

# Projeto de análise experimental para comparação entre algoritmos de ordenação

Marcos Renan Krul

<sup>1</sup>Universidade Estadual de Ponta Grossa (UEPG)

**Resumo.** *As técnicas de ordenação estão presentes em diversas aplicações e se caracterizam como um problema computacional muito bom para estudo da análise e projeto de algoritmos. O objetivo é realizar uma análise experimental entre três algoritmos de ordenação, a fim de definir em quais situações cada um deles possui uma vantagem. Os resultados validam o comportamento quadrático do Bubble sort em médio e pior caso, além da vantagem do quick sort em relação à ordenação digital para números com mais dígitos.*

## 1. Introdução

A ordenação de um conjunto de dados consiste, basicamente, no processo de rearranjo dos elementos, reorganizando-os em uma ordem ascendente ou descendente. Dessa forma, pode-se ordenar uma lista de valores numéricos de forma crescente, ou ainda uma lista de nomes em ordem lexicográfica (ordem alfabética). De forma prática, essa atividade está presente na maioria das aplicações onde os dados precisam ser pesquisados e recuperados, como dicionários, índices de livros, agendas telefônicas ou tabelas de arquivos. Portanto, um dos objetivos da ordenação é facilitar as futuras operações de pesquisa e recuperação na base de dados. [Ziviani 1999]

Os algoritmos de ordenação representam um bom exemplo de como fazer a análise e o projeto de algoritmos para resolver problemas computacionais. A partir das técnicas de ordenação, são estabelecidos diversos algoritmos que desempenham a mesma tarefa, porém, cada um possui seus pontos fortes e fracos, dependendo do contexto que estão inseridos. Dessa forma, se mostra importante a realização de análises teóricas e experimentais dos diversos algoritmos, para que, ao final da análise, seja possível determinar os melhores algoritmos para cada situação. [Ziviani 1999]

## 2. Desenvolvimento

O presente trabalho tem por objetivo desenvolver o projeto de uma análise experimental para comparar três algoritmos de ordenação, sendo eles o *Quick Sort*, *Bubble Sort* e ordenação por dígitos.

A realização dos testes para comparação entre os métodos requer uma série de etapas, sendo elas a determinação dos parâmetros, como a quantidade de dígitos dos números e o intervalo que representa a quantidade de elementos, a obtenção dos resultados e a análise dos mesmos, com plotagem de gráficos e cálculos estatísticos. É interessante, além dessas, realizar um mesmo teste mais de uma vez, visando resultados melhores, descartando (em parte) as interferências externas, como a sobrecarga de recursos do *hardware* utilizado. Diante disso, é válida a criação de uma sequência de teste automatizada.

## 2.1. Complexidade de algoritmos

O tempo de execução de um algoritmo pode ser expresso por uma função de complexidade  $f(n)$ , onde  $n$  representa o tamanho da entrada. Assim, o valor de  $n$  é um parâmetro que utilizamos para supor o tempo de execução de um programa: a medida que ele cresce, o tempo necessário aumenta. O importante nesta análise é saber qual a taxa de crescimento do tempo, e para isso utilizamos as classes de comportamento assintótico. [Ziviani 1999].

Um algoritmo linear, que possui complexidade de  $O(n)$ , possui um tempo de execução melhor em relação a um algoritmo quadrático ( $O(n^2)$ ), mesmo que para valores pequenos de  $n$  o segundo algoritmo teria uma vantagem. Essa análise está baseada na matemática assintótica, focando apenas em valores enormes de  $n$ .

A complexidade dos algoritmos *Quick Sort* e ordenação digital já foi estudada e definida, como mostra a Tabela 1. [Cormen 2014]. O valor de  $d$  representa a quantidade de dígitos ou de caracteres.

**Tabela 1. Complexidade de tempo dos algoritmos em caso médio**

Algoritmo	Complexidade de tempo
Quick sort	$O(n \cdot \log(n))$
Bubble sort	$O(n^2)$
Ordenação digital	$O(d \cdot n)$

## 2.2. Bubble sort

O algoritmo *Bubble Sort* utiliza uma técnica bem simples para realizar a ordenação dos elementos. Basicamente, cada varredura na base de dados irá comparar dois elementos adjacentes e, caso estejam na ordem indesejada, serão trocados de posição. A comparação e a decisão de troca dos elementos irá depender da estrutura de dados utilizada e da regra de negócio em específico. Assim, um conjunto simples de números pode ser ordenado de forma ascendente ou decendente, ou um conjunto de palavras ordenado de acordo com a ordem alfabética. [de Souza ]

A estratégia adotada pelo algoritmo é de percorrer toda a base de dados para comparar os elementos adjacentes e trocá-los de posição caso necessário. Ao analisar o maior elemento do conjunto, este será trocado de posição em todas as comparações, fazendo com que ao final da primeira iteração assuma sua posição correta. O menor elemento, por sua vez, será trocado de posição uma vez em cada iteração. Dessa forma, pode-se entender o motivo do nome *Bubble sort* (ordenação por bolha/flutuação), onde o menor elemento (o mais "leve") irá "subir" aos poucos como uma bolha.

O *Bubble sort* é um algoritmo estável e possui uma implementação muito simples. Porém, do ponto de vista de desempenho, deve ser utilizado apenas quando a base de dados é relativamente pequena ou quando se sabe que os dados estão parcialmente ordenados (melhor caso). Para uma grande quantidade de elementos, esse algoritmo é considerado lento, visto que sua complexidade de tempo de execução, para caso médio e pior caso, é quadrática ( $O(n^2)$ ). [de Souza ]

## 2.3. Tecnologias

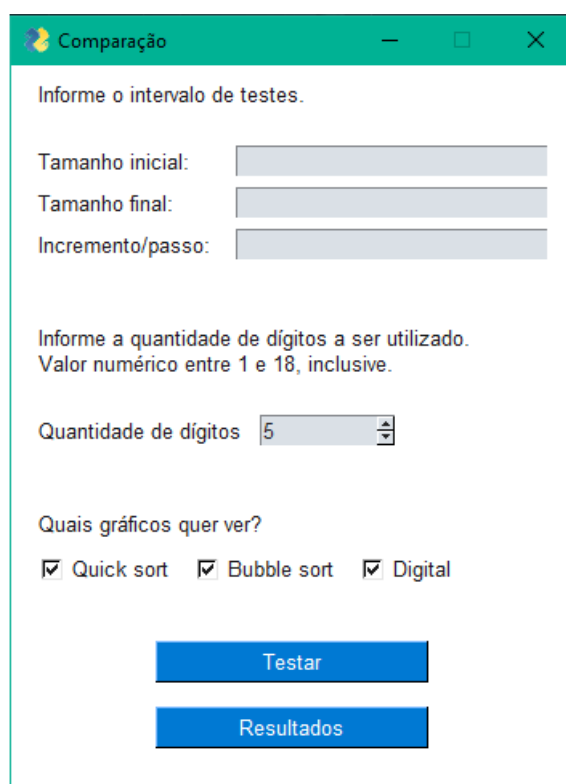
Os algoritmos de ordenação foram implementados utilizando a linguagem C. Apenas por questão de organização, as funções foram separadas em arquivos distintos. Existe, ainda,

uma função desenvolvida para realizar o preenchimento dos vetores (com valores pseudoaleatórios) e as chamadas dos algoritmos de ordenação.

A automatização dos testes foi implementada com *scripts* em linguagem Python. Para a execução da função em C, utilizou-se o módulo *ctypes*, que possibilita a utilização de funções em DLLs (*Dynamic Linked Libraries*, bibliotecas de linkedição dinâmica) compartilhadas. Para a plotagem dos gráficos, utilizou-se o módulo *matplotlib* em conjunto com o módulo *pandas*, para o tratamento dos dados a serem impressos no gráfico.

O *script* principal renderiza uma pequena interface gráfica bem simples, como mostra a [Figura 1](#), desenvolvida utilizando o módulo *PySimpleGUI*.

**Figura 1. Interface gráfica principal**



Após as escolhas dos valores, clicando no botão *Testar*, o algoritmo começa a executar as devidas ordenações. Os valores obtidos são armazenados em arquivo com extensão csv (*comma-separated-values*, valores separados por vírgula), e uma nova janela será aberta com o gráfico gerado a partir dos valores. Caso um novo teste seja realizado, os novos valores serão armazenados no arquivo, e o gráfico será gerado a partir da média dos mesmos. Caso já existam valores de testes, é possível clicar no botão *Resultados* para visualizar o gráfico.

## 2.4. Projeto da análise experimental

A análise dos algoritmos de ordenação faz muito sentido diante do fato que existem várias opções. A escolha deve ser estudada levando em consideração aspectos como tempo de execução (algoritmo mais rápido) e espaço utilizado (algoritmo que utiliza menos recursos, como memória). Assim, toda a classes desses algoritmos é investigada a fim de se encontrar a melhor escolha possível. [Ziviani 1999]

Existem algumas formas de se determinar o custo da execução do algoritmo. A análise experimental, em particular, utilizada a execução do programa em um computador real, podendo medir o tempo de execução diretamente. Porém, esses resultados podem variar de acordo com fatores externos, como o poder computacional da máquina utilizada e sua sobrecarga, assim como a escolha do compilador. Dessa forma, os resultados de análises experimentais, em parte, não devem ser generalizados. [Ziviani 1999]

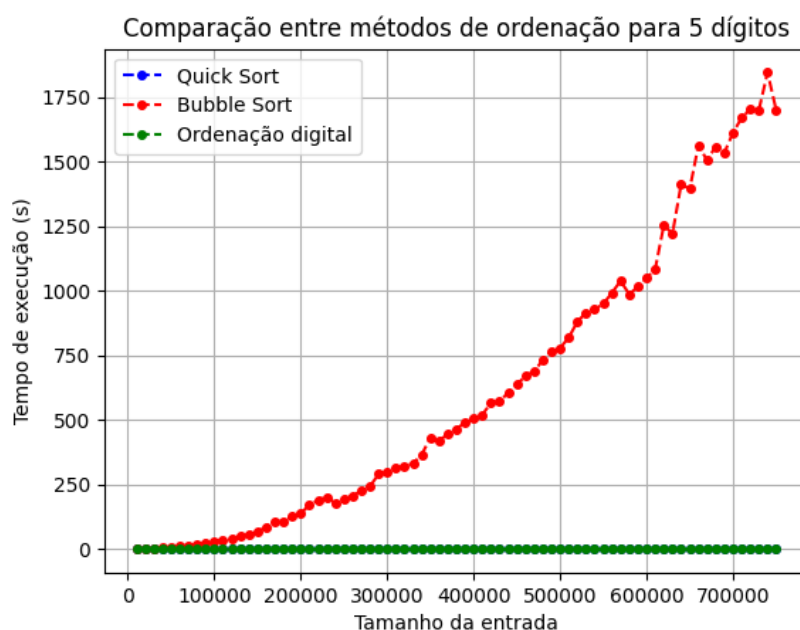
O valor do custo de execução de um algoritmo depende, principalmente, do tamanho da instância dos dados de entrada. Na caso dos algoritmos de ordenação, a distribuição dos dados também afeta o desempenho. A ordenação de um vetor parcialmente ordenado tende a ser menos custosa do ponto de vista computacional, cenário o qual denominamos de **melhor caso**. [Ziviani 1999]

Para essa análise, foram escolhidos valores altos para o tamanho da entrada, variando, para todos, a quantidade de dígitos dos números. Isso se deve ao fato de que, atualmente, possuímos um grande poder computacional, conseguindo realizar as operações de forma rápida, fazendo com que instâncias pequenas gerem resultados pouco significativos.

### 3. Resultados

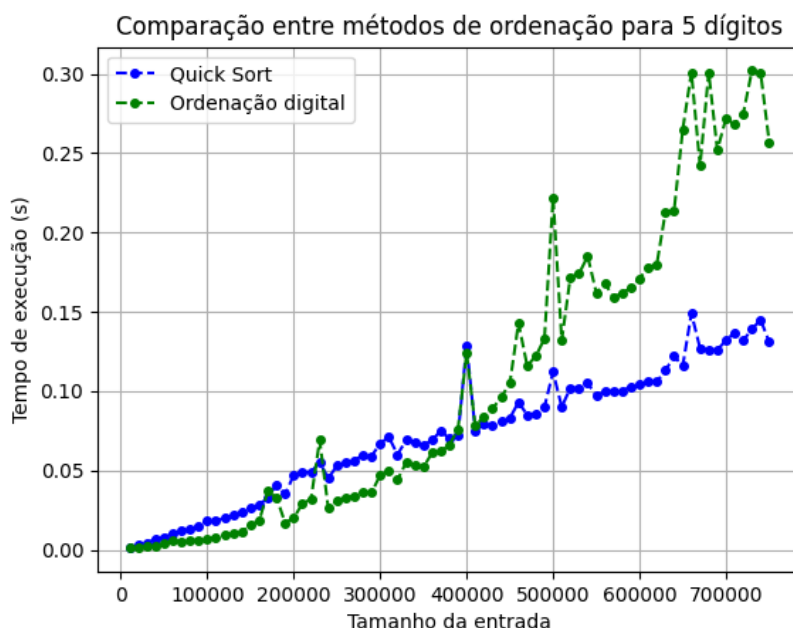
A primeira decisão tomada seria a de executar três vezes o mesmo teste para 5, 10 e 15 dígitos, onde o tamanho da entrada iria varia de 10.000 até 1.000.000, com passo de 10.000, totalizando 100 ordenações por teste para cada algoritmo. Todavia, esse modelo foi desconsiderado devido ao alto tempo de execução, principalmente do algoritmo *Bubble sort*. O teste foi abortado na septuagésima quinta (75ª) ordenação, momento no qual o *bubble sort* demorou, aproximadamente, 30 minutos para ser finalizado. O resultado final pode ser observado na Figura 2.

Figura 2. Resultado de 75 ordenações para 5 dígitos



Como pode ser observado pelo gráfico da [Figura 2](#), a plotagem dos resultados do *bubble sort* é muito superior em relação aos outros algoritmos, dificultando, inclusive, a comparação entre eles. Porém, já é possível validar o comportamento quadrático do mesmo. A [Figura 3](#) mostra os valores apenas do *quick sort* e do algoritmo de ordenação digital. Como foi realizado apenas um teste, pode-se observar que existem momentos que o tempo de execução aumenta em uma taxa muito alta. Isso pode ter ocorrido devido a sobrecarga de uso da máquina no momento do teste, entre outros fatores.

**Figura 3. Resultado de 75 ordenações para 5 dígitos (exceto *bubble sort*)**

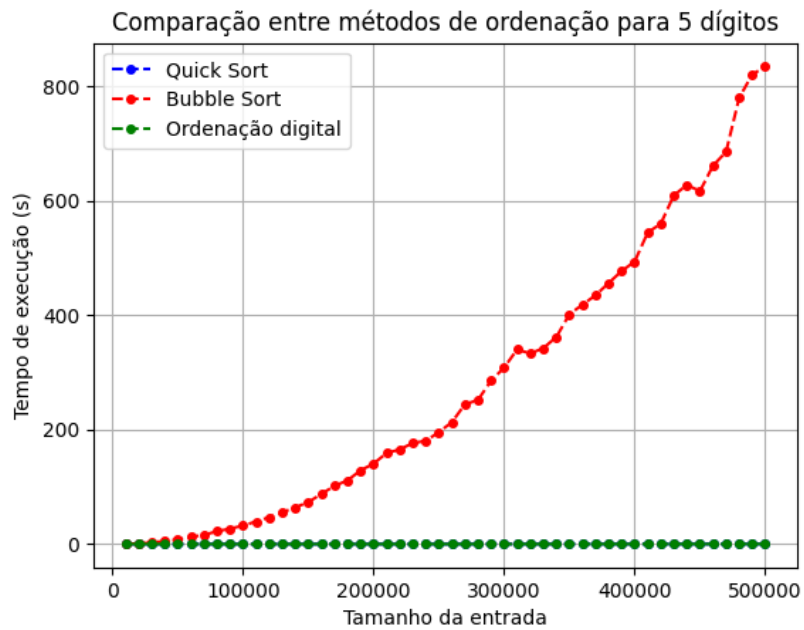


Diante do problema do tempo de execução, a estratégia adotada foi realizar os testes para tamanhos de entrada variando entre 10.000 e 500.000, com passo de 10.000. Os três algoritmos foram testados três vezes. Porém, para obter resultados melhores, outros sete testes foram realizados apenas com o *quick sort* e o algoritmo de ordenação digital.

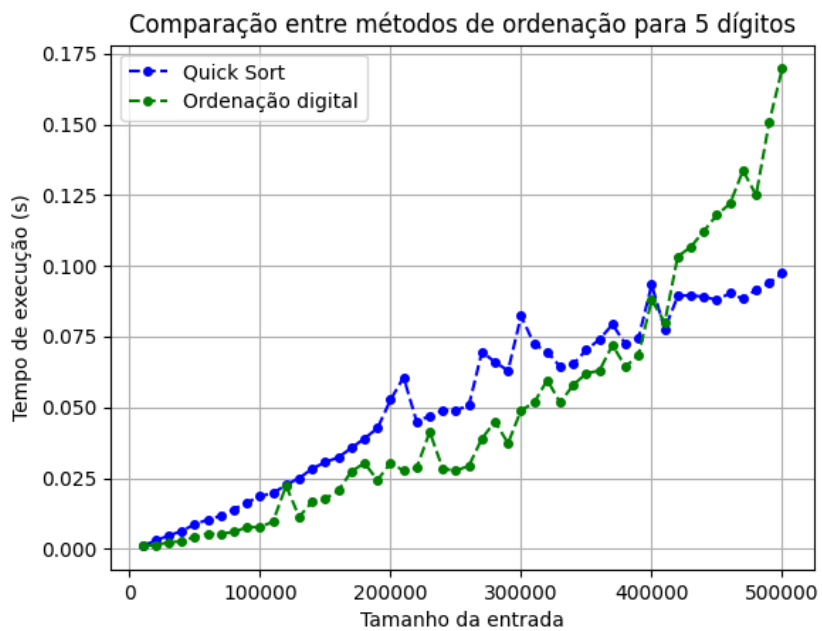
A figura [Figura 4](#) mostra os resultados obtidos para um total de 150 ordenações realizadas para 5 dígitos. A figura [Figura 4a](#) possui a plotagem dos resultados dos três algoritmos, porém, o *bubble sort* possui valores muito elevados, dificultando a comparação entre os algoritmos. Como exemplo, a última ordenação (para 500.000 elementos) durou aproximadamente 14 minutos. Para melhor visualização, a [Figura 4b](#) mostra as plotagens para *quicksort* e para o algoritmo de ordenação digital.

A [Figura 5](#) mostra uma melhor aproximação do comportamento dos algoritmos em relação ao tempo de execução, devido a realização de mais testes e obtendo os resultados a partir da média. Como dito anteriormente, essa estratégia foi adotada com a intenção de reduzir as interferências de outros fatores, como por exemplo a sobrecarga de uso da máquina. Pode-se observar que a ordenação digital possui uma leve vantagem em relação ao *quick sort* para até 400.000 elementos, quando o algoritmo passa a ter taxa de crescimento e tempo de execução maiores.

Figura 4. Três testes para 5 dígitos

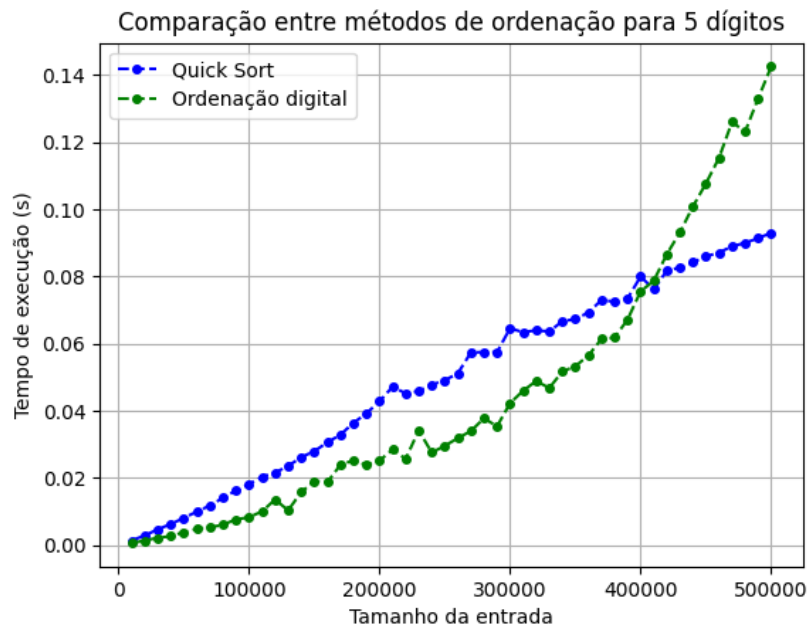


(a) com *bubble sort*



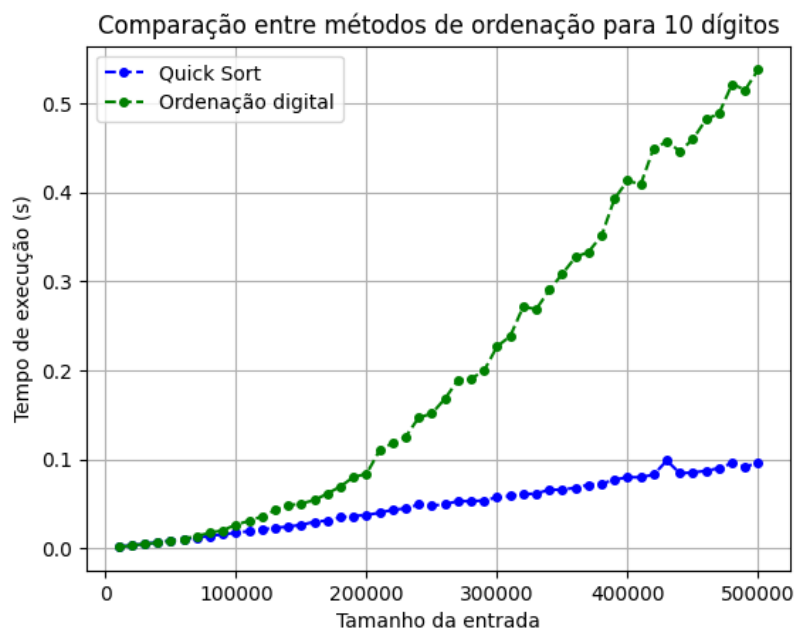
(b) sem *bubble sort*

**Figura 5. Dez testes para 5 dígitos**



A figura [Figura 7](#) mostra os resultados obtidos para as 150 ordenações realizadas para 10 dígitos (uma melhor aproximação está na [Figura 6](#)).

**Figura 6. Dez testes para 10 dígitos**



A partir desse momento, pode-se observar uma maior diferença entre os tempos de execução para os algoritmos *quick sort* e ordenação digital. Em geral, o *quick sort* não possui uma grande dependência com a quantidade de dígitos dos números (ou a quanti-

dade de caracteres das palavras), onde sua complexidade de tempo de execução, em caso médio, é dada por  $O(n \cdot \log(n))$ . A ordenação digital, por sua vez, apresenta uma maior dependência da quantidade de dígitos  $d$ , onde sua complexidade é dada por  $O(d \cdot n)$ .

A figura [Figura 8](#) mostra os resultados obtidos para as 150 ordenações realizadas para 15 dígitos (uma melhor aproximação está na [Figura 9](#)). Ao utilizarmos números com 15 dígitos é ainda mais fácil perceber a diferença de desempenho entre o *quick sort* e o algoritmo de ordenação digital. Para valores pequenos do tamanho da entrada  $n$  os algoritmos possuem um desempenho similar. Porém, a partir de 200.000 elementos, aproximadamente, o tempo de execução da ordenação digital cresce em uma taxa mais alta.

Os resultado obtido até aqui já revelam que o desempenho do algoritmo de ordenação digital depende da quantidade de dígitos dos elementos. Porém, apenas por validação, foram realizados mais três testes para 2 dígitos, onde o tamanho do conjunto varia de 10.000 até 1.000.000, com passo de 10.000. O resultado obtido está representado na [Figura 10](#). Como a quantidade de dígitos é baixa, pode-se observar um melhor desempenho do algoritmo de ordenação por dígitos, superando levemente o *quick sort*.

#### 4. Conclusão

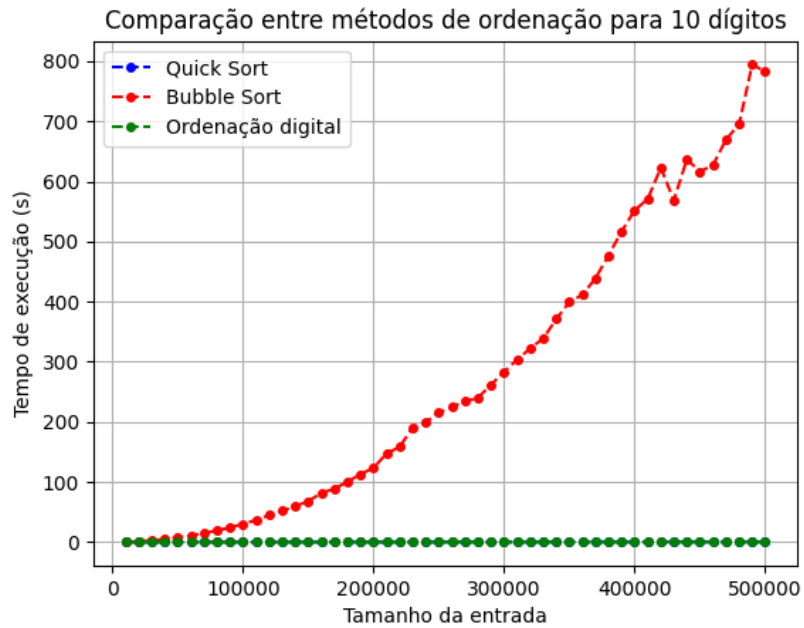
A análise e projeto de algoritmos se mostra muito importante para o desenvolvimento de sistemas ainda mais eficientes. Assim, devem ser realizadas as análises teóricas e experimentais. A análise experimental, em particular, utiliza os tempos coletados a partir da execução do algoritmo em um computador real. Porém, como existe a dependência do poder computacional, da sobrecarga de recursos no momento do teste e do compilador escolhido, os resultados não devem ser generalizados para todos os *hardwares*.

O *Bubble sort* é um algoritmo de ordenação estável que possui uma implementação muito simples. Os testes realizados validam o comportamento quadrático desse algoritmo em caso médio. Um algoritmo que possui complexidade de tempo de execução quadrática é considerado lento, visto que para uma base de dados grande o seu tempo de execução será muito elevado. O alto tempo de execução do *bubble sort* foi validado pelos vários testes realizados nesta análise, onde para 750.000 elementos, por exemplo, o algoritmo levou aproximadamente 30 minutos para finalizar.

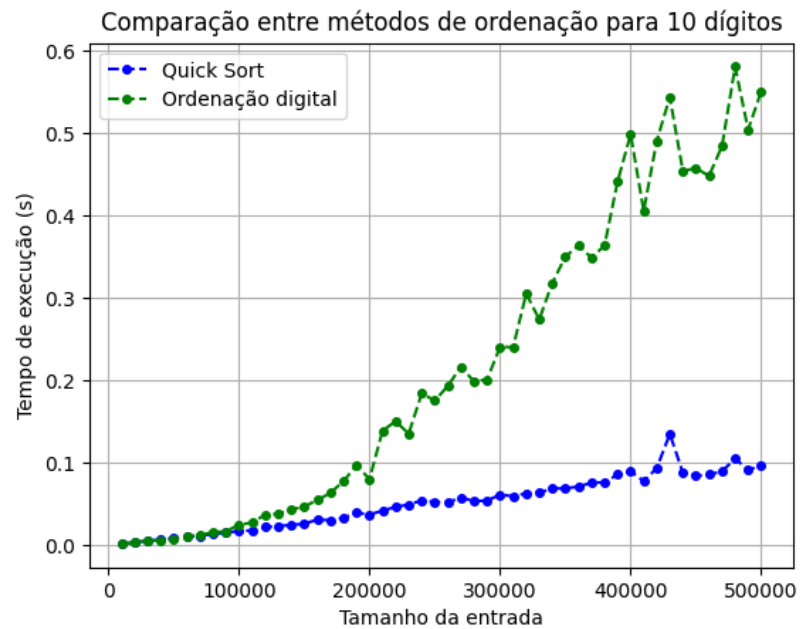
Os algoritmos *Quick sort* e ordenação digital possuem um melhor desempenho para o tempo de execução. O segundo algoritmo, em particular, possui uma complexidade dependente da quantidade de dígitos (ou caracteres) dos elementos da base de dados. Os testes realizados validam essa dependência, onde um conjunto de dados grande que possui números com bastante dígitos eleva o tempo de execução desse algoritmo. O *Quick sort*, por sua vez, não possui uma dependência direta à quantidade de dígitos. O desempenho desse algoritmo depende, principalmente, do tamanho do conjunto a ser ordenado e de uma boa implementação para que a escolha do pivô seja a melhor possível. Os testes validam o comportamento sólido do *quick sort* para diferentes quantidade de dígitos e tamanhos de entrada. Esse algoritmo, portanto, é uma das melhores opções de ordenação para a maioria dos casos, determinando, inclusive, o limite inferior de tempo de execução para algoritmos de ordenação por comparação entre chaves, ou seja, para casos médios, esse algoritmo possui o menor tempo de execução possível.



Figura 7. Três testes para 10 dígitos

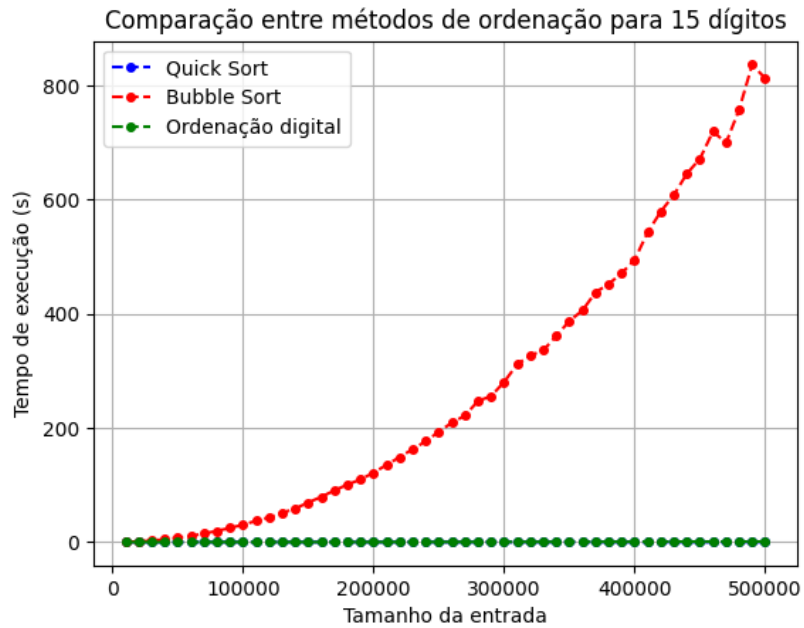


(a) com *bubble sort*

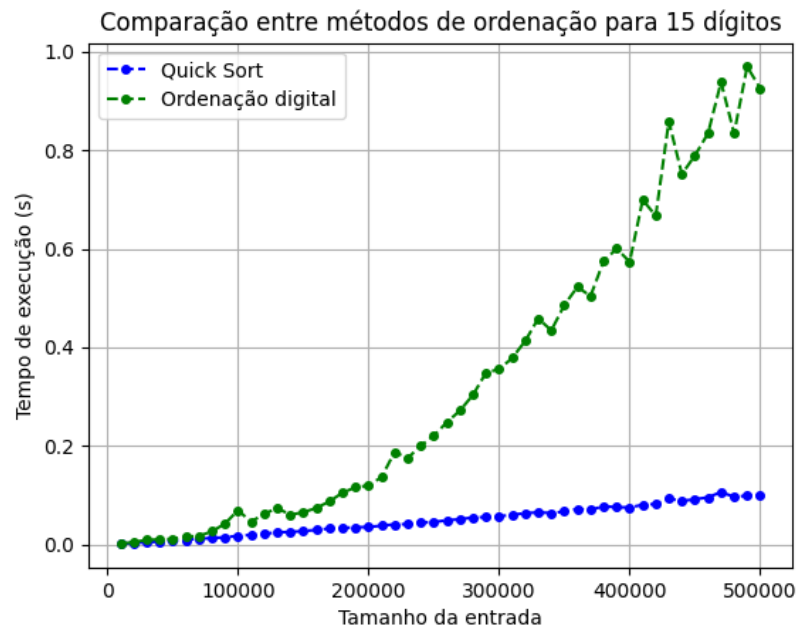


(b) sem *bubble sort*

**Figura 8. Três testes para 15 dígitos**

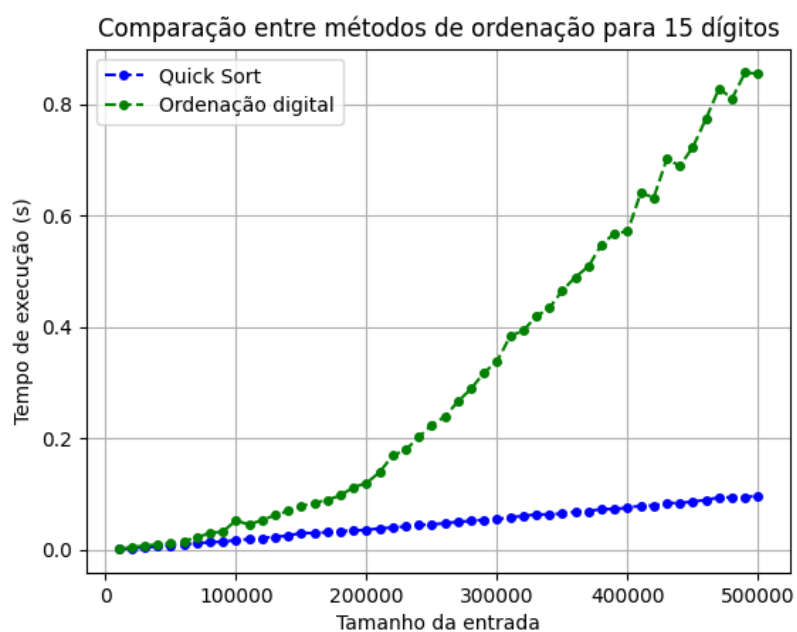


**(a) com *bubble sort***

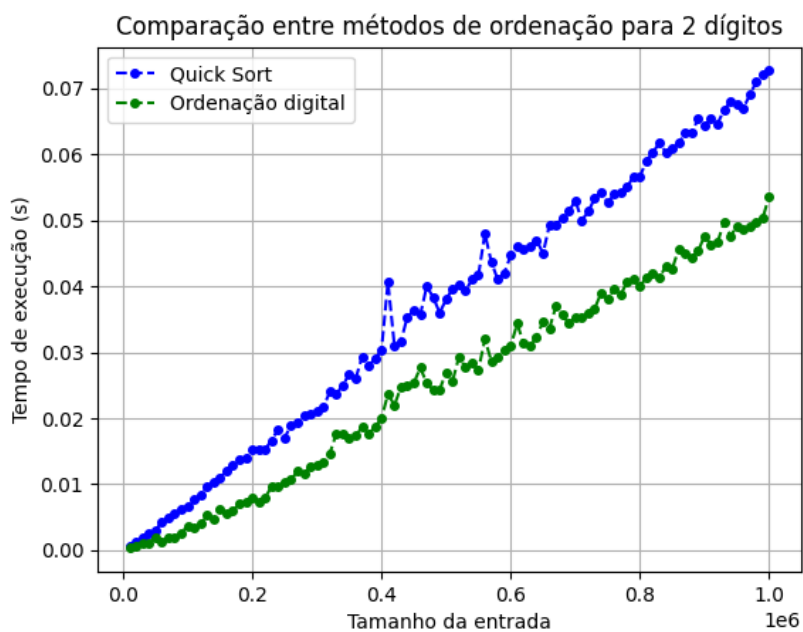


**(b) sem *bubble sort***

**Figura 9. Dez testes para 15 dígitos**



**Figura 10. Três testes para 2 dígitos**



## Referências

Cormen, T. H. (2014). *Desmistificando Algoritmos*. Elsevier, 1th edition.

de Souza, J. F. Método bubblesort. <https://bit.ly/3urKgbx>. Acesso em 05 abr. 2021.

Ziviani, N. (1999). *Projeto de Algoritmos com implementações em Pascal e C*. Editora Pioneira São Paulo, 4th edition.

## **Anexo**

[Link do repositório contendo todo o código.](#)