SUMÁRIO

1 ANÁLISE EMPÍRICA

Este capítulo tem como objetivo analisar empiricamente todos os algoritmos de ordenação apesentados anteriormente, comparando a análise teórica com os resultados obtidos na prática, no que diz respeito ao tempo de execução, número de comparações e número de movimentações.

1.1 Preliminares

Nesta seção, descreve-se o ambiente de desenvolvimento, indicando o hardware e os softwares utilizados na implementação dos algoritmos e na execução dos testes. Também é apresentada a organização do projeto em diretórios e arquivos. Além disso, são feitas definições necessárias para o melhor entendimento das seções seguintes. Por fim, descreve-se o que foi coletado e como essa coleta foi realizada.

1.1.1 Ambiente de desenvolvimento

Para a implementação dos algoritmos e execução dos testes foi utilizado um computador de mesa comum com o sistema operacional Ubuntu. Todos os algoritmos e funções auxiliares foram implementados com a linguagem de programação C. As linguagens de programação Bash e Python foram utilizadas de forma secundária para automatização dos testes e geração de gráficos, respectivamente. O hardware, softwares e versões utilizados são especificados na tabela 1.

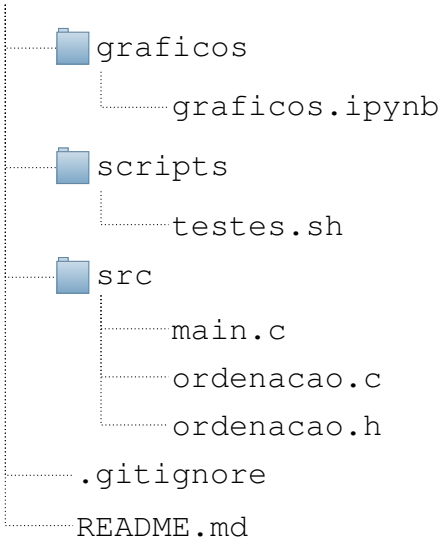
Tabela 1 – Hardware e softwares utilizados

Tipo	Componente	Produto
Hardware	Placa mãe	MSI B450M PRO-VDH MAX
	Processador	AMD® Ryzen 5 4600g
	Memória principal	2x XPG 8GB DDR4
Software	Sistema operacional	Linux Ubuntu 22.04 LTS
	Editor de texto	Microsoft Visual Studio Code
	Compilador C	GNU Compiler Collection GCC
	Linguagens de programação	C, Bash e Python

1.1.2 Organização do código fonte

A figura 1 ilustra a forma como código fonte do projeto foi organizado, enquanto que a tabela 2 dá uma breve descrição de cada arquivo.

Figura 1 – Árvore de diretórios e arquivos do projeto



Fonte: Elaborado pelo autor. Acesso público em https://github.com/jbrenorv/ordenacao

Tabela 2 – Descrição dos arquivos do projeto

Arquivo	Descrição
graficos.ipynb	Arquivo exportado do notebook criado no Google Colab
testes.sh	Implementa script em Bash para execução automatizada dos testes
main.c	Implementa a função principal
ordenacao.c	Implementa os algoritmos de ordenação e funções utilitárias
ordenacao.h	Declara os protótipos dos algoritmos e funções utilitárias
.gitignore	Declara diretórios e arquivos que o Git deve ignorar
README.md	Documenta o projeto com instruções de execução dos testes

1.1.3 Tipos de vetores

Além do tamanho do vetor, a forma como os elementos estão dispostos inicialmente pode interferir no tempo e na quantidade de operações que cada algoritmo executa. Pensando nisso, com o objetivo de analisar como cada algoritmo se comporta para diferentes tipos de entrada, foram considerados três tipos de vetores nos testes, como

mostra a tabela 3.

Tabela 3 – Tipos de vetores

Tipo	Descrição
1	Vetores já ordenados em ordem crescente
2	Vetores ordenados em ordem decrescente
3	Vetores gerados de forma pseudo-aleatória

1.1.4 Dados coletados

Esta seção visa apenas apresentar e definir quais dados foram coletados. A próxima seção descreve com mais detalhes como a coleta foi realizada. Para cada combinação de algoritmo, tamanho e tipo de vetor, foi registrado o número de comparações, o número de movimentações e o tempo de execução. Essas três métricas são definidas na tabela 4.

Obter essas informações de cada algoritmo é importante para sabermos na prática como eles se comportam para diferentes configurações de entrada. Por exemplo, um algoritmo que executa muitas movimentações pode não ser adequado em um cenário em que mudar os registros de posição seja uma operação muito lenta.

Tabela 4 – Dados coletados e suas definições

Dado	Definição
Número de comparações	Uma comparação ocorre quando um elemento do vetor é comparado com outro elemento do vetor ou com outra variável do programa
Número de movimentações	Uma movimentação ocorre sempre que um valor é atribuído a um elemento do vetor ou um elemento do vetor é atribuído a uma outra variável
Tempo de execução	Tempo decorrido entre o início e o fim da execução do algoritmo em segundos

1.1.5 Metodologia de coleta

1.1.5.1 Entrada

A função principal é o ponto de entrada para qualquer programa escrito em linguagem de C. De uma forma mais técnica, ao compilar e linkar o código fonte do projeto, obtém-se um arquivo binário executável que, ao ser executado, invoca a função principal para iniciar a execução do processo. Essa função recebe dois argumentos quando invocada.

O primeiro é um número inteiro comumente chamado de argc, que indica a quantidade de parâmetros recebidos. Já o segundo é uma lista de vetores de caracteres comumente chamado de argv, que são os parâmetros em si. O primeiro parâmetro é sempre o nome do arquivo executável ou o caminho absoluto até ele. A tabela 5 indica os parâmetros adicionais esperados.

Tabela 5 – Parâmetros da função principal

Posição (argv)	Parâmetro esperado
0	Nome ou caminho para o arquivo executável
1	Nome do arquivo onde deve ser escrito os resultados
2	Número representando o tamanho do vetor
3	Número igual a 1, 2 ou 3, representando o tipo do vetor
4	Número representando número da execução

O último parâmetro, que indica o número da execução, é explicado na subseção 1.1.5.4, que apresenta como os testes foram executados.

1.1.5.2 Processamento

O processamento consiste na execução sequencial das etapas necessárias para realizar os experimentos definidos neste trabalho. O pseudocódigo apresentado a seguir oferece uma visão de alto nível do fluxo executado pela função principal, desde a leitura dos parâmetros de entrada até o registro dos resultados obtidos. Optou-se por essa forma de apresentação para destacar a lógica geral do procedimento e evitar a exibição detalhada de trechos de código que não contribuem significativamente para a compreensão do funcionamento do sistema.

Algoritmo 1: Função principal

Entrada: A tupla (S, N, T, E) representado os parâmetros listados na tabela 5

início

$F \leftarrow$ Abre arquivo S

$O \leftarrow$ Aloca vetor de tamanho N com valores de acordo com o tipo T

$V \leftarrow$ Aloca vetor de tamanho N

$L \leftarrow$ Cria lista de pares de algoritmos[a]

para *cada par* $(A, A') \in L$ **faça**

$V \leftarrow$ Cópia de O

$t \leftarrow$ Executa A em V[b]

$V \leftarrow$ Cópia de O[c]

$(c, m) \leftarrow$ Executa A' em V[d]

Adiciona em F uma linha com os resultados[e]

fim

Libera a memória alocada e fecha o arquivo F

fim

[a] O primeiro elemento é o algoritmo original e o segundo é a versão modificada para obter o número de comparações e movimentações.

[b] Obtendo o tempo de execução t em segundos.

[c] Neste ponto V está ordenado, então é necessário resetar para o vetor original.

[d] Obtendo o número de comparações c e movimentações m.

[e] Precisamente, a linha contêm o nome do algoritmo, o tamanho do vetor, o tipo do vetor, o número da execução, o número de comparações, o número de movimentações e o tempo de execução.

1.1.5.3 Exemplo de saída

Se após compilar e linkar o código C, o arquivo executável resultante se chamar a.out, o comando a ser executado a partir de um Terminal Linux, aberto no mesmo diretório do executável, poderia ser, por exemplo:

```
1 ./a.out saida.csv 1000 3 1
```

Neste caso, seria gerado um vetor de forma pseudoaleatória (tipo 3) de tamanho 1000 e os resultados seriam escritos no arquivo saida.csv. Com exceção do cabeçalho, o conteúdo do arquivo saida.csv é representado na tabela 6. Além disso, a coluna do tempo pode conter valores diferentes.

Tabela 6 – Exemplo de saída

Algo.	Tam.	Tipo	Exec.	Comp.	Movi.	Tempo (s)
Bolha	1000	2	1	499224	745914	0.003385
Coquetel	1000	2	1	376244	745914	0.004085
Selecao	1000	2	1	499500	2970	0.000762
Insercao	1000	2	1	249630	250636	0.000220
Shellsort	1000	2	1	13038	20254	0.000148
Mergesort	1000	2	1	8698	19303	0.000126
Heapsort	1000	2	1	8779	12587	0.000137
Quicksort	1000	2	1	10777	21210	0.000135
QuicksortI	1000	2	1	11568	15621	0.000091
Introsort	1000	2	1	12291	9608	0.000084
Contagem	1000	2	1	0	2000	0.152234
Balde	1000	2	1	1046	4046	0.000036
RadixsortC	1000	2	1	0	18000	0.000029
RadixsortB	1000	2	1	0	18000	0.000075

1.1.5.4 Testes

Neste ponto o código implementado em linguagem C está pronto para ser invocado com diferentes parâmetros de entrada. Para automatizar este processo, foi criado um script em linguagem Bash. Este script segue os seguintes passos:

1. Compilação do código C e inicialização do arquivo CSV de saída.
2. Geração do tamanhos.
3. Três execuções para cada tamanho e tipo de vetor.

Optou-se por realizar três execuções por tamanho e tipo de vetor para reduzir o impacto de variações ocasionais no tempo de execução, como pequenas flutuações no uso da CPU. Esse número foi escolhido por ser suficiente para obter uma média representativa sem tornar a coleta de dados muito demorada.

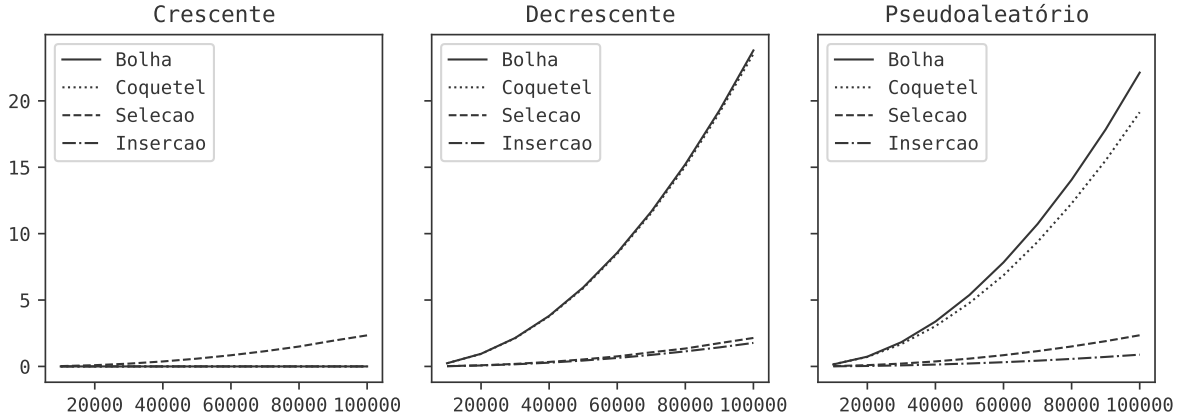
Os tamanhos gerados são melhores descritos na tabela 7. Foram considerados ao todo 207 tamanhos entre 10^3 e 10^8, inclusive. Além disso, cada tamanho foi usado três vezes para cada um dos três tipos, totalizando 1863 chamadas aos algoritmos.

Tabela 7 – Geração de tamanhos

Intervalo		Incremento	Subtotal	Total
Início	Fim			
10^3	$10^4 - 1$	5×10^2	18	207
10^4	$10^5 - 1$	4×10^3	23	
10^5	$10^6 - 1$	3×10^4	30	
10^6	$10^7 - 1$	2×10^5	45	
10^7	10^8	1×10^6	91	

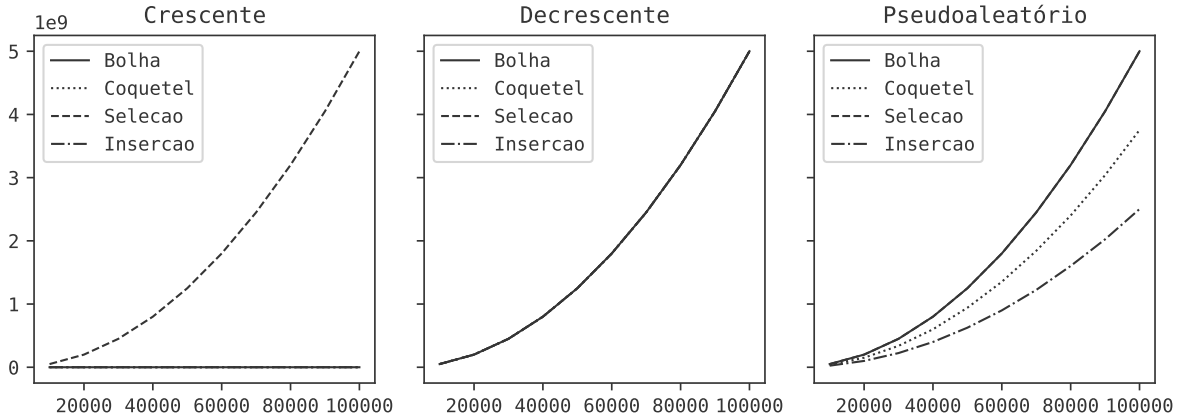
1.2 Métodos inferiores

Figura 2 – Métodos inferiores - Tamanho × Tempo.



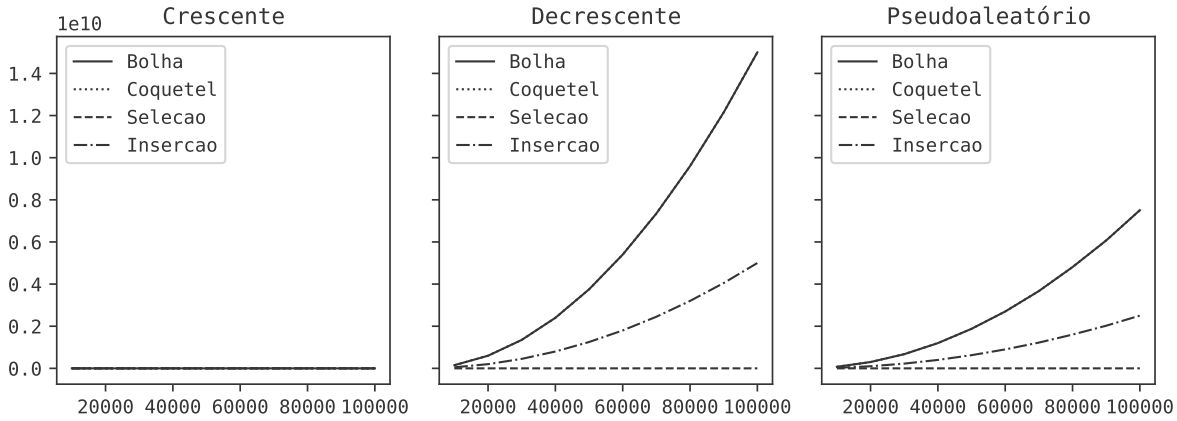
Fonte: Elaborado pelo autor

Figura 3 – Métodos inferiores - Tamanho × Comparações.



Fonte: Elaborado pelo autor

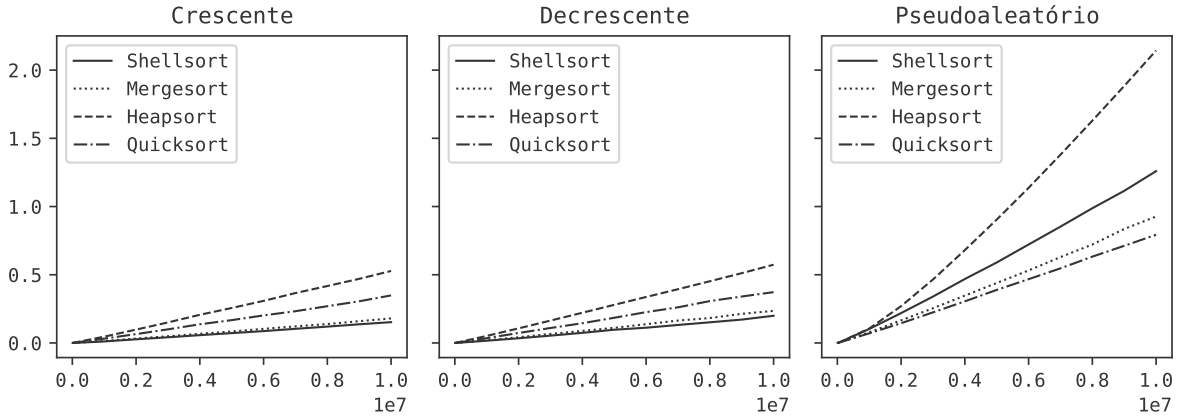
Figura 4 – Métodos inferiores - Tamanho × Movimentações.



Fonte: Elaborado pelo autor

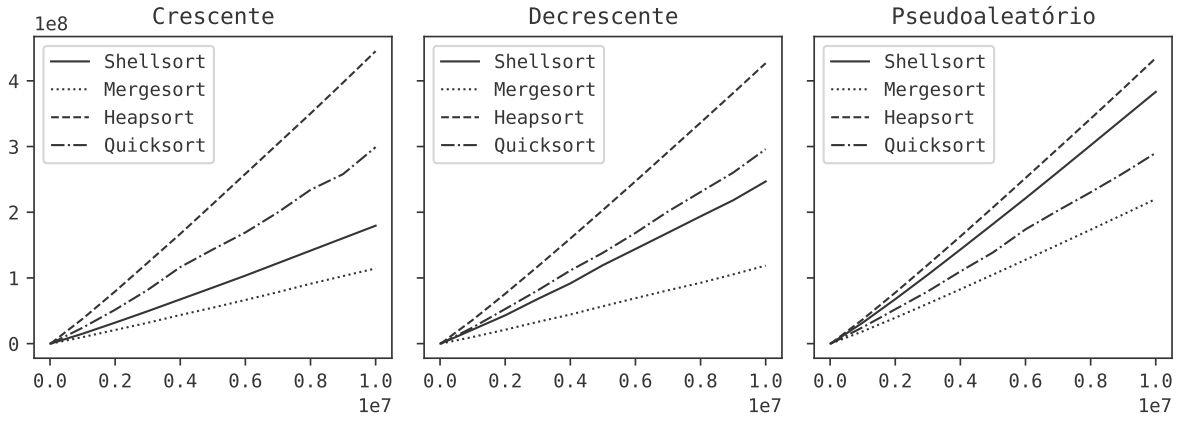
1.3 Métodos superiores

Figura 5 – Métodos superiores - Tamanho × Tempo.



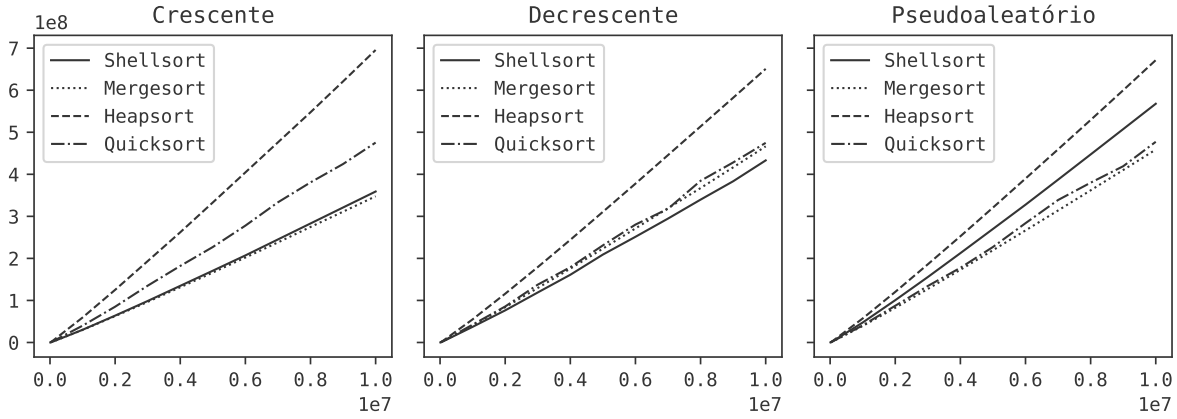
Fonte: Elaborado pelo autor

Figura 6 – Métodos superiores - Tamanho × Comparações.



Fonte: Elaborado pelo autor

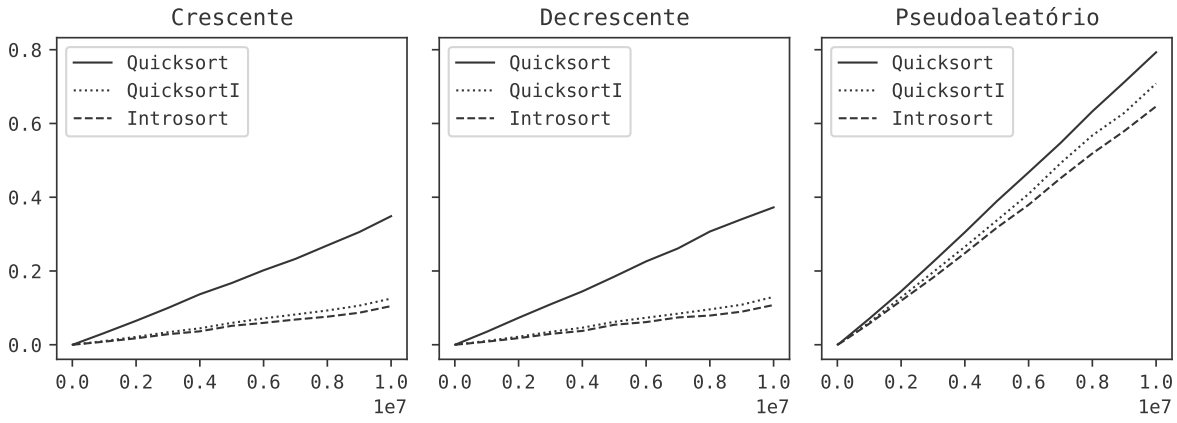
Figura 7 – Métodos superiores - Tamanho × Movimentações.



Fonte: Elaborado pelo autor

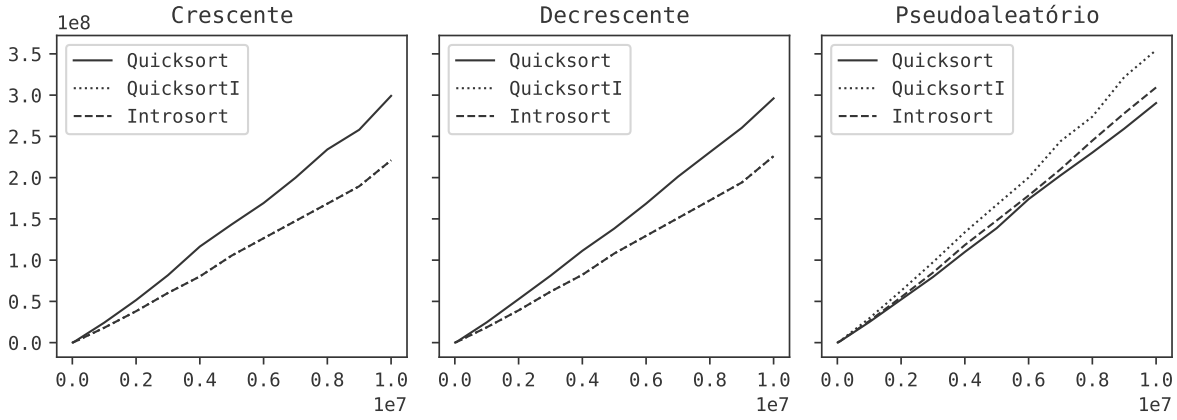
1.4 Quicksort e métodos híbridos

Figura 8 – Métodos híbridos - Tamanho × Tempo.



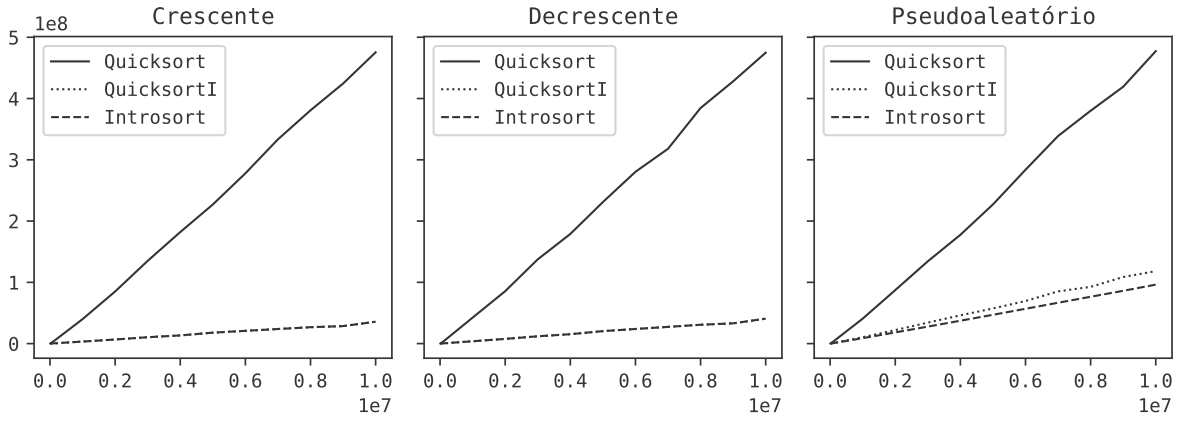
Fonte: Elaborado pelo autor

Figura 9 – Métodos híbridos - Tamanho × Comparações.



Fonte: Elaborado pelo autor

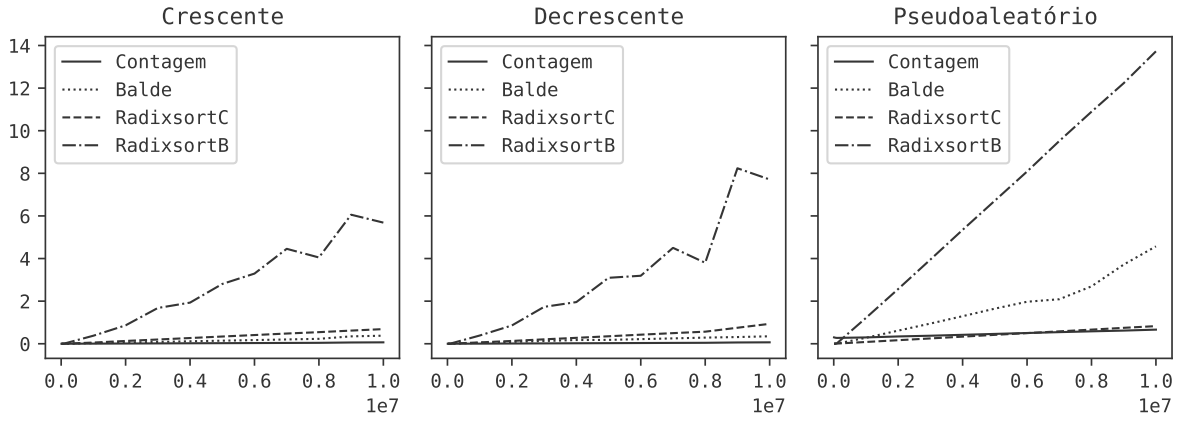
Figura 10 – Métodos híbridos - Tamanho × Movimentações.



Fonte: Elaborado pelo autor

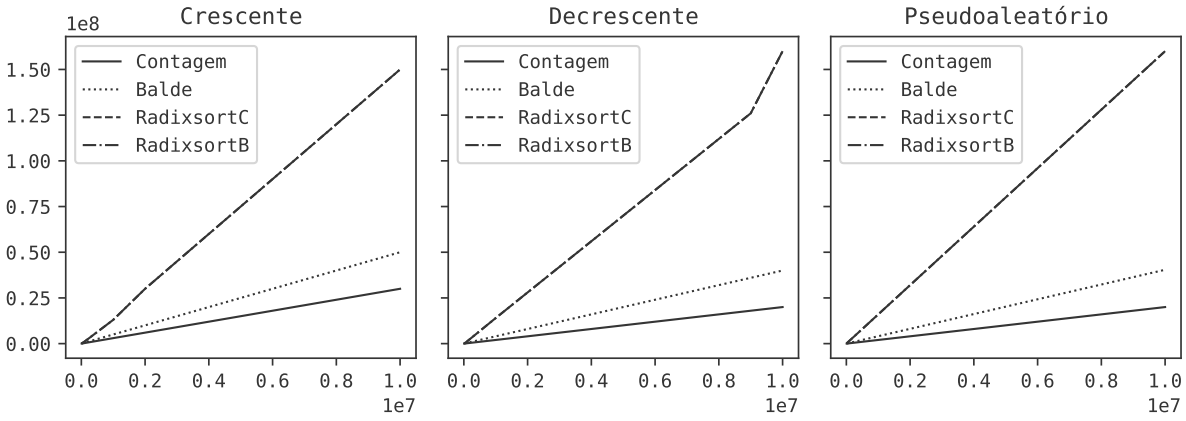
1.5 Métodos lineares

Figura 11 – Métodos lineares - Tamanho × Tempo.



Fonte: Elaborado pelo autor

Figura 12 – Métodos lineares - Tamanho × Movimentações.



Fonte: Elaborado pelo autor

REFERÊNCIAS