UNIVERSIDADE ESTADUAL DE PONTA GROSSA

SETOR DE CIÊNCIAS AGRÁRIAS E TECNOLÓGIA

DEPARTAMENTO DE INFORMÁTICA

MIKEIAS SILVA GOMES DE AZEVEDO

Análise empírica do comportamento assintótico de algoritmos de ordenação

PONTA GROSSA

2018

UNIVERSIDADE ESTADUAL DE PONTA GROSSA

SETOR DE CIÊNCIAS AGRÁRIAS E TECNOLÓGIA

DEPARTAMENTO DE INFORMÁTICA

MIKEIAS SILVA GOMES DE AZEVEDO

Análise empírica do comportamento assintótico de algoritmos de ordenação

Trabalho sobre Análise empírica do comportamento assintótico de algoritmos de ordenação, com objetivo de obtenção de nota parcial da disciplina de Análise de Algoritmos ministrada pelo Professor Luiz Pedro Petroski.

PONTA GROSSA

2018

**Sumário**

[1. Introdução 4](#_Toc525419510)

[2. Objetivo Geral 5](#_Toc525419511)

[3. Análise empírica do comportamento assintótico de algoritmos de ordenação 6](#_Toc525419512)

[3.1 Hardware e Sistema operacional utilizado 6](#_Toc525419513)

[3.2 Insertion Sort 6](#_Toc525419514)

[3.3 Selection Sort 7](#_Toc525419515)

[3.4 Quicksort 8](#_Toc525419516)

[3.5 Heap Sort 9](#_Toc525419517)

[3.7 Bucket sort 11](#_Toc525419518)

[4. Conclusão 13](#_Toc525419519)

[5. Referências 14](#_Toc525419520)

1. Introdução

Este trabalho trata sobre análise empírica do comportamento assintótico de algoritmos de ordenação, mais precisamente sobre os algoritmos Insertion Sort, Selection Sort, Quick Sort, Heap Sort, Radix Sort e Bucket Sort.

Tendo como objetivo demonstrar como funciona suas ordenações e o tempo de execução para cada algoritmo.

A metodologia utilizada foi a implementação dos algoritmos na linguagem C, juntamente com o Sistema Operacional Linux.

1. Objetivo Geral

Analisar a execução e comportamento dos algoritmos de ordenação em um arranjo de amostras ordenadas, desordenadas e inversa, apresentando juntamente com o hardware utilizado para obter os resultados apresentado nesse trabalho.

Sendo dessa forma colocando em prática as situações apresentadas em sala de aula sobre alguns modos de ordenação e a complexidade dos algoritmos e demonstrar sobre qual situação seria a melhor utilização observando o tempo de execução dos mesmos.

1. Análise empírica do comportamento assintótico de algoritmos de ordenação

Uma análise empírica é basicamente uma análise feita por tempo de execução não levando em consideração matemática e sim “temporização” que é uma alternativa as técnicas matemáticas, o hardware e os métodos utilizado para a execução, avaliando os melhores e piores casos de execução e ordenação, o que pode ser também um teste ou análise de performance.

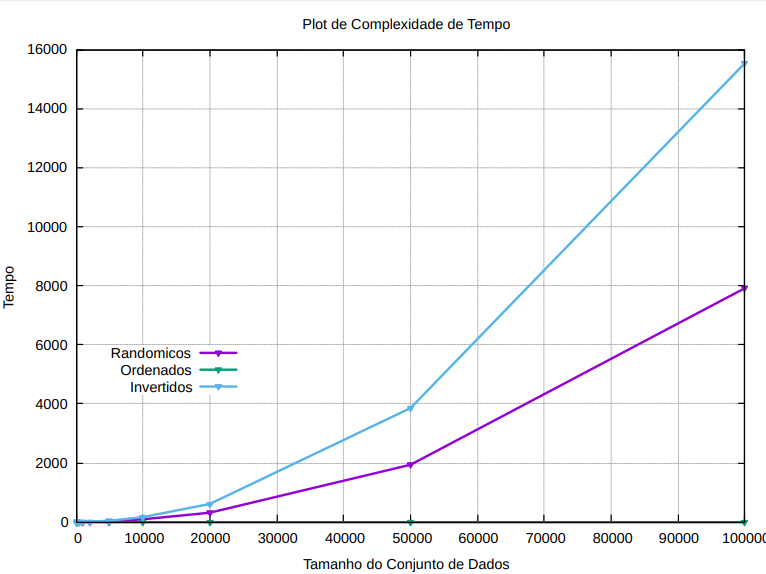
1. Hardware e Sistema operacional utilizado

* Processador: Inter Core i7-5500U 2.40GHz
* Memória RAM: DDR3 8,00 GB
* SSD: SanDisk SSD PLUS 240GB
* Sistema operacional: Linux Ubuntu 18.04.1 LTS

1. Insertion Sort

A análise empírica a seguir é do algoritmo insertion sort que consiste em dado uma lista, constrói uma matriz final com um elemento de cada vez. Assim como algoritmo de ordenação quadrática é bastante eficiente a pequenas entradas. Melhor caso O(n), quando a matriz está ordenada, médio caso O(n²/4), quando a matriz tem valores aleatórios sem ordem de classificação (crescente ou decrescente), pior caso O(n²), quando a matriz está em ordem inversa, daquela que deseja ordenar.

Na figura a baixo podemos observar os resultados obtidos após a execução.

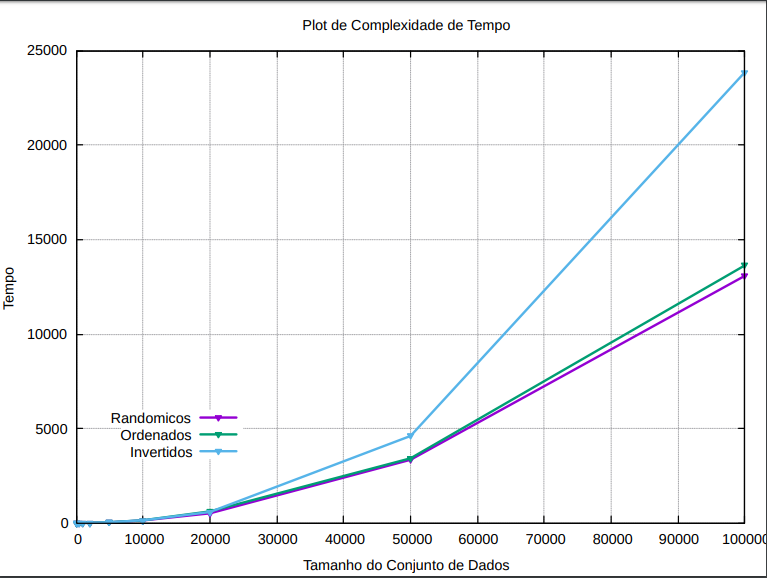


1. Selection Sort

O algoritmo Selection Sort é um algoritmo de ordenação baseado em se passar sempre o menor valor para o vetor para a primeira posição, depois o de segundo menor valor para a segunda posição e assim sucessivamente até o elemento n – 1.

A complexidade de pior caso nesse algoritmo é O(n²), no caso médio é de O(n²) e o melhor caso O(n²). Dessa forma podemos entender então que não existe um melhor e pior caso mesmo que o arranjo esteja ordenado ou em ordem inversa.

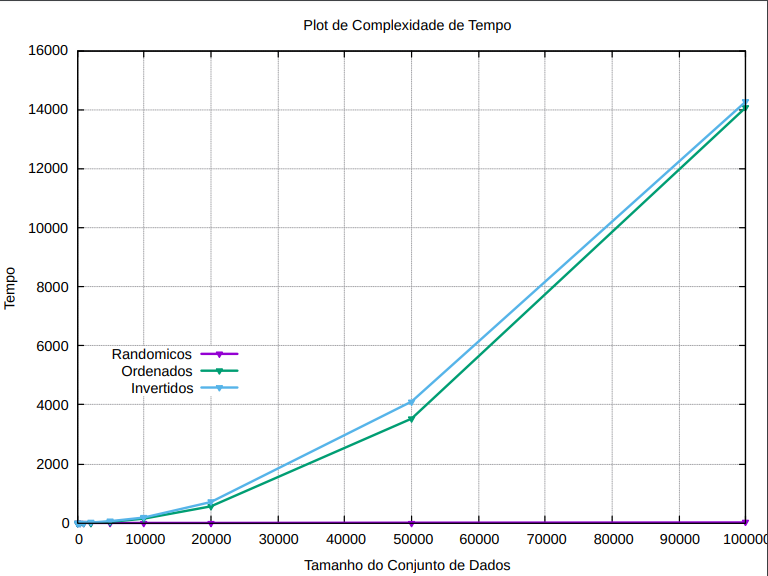
A figura abaixo contém os resultados obtidos na execução do algoritmo Selection Sort.



1. Quicksort

O quicksort adota a estratégia de divisão e conquista. A estratégia consiste em rearranjar as chaves de modo que as chaves "menores" precedam as chaves "maiores". Em seguida o quicksort ordena as duas sublistas de chaves menores e maiores recursivamente até que a lista completa se encontre ordenada. A complexidade de pior caso para o algoritmo quicksort é O(n²), caso médio O(n log n), O(n log n).

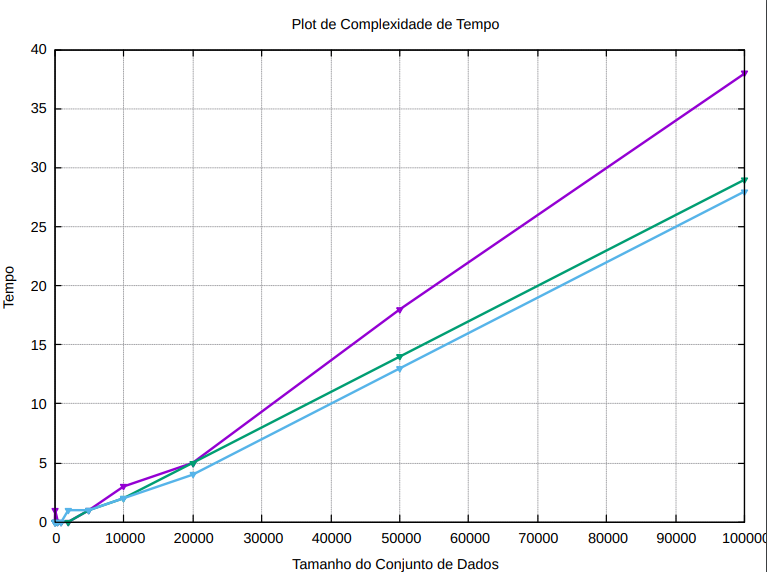
Abaixo os resultados obtidos na execução do quicksort.



1. Heap Sort

O heapsort utiliza uma estrutura chamada heap, para ordenar os elementos à medida que insere na estrutura. Assim ao final das inserções os elementos podem ser sucessivamente removidos da raiz da heap, na ordem desejada. A complexidade do algoritmo heap sort do pior caso, caso médio e melhor caso é o mesmo, O(n log n).

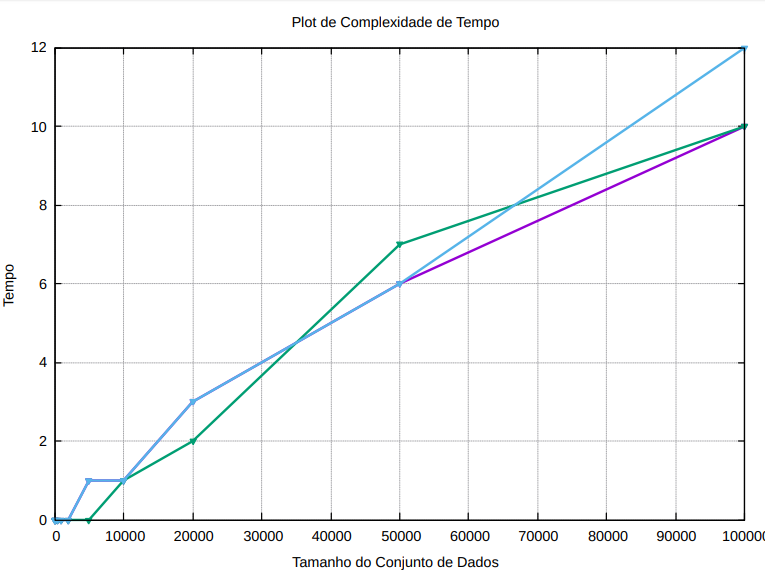
Na sequência os resultados obtidos na execução da análise empírica.



1. **Radix sort**

Em poucas palavras, a ideia do Radix Sort é fazer ordenação por dígito a partir do dígito menos significativo até o dígito mais significativo. Tempo de execução pior caso O(nk).

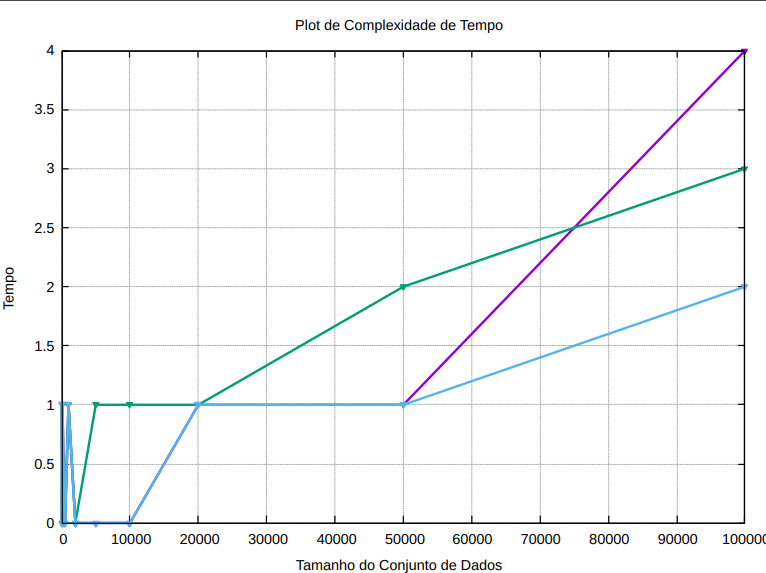
Abaixo os resultados obtidos na execução do algoritmo radix sort.



3.7 Bucket sort

Algoritmo de ordenação que divide um vetor em um numero finito de recipiente e cada recipiente é ordenado individualmente, seja usando outro algoritmo ou usando o buckset sort recursivamente. Complexidade de pior caso O(n²), caso médio O(n + k), melhor caso O(n + k).

A execução e resultados obtidos do algoritmo se encontra a seguir



1. Conclusão

Podemos observar os diferentes comportamentos dos métodos de ordenação analisados e perante a análise empírica que consistiu em ordenação de arranjos ordenados, desordenados e inversamente ordenados, assim como cada complexidade de tempo pelos resultados das tabelas, que em determinados casos a ordenação se sai muito melhor que em outros obtendo uma eficiência melhor.

1. Referências

Radix Sort: disponível em <https://pt.wikipedia.org/wiki/Radix\_sort> acesso 22 de setembro de 2018.

Radix Sort: disponível em <https://www.geeksforgeeks.org/radix-sort/> acesso 22 de setembro de 2018.

Bucket Sort: disponível em <https://pt.wikipedia.org/wiki/Bucket\_sort> acesso 22 de setembro de 2018.

Heapsort: disponível em <https://pt.wikipedia.org/wiki/Heapsort> acesso 22 de setembro de 2018.

Quicksort: disponível em <https://pt.wikipedia.org/wiki/Heapsort> acesso 22 de setembro de 2018.

Selection Sort: disponível em <https://pt.wikipedia.org/wiki/Selection\_sort> acesso 22 de setembro de 2019.

Algoritmos: https://www.cin.ufpe.br/~ara/algoritmos-%20portugu%EAs-%20cormen.pdf