Trabalho 1 - Geometria Computacional

Samuel Vieira de Paula Farias - 498844

16 de Abril de 2023

Abstract

Código disponível em: Github Page

1 Questão 1.

Assinale V ou F nas afirmações abaixo, justificando suas respostas:

1. Um algoritmo é $O(f(n))$ quando existe k, n_0 tal que existe alguma instância de tamanho $n \geq n_0$, onde o número de passos é $\geq k \cdot f(n)$.

   Falso, um algoritmmo O(f(n)) define um limite superior de execução em f(n), logo, para esse $n \geq n_0$, deve-se executar em um número de passos $\leq k.f(n)$

2. Um algoritmo é $\Omega(f(n))$ quando existe k, n_0 tal que para qualquer instância de tamanho $n \geq n_0$, o número de passos é $\leq k \cdot f(n)$.

   Falso, um algoritmmo $\Omega(f(n))$ define um limite inferior de execução em $f(n)$, logo, para esse $n \geq n_0$, deve-se executar em um número de passos $\geq k.f(n)$

3. Se um algoritmo é $O(n)$, ele é também $O(n^2)$.

   Verdadeiro, como a notação O define um limite superior de execução e $n^2 > n$, pode-se afirmar que todo algortitmo $O(n)$ também é $O(n^2)$

4. Todo algoritmo é pelo menos $\Omega(n)$.

   Falso, desconsiderando os custos para alocação de memória, todo algoritmo é pelo menos $\Omega(1)$

5. A etapa mais importante no algoritmo de ordenação quicksort é a combinação, enquanto que no algoritmo de ordenação mergesort é a separação.

   Falso, o trabalho majoritario do algoritmo de QuickSort é a separação do problema em sub-arrays para fazer a ordenação, já no algoritmo de MergeSort o trabalho principal é juntar as partes que já foram ordenadas.

2 Questão 2.

Implemente o algoritmos de ordenação por seleção, testando com um exemplo com 10, 100 e 1000 elementos gerados de forma aleatória. Além disso, teste com um exemplo já ordenado de 10 elementos. Faça uma análise dos resultados obtidos na sua implementação.

```
(SelectionSort)Antes da ordenação:
 Vec10: false
 Vec100: false
 Vec1000: false
Tempo do SelectionSort10 0.000001 sec
Tempo do SelectionSort100 0.00003 sec
Tempo do SelectionSort1000 0.00239 sec
(SelectionSort)Após a ordenação:
 Vec10: true
 Vec100: true
 Vec1000: true
```

Figure 1: Resultado do SelectionSort.

3 Questão 3.

Implemente o algoritmos de ordenação por quicksort, testando com um exemplo com 10,100 e 1000 elementos gerados de forma aleatória. Além disso, teste com um exemplo já ordenado de 10 elementos. Faça uma análise dos resultados obtidos na sua implementação.

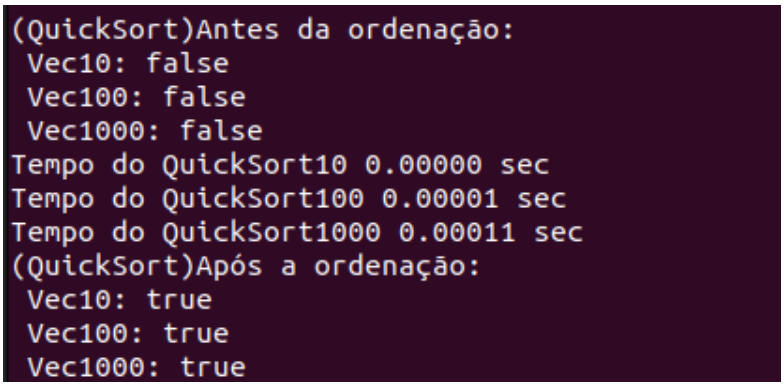

Figure 2: Resultado do QuickSort.

4 Questão 4.

Implemente o algoritmos de ordenação por mergesort, testando com um exemplo com 10, 100 e 1000 elementos gerados de forma aleatória. Além disso, teste com um exemplo já ordenado de 10 elementos. Faça uma análise dos resultados obtidos na sua implementação.

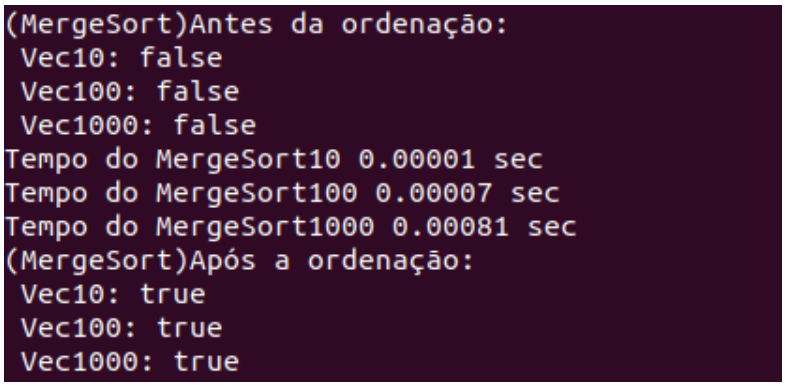

Figure 3: Resultado do MergeSort.

5 Questão 5.

Compare os resultados dos três algoritmos implementados, usando os exemplos com 10,100 e 1000 elementos gerados de forma aleatória e o exemplo já ordenado de 10 elementos.

Analisando o tempo de execução dos algoritmos, constata-se que eles seguem o padrão espera da complexidade assintótica dos algoritmos estudados, no qual o MergeSort se destaca por, nos seus piores casos com entradas maiores, performar melhor.

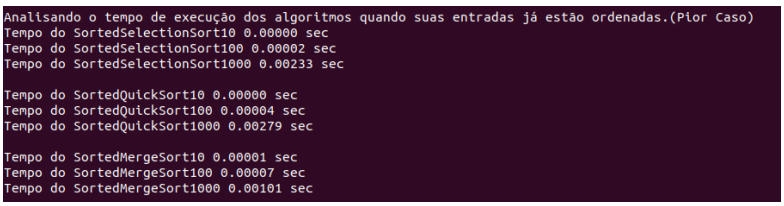

Figure 4: Resultado dos algoritmos quando a entrada já é um vetor ordenado.

6 Questão 6.

Ache uma cota inferior e superior para o problema de linha poligonal através do mecanismo de redução. Assuma que linha poligonal é uma linha que une vértices que não podem se interceptar. Mostre que a redução é linear e que existe um algoritmo ótimo para linha poligonal. Implemente um algoritmo para linha poligonal usando a redução por ordenação e teste para os seus algoritmos de ordenação implementados, usando exemplos com 10, 100 e 1000 elementos gerados de forma aleatória. Analise os resultados da sua implementação.

Sabe-se que o problema da linha poligonal consiste em fixar um eixo e ordená-lo, tornando esse problema redutível a uma ordenação. Para a cota inferior:
Levando em conta uma árvore de decisões, sabe-se que existem $n!$ possíveis ordenações para uma certa entrada de tamanho n, a árvore gerada tem p níveis, logo:

$$2^p \geq n!$$

$$p \geq log_2 n!$$

Com a aproximação de Stirling:

$$n! \approx \sqrt{2\pi n)} * (\frac{n}{e})^n$$

Logo:

$$log_2 n! \approx log_2(\sqrt{2\pi n}) + nlog_2 n - nlog_2 e \geq knlogn$$

Como no caso do MergeSort que possui cota superior = $nlogn$ **, concluímos que o problema de ordenação é Θn log n**