GOVERNO DO ESTADO DO PIAUÍ
UNIVERSIDADE ESTADUAL DO PIAUÍ – UESPI
CAMPUS PROFESSOR ANTONIO GEOVANNE ALVES DE SOUSA CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

DISCIPLINA: ESTRUTURA DE DADOS
PROF.: FÁBBIO ANDERSON SILVA BORGES
ALUNO: MARIA CLARA DE ARAÚJO PEREIRA; MOISÉS CUNHA PIMENTEL

Relatório Bubble Sort

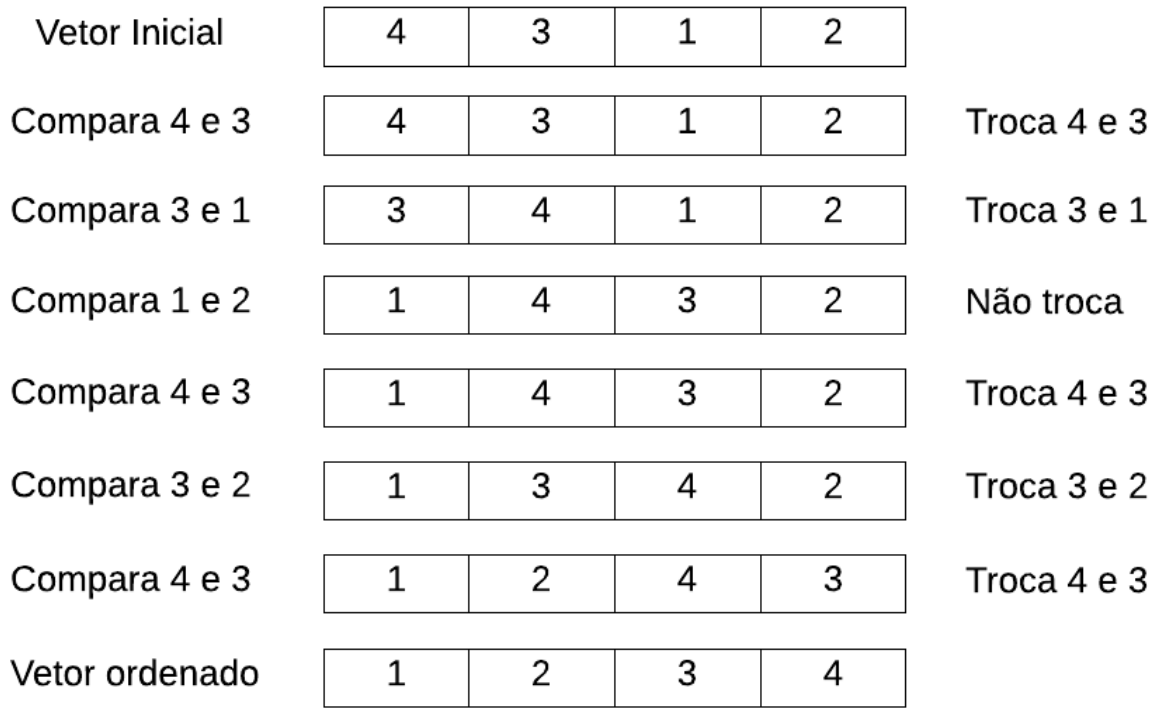
2023

1. Introdução

O *Bubble sort* é um dos algoritmos de ordenação mais conhecido, além de ter uma simples implementação. No entanto, tem maior tempo de execução em comparação com outros algoritmos de ordenação. Este trabalho tem como objetivo implementar e realizar testes para obter o tempo de execução utilizando o Bubble sort para ordenar vetores de quatro tamanhos distintos (100, 1000, 10000 e 100000) e três formas de inserção de valores (ordenado, inversamente ordenado e aleatório).

2. Implementação

O código tem uma classe *bubble.c* que irá implementar o *Bubble sort*, além de retornar a quantidade de comparações e movimentações feita em cada vetor testado. A classe *bubble_test.c* criará os três modos de organização dos vetores, como também o método que irá retornar o tempo de execução de cada teste em segundos. A estrutura de dados utilizada foi um vetor de inteiros que está exemplificado no diagrama abaixo, além do funcionamento do algoritmo:



A entrada dos dados é feita através do terminal, onde o usuário pode optar pelo tamanho do vetor dentre os quatro tamanhos pré definidos, além da escolha do tipo de ordenação que será usada. A saída será o tempo de execução (em segundos) do algoritmo e a quantidade de comparações e movimentações feitas durante a execução do método de ordenação.

3. Listagem de testes executados

Cada teste foi realizado 10 vezes para obter a média do tempo de execução com os diferentes tamanhos e modos de organização dos dados. Veja a tabela à seguir:

TIPO DE ORDENAÇÃO	TAMANHO DO VETOR	QUANTIDADE DE TESTES	MÉDIA DO TEMPO DOS TESTES (segundos)
Ordenado	100	10	0,000009
Inversamente Ordenado	100	10	0,000024
Aleatório	100	10	0,000028
Ordenado	1000	10	0,000912
Inversamente Ordenado	1000	10	0,002265
Aleatório	1000	10	0,003363
Ordenado	10000	10	0,084789
Inversamente Ordenado	10000	10	0,181970
Aleatório	10000	10	0,251101
Ordenado	100000	10	9,068223
Inversamente Ordenado	100000	10	21,836346
Aleatório	100000	10	31,969803

4. Conclusão

A implementação desse algoritmo foi fácil, basicamente sendo formado por dois laços de repetições aninhados com uma estrutura condicional e um *swap* dos elementos. Após a realização dos testes, foi observado que o algoritmo terá melhor tempo de execução quando o vetor já está ordenado, e pior tempo com o vetor aleatório. O algoritmo possui uma alta complexidade de tempo, tornando-o ineficiente em vetores com grandes volumes de dados.

5. Bibliografia

GEEKSFORGEEKS. How to measure time taken by a function in C?. Disponível em: https://www.geeksforgeeks.org/how-to-measure-time-taken-by-a-program-in-c/. Acesso em: 13 nov. 2023.

GEEKSFORGEEKS. Generating random number in a range in C. Disponível em: https://www.geeksforgeeks.org/generating-random-number-range-c/. Acesso em: 15 nov. 2023.