Relatório de CCI

Aluno: Henrique Fernandes Feitosa

1. Implementação do produto interno entre dois vetores

Inicialmente, implementou-se o código de duas formas: usando as operações nativas da linguagem e usando laços de repetição. A figura 1 mostra uma foto das duas implementações.

```
for n=1:1000
    tic;
    x*y';
    t1(n)=toc;
end
for n=1:1000
    % criando a variável que vai armazenar o valor do produto escalar
    soma=0
    tic
    for w=1:10000000
        soma=soma+x(w)*y(w);
    end
    t2(n)=toc
end
```

Figura 1: Mostra a foto da implementação usando laços de repetição e da implementação usando operações nativas.

Após realizar as mil operações para cada forma de calcular, fez-se o gráfico do tempo gasto pra realizar a operação pelo número da interação. Para diminuir o ruído da informação, o elemento de posição i no vetor recebeu a média dos elementos da posição 1 até a posição i, assim como mostra a figura 2.

```
for n=1:1000
    n1(n)=mean(t1(1:n));
    n2(n)=mean(t2(1:n));
end
```

Figura 2: Mostra a manipulação feita com os valores de tempo para mil interações com o intuito de diminuir o ruído da informação.

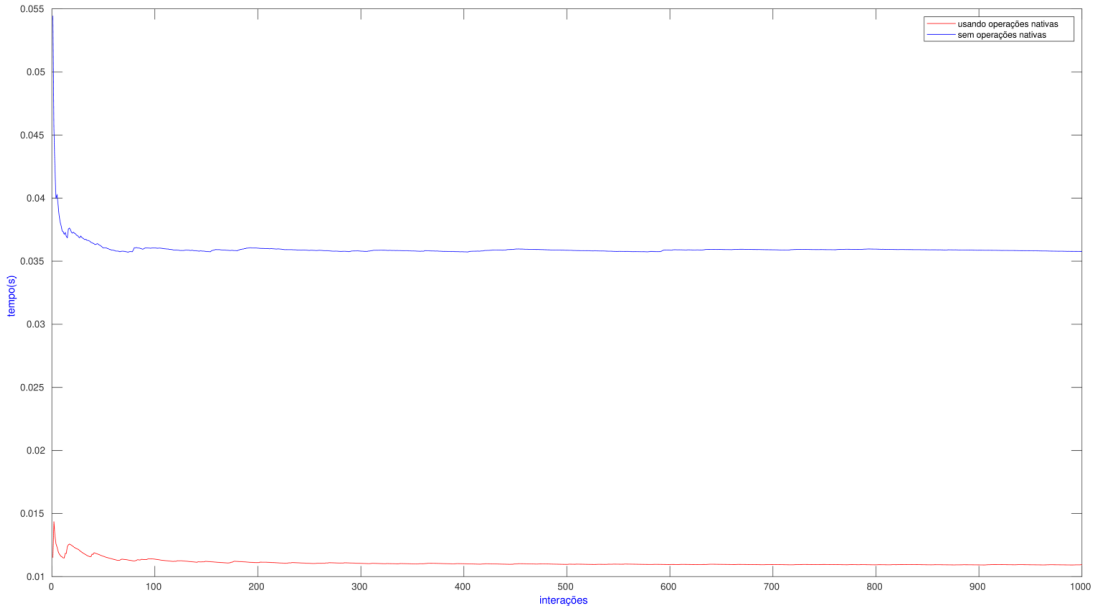

Figura 2: Gráfico mostrando o tempo gasto para fazer a operação do produto interno de duas maneiras diferentes.

Em seguida, fez-se o gráfico que está mostrado na figura 3. Por fim, pode-se perceber que o cálculo do produto interno usando operações nativas são muito mais eficientes e demoram uma quantidade de tempo significantemente menor do que a implementação usando laços de repetição.

2. Implementação da multiplicação de matrizes usando MATLAB e C++

Nesse tópico, implementou-se a multiplicação de matrizes de três formas diferentes: Usando laços de repetição no MATLAB, usando C++ e usando as operações nativas do MATLAB. Segue nas figuras 3, 4 e 5 fotos das respectivas implementações.

```
for i=1:n
    for j=1:n
        for k=1:n
            z(i,j)=z(i,j)+x(i,k)*y(k,j);
        end
    end
end
tempo=toc;
```

Figura 3: Foto da implementação usando laços de repetição no MATLAB

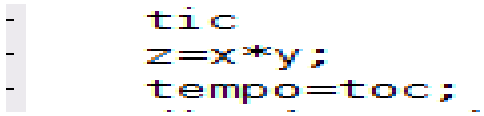

Figura 4: Foto da implementação usando operações nativas do MATLAB

```
for(int i =0; i < n; ++i)
        for(int j = 0 ; j< n; ++j)
                for(int k=0; k < n; ++k)
                        z[i][j]=z[i][j]+x[i][k]*y[k][j];
```

Figura 5: Foto da implementação usando laços de repetição no C++

Depois de implementação, realizou-se cada operação com matrizes quadradas de ordem 10, 100 e 1000, medindo o tempo gasto para a realização da multiplicação em cada implementação. Na tabela 1, mostram-se os tempos gastos para cada implementação e para cada tamanho da matriz.

Tabela 1: Tabela que mostra os tempos gastos para cada implementação e para cada tamanho da matriz

Implementação/ Ordem da matriz	10	100	1000
Operações nativas do MATLAB	$1{,}03 \times 10^{-4}$ s	0,010 s	0,11 s
Laços no MATLAB	0,0052 s	0,0481 s	5,93 s
Laços no C++	$1{,}9 \times 10^{-5}$ s	0,0174 s	5,66 s

Assim, observa-se , que , em geral, as operações nativas do MATLAB são mais rápidas e eficientes do que as operações usando laços em C++, e essas operações são bem mais rápidas do que as que usam laços de repetição no MATLAB.

3. Comparação do bubblesort, mergesort e a operação nativa sort usando MATLAB

Nesse tópico, implementou-se dois algoritmos de ordenação: bubblesort e merge sort. Após a implementação, o programa foi testado para vetores aleatórios que variavam o tamanho de 1000 até 10000, com o passo de 10 e os tempos foram anotados. Por fim, fez-se um gráfico do tempo pelo tamanho do vetor, esse gráfico é apresentado na figura 6.

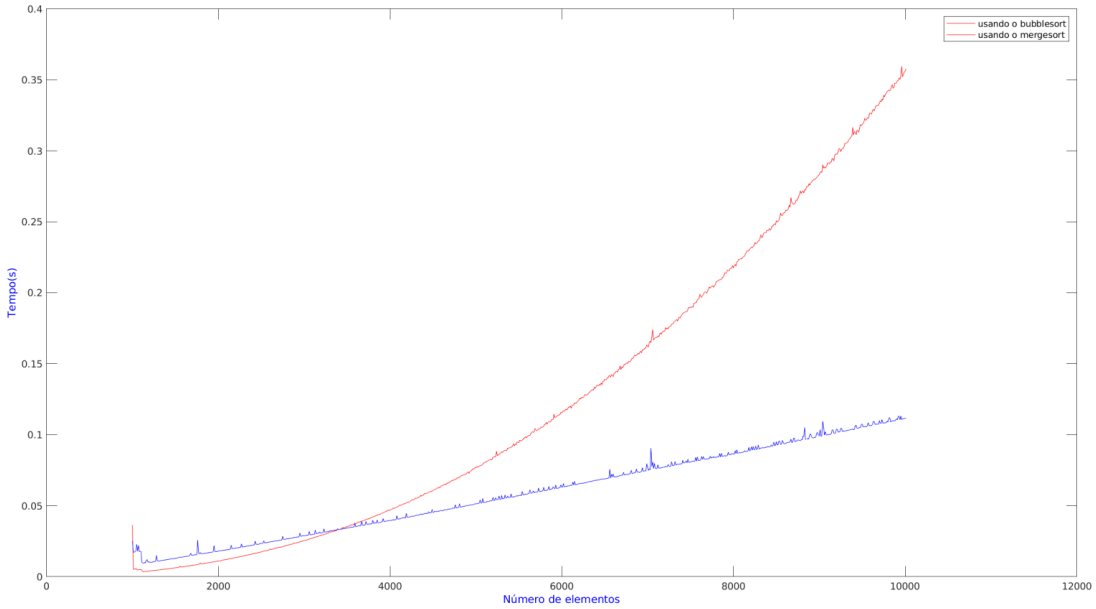

Figura 6: Gráfico das implementações do mergesort e do bubble sort

Assim, tivemos o resultado como esperado. No início, o bubble sort e melhor que o bubble sort e quando o número de elementos cresce, o merge sort fica mais eficiente, isso com firma as complexidades dos algoritmos que são $O(n^2)$ e de $O(nlog(n))$ respectivamente.

Por fim, fez-se uma comparação com a função sort, que é uma função nativa do MATLAB. O gráfico dos tempos encontra-se na figura 7.

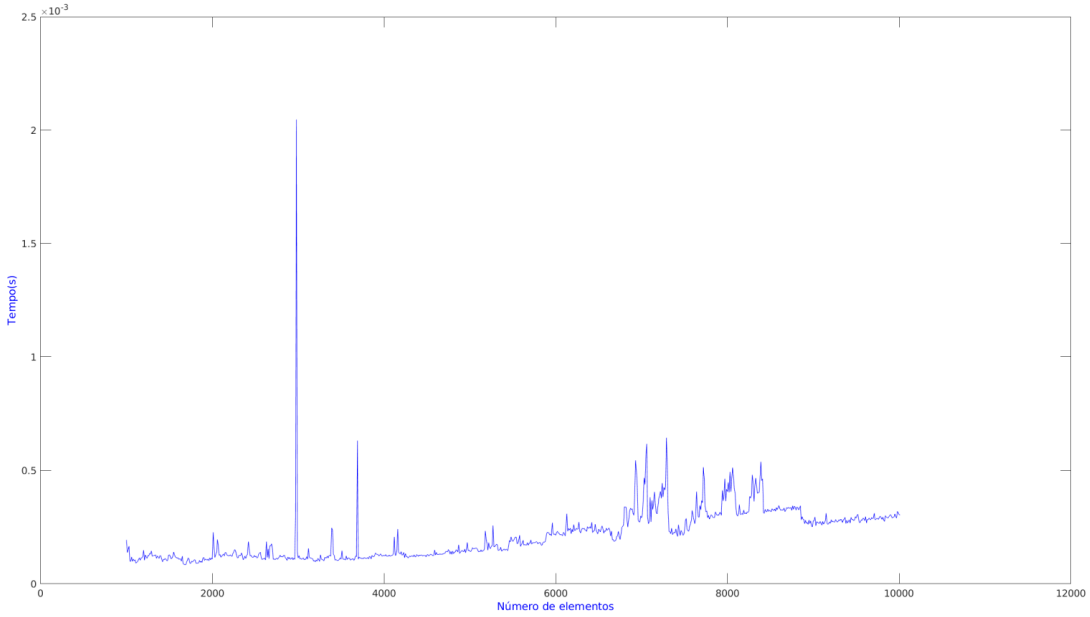

Figura 7: Gráfico das implementação usando a função nativa sort

Assim, percebe-se que a função sort faz a ordenação em um tempo consideravelmente menor, aproximadamente na ordem de milhares de vezes.