Algoritmo e Estrutura de Dados
Relatório do Trabalho 2 Parte 1- Algoritmos de Ordenação
Autores:

Alex Galhardo
Numero USP: 10408344

Ian Castilho Caldeira Brant
Numero USP: 10133967

ICMC USP

São Carlos, 22 de Novembro de 2017

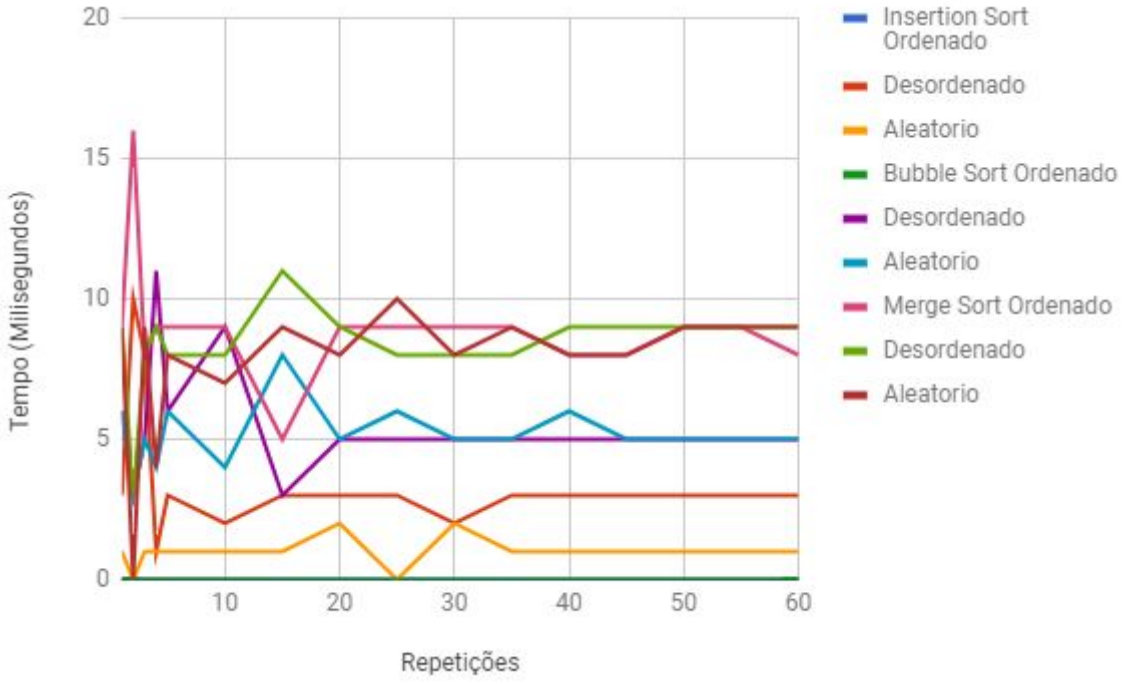
- ## Introducao

Esse relatório tem como objetivo analisar os resultados obtidos em testes realizados com os algoritmos de ordenação Insertion Sort, Bubble Sort e Merge Sort. Cada algoritmo foi testado ordenando um conjunto já ordenado, um conjunto completamente desordenado e um conjunto aleatório. Foram testados diferentes tamanhos de conjuntos, sendo medido o tempo de ordenação em cadas teste. Para não deixar variáveis computacionais externas interferirem nos resultados, foram feitas várias repetições para cada teste, utilizando a média de tempo como resultado

Feita as medições, comparou-se os resultados obtidos com os resultados esperados de complexidade de cada algoritmo. A análise foi feita graficamente, comparando o crescimento do tempo de ordenação de cada algoritmo com funções de complexidade encontrada na literatura para cada um dos casos.

- **Repetições**

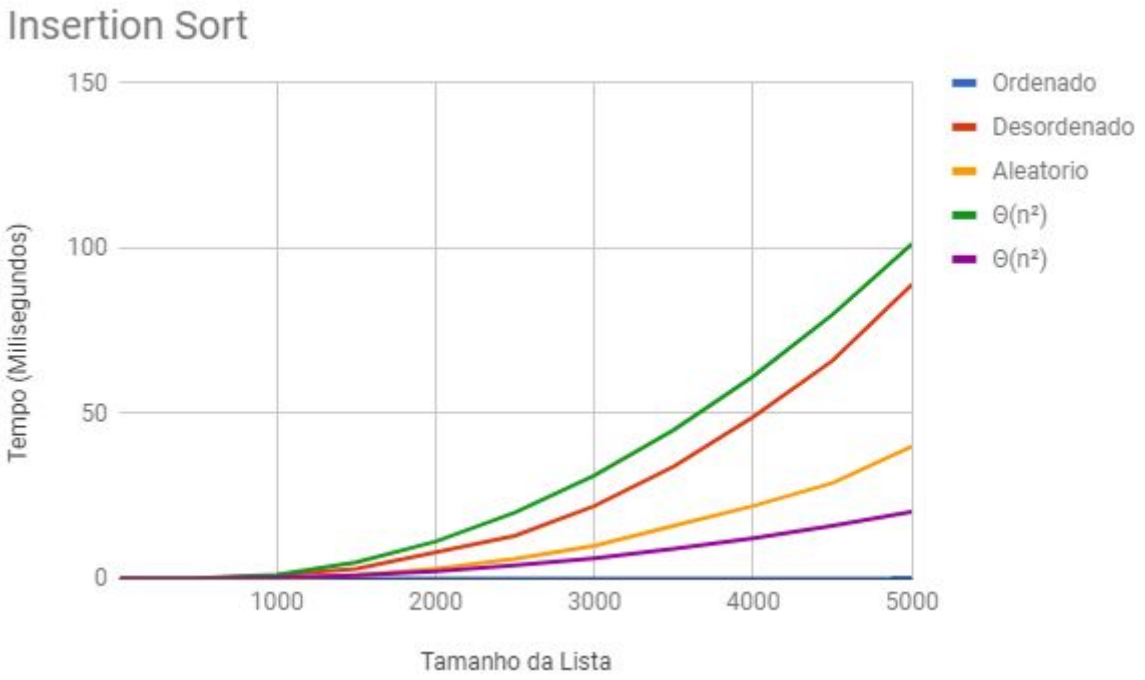
Para obter o número ideal de repetições para chegar em uma boa média, foi fixado um conjunto de 1000 elementos e variou-se o número de repetições. O número de repetições foi sendo aumentado até que não se observou mais grandes alterações nos resultados obtidos, como é mostrado no gráfico abaixo:



Pode-se observar pelo gráfico, que a partir de 50 repetições não há mais grandes variações nos valores obtidos. Como também não poderíamos pegar um valor muito alto de repetições, devido ao tempo que levaria para cada teste, principalmente para conjuntos muito grandes, fixamos o valor de repetições em 50.

- **Insertion Sort**

Fixado o número de repetições em 50, testou-se o Insertion Sort com conjuntos de 1 a 5000 elementos. Os resultados dos tempos de ordenação são apresentados no gráfico abaixo:

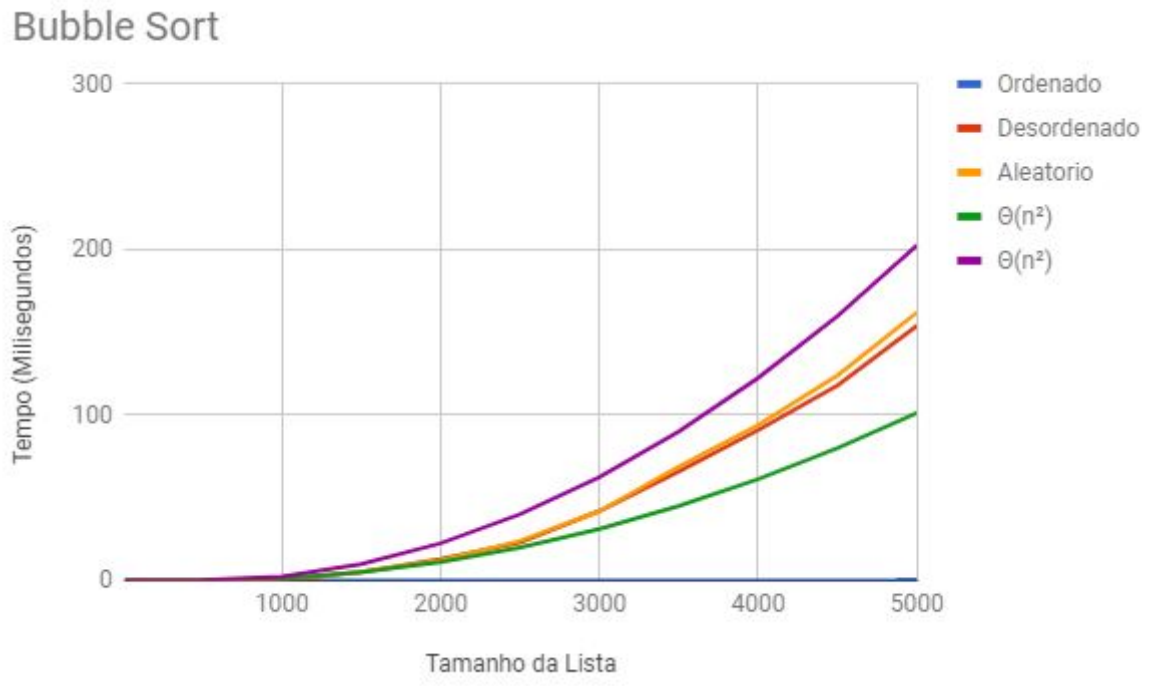


No melhor caso do Insertion Sort (conjunto já ordenado) a ordenação foi praticamente instantânea para a escala de milisegundos, não tendo sido possível observar variações no tempo de execução para diferentes tamanhos de entrada.

Já para o pior caso (conjunto completamente desordenado) e caso médio (conjunto aleatório) podemos observar pelo gráfico que o tempo de execução (complexidade) do algoritmo cresce proporcionalmente a funções $\Theta(n^2)$, como já era esperado pelas análises desse algoritmo encontradas na literatura. Podemos ver ainda que o pior caso tem um maior tempo de execução que o caso médio, como também já era esperado.

- **Bubble Sort**

Fixado o número de repetições em 50, testou-se o Bubble Sort com conjuntos de 1 a 5000 elementos. Os resultados dos tempos de ordenação são apresentados no gráfico abaixo:

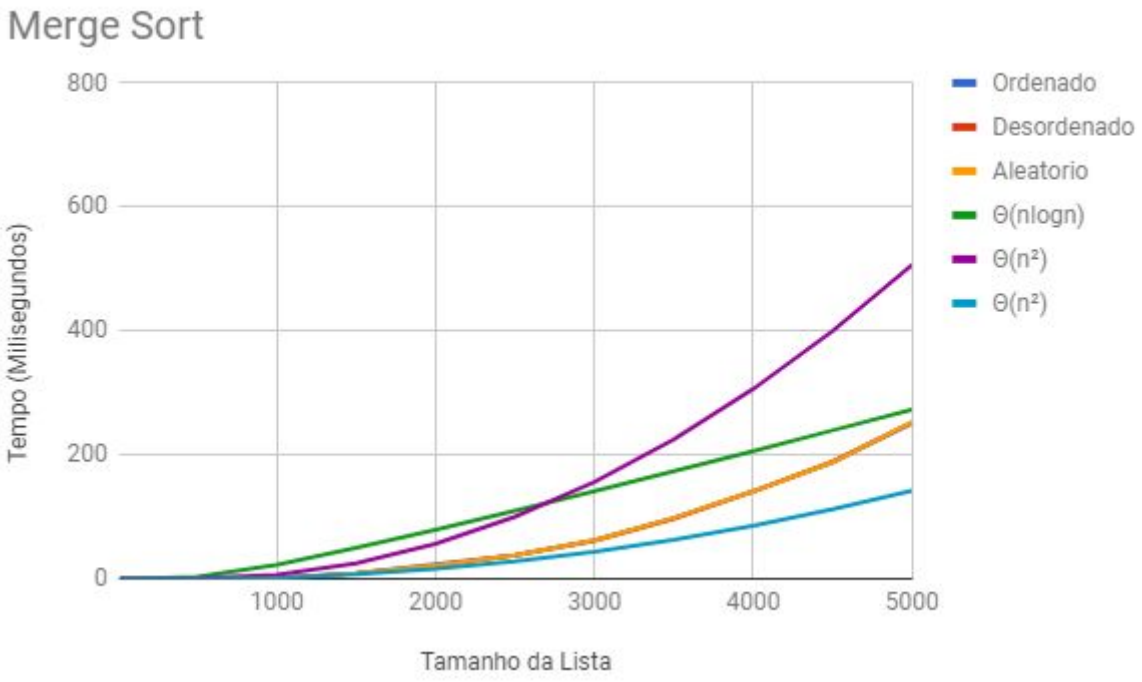


Também no Bubble Sort não foi possível observar variações no tempo de execução no melhor caso (conjunto já ordenado).

Pelo gráfico, é possível observar que tanto o pior caso (conjunto completamente desordenado) quanto o caso médio (conjunto ordenado) possuem complexidade $\Theta(n^2)$, como já era esperado pela complexidade desse algoritmo encontrada na literatura. Nesse caso observou-se também que a diferença de tempo de execução entre o pior caso e o caso médio foi bem baixa.

- **Merge Sort**

Fixado o número de repetições em 50, testou-se o Merge Sort com conjuntos de 1 a 5000 elementos. Os resultados dos tempos de ordenação são apresentados no gráfico abaixo:



O Merge Sort apresentou o mesmo tempo de execução para todos os casos (conjunto ordenado, completamente desordenado e aleatório), que condiz com o esperado para esse algoritmo, que possui a mesma complexidade para todos os casos, de acordo com a literatura.

Podemos observar pelo gráfico que o tempo de ordenação cresce proporcionalmente a funções $\Theta(n^2)$, o que não condiz com o esperado para esse algoritmo, que de acordo com a literatura possui complexidade $\Theta(n\log n)$ para todos os casos.

O provável motivo para essa inconsistência entre resultado obtido e esperado é a forma como o algoritmo foi implementado. Devido à exigência de fazer a implementação para conjuntos dinâmicos encadeados, foram feitas algumas adaptações para que a implementação fosse possível, como por exemplo o uso de uma função que retorna um ponteiro para o NÓ de acordo com a sua posição na lista. Provavelmente não conseguimos fazer uma implementação eficiente desse algoritmo. Apesar de termos testados diferentes formas, a implementação que usamos foi a que obteve melhores resultados.