Universidade Federal de Pernambuco - UFPE
Centro de Informática - Graduação em Sistemas de Informação
3º Projeto de Implementação - Algoritmos e Estruturas de Dados (IF969)

Análise experimental de algoritmos: *insertion sort*, *quicksort* e *radix sort*

Aluno: Vinícius Giles Costa Paulino (vgcp)
Professor: Renato Vimieiro (rv2)

1. Resumo

Este relatório tem o objetivo de apresentar resultados de análises experimentais dos comportamentos de algoritmos de ordenação *insertion sort*, *radix sort* e *quicksort*, avaliados de acordo com seus respectivos tempos de execução em diversos casos de teste.

2. Detalhes da implementação

O projeto foi implementado na linguagem Python utilizando algumas bibliotecas adicionais. Logo, tem-se como pré-requisitos para executar o código desse teste os seguintes módulos:

1. **pandas**, para uso da estrutura de dados *DataFrame* como um correto armazenamento dos dados obtidos com os experimentos;
2. **ggplot**, para plotar gráficos com os dados obtidos dos experimentos e identificar o comportamento dos algoritmos;
3. **numpy**, para realizar o cálculo da média e desvio padrão dos dados e também utilizar arrays;
4. **time** e **random** (ambas da *Python Standard Library*), usadas para medir tempos de execução e gerar vetores de números pseudoaleatórios.

O Microsoft Office Excel foi usado como ferramenta para fazer a regressão linear. Foram também aproveitados alguns conceitos de programação orientada a objetos vistas no início da disciplina, como classes, métodos e atributos públicos e privados, encapsulamento, etc.

A classe *Sorting* implementa os algoritmos de ordenação, além de conter métodos privados de uso particular de cada algoritmo. Os métodos dessa classe, *insertionSort()*, *quickSort()* e *radixSort()*, devem ser chamados com dois parâmetros obrigatórios (um vetor v e seu tamanho n), e todos eles retornam os seus respectivos tempos de execução.

A classe *Experiment* implementa os testes que serão realizados com os algoritmos de ordenação e de que forma os dados obtidos serão avaliados. O principal método dessa classe, *test()*, executa de uma vez toda a bateria de testes analisada neste relatório.

3. Metodologia

Os algoritmos de ordenação foram submetidos a testes com 60 vetores de números inteiros, de 0 até 100.000 elementos em intervalos regulares de 5.000 elementos, sendo 20 vetores ordenados crescentemente, 20 ordenados decrescentemente e 20 em ordem aleatória.

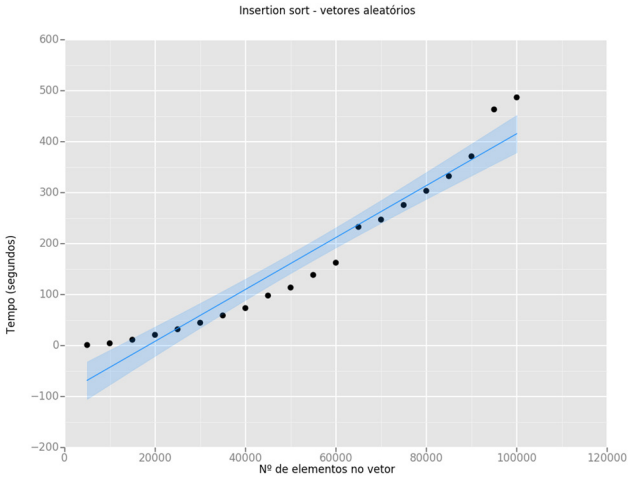
Cada ordenação foi repetida 10 vezes por cada algoritmo para obter a média dos tempos para diminuir o impacto de tarefas paralelas na mensuração do tempo.

4. Resultados

4.1 – Insertion sort

O algoritmo insertion sort tem complexidade de tempo O(n) no melhor caso e O(n²) no pior caso [1].

4.1.1 - Vetores aleatórios



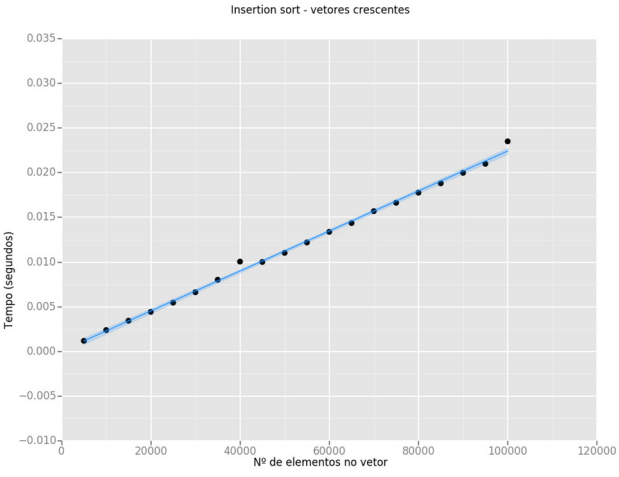
C.linear	-93,69336122
C.angular	0,005093813
RESUMO DOS RESULTADOS	
Estatística de regressão	
R múltiplo	0,967346277
R-Quadrado	0,93575882
R-quadrado aju	0,932189865
Erro padrão	40,56134806
Observações	20

Ao analisar os dados dos experimentos do insertion sort e fazer a regressão linear, descobre-se a equação da reta que descreve o tempo de execução T para n elementos no vetor aleatório como sendo aproximadamente:

$$T(n) = 0{,}00509n - 93{,}69336$$

O erro padrão obtido na regressão está elevado porque a curva por vezes se afasta da reta.

4.1.2 - Vetores crescentes



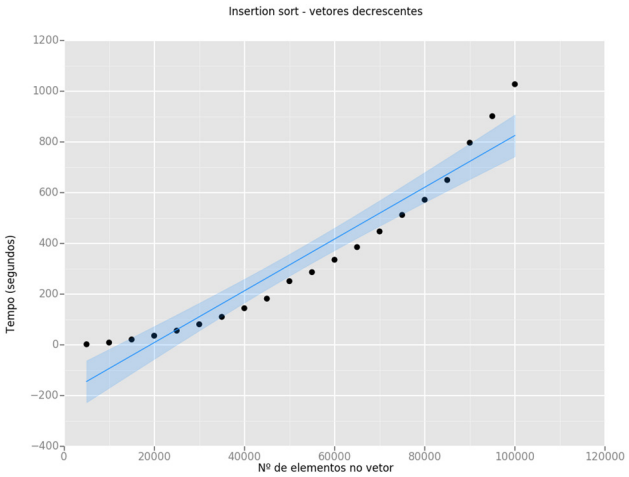
C. linear	5,86858E-05
C.angular	2,23705E-07
RESUMO DOS RESULTADOS	
Estatística de regressão	
R múltiplo	0,998325007
R-Quadrado	0,996652821
R-quadrado aju	0,996466866
Erro padrão	0,000393992
Observações	20

Analisando os dados dos experimentos do insertion sort e fazer a regressão linear, descobre-se a equação da reta que descreve o tempo de execução T para n elementos no vetor crescente como sendo aproximadamente:

$$T(n) = 2{,}23\times10^{-7}n - 5{,}8685$$

O insertion sort tem custo O(n) quando o vetor já está (ou quase) ordenado. Com isso, o algoritmo não faz trocas de posição, logo o desempenho se torna linear [2].

4.1.3 - Vetores decrescentes



C.linear	-195,4347087
C.angular	0,010208154
RESUMO DOS RESULTADOS	
Estatística de regressão	
R múltiplo	0,95943707
R-Quadrado	0,920519491
R-quadrado aju	0,916103907
Erro padrão	91,16027411
Observações	20

Ao analisar os dados dos experimentos do insertion sort e fazer a regressão linear, descobre-se a equação da reta que descreve o tempo de execução T para n elementos no vetor decrescente como sendo aproximadamente:

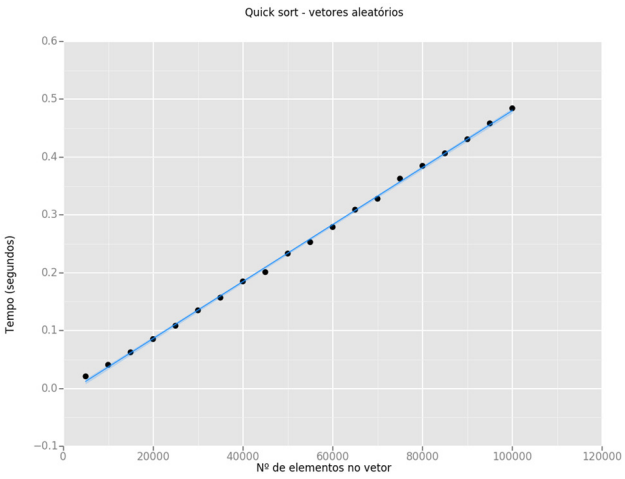
$$T(n) = 0{,}01n - 195{,}434$$

O insertion sort tem custo $O(n^2)$ vetor já está ordenado decrescentemente. Isso explica a curva mais acentuada deste experimento.

4.2 – Quicksort

O algoritmo quicksort tem complexidade de tempo O(n lgn)[4] no seu melhor caso enquanto no pior caso é $O(n^2)$[5]. A versão do quicksort implementada neste projeto escolhe um pivô aleatório em vez do primeiro elemento, o que pode diminuir a probabilidade de um pior caso ocorrer.

4.2.1 - Vetores aleatórios

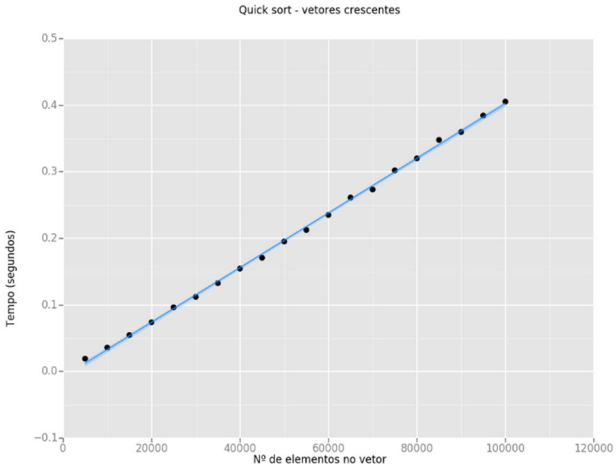


C. linear	-0,012353897
C. angular	4,92801E-06
RESUMO DOS RESULTADOS	
Estatística de regressão	
R múltiplo	0,999615551
R-Quadrado	0,999231249
R-quadrado ajustado	0,99918854
Erro padrão	0,004154093
Observações	20

Ao analisar os dados dos experimentos do quicksort e fazer a regressão linear, descobre-se a equação da reta que descreve o tempo de execução T para n elementos no vetor aleatório como sendo aproximadamente:

$$T(n) = 4{,}9\times10^{-6}n - 0{,}012353$$

4.2.2 - Vetores em ordem crescente

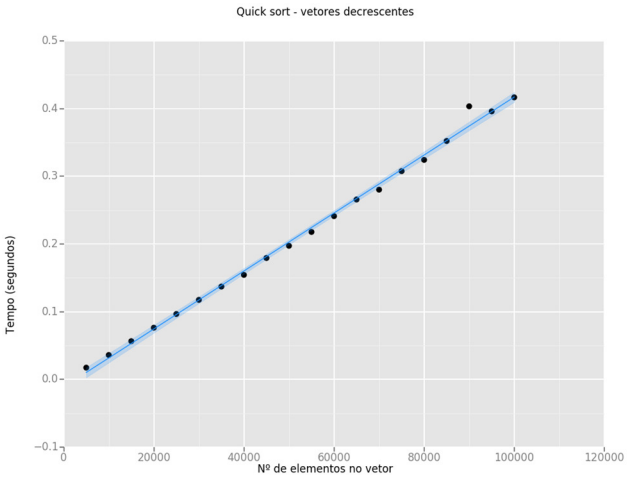


C. linear	-0,00802615
C. angular	4,10529E-06
RESUMO DOS RESULTADOS	
Estatística de regressão	
R múltiplo	0,999526102
R-Quadrado	0,999052428
R-quadrado ajustado	0,998999785
Erro padrão	0,003842377
Observações	20

Ao analisar os dados dos experimentos do quicksort e fazer a regressão linear, descobre-se a equação da reta que descreve o tempo de execução T para n elementos no vetor crescente como sendo aproximadamente:

$$T(n) = 4{,}1\times10^{-6}n - 0{,}008026$$

4.2.3 - Vetores em ordem decrescente



C. linear	-0,011114563
C. angular	4,28182E-06
RESUMO DOS RESULTADOS	
Estatística de regressão	
R múltiplo	0,998037991
R-Quadrado	0,996079832
R-quadrado ajustado	0,995862045
Erro padrão	0,008163533
Observações	20

Ao analisar os dados dos experimentos do quicksort e fazer a regressão linear, descobre-se a equação da reta que descreve o tempo de execução T para n

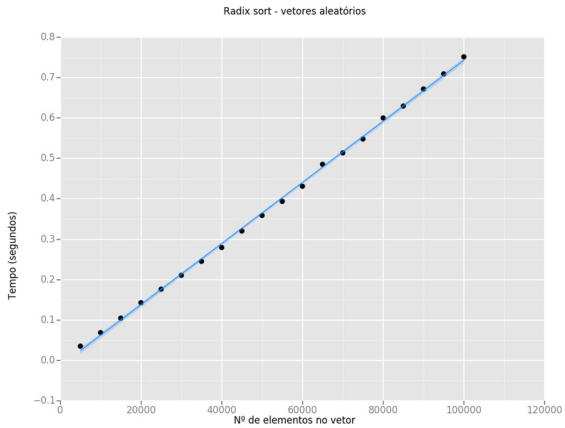
elementos no vetor decrescente como sendo aproximadamente:

$$T(n) = 4{,}2\times10^{-6}n - 0{,}011114$$

4.3 – Radix sort

O algoritmo radix sort tem complexidade de tempo O(n+k), onde n é o número de elementos do vetor de teste e k é o número de dígitos. Neste projeto, o radix sort foi implementado com base em algarismos decimais.

4.3.1 – Vetores aleatórios

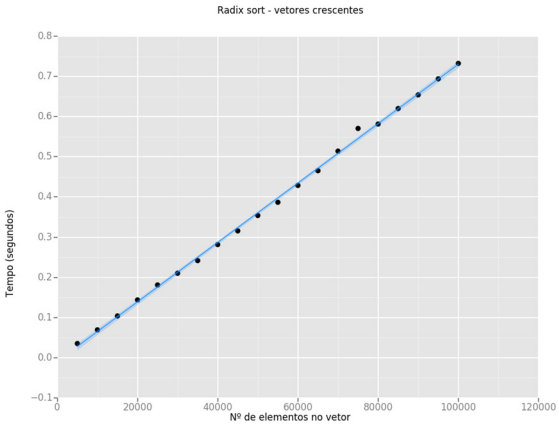


C.linear	-0,013341664
C.angular	7,56655E-06
RESUMO DOS RESULTADOS	
Estatística de regressão	
R múltiplo	0,999547434
R-Quadrado	0,999095073
R-quadrado ajus	0,999044799
Erro padrão	0,006920632
Observações	20

Assim como os moldes teóricos, os experimentos mostraram um desempenho linear do radix sort decimal. A equação da reta que expressa o tempo T no caso de vetores aleatórios é a seguinte:

$$T(n) = 7{,}56\times10^{-6}n - 0{,}013341$$

4.3.2 – Vetores crescentes

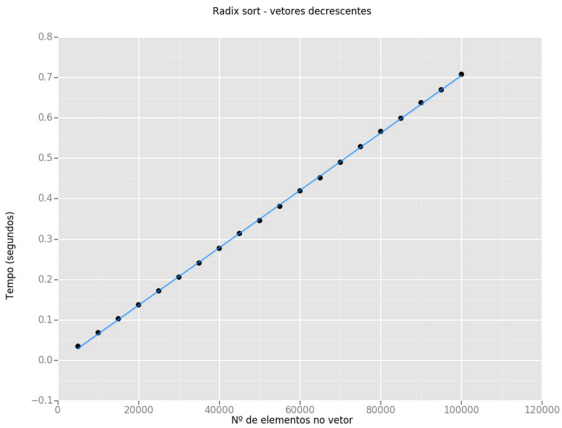


C.linear	-0,00873852
C.angular	7,38987E-06
RESUMO DOS RESULTADOS	
Estatística de regressão	
R múltiplo	0,999342242
R-Quadrado	0,998684916
R-quadrado ajus	0,998611855
Erro padrão	0,008149743
Observações	20

Com vetores em ordem crescente, os experimentos do radix sort mostraram um desempenho também linear. A equação da reta que expressa o tempo T no caso de vetores em ordem crescente é:

$$T(n) = 7{,}38\times10^{-6}n - 0{,}008738$$

4.3.3 – Vetores decrescentes



C. linear	-0,030280644
C. angular	7,35747E-06
RESUMO DOS RESULTADOS	
Estatística de regressão	
R múltiplo	0,971495589
R-Quadrado	0,943803679
R-quadrado ajus	0,940681662
Erro padrão	0,054561453
Observações	20

Dessa vez com vetores em ordem decrescente, o desempenho do radix também se mostra linear. A equação da reta que expressa o tempo T no caso de vetores em ordem decrescente é:

$$T(n) = 7{,}38\times10^{-6}n - 0{,}008738$$

5. Conclusão

A análise experimental dos algoritmos de ordenação insertion sort, quick sort e radix sort nos permite ter uma compreensão além da teoria que é vista em sala de aula, principalmente em relação às noções de complexidade assintótica.

6. Referências

[1] ZIVIANI, Nívio. **Projeto de Algoritmos com implementações em Java e C++**. p. 118. São Paulo: Cengage Learning, 2007.

[2] ZIVIANI, Nívio. **Projeto de Algoritmos com implementações em Java e C++**. p. 118. São Paulo: Cengage Learning, 2007.

[3] LEVITIN, Anany. **Introduction to design & analysis of algorithms**. 3. ed. p. 135. Pearson, 2012.

[4] ZIVIANI, Nívio. **Projeto de Algoritmos com implementações em Java e C++**. p. 123. São Paulo: Cengage Learning, 2007.

[5] ZIVIANI, Nívio. **Projeto de Algoritmos com implementações em Java e C++**. p. 123. São Paulo: Cengage Learning, 2007.