

**Algoritmos e Estruturas de Dados**

**Sorting Methods**

**Professores:**

Tomás Oliveira e Silva ([tos@ua.pt](mailto:tos@ua.pt))  
Pedro Lavrador ([plavrador@ua.pt](mailto:plavrador@ua.pt))

**Pedro** **Sobral**, 98491 – XX%

**André Freixo**, 98495 – XX%

**Marta Fradique**, 98626 – XX%

XX/01/2021

**Índice**

[**1 - Introdução** 3](#_Toc60173588)

[**2 - Introdução ao Problema** 4](#_Toc60173589)

[2.1 - Resumo 4](#_Toc60173590)

[2.2 – Compilação e Execução 4](#_Toc60173591)

[**3 - Resultados** 5](#_Toc60173592)

[3.1 – Bubble Sort 5](#_Toc60173593)

[3.2 – Shaker Sort 6](#_Toc60173594)

[3.3 – Insertion Sort 7](#_Toc60173595)

[3.4 – Shell Sort 8](#_Toc60173596)

[3.5 – Quick Sort 10](#_Toc60173597)

[3.6 – Merge Sort 12](#_Toc60173598)

[3.7 – Heap Sort 14](#_Toc60173599)

[3.8 – Rank Sort 15](#_Toc60173600)

[3.9 – Selection Sort 16](#_Toc60173601)

[3.10 – Resultados Totais 17](#_Toc60173602)

[**Conclusão** 17](#_Toc60173603)

[**Bibliografia** 18](#_Toc60173604)

# **1 - Introdução**

No âmbito da unidade curricular de AED, foi-nos apresentada a realização deste trabalho prático, sendo este relatório o resultado do problema “Sorting Methods”. Todo o código fonte e informações deste trabalho prático podem ser encontradas neste [repositório do GitHub](https://github.com/TheScorpoi/AED_Trabalho02)1 (mais informações, ler README.md do repositório).

O trabalho prático, consiste essencialmente em estudar os tempos de execução de uma série de rotinas de ordenação. Os algoritmos de ordenação foram implementados em C, sendo os mesmos fornecidos pelos docentes da unidade curricular.

A linguagem de programação C, é uma linguagem muito poderosa, pois dá ao programador um controlo íntegro de todo o processo programado, sendo uma linguagem onde o programador tem de lidar com todos os pormenores, torna-se consideravelmente eficiente e otimizada. Desta forma, conseguiremos execuções mais eficientes, pois toda a implementação é feita em C, como já referido anteriormente.

Com a realização deste trabalho prático, esperamos veemente alargar os nossos conhecimentos em C, e principalmente em conhecimento sobre algoritmos de ordenação. Esperamos também conseguir concluir com êxito todos os objetivos que são propostos no início (em comentário) do programa sorting\_methods.c.

1Por motivos de privacidade o repositório encontra-se privado, para visualização é favor entrar em contacto com os autores do trabalho prático.

# **2 - Introdução ao Problema**

## 2.1 - Resumo

….

A very short introduction describing how you got your results

## 2.2 – Compilação e Execução

Para compilar o programa é necessário à partida ter um compilador de C instalado na máquina, por exemplo o *gcc*.

Posto isto, para compilar usamos o *makefile*:

make sorting\_methods

O programa está capacitado com um teste, para executar o teste basta passar a seguinte linha de código:

./sorting\_methods -test

Para correr o programa, e guardar os resultados dos tempos de execução, num ficheiro .txt, executamos a seguinte linha:

./sorting\_methods -measure | tee output.txt

# **3 - Resultados**

Como já referido anteriormente, os algoritmos e toda a implementação foi fornecida pelos docentes da unidade curricular, sendo então aqui no ponto 3 do relatório, que iremos expor os nossos resultados para cada rotina de ordenação.

## 3.1 – Bubble Sort

## 3.2 – Shaker Sort

## 3.3 – Insertion Sort

## 3.4 – Shell Sort

A estratégia de ordenação, *Shell Sort*, é de forma genérica uma sucessiva aplicação da estratégia de ordenação, *Insertion Sort*, onde são criados sub-arrays do array original. A ideia do *Shell Sort*, é que seja possível a troca de itens distantes , criamos o array-h para um grande valor de h, e vamos reduzindo o valor de h, até que este se torne 1

A nível da complexidade computacional desta rotina de ordenação, é um pouco manhosa, pois ainda não é conhecida para nenhum dos 3 casos (best, worst e average), depende muito da sequência de *strides* que são usadas. São conhecidos alguns valores para h, de modo a que o algoritmo seja **o(n2),** por exemplo, quando h = 9 \* 2s - 9 \* 2s/2 + 1 , para *s* par.

A tabela com os tempos de execução obtida foi a seguinte(Fig.XX):

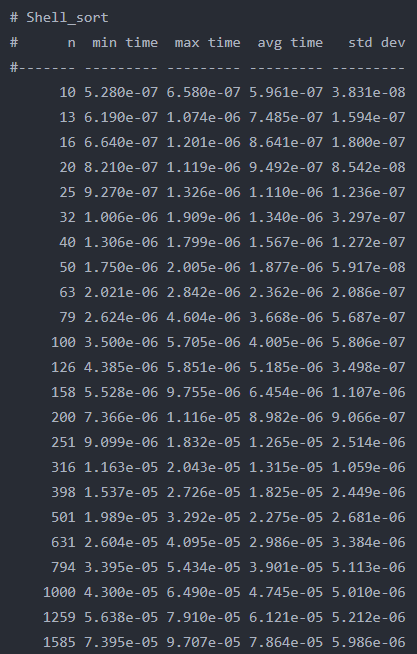
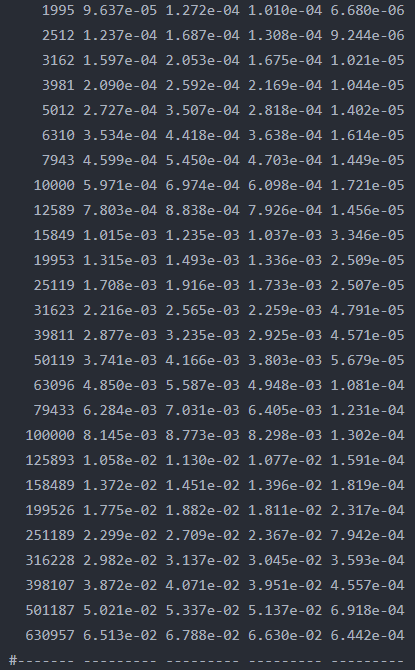


Figura - Tabela Shell Sort

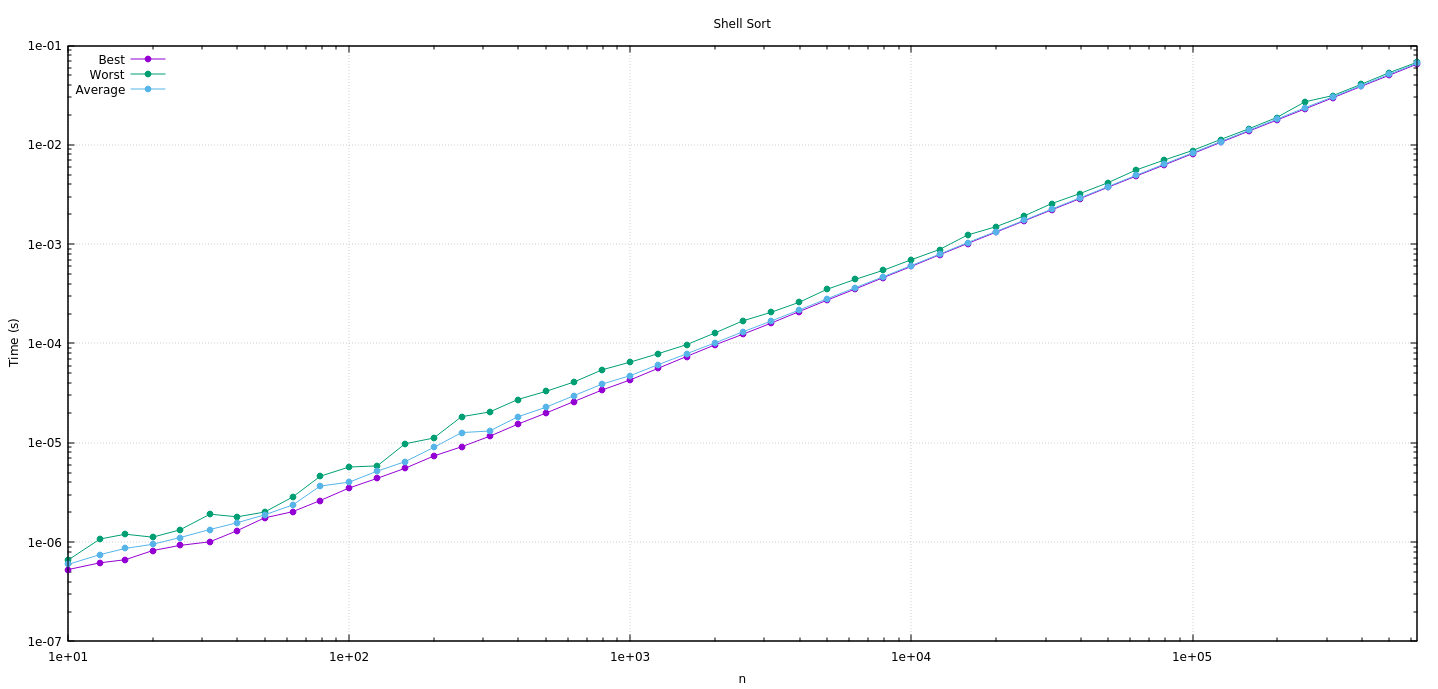
O gráfico obtido para esta estratégia de ordenação foi o seguinte(Fig.XX):

Figura - Gráfico Shell Sort

Através de uma análise cuidada do gráfico, consegue-se perceber que para valores de *n* menores que 104 , nota-se alguma diferença, a nível de tempos de execução entre o *Best* e o *Worst*, porém desses valores para cima, as linhas que o gráfico traça, são muito próximas umas das outas, estando em algumas partes até sobrepostas.

## 3.5 – Quick Sort

A rotina de ordenação Quick Sort, é uma rotina que como o próprio nome indica é rápida, é implementada através de um método recursivo, que usa um algoritmo *Divide and Conquer*.

O algoritmo para ordenar arrays mais pequenos (menores que 20 a 30, discutível) usa a rotina *Insertion Sort*, pois torna-se mais eficiente, para arrays com tamanho maior, o algoritmo vai escolher um pivot, pivot esse que é escolhido aleatoriamente, neste contexto, faz-se uma passagem pelo array, e para a esquerda do pivot, colocamos os valores mais pequenos que o array, e para a direita os valores maiores, sendo a parte dos valores igual ao pivot discutível de que lado do pivot é que ficam. De seguida, é feita a mesma estratégia para cada sub-array, um da parte esquerda do pivot, outro da parte direita. O Quick Sort, tem uma complexidade computacional para o *Best e para o Average de* ***O(n log n)*,** porém para *Worst* ***O(n2)*.**

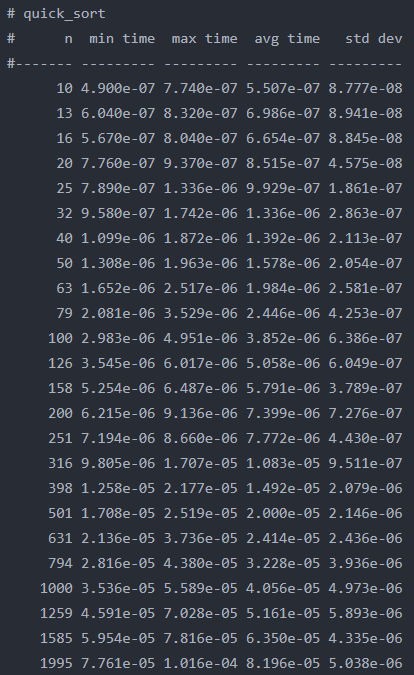
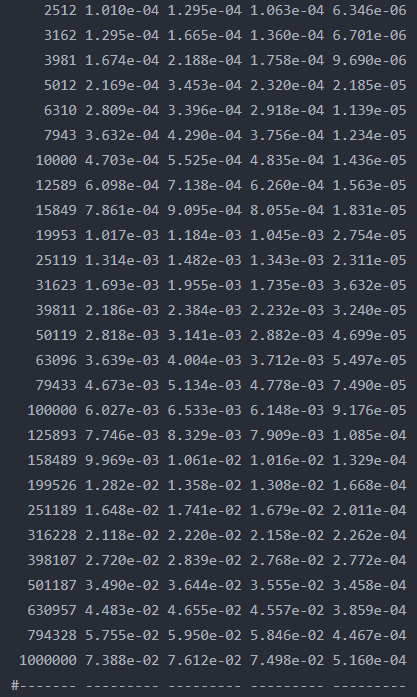
A tabela com os tempos de execução é a seguinte(Fig.XX):

Figura - Tabela Quick Sort

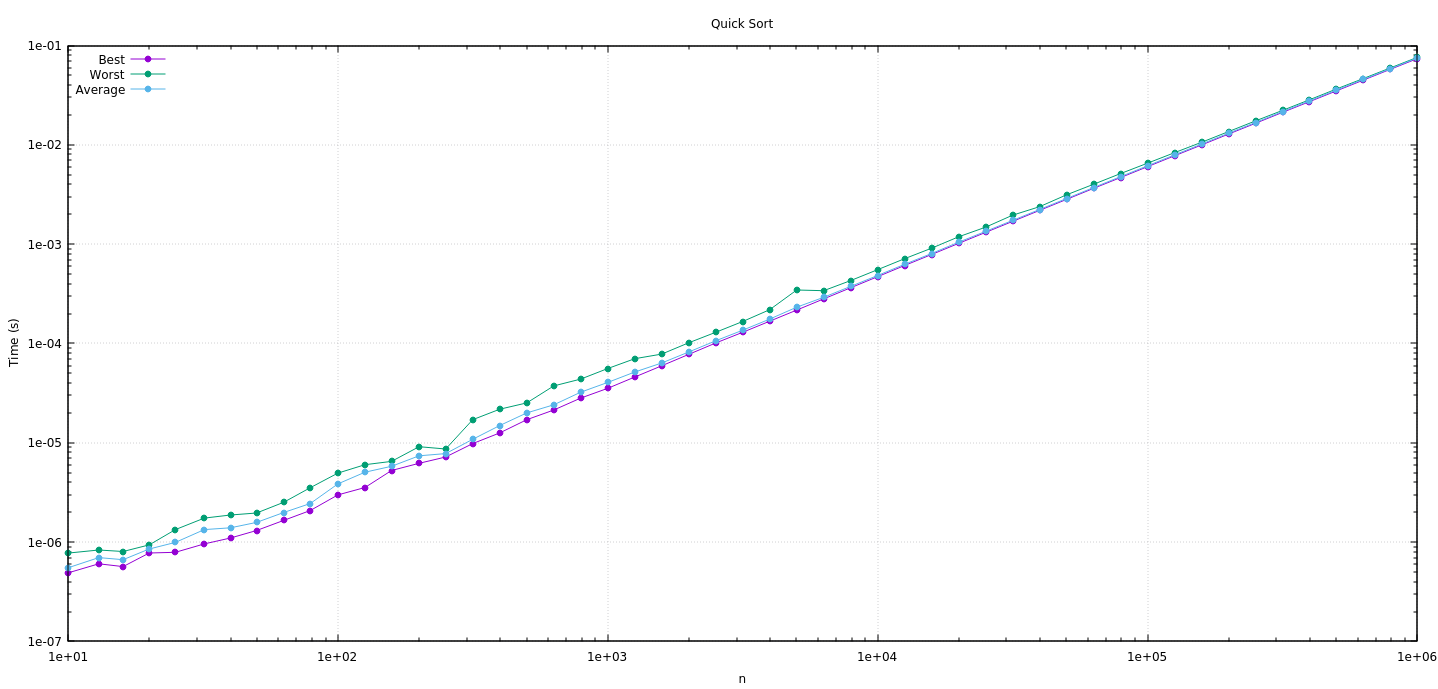
O gráfico com os tempos de execução para o Quick Sort, é o seguinte:

Figura - Gráfico Quick Sort

De acordo com o gráfico (Fig.XX), podemos concluir que a partir de um certo valor de *n*, sensivelmente, 104 , os tempos de execução para *Best, Average, e Worst* ficam bastante semelhantes …………………….

## 3.6 – Merge Sort

O Merge Sort é um algoritmo de ordenação, que usa uma estratégia de *Divide and Conquer,* é implementado de forma recursiva, e em traços gerais pode ser descrita como: Primeiramente divide-se o array em 2 partes iguais (ou, parcialmente igual, se o tamanho do mesmo for um número ímpar), divide-se recursivamente até que fiquemos com cada elemento sozinho, e a partir dessemomento, entra a parte de *Conquer*, onde se liga as partes dos sub-arrays ordenados. Esta rotina de ordenação tem uma complexidade computacional igual para os três casos, *Best, Avevage, e Worst*, **O(n log 10)**.

A tabela com os tempos de execução da rotina Merge Sort, é a seguinte (Fig.XX):

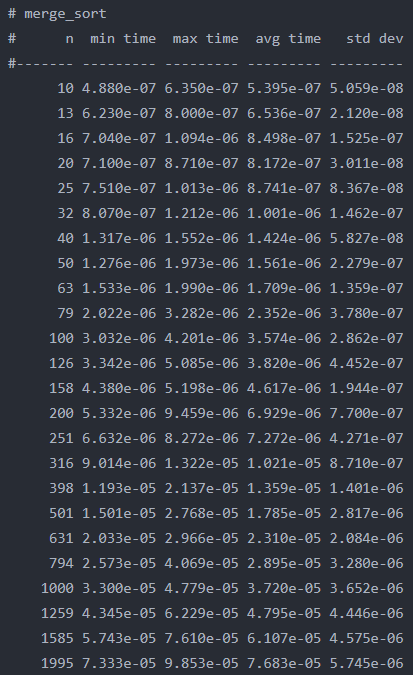
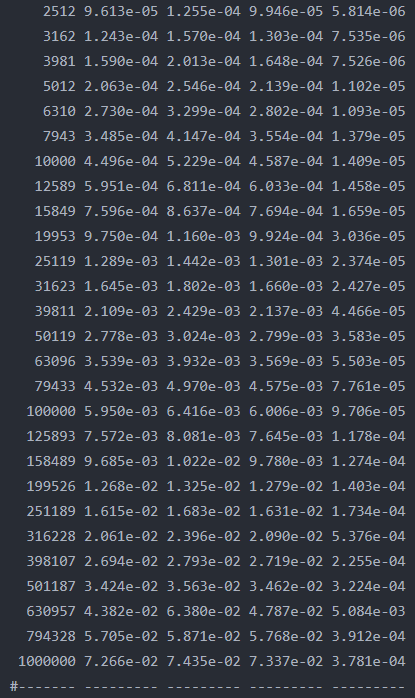


Figura - Tabela Merge Sort

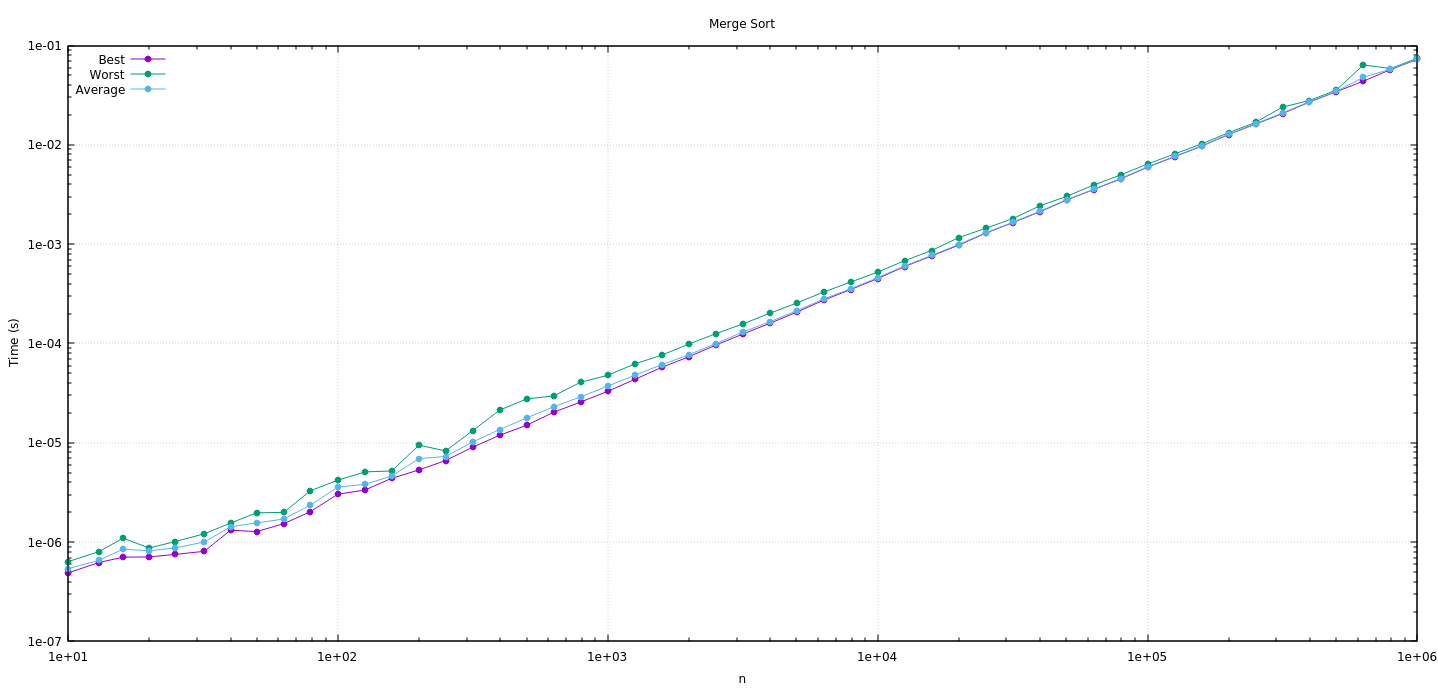
O gráfico com os tempos de execução da rotina Merge Sort, é a seguinte (Fig.XX):

Figura - Gráfico Merge Sort

De acordo com o gráfico da (Fig.XX)

## 3.7 – Heap Sort

## 3.8 – Rank Sort

## 3.9 – Selection Sort

## 3.10 – Resultados Totais

# **Conclusão**

# **Bibliografia**

Silva, Tomás Oliveira e. Lecture notes: Algorithms and Data Structute (AED - Algoritmos e Estruturas de Dados), LEI, MIEC, 2020/2021