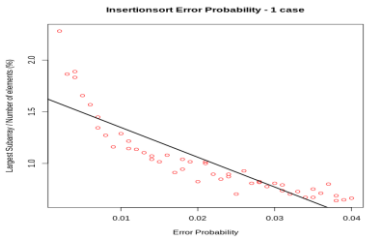


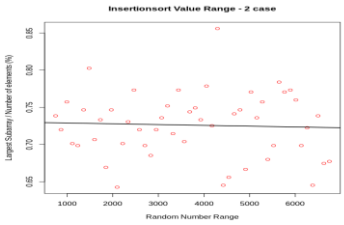


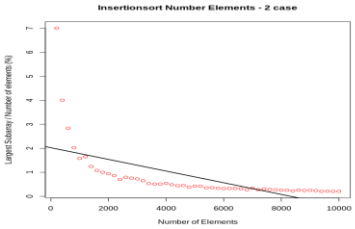


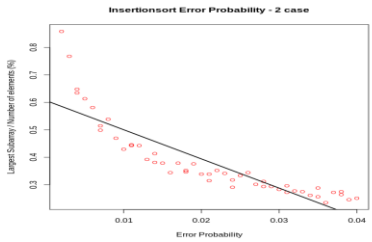
Insertionsort











Analysis:

For the value range, we were able to see that the relation longest subarray/number elements tend to decrease when we increase the value range. When we have a small pool of numbers to populate our array, there tends to be several repeated numbers which means that when an error occurs and we wrongly compare it, the probability of the adjacent numbers being equal to itself is fairly high.

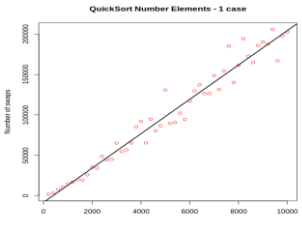
We can also see that see that this is where there might be some outliers. however this may not be case since the values are more disperse in comparison to the remaining graphs.

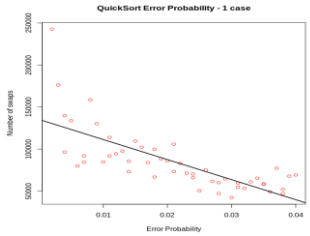
For the number of elements, we can see that the two environments have different points of steepest decline. We can see that for all algorithms on the environment 1 have the decline at a lower number of elements (more or less at 1000) whereas on the environment 2 they have the decline later (more or less at 2000). We also can see that the longest subarray/number of elements tend do stagnate having little impact if we increase the number of elements.

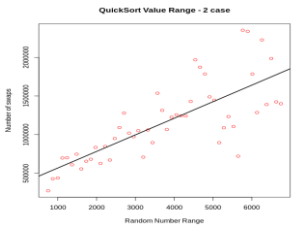
For the error probability, we can see that it varies in the same way as the number of elements. When we increase the error probability, the longest sorted subarray tends to be smaller tending to stagnate towards a low value. In this variable, we can also see the differences between the algorithms. Quicksort, Mergesort and Insertionsort tend to stagnate lower than Bubblesort, the last one longer sorted subarrays with a smaller error probability which we believe is due to the number of comparations in the algorithms. The lower the number of comparations, less prone to error it is so it will have bigger sorted arrays.

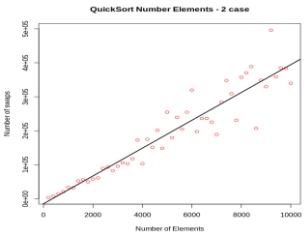
Variable – Number of Swaps to re-organize the faulty array

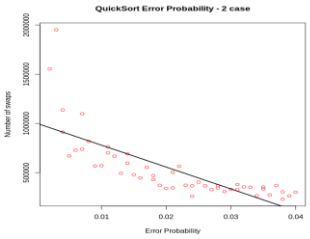
Quicksort



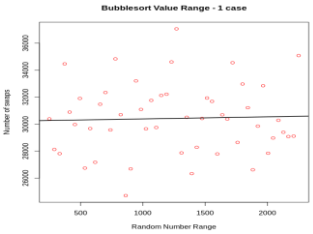


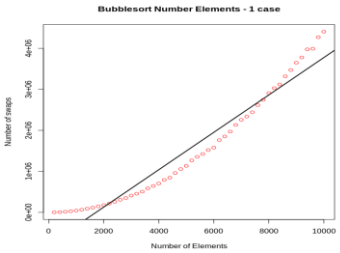


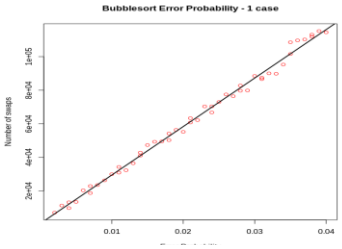


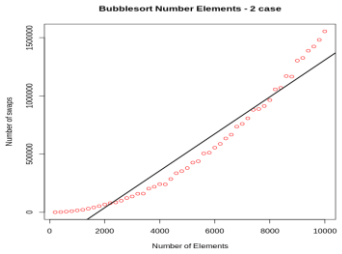


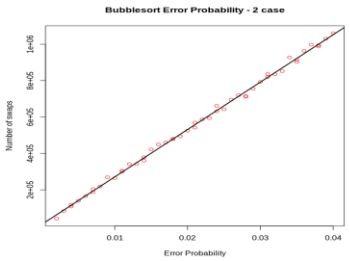
Bubblesort



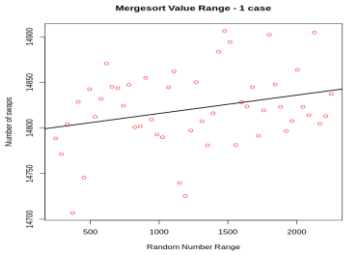


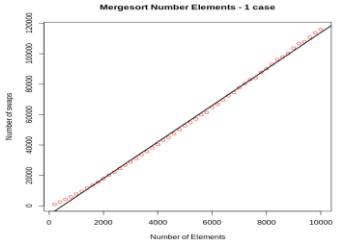


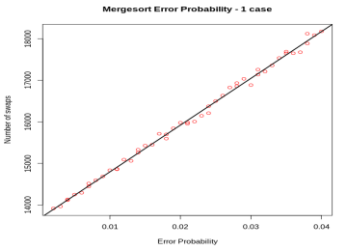


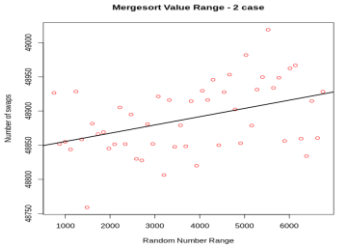


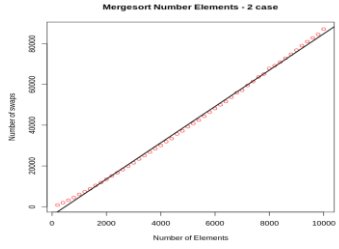
Mergesort

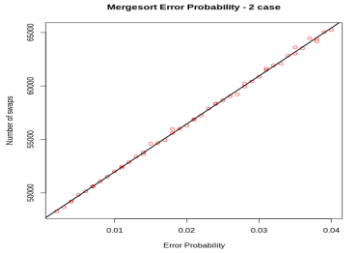




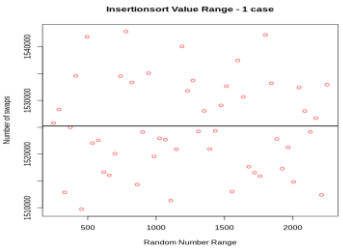


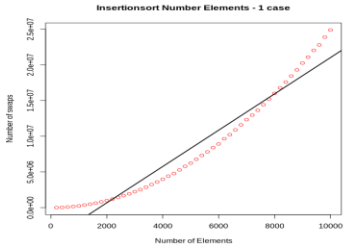


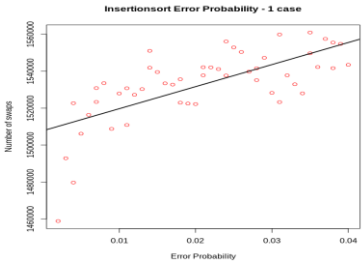


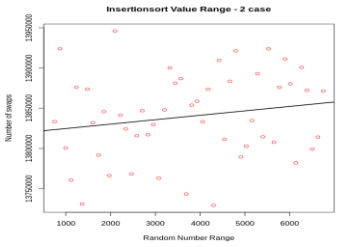


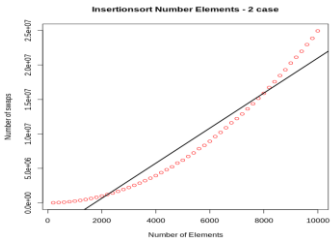
Insertionsort

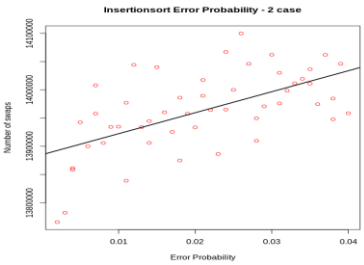












Analysis:

For the value range, we can see that Quicksort's number of swaps tends to grow as the value range increases, for the other three algorithms the values don't vary that much, so we can conclude that the variation of value range doesn't affect much the number of swaps. This is best seen by the size of window of the graphs and in the standard deviation calculated below.

For the number of elements, Quicksort and Mergesort both have a linear grow when increasing the number of elements in the array, as the number of swaps grows linear with the increase in elements. Insertion sort and Bubble sort grow much faster, reaching values at least 5 times higher than the other two algorithms.

Last but not least, the error probability in most of the graphs go according to our expectations, they all tend to grow in a similar way, as the error probability increases the number of swaps also increases. This is true for all graphs except for quicksort, which with the increase in error probability, the number of swaps needed to sort the array decreases. We can't reach a possible conclusion to why this is happening.

Median / Standard Deviation

For our experiments, we decided to develop a table to see the standard deviation and median.

Var Dependent	Algorithm	Var Independent	Environment	Median	Standard Deviation
Longest Subarray	Quicksort	Value range	1	3.199	0.349
Longest Subarray	Quicksort	Value range	2	2.640	0.237
Longest Subarray	Quicksort	Number of elements	1	0.697	1.099
Longest Subarray	Quicksort	Number of elements	2	1.339	1.891
Longest Subarray	Quicksort	Error probability	1	2.674	1.965
Longest Subarray	Quicksort	Error probability	2	0.966	0.668
Longest Subarray	Bubblesort	Value range	1	16.409	3.782
Longest Subarray	Bubblesort	Value range	2	16.626	3.599
Longest Subarray	Bubblesort	Number of elements	1	3.004	3.213
Longest Subarray	Bubblesort	Number of elements	2	5.327	5.738
Longest Subarray	Bubblesort	Error probability	1	14.093	7.732
Longest Subarray	Bubblesort	Error probability	2	5.231	3.824
Longest Subarray	Mergesort	Value range	1	2.716	0.226
Longest Subarray	Mergesort	Value range	2	2.401	0.289
Longest Subarray	Mergesort	Number of elements	1	0.549	0.894
Longest Subarray	Mergesort	Number of elements	2	1.092	1.843
Longest Subarray	Mergesort	Error probability	1	2.173	1.705
Longest Subarray	Mergesort	Error probability	2	0.749	0.667
Longest Subarray	Insertionsort	Value range	1	1.230	0.094
Longest Subarray	Insertionsort	Value range	2	0.726	0.043
Longest Subarray	Insertionsort	Number of elements	1	0.501	0.720
Longest Subarray	Insertionsort	Number of elements	2	0.784	1.141
Longest Subarray	Insertionsort	Error probability	1	1.031	0.369
Longest Subarray	Insertionsort	Error probability	2	0.383	0.137
Number of Swaps	Quicksort	Value range	1	136230.8	44710.67
Number of Swaps	Quicksort	Value range	2	1157585	499475.4
Number of Swaps	Quicksort	Number of elements	1	100141.3	63147.3
Number of Swaps	Quicksort	Number of elements	2	193965.7	126592.6
Number of Swaps	Quicksort	Error probability	1	85181.35	36622.49
Number of Swaps	Quicksort	Error probability	2	543164.9	329747.5
Number of Swaps	Bubblesort	Value range	1	30425.3	2589.476
Number of Swaps	Bubblesort	Value range	2	89433.01	6335.545
Number of Swaps	Bubblesort	Number of elements	1	1540939	1367044
Number of Swaps	Bubblesort	Number of elements	2	531167.8	477403
Number of Swaps	Bubblesort	Error probability	1	61224.86	32685.8
Number of Swaps	Bubblesort	Error probability	2	557393.4	293795.9
Number of Swaps	Mergesort	Value range	1	14820.96	42.816
Number of Swaps	Mergesort	Value range	2	48888.71	51.095
Number of Swaps	Mergesort	Number of elements	1	55171.34	34922.43
Number of Swaps	Mergesort	Number of elements	2	41173.68	25988.39
Number of Swaps	Mergesort	Error probability	1	16033.25	1271.912
Number of Swaps	Mergesort	Error probability	2	56900.84	5063.522
Number of Swaps	Insertionsort	Value range	1	1525228	8704.189
Number of Swaps	Insertionsort	Value range	2	13839886	53660.98
Number of Swaps	Insertionsort	Number of elements	1	8559484	7638196
Number of Swaps	Insertionsort	Number of elements	2	8549711	7648768
Number of Swaps	Insertionsort	Error probability	1	1532833	19544.45
Number of Swaps	Insertionsort	Error probability	2	13963174	70559.19

Table 4 – Means and standard deviations

Conclusion

Most of the results we got were as expected, with the increase of value range, number of elements or error probability, the largest subarray (% of full array) tended to decrease and the number of swaps tended to increase as well.

The ones we didn’t expect were the value range tests corresponding to the number of swaps of all algorithms except quicksort because the value range didn’t really affect the number of swaps. Another exception was the error probability graph of quicksort, since the relation is the opposite of what we expected, the number of swaps decreases with the increase of error probability. We fail to explain why.

Bubblesort got a higher median of largest subarray vs the other algorithms and we believe it to be because of the number of comparisons. Due to having less comparations it causes less errors which is a good thing, but it also needs a high number of swaps to really sort the array, which is a bad thing since we want to test algorithms in a faulty environment, and more swaps indicates more memory access which results in more errors.

In overall ranking Insertion sort is the worst one achieving a mean of swaps a lot higher than the rest of the algorithms. Mergesort and quicksort are kinda similar in faulty environments except for the error probability where quicksort takes the first place.