Análise de Desempenho de Algoritmos de Ordenação

Rafael Passos Domingues

I. Introdução

A ordenação numa aplicação computacional é o processo de organizar dados numa determinada ordem. Atualmente, vivemos uma era de grandes volumes de dados e, para isso, a ordenação é essencial, especialmente se tratando de processamento de dados em tempo real.

A ordenação melhora a eficiência da pesquisa de dados específicos dados no computador. Há muitas técnicas sofisticadas para ordenação: Nesta análise, serão investigados os algoritmos de ordenação Bubble Sort, Selection Sort e Insertion Sort aplicados a três distindos arranjos de vetores: em ordem crescente, decrescente e aleatório. O desempenho desses algoritmos será avaliado com base no número de iterações. No algoritmo foram implementados contadores a cada troca de endereçamento dos elementos, variando o tamanho dos vetores de 100 a 10000, de 100 em 100.

II. Referencial Teórico

Incluir aqui o referencial teórico relevante para a análise de desempenho de algoritmos de ordenação. Cite artigos, livros ou outras fontes científicas que abordem o assunto.

III. Material Utilizado

Para realizar esta análise, foram utilizados dois computadores: um computador pessoal com sistema operacional Linux e SSD de 256 GBytes e um dos computadores no laboratório do BCC-UNIFAL com sistema operacional Linux, porém com HD. O código foi implementado em C++ e compilado com o compilador g++, também foi utilizado o ambiente NetBeans e a IDE VSCode.

IV. Métodos Implementados

O código em questão implementa os seguintes métodos:

- Inclusão das bibliotecas necessárias: `iostream`, `fstream` e `ctime`.
- Definição das funções de classificação: `bubbleSort`, `insertionSort` e `selectionSort`.
- Definição da função `randomArrayGenerator` para gerar um array aleatório.
- Definição da função `copyArray` para reconstruir os vetores originais.
- Definição da função `saveResultsToFile` para salvar os resultados em um arquivo CSV.
- Implementação da função principal (`main`) que realiza as seguintes ações:
  1) Definição de constantes para determinar o tamanho inicial, tamanho final e incremento.
  2) Declaração dos arrays e variáveis necessários para armazenar os resultados.
  3) Preenchimento do array de tamanhos com valores incrementais.
  4) Execução dos algoritmos de classificação para diferentes tamanhos de arrays (aleatório, crescente e decrescente).
  5) Armazenamento do número de utilizações realizadas em cada algoritmo de classificação.
  6) Chamada da função `saveResultsToFile` para salvar os resultados em um arquivo CSV chamado ”sort_usages.csv”.
  7) Retorno de 0 para indicar que o programa foi executado com sucesso.

V. Resultados Obtidos

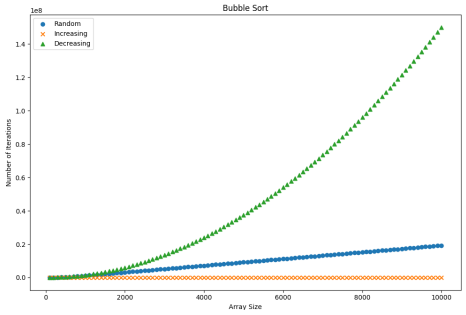

Fig. 1. Número de utilizações do Bubble Sort para diferentes tamanhos de arrays.

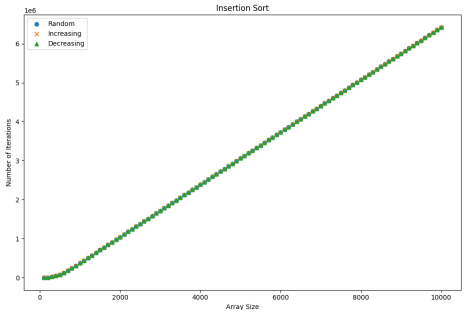

Fig. 2. Número de utilizações do Insertion Sort para diferentes tamanhos de arrays.

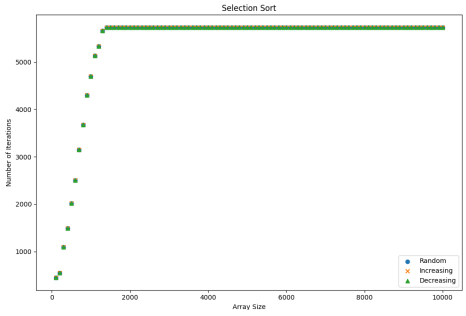

Fig. 3. Número de utilizações do Selection Sort para diferentes tamanhos de arrays.

Os resultados da tabela mostram o número de utilizações de cada algoritmo de ordenação para diferentes tamanhos de arrays. Observa-se que o Bubble Sort tem a maior contagem de utilizações, seguido pelo Insertion Sort e pelo Selection Sort.

VI. Conclusão

A escolha de uma determinada técnica de ordenação depende do tipo de dados: Com base nos resultados obtidos, conclui-se que o método de ordenação por seleção se mostrou o menos performático, pois o número de iterações cresce exponencialmente e estabiliza em um grande número de iterações. O método de inserção é mais linear com o aumento do tamanho do vetor, além de apresentar um comportamento semelhante aos distintos arranjos de vetores. O método *Bubble Sort* se mostrou mais performático com vetores em ordem decrescente, apresentando uma performance razoável com vetores aleatórios e diminuindo a performance em uma relação tamanho ao quadrado do vetor crescente.

VII. Referências Bibliográficas

1) Al-Kharabsheh, K. S., AlTurani, I. M., AlTurani, A. M. I., & Zanoon, N. I. (2013). Review on sorting algorithms a comparative study. International Journal of Computer Science and Security (IJCSS), 7(3), 120-126.
2) Prajapati, P., Bhatt, N., & Bhatt, N. (2017). Performance comparison of different sorting algorithms. vol. VI, no. Vi, 39-41.
3) Friedman, J. H., Bentley, J. L., & Finkel, R. A. (1977). An algorithm for finding best matches in logarithmic expected time. ACM Transactions on Mathematical Software (TOMS), 3(3), 209-226.