Relatório Merge Sort e Gnome Sort

ferraz.freire

November 2024

1 Introdução

Este é um trabalho onde foi feito um código que contém a aplicação de dois algoritmos de ordenação, sendo eles Merge Sort e Gnome Sort. Ambos os algoritmos foram colocados em situações onde poderemos ver como eles irão desempenhar com vetores de tamanhos específicos.

2 Gnome Sort

O Gnome Sort compara os elementos de um array passando por ele todo, porém quando encontra um elemento fora de posição, ele o leva até sua posição correta. Pode ser bem eficiente quando se trata de arrays pequenos, porém em arrays grandes se torna extremamente lento. Segue um exemplo abaixo:

Aqui será quando ele estiver na posição 0 e irá seguir até achar algo fora de ordem no array. Na posição 2, ele acaba por achar o 4 fora de ordem, então o leva até a posição 3 onde ficará atrás do 1, pois é menor que ele. O mesmo acontece agora com o 3, por ser maior que o 1 é levado para uma posição onde ficará de forma correta, entre o 1 e o 4. E acontece o mesmo com o 2 e assim irá seguir até que todos os números estejam ordenados na ordem numérica de menor para o maior.

Figure 1: Passo 1 do Gnome Sort

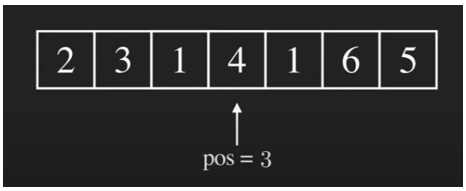


Figure 2: Passo 2 do Gnome Sort

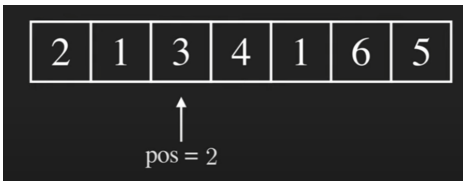


Figure 3: Passo 3 do Gnome Sort

Figure 4: Passo 4 do Gnome Sort

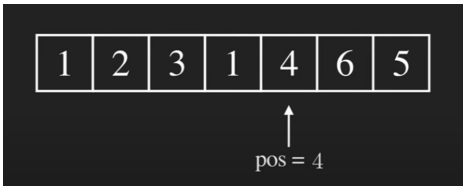


Figure 5: Passo 5 do Gnome Sort

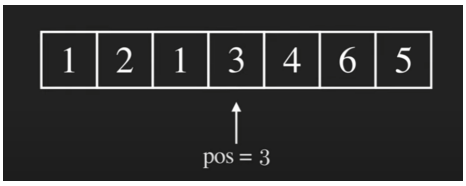


Figure 6: Passo 6 do Gnome Sort

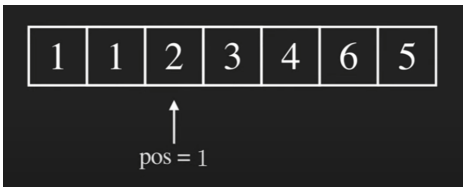


Figure 7: Passo 7 do Gnome Sort

3 Como ficou no código

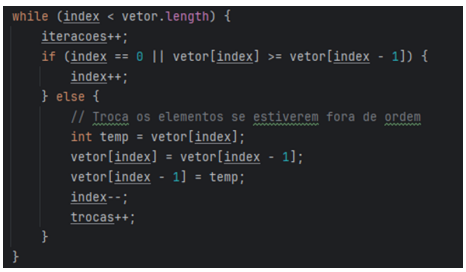
Na minha implementação do Gnome Sort ficou assim:

```
// Implementação do Gnome Sort
// O gnome vai tentar ordenar o algoritimo em sua devida ordem
private static Resultado ordenacaoGnome(int[] vetor) {  1 usage
    long tempoInicio = System.currentTimeMillis();
    long trocas = 0;
    long iteracoes = 0;

    int index = 0;
    while (index < vetor.length) {
        iteracoes++;
        if (index == 0 || vetor[index] >= vetor[index - 1]) {
            index++;
        } else {
            // Troca os elementos se estiverem fora de ordem
            int temp = vetor[index];
            vetor[index] = vetor[index - 1];
            vetor[index - 1] = temp;
            index--;
            trocas++;
        }
    }

    long tempoFim = System.currentTimeMillis();
    return new Resultado( tempo: tempoFim - tempoInicio, trocas, iteracoes);
}
```

Figure 8: Implementação do Gnome Sort

Onde podemos separar por partes para explicar. Ele tem início nas suas definições onde temos o `tempoInicio`, que será o que contará o tempo desde o início da execução do array, e definimos as `trocas` e `iteracoes` como 0 para que esses números sejam atualizados ao decorrer do código.

Logo após isso teremos esse Loop:



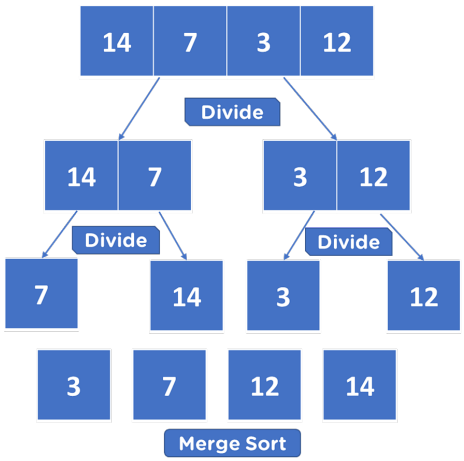
```
while (index < vetor.length) {
    iteracoes++;
    if (index == 0 || vetor[index] >= vetor[index - 1]) {
        index++;
    } else {
        // Troca os elementos se estiverem fora de ordem
        int temp = vetor[index];
        vetor[index] = vetor[index - 1];
        vetor[index - 1] = temp;
        index--;
        trocas++;
    }
}
```

Figure 9: Loop principal do Gnome Sort

Onde o `while` irá percorrer todo o vetor, incrementando o contador de iterações a cada passo. Após isso teremos o `if` que irá comparar o elemento atual com o anterior, daí caso esteja na ordem correta, ele avançará; caso contrário, trocará os elementos e moverá o índice para trás.

4 Merge Sort

Primeiramente uma breve explicação, o merge sort ele vai dividir uma array varias vezes ate que tenha exista apenas sub-arrays com um único elemento, e após separara a array em vários elementos, ele ira juntar todos em uma array de forma ordenada, assim como pode ser visto no exemplo mostrado a baixo

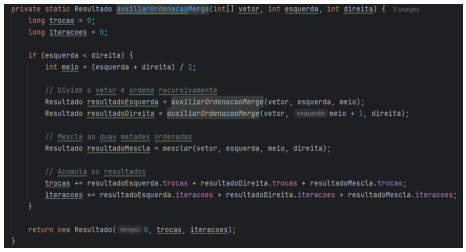


No meu código a implementação do Merge Sort ficou dividida em 3 principais funções, a" ordenacaoMerge", "auxiliarOrdenacaoMerge"e a "mesclar". Sobre a ordenacaoMerge:

```
private static Resultado ordenacaoMerge(int[] vetor) {  1 usage
    long tempoInicio = System.currentTimeMillis();
    long trocas = 0;
    long iteracoes = 0;

    // Chama o metodo auxiliar recursivo
    Resultado resultado = auxiliarOrdenacaoMerge(vetor,  esquerda: 0,  direita: vetor.length - 1);
    trocas += resultado.trocas;
    iteracoes += resultado.iteracoes;

    long tempoFim = System.currentTimeMillis();
    return new Resultado( tempo: tempoFim - tempoInicio, trocas, iteracoes);
}
```

Assim como a função de ordenação do Gnome, aqui teremos tempoInicio, que será oque contara o tempo dês do início da execução da array, e definimos as trocas e iteracoes como 0 para que esses números sejam atualizados ao decorrer do código. Logo a baixo ela ira chamar a função recursiva auxiliarOrdenacaoMerge, aqui também sera onde acumulara o tempo de trocas e interações e calculara o tempo total da execução, retornando esse valores como um objeto chamado Resultado.

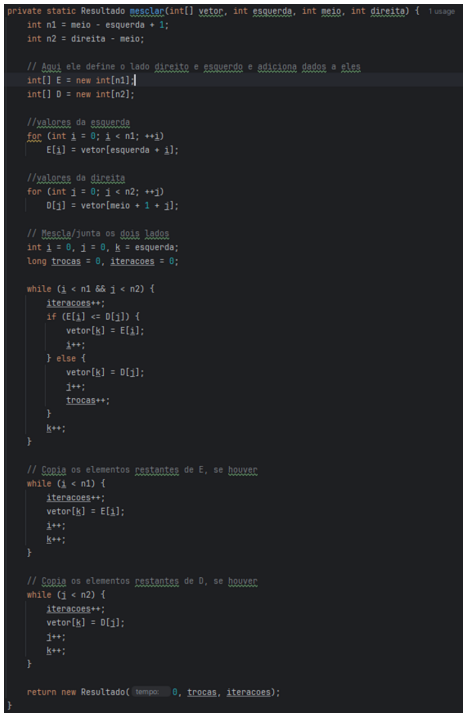
Sobre auxiliarOrdenacaoMerge:



```
private static Resultado auxiliarOrdenacaoMerge(int[] vetor, int esquerda, int direita) {
    long trocas = 0;
    long iteracoes = 0;

    if (esquerda < direita) {
        int meio = (esquerda + direita) / 2;

        // Divide o vetor e ordena recursivamente
        Resultado resultadoEsquerda = auxiliarOrdenacaoMerge(vetor, esquerda, meio);
        Resultado resultadoDireita = auxiliarOrdenacaoMerge(vetor, meio + 1, direita);

        // Mescla as duas metades ordenadas
        Resultado resultadoMescla = mesclar(vetor, esquerda, meio, direita);

        // Acumula os resultados
        trocas += resultadoEsquerda.trocas + resultadoDireita.trocas + resultadoMescla.trocas;
        iteracoes += resultadoEsquerda.iteracoes + resultadoDireita.iteracoes + resultadoMescla.iteracoes;
    }

    return new Resultado(0, trocas, iteracoes);
}
```

Essa função será a responsável por dividir o vetor ao meio. Após isso ela ira chamar a recursividade para as metades esquerda e direita, e após isso ela vai mescla as duas metades usando a função Mesclar, assim acumulando o número de trocas e iterações de todas as operações

Função Mesclar



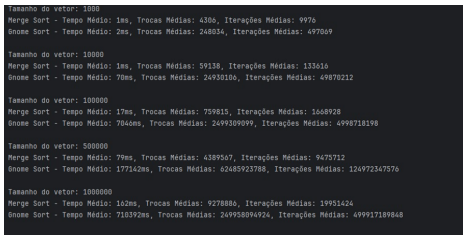
```
private static Resultado mesclar(int[] vetor, int esquerda, int meio, int direita) {
    int n1 = meio - esquerda + 1;
    int n2 = direita - meio;

    // Aqui ele define o lado direito e esquerdo e adiciona dados a eles
    int[] E = new int[n1];
    int[] D = new int[n2];

    //valores da esquerda
    for (int i = 0; i < n1; ++i)
        E[i] = vetor[esquerda + i];

    //valores da direita
    for (int j = 0; j < n2; ++j)
        D[j] = vetor[meio + 1 + j];

    // Mescla/junta os dois lados
    int i = 0, j = 0, k = esquerda;
    long trocas = 0, iteracoes = 0;

    while (i < n1 && j < n2) {
        iteracoes++;
        if (E[i] <= D[j]) {
            vetor[k] = E[i];
            i++;
        } else {
            vetor[k] = D[j];
            j++;
            trocas++;
        }
        k++;
    }

    // Copia os elementos restantes de E, se houver
    while (i < n1) {
        iteracoes++;
        vetor[k] = E[i];
        i++;
        k++;
    }

    // Copia os elementos restantes de D, se houver
    while (j < n2) {
        iteracoes++;
        vetor[k] = D[j];
        j++;
        k++;
    }

    return new Resultado(0, trocas, iteracoes);
}
```

A função Mesclar ela ira criar duas arrays temporárias E(esquerda) D(direita), copiando os valores da array original para as temporárias. Aqui teremos uma comparação e a mescla dos elementos de "E" e "D" de volta a seu vetor original e de forma ordenada Esta adição completa a seção sobre o Merge Sort, fornecendo uma explicação geral do algoritmo e detalhando as três principais funções utilizadas na implementação. O texto descreve o processo de divisão e

mesclagem característico do Merge Sort, bem como a estrutura e o propósito de cada função no código.

5 Tabela Merge Sort x Gnome Sort

Ao executar o código, obtive esses valores como resultado:



E com esse resultado, podemos observar que o Merge Sort teve um melhor desempenho em relação ao Gnome Sort em todos os aspectos. Segue abaixo uma tabela com os resultados de uma forma melhor apresentada:

Table 1: Comparação entre Merge Sort e Gnome Sort

2*Tamanho	Merge Sort			Gnome Sort		
	Tempo (ms)	Trocas	Iterações	Tempo (ms)	Trocas	Iterações
1.000	1	4.306	9.976	2	248.034	497.069
10.000	1	59.138	133.616	70	24.930.106	49.870.212
100.000	17	759.815	1.668.928	7.046	2.499.309.099	4.998.718.198
500.000	79	4.389.567	9.475.712	177.142	62.485.923.788	124.972.347.576
1.000.000	162	9.278.886	19.951.424	710.392	249.958.094.924	499.917.189.848