Report **Exploratory Data Analysis, Hypothesis Testing and Linear Regression**

Experimental Methods in Computer Science, 5 December 2019

Students: António Eloi (2016231383), Isabel Carvalho (2016212943), José Pereira (2016228030)

Problem Statement

How are sorting algorithms affected by memory faults?

Description Statement

Errors due to memory faults are very common (5000 failures in time per MBit).
Although there are already techniques that detect and correct memory faults, sorting algorithms are strongly affected by this, since they can take the wrong path after reading corrupted values, propagating the error throughout their execution.
In this work, we studied how different sorting algorithms can be affected by this kind of error. In order to do this, we used a memory fault model, that, with a given probability, caused faulty memory access between values of a given array.

Assumptions and Recommendations

We considered for this assignment an array of integers in non-decreasing order and that was based on pairwise comparisons. The sort algorithms used were:

- Quicksort (represented in the graphics by the colour **orange**)
- Mergesort (represented in the graphics by the colour **green**)
- Insertion sort (represented in the graphics by the colour **red**)
- Bubblesort (represented in the graphics by the colour **blue**)

We simulated the memory fault model explicitly in the code of each sorting algorithm, that is, whenever a comparison was performed between two values in the array, there was a probability that a failure would occur and that the outcome of the comparison would be wrong.

Each of the four implementations received a file from stdin as input. This file contained a single line. It started with a real number between 0 and 1, which corresponded to the probability that a fault would occur as above. Then, it was followed by a positive integer corresponding to the size of the array to be sorted.
Then, a list of n positive integers would follow, which corresponded to the content of the
array.

To accomplish this we produced a bash script and some python scripts too. To generate all input/output/results we ran executable.sh. The script arguments were N_MIN, N_MAX, and the STEP. The array created by this script followed a uniform distribution.

Each implementation of a sorting algorithm produced an output structured as follows:

1. Size of the array used - *n ;*
2. Memory Fault probability change - *eps ;*
3. Max random number used in the array - *max_r ;*
4. The size of the largest sorted subarray ;
5. The number of unsorted sub-arrays ;
6. The number of elements unsorted, considering the whole array;
7. The number of comparisons with memory fault;
8. The number of comparisons without any memory fault.

This output was considered as a rough indicator of the quality of the array produced under the memory fault model.

Time was not considered as an indicator since we considered it was not relevant to the problem statement - “How are sorting algorithms affected by memory faults?” - This is because the number of memory faults wouldn’t be influenced by it. Instead, we considered the number of comparisons made by the faulty algorithms and the well-implemented algorithms, which reflected the time, since an algorithm takes longer to run if it has to make more comparisons.

Variables identification

Independent Variables

- Sequence size;
- Memory fault probability;
- The maximum range used by the function "randint";
- Sorting algorithms.

Dependent Variables

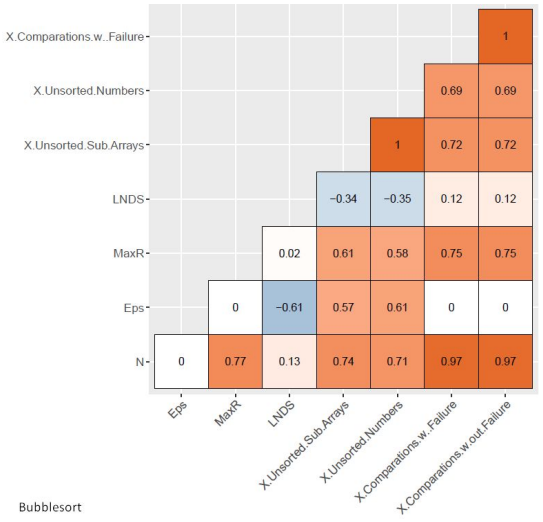
- Length of the longest non-decreasing sequence;
- Number of unsorted sub-arrays;
- Number of unsorted elements;
- Number of comparisons with memory fault;
- Number of comparisons without any memory fault.

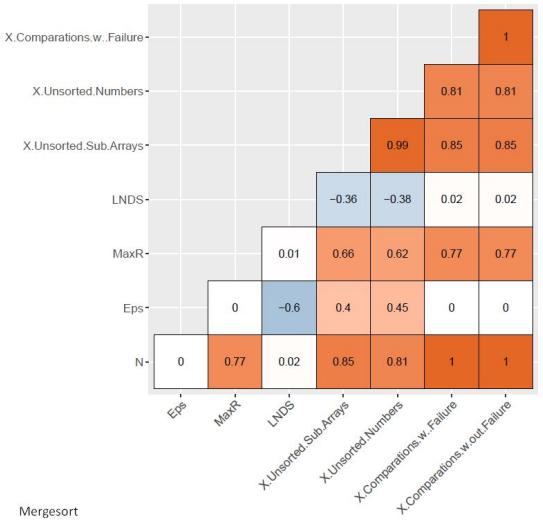
Exploratory data analysis

Histograms

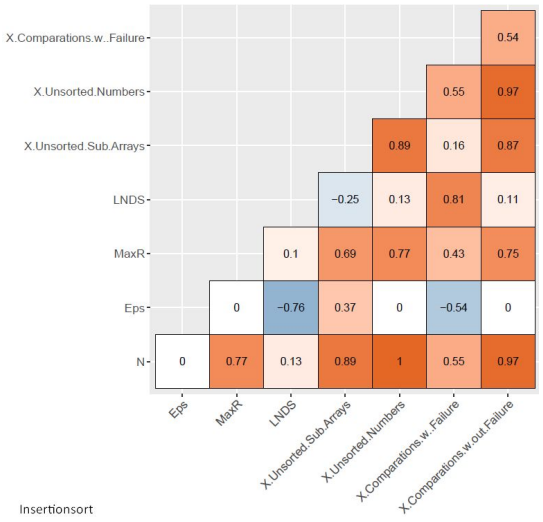
Analysing the histograms from all algorithms we noticed the data is mostly left-skewed making it good to use boxplot since it uses median. We also noticed a difference between the # of comparisons between the faulty and non-faulty algorithm which was later confirmed as will be shown.

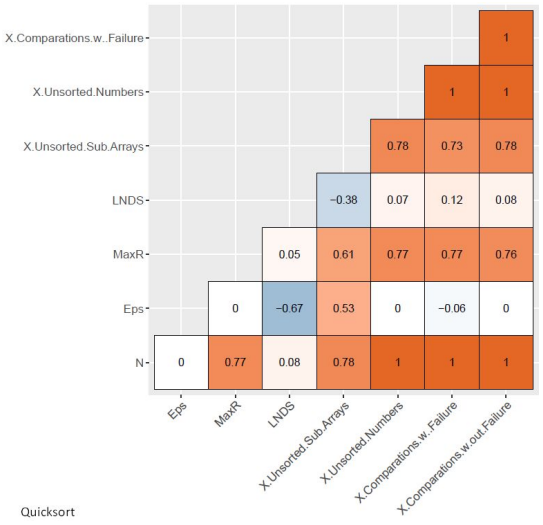
Correlations





Figures 1 and 2: Bubblesort’s correlations (Left) and Mergesort’s correlations (Right)





Figures 3 and 4: Insertionsort's correlations (Left) and Quicksort's correlations (Right)

- All algorithms showed a very high correlation between **array size** and t**he number of comparisons done by the faulty algorithm** with the exception of *insertion sort.* This can be explained by how the algorithm works. Since the insertion sort iterates over the array, looking for a number bigger than the one being iterated over, if a memory fault occurs it's going to do fewer comparisons than it's supposed to.

- As expected the **array size** is also highly correlated with the **number of unsorted elements**.

- There's no correlation between the **array size** and the **longest non-decreasing sequence**. The **LNDS** is highly influenced by the memory fault probability, **eps**.

- The **memory fault probability** has a negative correlation with the **longest non-decreasing sequence** meaning the values will decrease as the memory fault probability increases.

- In Insertion sort there's also a negative correlation regarding the **memory fault probability**/**longest non-decreasing sequence** (positive correlation) and the **number of comparisons by the faulty algorithm**.

The ratio of the number of comparisons between a faulty algorithm and the correct one

In what concerns the difference of the number of comparisons done by an algorithm with memory fault against the same algorithm with no memory fault we had a theoretical assumption as we run the tests. Our supposition was that with memory fault errors the algorithm would need fewer comparisons on insertion algorithms as it would find sooner a place to insert the element that would satisfy the sorting condition.
We considered the **Ratio** as the division of the number of comparisons realized in the faulty sort over the corrected sort.

Insertion Sort

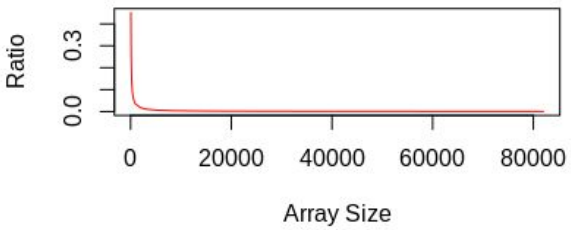


Figure 5: Relation between Ratio and Array Size for Insertion Sort

In this situation we see the ratio tends to 0 as the algorithm does more comparisons when it does not have any fault. This happens because, as we order an array, it is easier to insert an element anywhere that satisfies the compare condition. For this reason, we also have to verify that there are more elements out of order as the N increases.

Bubble Sort

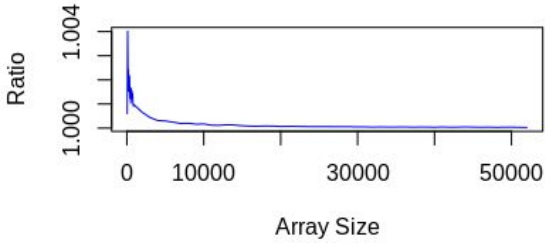


Figure 6: Relation between Ratio and Array Size for Bubble Sort

In this situation we see on the limit of N to infinite the ratio analysed is getting close to 1 as the algorithm does more comparisons when it has memory fault. This happens because when there is a memory fault the algorithm has to compare all the elements again to sort it since the code was made in a way to verify that if the algorithm was already sorted it wouldn't have to keep comparing the elements.

Merge Sort

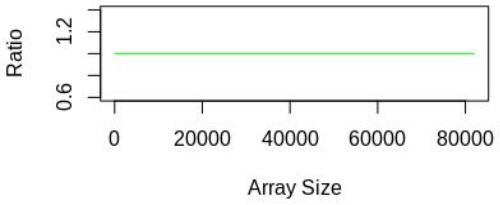


Figure 7: Relation between Ratio and Array Size for Merge Sort

The result obtained was expected because the mergesort algorithm proceeds in the same way regardless of the array elements' order.

Quick Sort

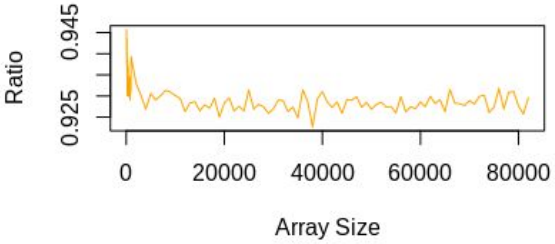


Figure 8: Relation between Ratio and Array Size for QuickSort

About this information, we saw that generally the quicksort works better and is faster when there is no memory fault. This helps to choose the right pivot and optimize the search for the solution.

The number of unsorted subarrays by the size of the array

As we expected the number of unsorted subarrays get larger as the size array gets bigger, there isn't much data we can infer from it.

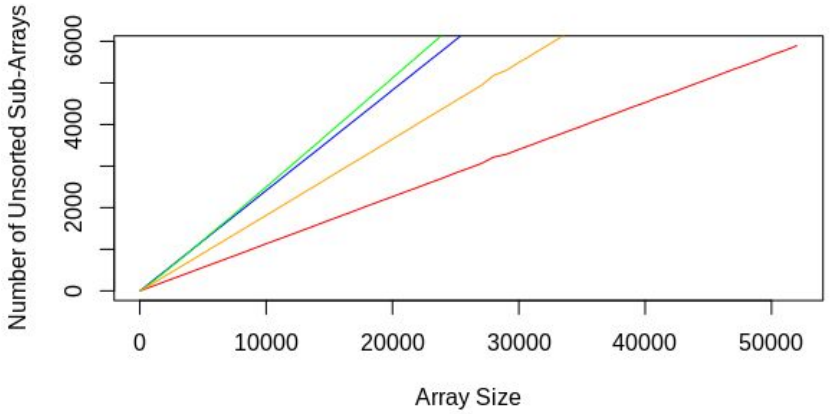
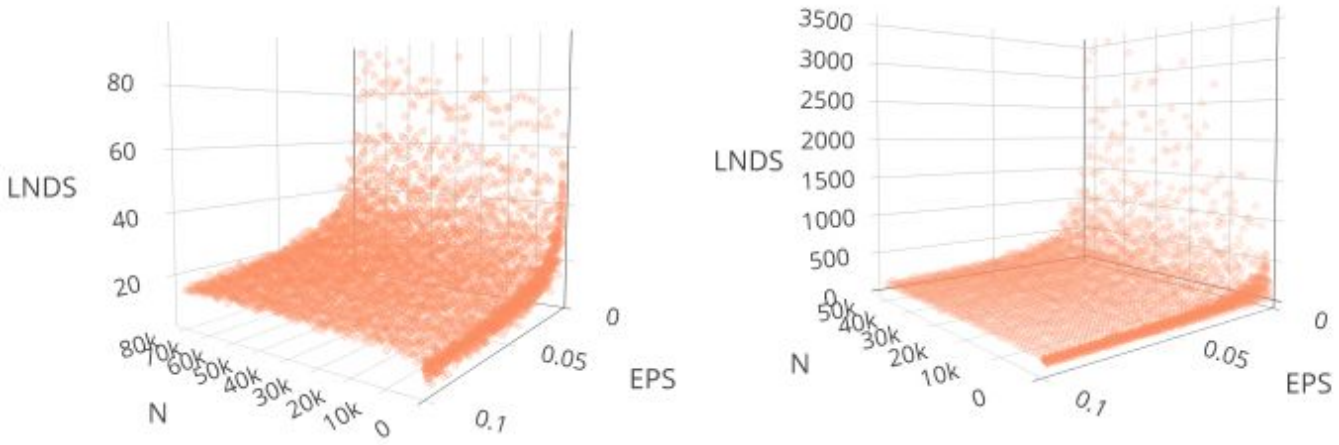


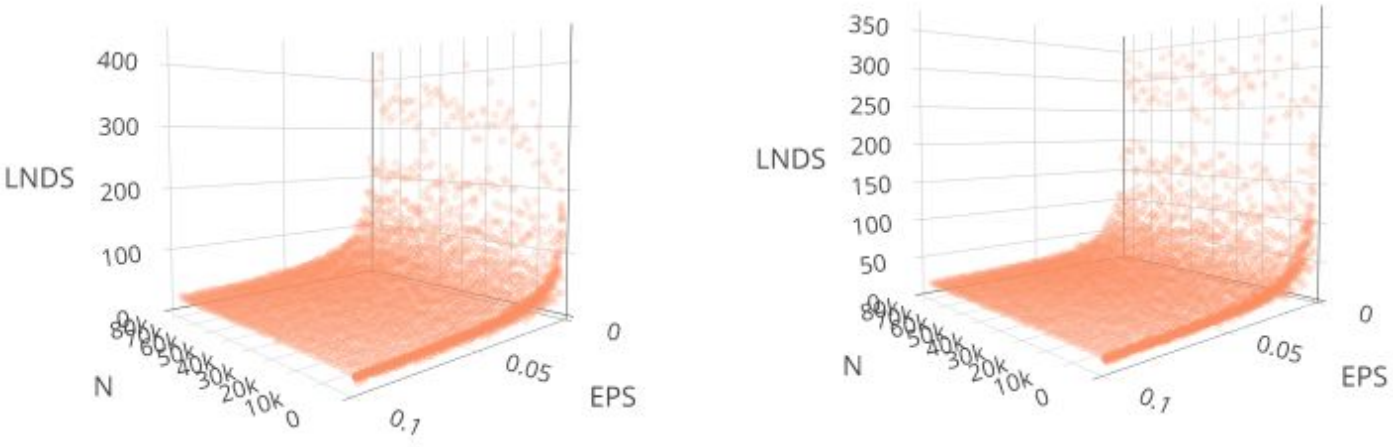
Figure 9: Relation between Array Size and Number of Unsorted Sub-Arrays for each algorithm tested. Green represents Mergesort. Blue represents Bubblesort. Orange represents Quicksort. Red represents Insertionsort.

The relation between Array Size, Memory Fault Probability & Longest Non-Decreasing Sequence

As we saw by the previous correlation matrix, the **longest non-decreasing sequence** values are highly influenced by the **memory fault probability**, not by the **array size.**
Below we can see the relationship between those variables on a 3D scatter plot for each sorting algorithm.



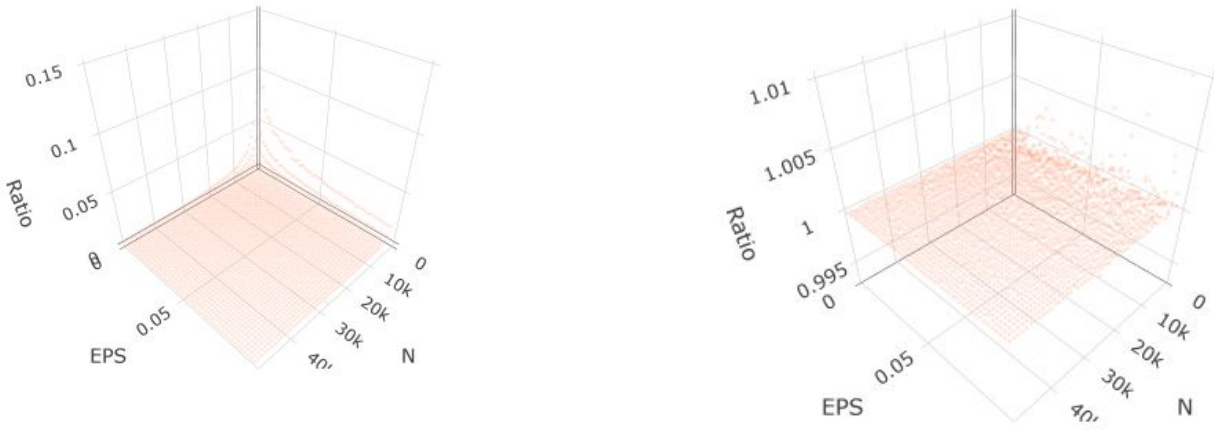
Figures 10 and 11: 3D Scatter Plots for Insertion Sort (Left) and for Bubble Sort (Right)



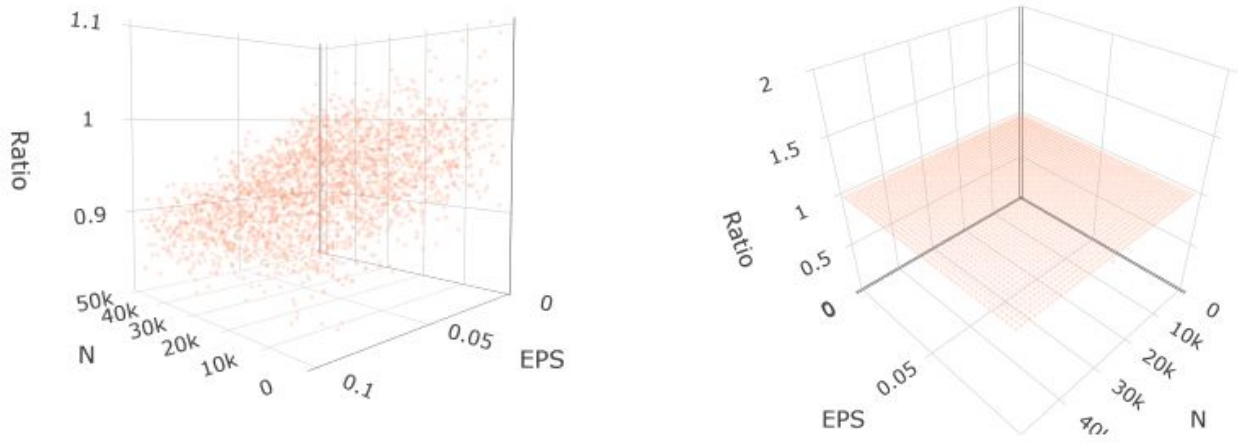
Figures 12 and 13: 3D Scatter Plots for Quick Sort (Left) and for Merge Sort (Right)

The relation between Array Size, Memory Fault Probability & Ratio Comparisons with error / without

We found interesting the individual plots for the relation between the size of the arrays and the ration so we decided to plot a 3d scatter plot for each algorithm. We wanted to test for all algorithms aggregated on a single plot but we didn't manage to achieve a good plot visualization so we gave up and moved on.



Figures 14 and 15: 3D Scatter Plots for Insertion Sort (Left) and for Bubble Sort (Right)



Figures 16 and 17: 3D Scatter Plots for Quick Sort (Left) and for Merge Sort (Right)

Conclusion:

With this assignment, we concluded that the memory fault errors on sorting algorithms have an interesting effect on the run-time and comparisons made on it. It may cause them to run faster as they won't iterate over the array as they would on the correct algorithm. For sorting algorithms that have to iterate all over again, they may take more time / do more comparisons. Our research and results also proved what we expected prior to the start of the development of the assignment. With more probability of memory fault, more errors happen, with the size of an array taking no effect on it.

Observações da análise exploratória de dados

1. O algoritmo **bubblesort** é o algoritmo que é menos afetado com os erros de memória. No entanto pesa na sua computação, exigindo mais tempo de execução que um outro algoritmo abordado. Para além disto, revela comportamentos como:
    a. Para um determinado N, aumentando probabilidade de **erro de memória**, o número de comparações mantém-se estável.

2. **Para um qualquer dos algoritmos bitwise-comparison analisados**, **aumentando a probabilidade de erro de memória**, o tamanho da maior subarray ordenada diminui.

3. **Para um qualquer dos algoritmos analisados,** o tamanho da array inicial influencia o tamanho da maior subarray ordenada.

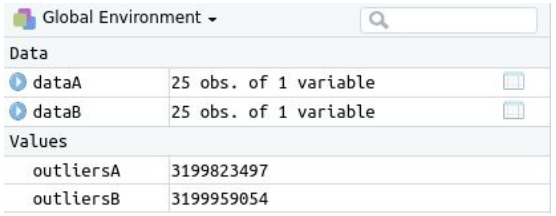
4. O algoritmo **mergesort** é um algoritmo em que o ratio de comparações feitas entre um ***faulty algorithm*** e o algoritmo correcto apresenta-se constante e igual a 1 para qualquer entrada de variáveis independentes.

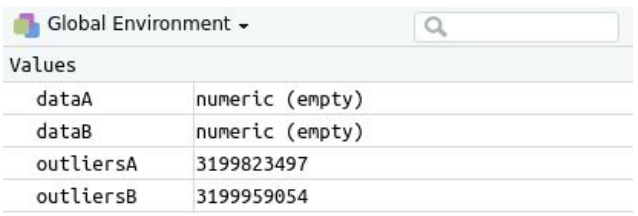
Com estas observações decidimos analisar alguns pontos e formular hipóteses de teste relativamente a elas.

Hypothesis testing

H_0 - No bubblesort, a probabilidade de erro de memória (eps) influencia o número de comparações para o algoritmo com falhas.

H_1 - No bubblesort, a probabilidade de erro de memória (eps) não influencia o número de comparações para o algoritmo com falhas.





Figuras 18 e 19: Informação dos outliers

No decorrer destes testes não removemos os outliers, pois dado que o tamanho da amostra era pequeno e os valores semelhantes, qualquer alteração era considerada outlier, resultando em que as amostras ficassem vazias aquando da remoção destes valores, como mostrado nas figuras 1 e 2.

```
        Welch Two Sample t-test

data:  dataA and dataB
t = -10.197, df = 24.003, p-value = 3.35e-10
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -70529.16 -46784.44
sample estimates:
 mean of x  mean of y
3199900995 3199959651
```

Figura 20: Resultado do t student

Como podemos ver na figura 3, para este teste o valor de P é inferior a 5%, pelo que H_0 é rejeitada.

Assim, podemos concluir que o eps não influencia significativamente o número de comparações realizadas para o algoritmo bubblesort com falhas.

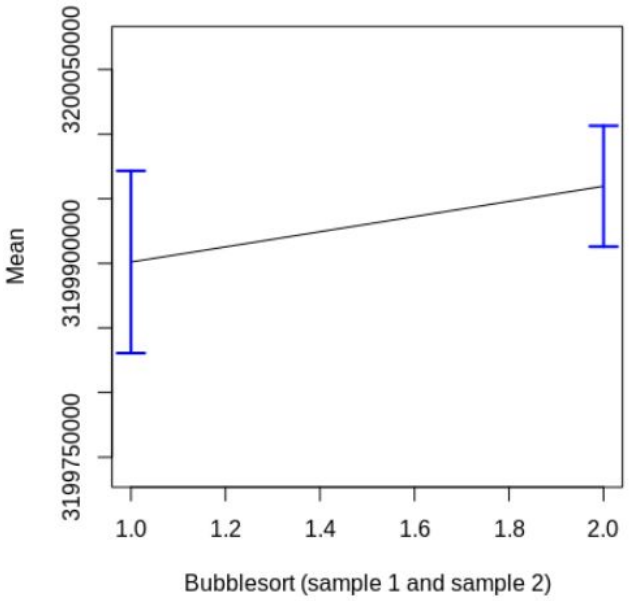


Figura 21: Gráfico que representa a média para cada amostra testada

A figura 4 representa a média para as duas amostras testadas e o intervalo de confiança correspondente, representado como uma barra de erro.

Conforme indicado nas nossas observações da análise exploratória de dados, o tamanho da array inicial influenciava o tamanho da maior subarray ordenada (**LNDS**). Não obstante, decidimos formalmente confirmar estes pressupostos recorrendo a testes à nossa hipótese.
Considerando as seguintes hipóteses nula e alternativa:

H_0 - No algoritmo de ordenamento **quicksort**, o tamanho da array N não influencia o tamanho da maior subarray ordenada (LNDS)

H_1 - No algoritmo de ordenamento **quicksort**, o tamanho da array N influencia o tamanho da maior subarray ordenada (LNDS)

Usando *Welch Two-Sample T-Test*:

```
        Welch Two Sample t-test

data:  dataA and dataB
t = -11.462, df = 44.122, p-value = 8.097e-15
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -16.08512 -11.27488
sample estimates:
mean of x mean of y
    23.12     36.80
```

Figura 22: Resultado do t-test

Como podemos ver na figura 5, para este teste o valor de P é inferior a 5%, pelo que H_0 é rejeitada.
Assim, podemos concluir que o tamanho do array (N) influencia o tamanho da maior subarray ordenada (LNDS).

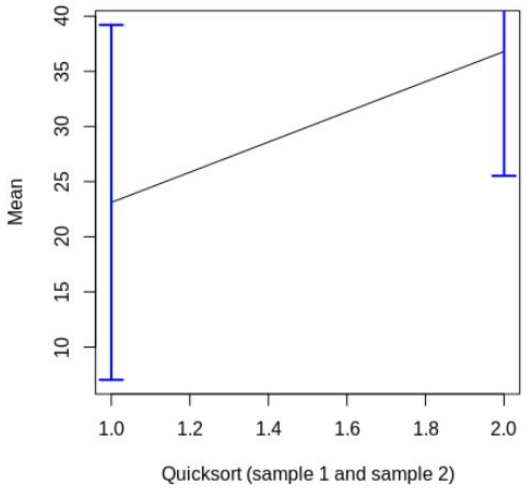


Figura 23: Gráfico que representa a média para as amostras testadas

A figura 6 representa a média para as duas amostras testadas e o intervalo de confiança correspondente, representado como uma barra de erro a azul.

H_0 - No algoritmo de ordenamento **mergesort**, o tamanho da array N não influencia o tamanho da maior subarray ordenada (LNDS)

H_1 - No algoritmo de ordenamento **mergesort**, o tamanho da array N influencia o tamanho da maior subarray ordenada (LNDS)

```
        Welch Two Sample t-test

data:  dataA and dataB
t = -4.3168, df = 44.608, p-value = 8.687e-05
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -4.810736 -1.749264
sample estimates:
mean of x mean of y
    18.04     21.32
```

Figura 24: Resultado do t student

Como podemos ver na figura 5, para este teste o valor de P é inferior a 5%, pelo que H_0 é rejeitada. Assim, podemos concluir que o tamanho do array (N) influencia o tamanho da maior subarray ordenada (LNDS).

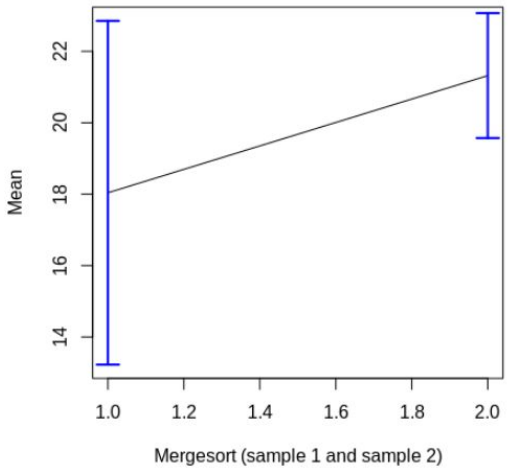


Figura 25: Gráfico que representa a média para as amostras testadas

A figura 8 representa a média para as duas amostras testadas e o intervalo de confiança correspondente, representado como uma barra de erro.

Como indicado nas nossas observações da análise exploratória de dados, o algoritmo **mergesort** é um algoritmo em que o ratio de comparações feitas entre um faulty algorithm e o algoritmo correcto apresenta-se constante e igual a 1 para qualquer entrada de variáveis independentes. Não obstante, decidimos formalmente confirmar estes pressupostos recorrendo a testes à nossa hipótese.

Considerando as seguintes hipóteses nula e alternativa:

H_0 - No mergesort, o ratio (número de comparações no algoritmo com falhas sobre o número de comparações no algoritmo sem falhas) não é influenciado pelo tamanho do array (N)

H_1 - No mergesort, o ratio (número de comparações no algoritmo com falhas sobre o número de comparações no algoritmo sem falhas) é influenciado pelo tamanho do array (N)

```
> t.test(data$V1, data$V2, var.equal = T)
Error in t.test.default(data$V1, data$V2, var.equal = T) :
  data are essentially constant
>
```

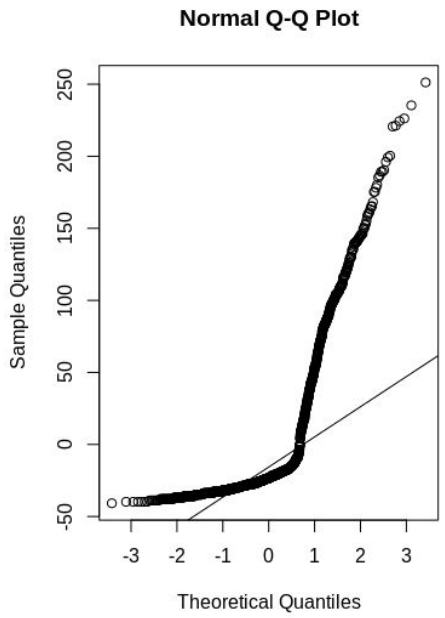
Figura 26: Resultado do t-test

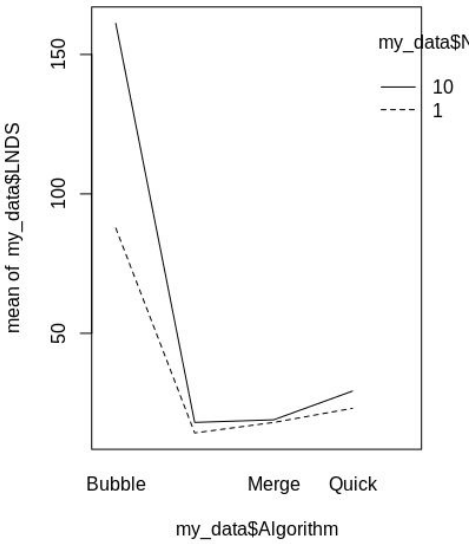
Visto não existir variância nos dados, como observado acima, H_0 é aceite. Este resultado é bastante interessante na medida que comprova os nossos conhecimentos à priori do algoritmo. A natureza do algoritmo é a principal razão destes resultados, a divisão da array a metade e o seu merge não é afetado pelos valores dos dados, sejam eles os correctos ou desordenados.

H_0^S - Os algoritmos analisados têm valores de média iguais.

H_0^P - Os tamanhos das arrays analisadas têm médias iguais.

H_0^{SP} - Não existe interação entre os algoritmos analisados.





Figuras 27 e 28: Gráfico do teste de normalidade e gráfico de interação (respectivamente)

```
        Bartlett test of homogeneity of variances

data:  LNDS by interaction(Algorithm, N)
Bartlett's K-squared = 3595.4, df = 7, p-value < 2.2e-16
```

```
        Shapiro-Wilk normality test

data:  aov.out$res
W = 0.83079, p-value < 2.2e-16
```

Figuras 29 e 30: Resultado do teste de homogeneidade e normalidade segundo Bartlett e Shapiro-Wilk

Tendo em conta que *não se verifica nem a normalidade nem a homogeneidade*, **p<0.05**, procedemos à realização do teste não paramétrico Two-Way ANOVA.

Após a análise dos resultados obtido, com ***p-value* igual a zero** para H_0^S , H_0^P e H_0^{SP} concluiu-se a rejeição de todas as hipóteses nulas isto é, o LNDS é afetado tanto pelo algoritmo como pelo tamanho da array, existindo interação entre os algoritmos e o tamanho da array.

Para a realização dos testes paramétricos, foi usada a média dos testes para cada valor da variável independente em inves dos valores das medições.

Tamanho da array entre 500 até 50 000 com intervalos de 500

Probabilidade de erro fixada em: 2%

Quicksort

```
Call:lm(formula = 1/log(N_LNDS$N) ~ N_LNDS$LNDS)

Residuals:
      Min        1Q    Median        3Q       Max
-0.014443 -0.006441 -0.001773  0.004249  0.046467

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept)  1.469e-01  2.400e-03    61.2   <2e-16
***
N_LNDS$LNDS -7.378e-04  3.945e-05   -18.7   <2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.009295 on 598 degrees of freedom
Multiple R-squared:  0.369,   Adjusted R-squared:  0.368
F-statistic: 349.8 on 1 and 598 DF,  p-value: < 2.2e-16
```

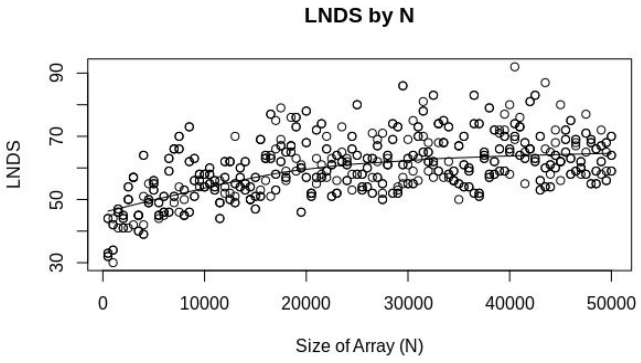
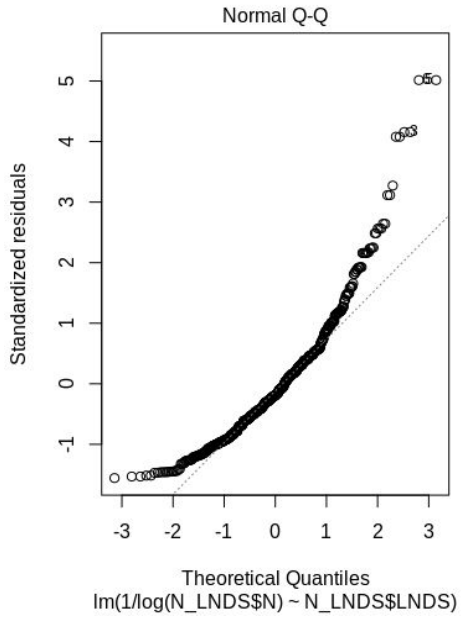
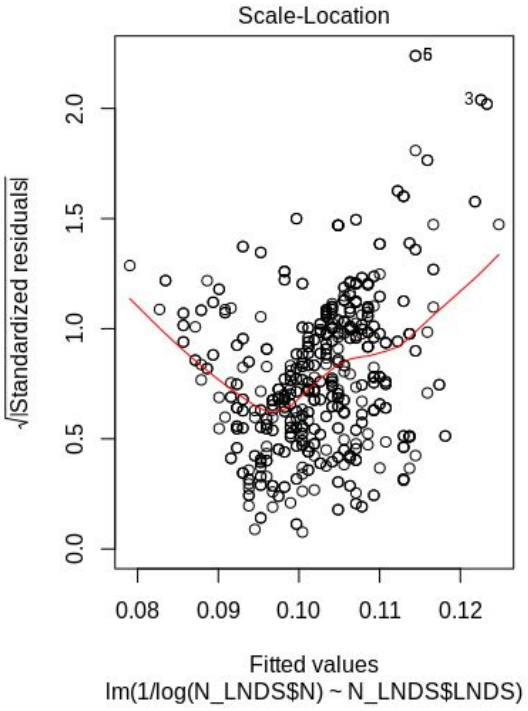


Figura 31: Scatter Plot de LNDS em função de N com regressão linear





Figuras 32 e 33: Normal Q-Q plot (Left) and Scale-Location plot (Right)

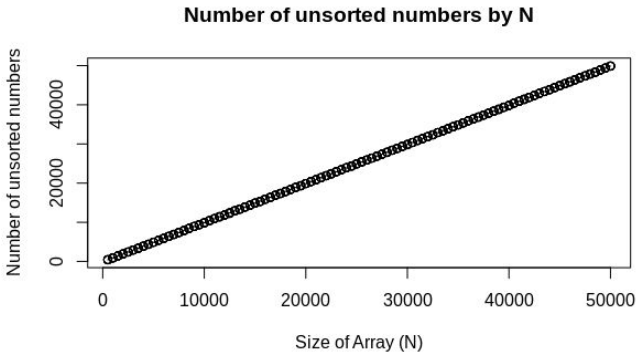


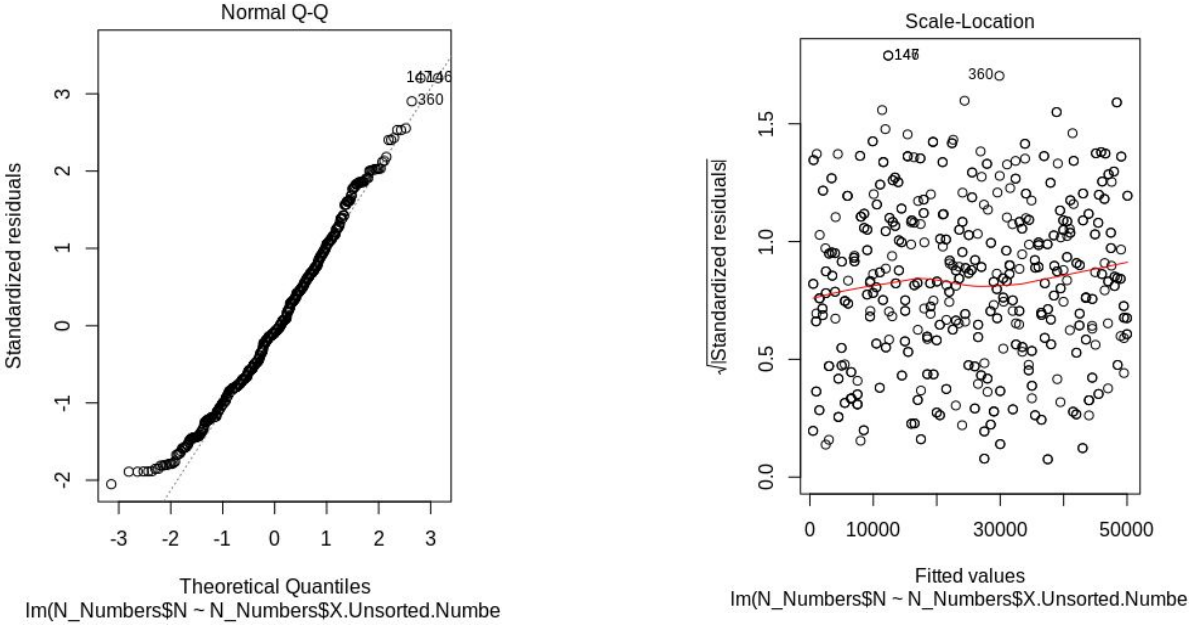
Figura 34: Scatter plot of the number of unsorted numbers by size of array.

```
Call:lm(formula = N_Numbers$N ~ N_Numbers$X.Unsorted.Numbers)

Residuals:
    Min      1Q  Median      3Q     Max
-94.002 -34.415  -3.891  29.885 146.673

Coefficients:
                             Estimate Std. Error t value Pr(>|t|)
(Intercept)                 1.065e+02  3.760e+00   28.33   <2e-16 ***
N_Numbers$X.Unsorted.Numbers 1.001e+00  1.298e-04 7709.75   <2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 45.86 on 598 degrees of freedom
Multiple R-squared:      1,   Adjusted R-squared:      1
F-statistic: 5.944e+07 on 1 and 598 DF,  p-value: < 2.2e-16
```



Figuras 35 e 36: Normal Q-Q plot (Left) and Scale-Location plot (Right)

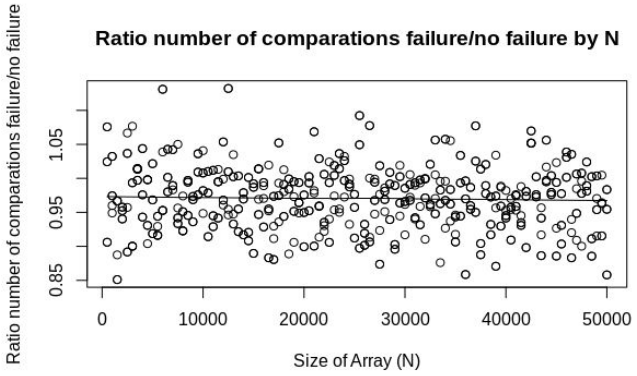


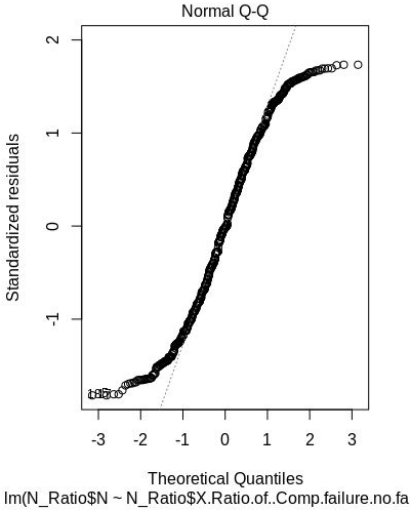
Figura 37: Scatter plot of the ratio by size of array

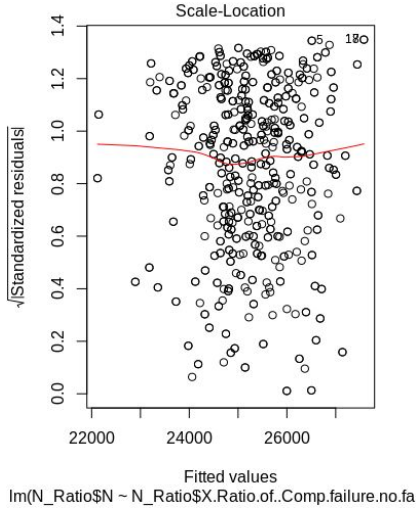
```
Call:lm(formula = N_Ratio$N ~ N_Ratio$X.Ratio.of..Comp.failure.no.failure)
Residuals:
     Min       1Q   Median       3Q      Max
-26078.1 -12300.4   -231.3  12574.2  24991.6

Coefficients:
                                          Estimate Std. Error t value Pr(>|t|)
(Intercept)                                  44102      12070   3.654 0.000281 ***
N_Ratio$X.Ratio.of..Comp.failure.no.failure  -19415      12415  -1.564 0.118386
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 14430 on 598 degrees of freedom
Multiple R-squared:  0.004073,Adjusted R-squared:  0.002407
F-statistic: 2.446 on 1 and 598 DF,  p-value: 0.1184
```

*As informações obtidas revelam não ser facilmente encontrar uma regressão linear aceitável para descrever o ratio entre o número de comparações com falhas / número de comparações sem falhas para o algoritmo **quicksort**. Isto acontece pela natureza do algoritmo na selecção do pivot.*





Figuras 38 e 39: Normal Q-Q plot (Left) and Scale-Location plot (Right)

Mergesort

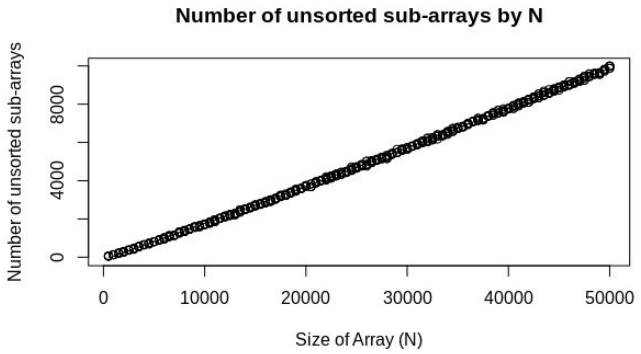


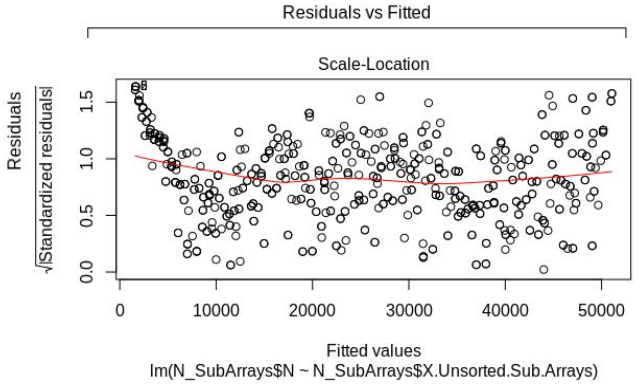
Figura 40: Scatter plot of the number of unsorted sub-arrays by size of array

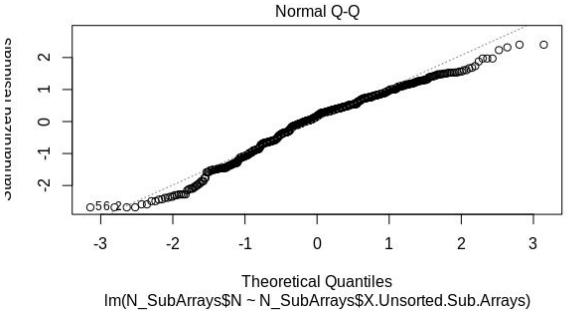
```
Call: lm(formula = N_SubArrays$N ~ N_SubArrays$X.Unsorted.Sub.Arrays)

Residuals:
     Min       1Q   Median       3Q      Max
-1145.13  -273.15    72.97   311.53  1027.36

Coefficients:
                                  Estimate Std. Error t value Pr(>|t|)
(Intercept)                     1.337e+03  3.385e+01   39.49   <2e-16 ***
N_SubArrays$X.Unsorted.Sub.Arrays 4.973e+00  6.028e-03  824.94   <2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 428.4 on 598 degrees of freedom
Multiple R-squared:  0.9991,  Adjusted R-squared:  0.9991
F-statistic: 6.805e+05 on 1 and 598 DF,  p-value: < 2.2e-16
```





Figuras 41 e 42: Scale-Location plot (Left) and Normal Q-Q plot (Right)

O gráfico de resíduos vs fitted não descarta a relação linear entre as duas variáveis.
O gráfico de normal Q-Q ajuda a comprovar distribuição normal dos dados O gráfico de Scale-Location permite aferir que existe uma variância semelhante entre resíduos.

Variação de Eps entre 0.25% e 8%, com incrementos de 0.25%

N fixado em: 20000

Quicksort

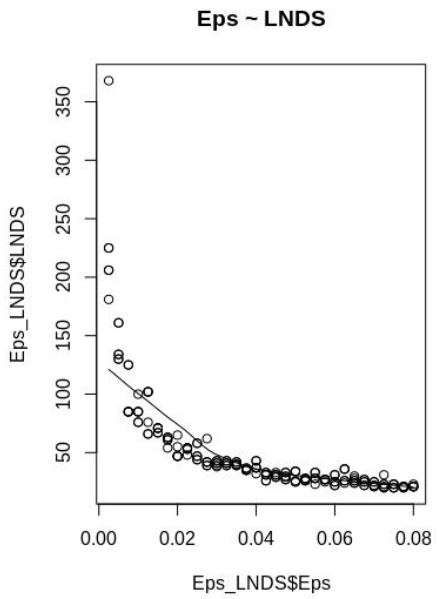


Figura 43: Scatter plot of the LNDS by Eps

Call: **lm(formula = 1/log(Eps_LNDS$Eps) ~ Eps_LNDS$LNDS)**

Residuals:

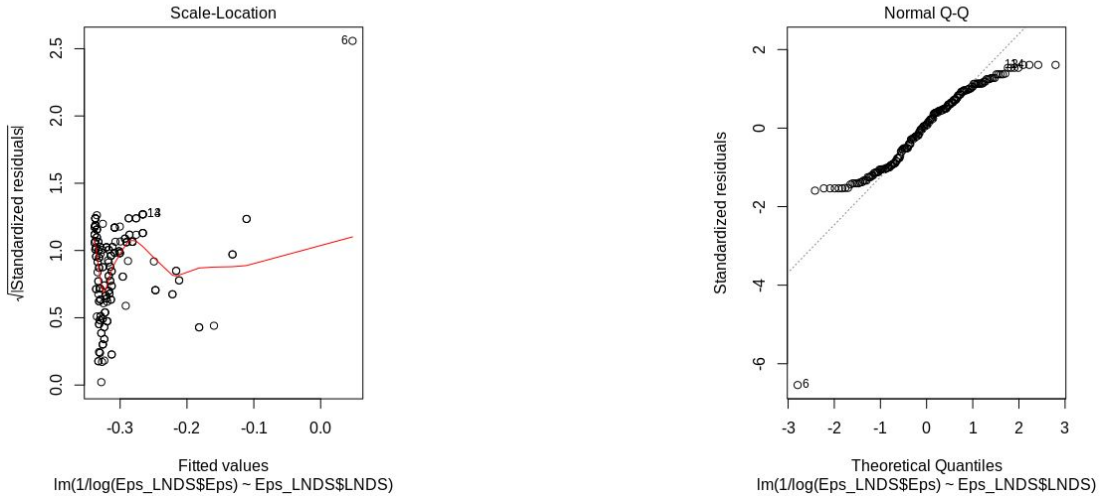
Min	1Q	Median	3Q	Max
-0.214636	-0.032688	0.003487	0.030239	0.061695

Coefficients:

	Estimate	Std. Error	t value	Pr(>|t|)	
(Intercept)	**-3.603e-01**	4.174e-03	-86.32	<2e-16	***
Eps_LNDS$LNDS	**1.109e-03**	6.272e-05	17.68	<2e-16	***

Signif. codes: 0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.0385 on 190 degrees of freedom
Multiple R-squared: 0.622, Adjusted R-squared: 0.62
F-statistic: 312.6 on 1 and 190 DF, p-value: < 2.2e-16



Figuras 44 e 45: Scale-Location plot (Left) and Normal Q-Q plot (Right)

Conclusão Final

Há algoritmos de ordenamento que são mais afetados por problemas de memória que outros. O algortimo ***Bubble sort*** é o algoritmo que apresenta melhores resultados em termos de eficácia no ordenamento à custa de um maior tempo de execução. O algoritmo ***Merge sort*** não é afetado por este problema pois a natureza do algoritmo, divisão da array, não sofre impacto pelo desordenamento que advém de más comparações.

No geral os algoritmos sofrem impacto no seu tempo de execução / número de comparações.