Análise de Complexidade e Experimental de Algoritmos de Ordenação

Bruno Santos de Lima
Faculdade de Ciências e Tecnologia
Universidade Estadual Paulista
Presidente Prudente, Brasil
brunoslima4@gmail.com

Leandro Ungari Cayres
Faculdade de Ciências e Tecnologia
Universidade Estadual Paulista
Presidente Prudente, Brasil
leandroungari@gmail.com

***Resumo*—O**

Palavras-chave: ordenação, análise assintótica e análise experimental.

I. Introdução

Diversas aplicações da atualidade envolvem um grande volume de dados, desde aplicações comerciais simples a grande aplicações científicas, todas estão nesse contexto. A organização estrutural de conjunto de dados, além de prover melhor usabilidade, também otimiza tempo e o consumo de recursos, tanto de processamento quanto de memória para a execução.

Neste contexto, este trabalho apresenta uma análise dos principais algoritmos de ordenação, o qual está dividido nas seguintes seções: na Seção 2, são apresentados os algoritmos de ordenação utilizados, indicando a abordagem utilizada no processo juntamente com a sua respectiva análise assintótica. Na Seção 3, são apresentados os estudos comparativos analisando a variações dos conjunto de dados de entrada e abordagens utilizadas na ordenação. Por fim, na Seção 4, são apresentadas as considerações finais do estudo.

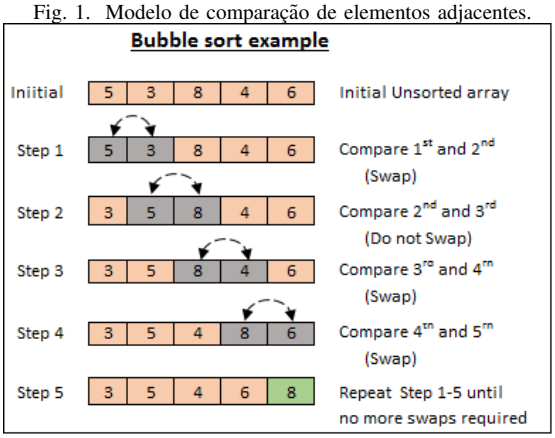
II. Algoritmos de Ordenação

A. Algoritmos baseados em Troca

1) Bubble Sort: O presente algoritmo utilizada a abordagem de troca, através da permutação de elementos vizinhos, seguindo a ideia de densidade dos elementos, em que pode-se optar por varrer o arranjo levando o maior elemento (mais pesado) ao fim do vetor, ou conduzir o menor elemento (mais leve) ao início desse. Esse passo (uma das duas opções excludentemente) é realizado sucessivamente para os subvetores não ordenados remanescentes até que o vetor esteja ordenado como um todo.

A abordagem original prevê que todos os elementos adjacentes sejam comparados através de $n-1$ iterações, porém é possível que o arranjo esteja ordenado sem que sejam necessárias todos esses ciclos. De modo a prevenir operações desnecessárias, algumas abordagens utilizam-se de *flags* para a detecção de trocas, caso a última iteração tenha ao menos uma ocorrência, há necessidade de pelo menos mais uma iteração, em caso negativo, o processo pode ser encerrado.

Fig. 1. Modelo de comparação de elementos adjacentes.



A Figura 1 apresenta a abordagem de ordenação utilizada pelo algoritmo referido.

Complexidade:

- **Melhor caso:** $O(n)$, para vetor crescente somente na implementação com melhoria.
- **Caso médio:** $O(n^2)$.
- **Pior caso:** $O(n^2)$, para vetor decrescente e aleatório.

2) Quick Sort: O algoritmo foi proposto por C.A.R. Hoare em 1962, tem como estratégia a divisão do arranjo original em partições a partir da determinação de um pivô. Para qualquer abordagem de escolha do pivô, em cada partição busca-se encontrar os elementos maiores que o pivô a partir do início e os menores a partir do fim do arranjo, os quais são trocados de posição aos pares, de modo a ordenar a partição, se necessário também através da quebra de subpartições.

Em relação a abordagem de escolha de pivô, há diversas técnicas que buscam explorar possíveis características dos elementos do arranjo. Dentre as principais, pode-se destacar a escolha do elemento inicial ou final do vetor, o uso de propriedades estatísticas como a média e mediana, entre outras. O principal objetivo de qualquer uma dessas estratégias é minimizar a ocorrência dos piores casos de recorrência, resultando em um comportamento assintótico quadrático.

Geralmente, os piores caso desse algoritmo consistem na escolha de pivô como elemento mínimo ou máximo, para entradas de dados crescentes e decrescentes.

Complexidade:

- **Melhor caso:** $O(nlogn)$.
- **Caso médio:** $O(nlogn)$.
- **Pior caso:** $O(n^2)$, em geral para o pivô mínimo e máximo.

B. Algoritmos baseados em Inserção

1) Insertion Sort: A estratégia de ordenação por inserção consiste em um dos métodos mais simples, sendo extremamente eficiente em conjuntos pequenos, com estratégia de percorrer o esquerdo da esquerda para a direita deixando os elementos ordenados à esquerda. Uma situação cotidiana que aplica a abordagem referida é a inserção de cartas na mão de um jogador, seguindo a estrutura de dados deque.

Complexidade:

- **Melhor caso:** $O(n)$, para arranjo crescente.
- **Caso médio:** $O(n^2)$.
- **Pior caso:** $O(n^2)$.

2) Shell Sort: O algoritmo Shell Sort foi proposto por Donald Shell em 1959, consistindo em um dos métodos de ordenação mais eficientes dentre os modelos quadráticos, baseando-se no Insertion Sort, a partir de comparações com elementos não adjacentes, desse modo, facilitando o deslocamento dos menores elementos para o início do vetor através do uso de saltos regressivos de tamanho.

O grande diferencial desse método é a utilização dos saltos, porém não existe uma abordagem única para o cálculo do tamanho dos saltos, consistindo em um fator determinante na mensuração da complexidade assintótica. Dentre os principais modelos têm-se o n primeiros múltiplos de um fator ou combinação de fatores (ex. 2^p3^q) ou os n primeiros números primos.

A Figura 2 apresenta a ordenação das partições de elementos distantes, com saltos de tamanho 40, 13 e 4 respectivamente, partindo do arranjo inicial e alcançando o conjunto ordenado.

Complexidade:

- **Caso médio:** $O(n^{3/2})$

C. Algoritmos baseados em Seleção

1) Selection Sort: O algoritmo Selection Sort, trata-se modelo mais básico de algoritmo de ordenação, utilizada

Fig. 2. Modelo de partições com elementos distantes em diferentes tamanhos.

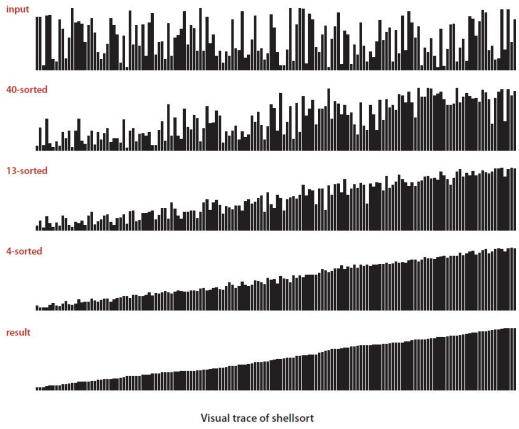
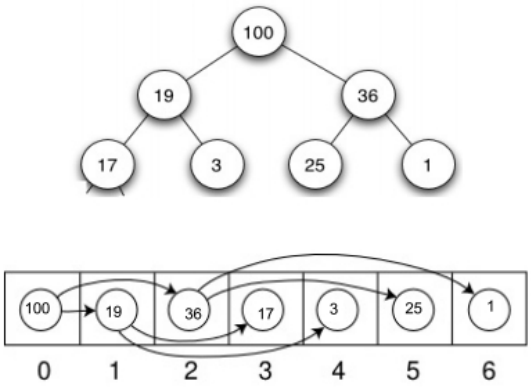


Fig. 3. Equivalência entre a árvore e arranjo.



muitas em ambientes naturais e é mais intuitivo para os seres humanos, pois se utiliza da abordagem de força bruta, sem aproveitar nenhuma peculiaridade do conjunto de dados ou vantagem que possa ser tomada. Sua estratégia consiste em percorrer todo o arranjo n, comparando todos os elementos de forma a encontrar o menor, em seguida, o segundo menor e assim por diante, de modo a organizar todo o conjunto.

O grande destaque para esse algoritmo, dentro do contexto das abordagens quadráticas, consiste no reduzido número de trocas, que no máximo ocorre n vezes.

Complexidade:

- **Caso médio:** $O(n^2)$, de modo invariante.

2) Heap Sort: O algoritmo Heap Sort, proposto por J.W.J. Williams em 1964, possui uma das implementações mais sofisticadas baseando-se em uma fila de prioridades, através da utilização de um vetor para a representação de uma árvore binária, caracterizando como um dos algoritmos mais estáveis de ordenação, independentemente do modelo de dados de entrada.

A Figura 3 apresenta o processo de equivalência entre a estrutura de dados Heap na forma de árvore e arranjo.

O algoritmo consiste na construção de um *max heap* (cuja principal propriedade consiste em que sempre o elemento raiz é maior que seus filhos, para todas as sub-árvores possíveis), em seguida, aplica-se a troca do primeiro elemento pelo último para todos os elementos do conjunto (equivalente a remover todos os elementos do arranjo), e posteriormente, rearranjar a estrutura para cada elemento, desse modo é obtido um vetor ordenado de modo crescente ao termino do processo.

Complexidade:

- **Caso médio:** $O(nlogn)$, de modo invariante.

D. Algoritmos baseados em Intercalação

1) Merge Sort: Este algoritmo utiliza o princípio de divisão e conquista, de modo a quebrar um problema complexo em sub-problemas, até o caso base (restante um ou dois elementos no sub-vetor), de modo que seja possível ordená-los. A partir da garantia de que todos os sub-arranjos estão ordenadas, inicia-se o processo de combinação entre eles, de modo recursivo, até alcançar o problema original, de modo ordenado.

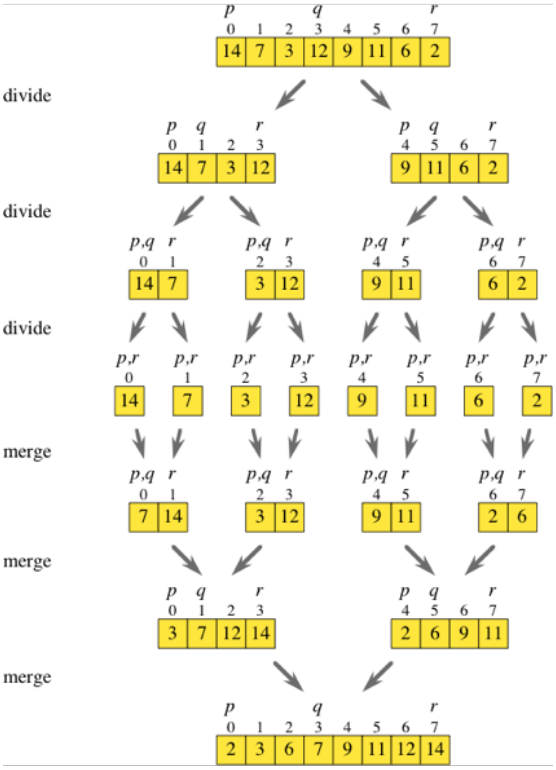
As posições situações de ocorrência são as seguintes:

- em seu seu melhor caso nunca é necessário realizar trocas após comparações;
- o caso médio ocorre quando nem sempre é necessário realizar trocas após comparações;
- por fim, o pior caso ocorre quando sempre é necessário realizar trocas após comparações.

Contudo, de modo invariável, todos os casos tem a mesma complexidade assintótica. Adicionalmente, vale-se destacar que diferentemente dos demais algoritmos que não se utilizam de memória adicional, o algoritmo Merge Sort tem complexidade espacial de tamanho n, que em dadas situações, deve ser levada em consideração.

A Figura 4 apresenta o processo de divisão e conquista utilizado no algoritmo.

Fig. 4. Abordagem de divisão e conquista do algoritmo.



Complexidade:

- **Caso médio:** $O(nlogn)$, de modo invariante.

III. Análise Experimental

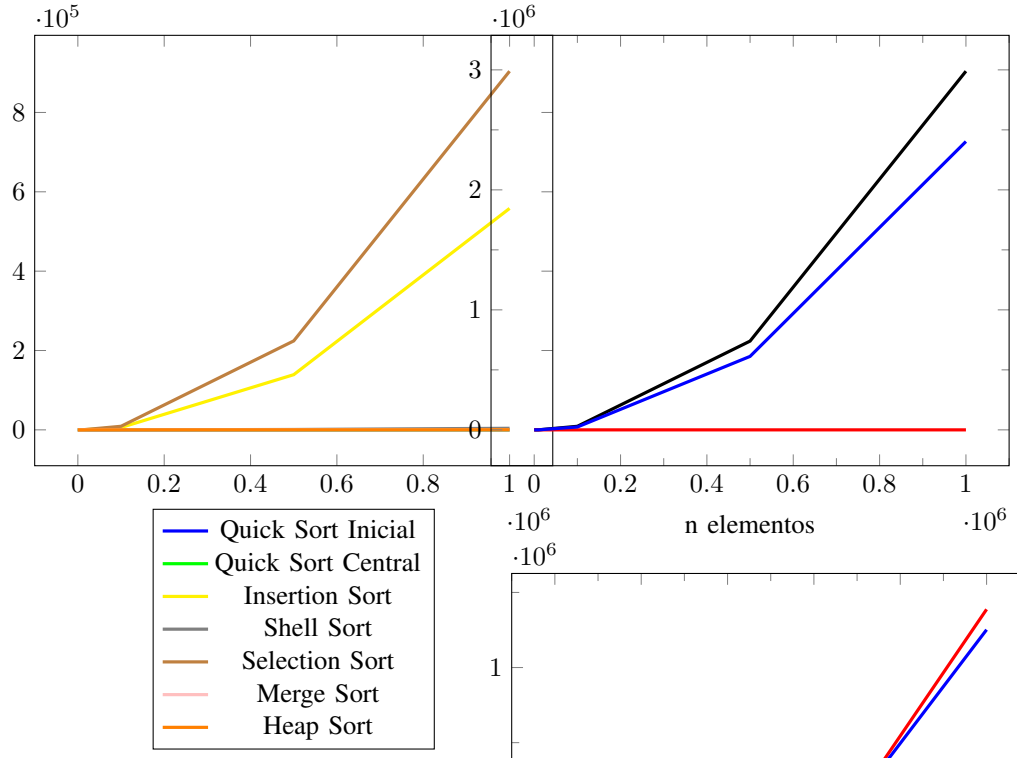
De modo a prover uma análise experimental em relação aos algoritmos, foram conduzidos estudos com diferentes tamanhos de entrada, cujas dimensões foram: 10, 100, 1 000, 10 000, 100 000 e 1 000 0000 de elementos, em que para todos os casos foram utilizados como teste um vetor crescente, decrescente e elementos dispersos pelo vetor de forma aleatória (vale ressaltar que para este último foi utilizado um arquivo que armazena os elementos, para que esta sequência aleatória seja sempre a mesma para execução em todos os algoritmos). Todos os casos foram avaliados em relação ao tempo medido em milissegundos (ms).

Para a realização da análise experimental foi utilizado um notebook com as seguintes especificações: processador Intel Core i7-7700HQ CPU @ 2.80 GHz de 4 núcleos físicos e 8 threads, com memória caches L1 de 256 KB, L2 de 1 MB, L3 de 6 MB e memória RAM de 8 GB DDR4.

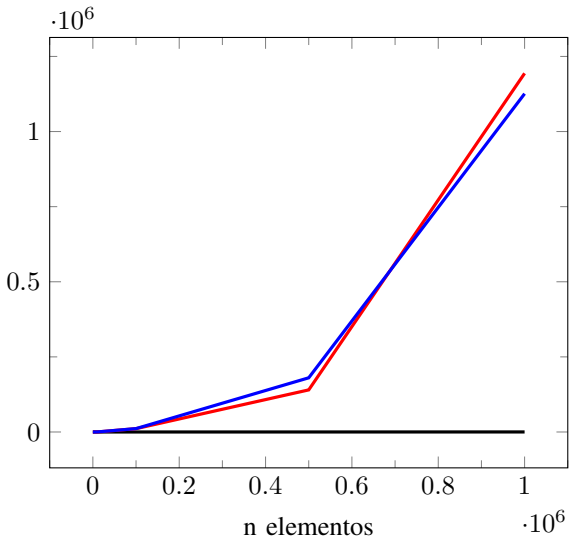
A. Modelos de entrada de dados

1) Entrada crescente:

2) Entrada decrescente:



3) Entrada aleatória:

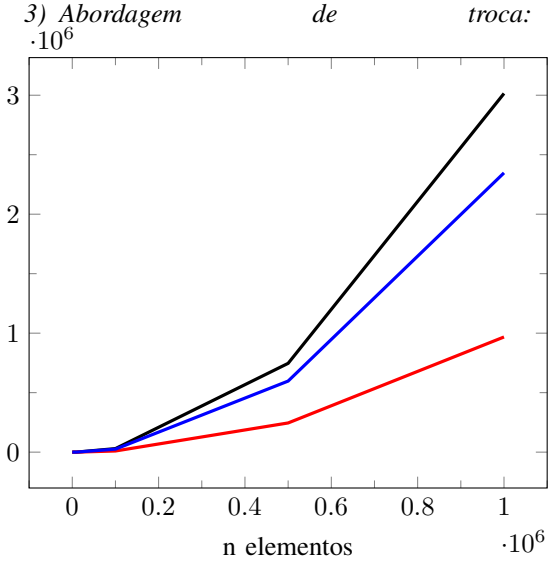


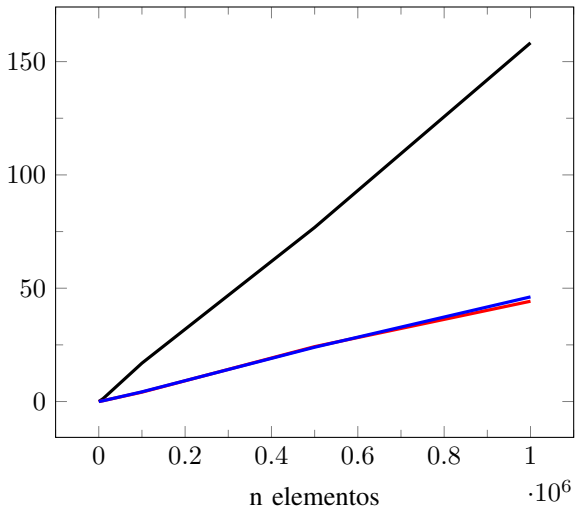
B. Comparativo de abordagem

1) Abordagem de inserção:

2) Abordagem de seleção:

3) Abordagem de troca:

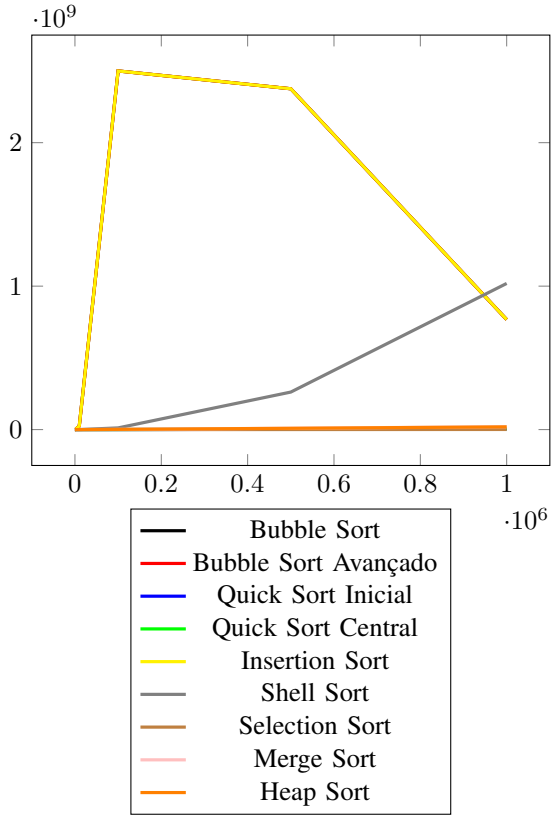


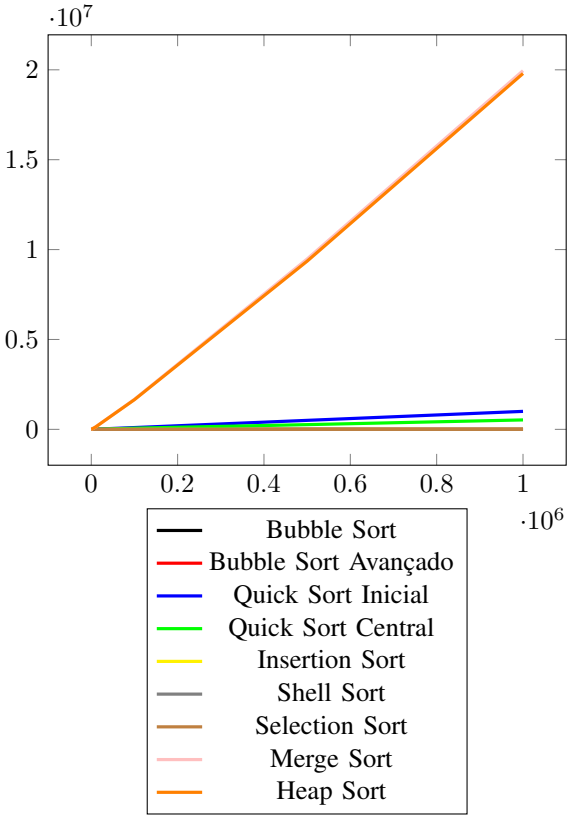


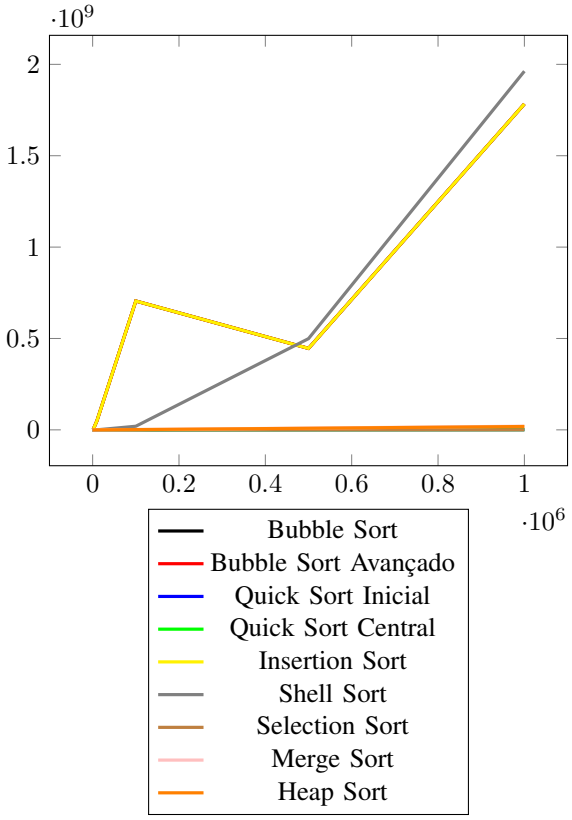
IV. Análise quantidade de trocas realizadas

Além da análise experimental que visa comparar os tempos de execução entre os diversos algoritmos de ordenação, também foi aplicada uma análise com objetivo de comparar a quantidade de trocas de posições de elementos durante uma

ordenação. Foi medido durante a execução dos algoritmos a quantidade máxima de troca que eles realizam de acordo com os três tipos de entrada (aleatória, crescente e decrescente) em diferentes dimensões.



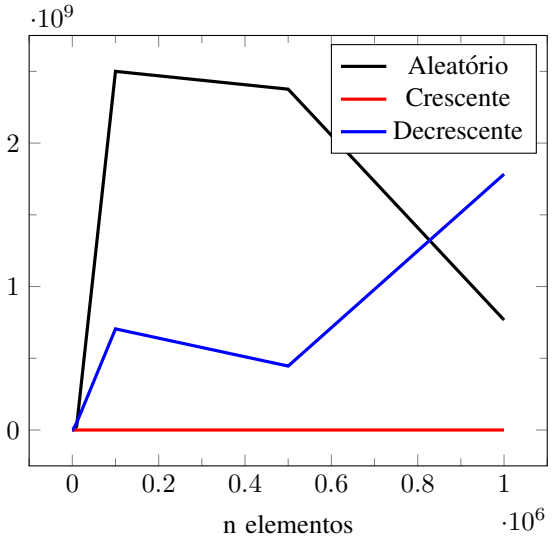




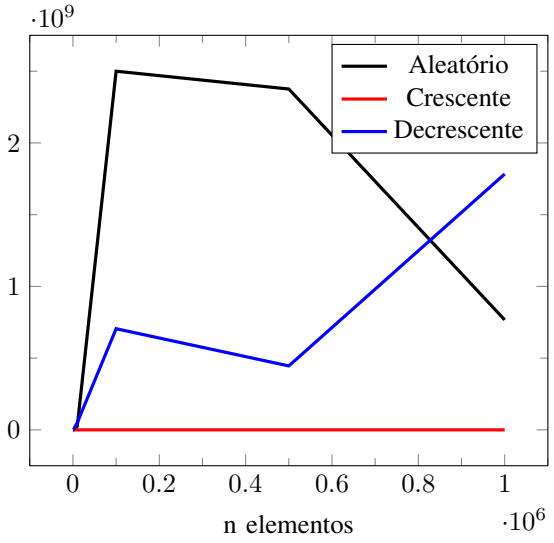
A. *Análise número de trocas por algoritmo*

Após observar o comportamento do número de trocas com uma visão de todos os algoritmos juntos, agora é exposto esse comportamento considerando cada algoritmo individualmente, afim de perceber as diferenças existentes nos três tipos distintos de entradas (aleatório, crescente e decrescente).

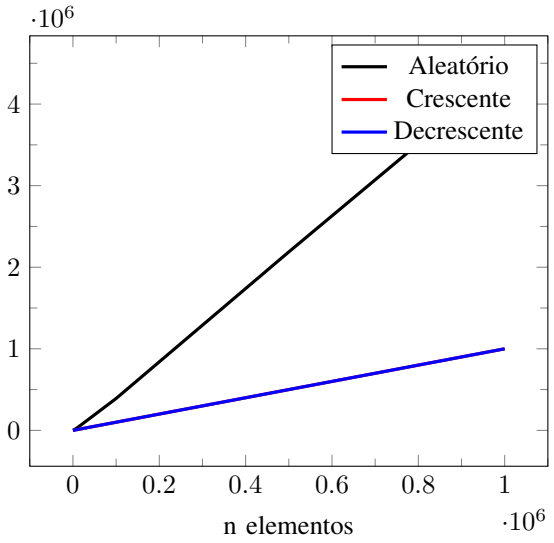
1) Bubble Sort Clássico:



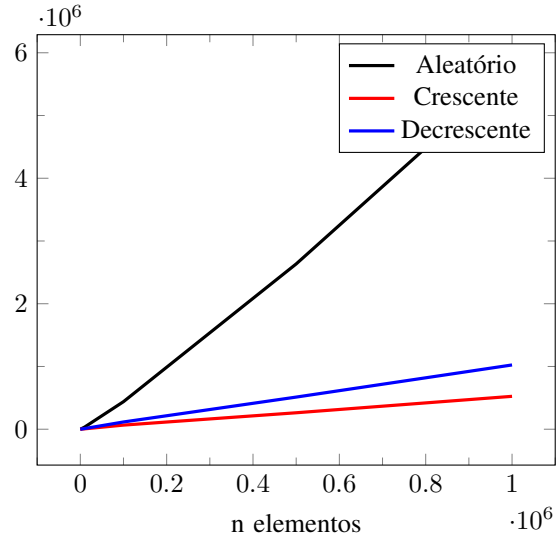
2) Bubble Sort Melhorado:



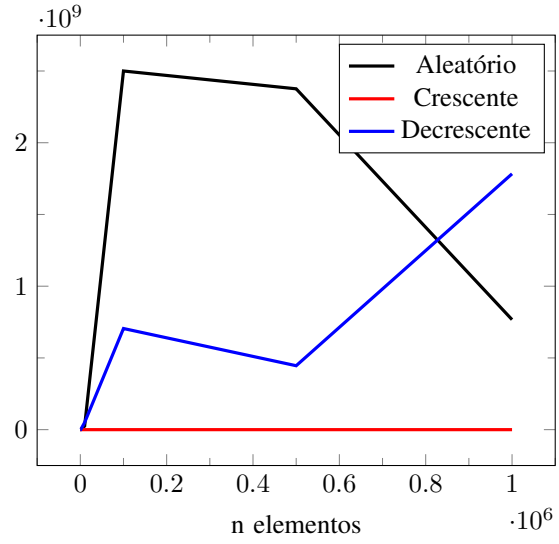
3) Quick Sort Pivô inicial: O algoritmo Quick Sort com implementação colocando sempre o pivô como ponto inicial tem números de trocas que apresentam comportamento linear. O maior número de trocas ocorre quando a entrada está organizada em ordem aleatória, entretanto os casos em que a entrada está em ordem crescente ou decrescente a quantidade de trocas é a mesma para essas duas entradas, vale ressaltar ainda que a quantidade de trocas na entrada em ordem aleatória é bem maior que a quantidade de trocas nas duas outras configurações de entrada. Observe o Gráfico IV-A3



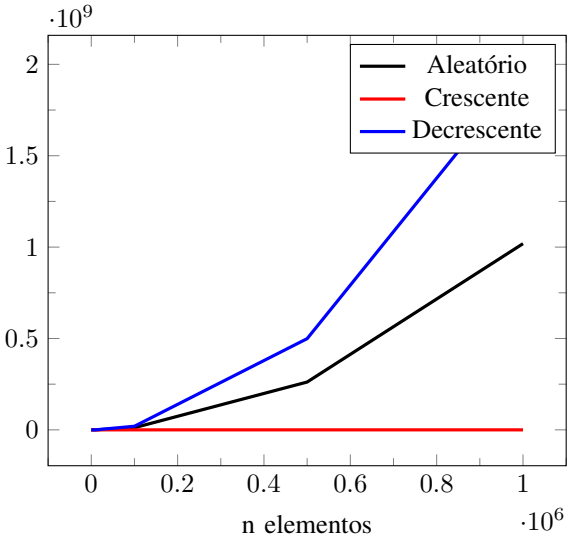
4) Quick Sort Pivô Central: Quando o algoritmo Quick Sort é implementado com o pivo sendo sempre o elemento central o comportamento de número de trocas é diferente a implementação com pivô sendo sempre o inicial. Neste caso, o maior número de trocas absoluto continua sendo para a entrada organizada de modo aleatório, contudo a diferença entra a implementação discutida anteriormente está que o número de trocas para entradas em ordem crescente e decrescente é diferente, tendo maior quantidade de trocas para a decrescente e menor para a entrada em ordem crescente. Observe o Gráfico IV-A4.



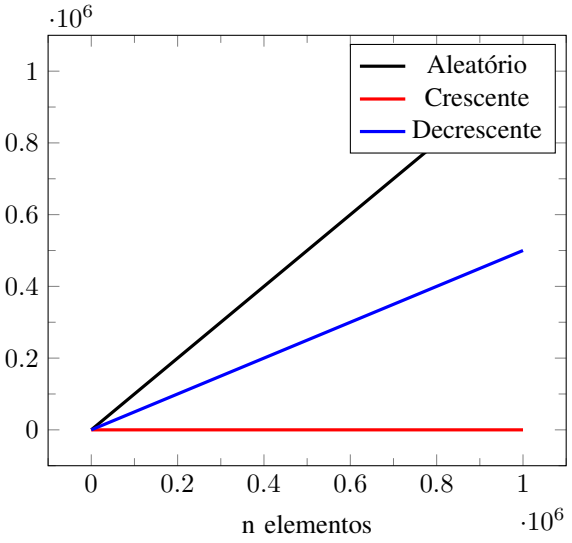
5) Insertion Sort:



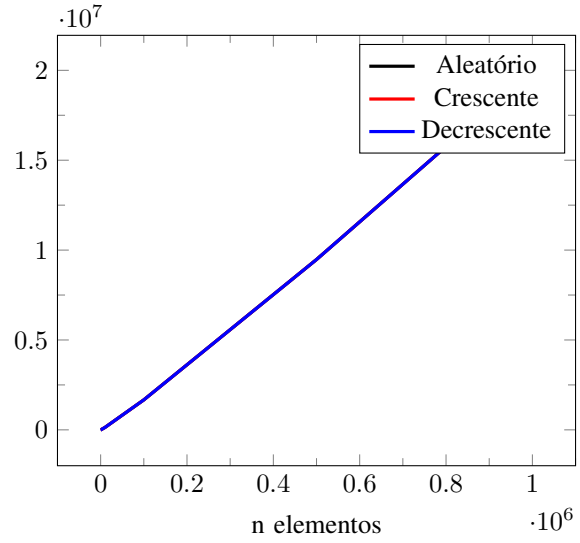
6) Shell Sort: O Shell Sort não realiza nenhuma troca quando a entrada de dados está em ordem decrescente, a verificação presente no corpo do algoritmo garante isso. Contudo, para os outros demais casos existem uma quantidade de trocas significativa de comportamento não linear, porem quanto maior a dimensão da entra maior o número de trocas. A maior quantidade de trocas ocorre com a entrada em ordem decrescente e a segunda maior em ordem aleatória. Observe o Gráfico IV-A6.



7) Selection Sort: No algoritmo Selection Sort todas as quantidades de trocas tem valores bem espaçados entre as três configurações de entrada. Neste caso, a que menos realiza trocas são quando a entrada está em ordem crescente, neste caso, nenhuma troca é realizada, pois, o algoritmo verifica que os elementos já estão em ordem correta. Já nas duas demais configurações de entrada ocorrem um número de trocas acentuado, sendo o maior deles na entrada em ordem aleatória e a segundo maior na entrada em ordem decrescente. Observe o Gráfico IV-A7.

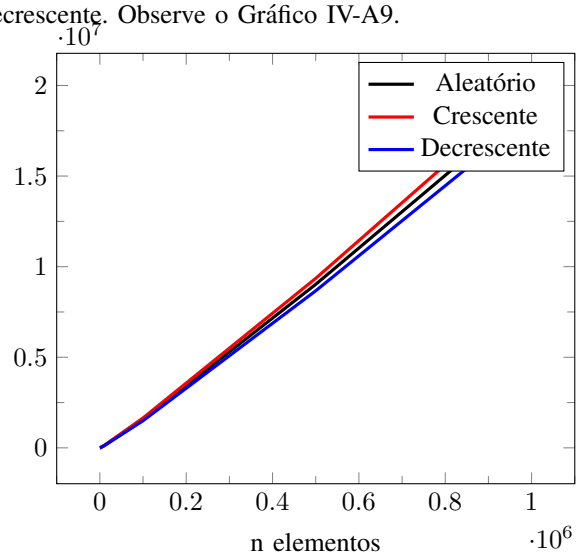


8) Merge Sort: O algoritmo Merge Sort independentemente da configuração de entrada sempre irá realizar o mesmo número de trocas em seu processo de ordenação. Assim, independente se a entrada contem elementos organizados em ordem aleatória, crescente ou decrescente, o número de troca será o mesmo. Vale ressaltar ainda que conforme a quantidade de elementos aumenta, maior será o número de trocas formando um comportamento linear. Observe o Gráfico IV-A8.



9) Heap Sort: No caso do algoritmo Heap Sort, é constatado também que conforme o número de elementos a serem ordenados aumentam maior será o número de trocas em todas as configurações de entrada, formando um comportamento bem próximo ao linear.

Contudo, existe uma mínima diferença entre a quantidade de trocas em cada configuração de entrada. A menor quantidade de trocas acontece quando a entrada de dados são elementos em ordem decrescente, já a maior quantidade de trocas ocorre quando os elementos está dispostos em ordem aleatória, caso os elementos estejam em ordem aleatória a quantidade de trocas é menor que a crescente e maior que a de ordem decrescente. Observe o Gráfico IV-A9.



V. Considerações Finais

REFERÊNCIAS

[1] Programa das Nações Unidas para o Desenvolvimento (PNUD). *Atlas do desenvolvimento humano do Brasil*. PNUD; 2003. Disponível em: http://www.pnud.org.br/atlas/

[2] Sen AK. *Desenvolvimento como liberdade*. São Paulo: Companhia das Letras; 2000.