INTRODUÇÃO

Neste relatório iremos fazer uma breve análise do problema da seleção. Segundo Cormen et al. (2009), o problema da seleção pode ser apresentado da seguinte forma:

“We formally specify the selection problem as follows:

Input: A set A of n (distinct) numbers and an integer i, with $1 \leq i \leq n$.

Output: The element $x \in A$ that is larger than exactly i - 1 other elements of A.” (CORMEN et al., 2009, p. 213).

A aplicação do problema da seleção, ainda segundo Cormen et al. (2009), está relacionada à busca do menor e/ou do maior valor em um conjunto de elementos comparáveis e à busca da mediana dessa sequência de valores. Neste trabalho iremos analisar dois algoritmos que resolvem o problema da seleção:

1) **Selecao1(A, i)**: para esta função, utilizamos o algoritmo *Merge Sort*. Seguem abaixo a invariante e o tempo de processamento deste algoritmo:

   Invariante: conforme estudado no curso, a propriedade invariante desse algoritmo é que “antes da t-ésima iteração começar, vale que k = início + t- 1, o vetor a[início..k-1] está ordenado e os elementos b[i..n_1] e c[j..n_2] são maiores ou iguais aos elementos de a[início..k-1]”.

   Tempo de Processamento: também como demonstrado no curso, o tempo de processamento do *Merge Sort* é $T(n) = c.n.log_2 + n.d \in \theta(n\log(n))$. Como a linha *return A[i]* é $\theta(1)$, então este trecho não irá alterar de forma significativa o tempo exposto acima.

2) **Selecao2(A, i)**: para este caso, utilizamos a função Particao(A) e chamadas recursivas da função Selecao2**.** Seguem abaixo a invariante e o tempo de processamento deste algoritmo:

   Invariante: a invariante pode ser analisada através da função de partição. A cada chamada recursiva, a função irá retornar um subvetor no qual

todos os valores à esquerda de q (pivô) são menores ou iguais ao pivô e todos os elementos após q serão maiores que o pivô.

Tempo de Processamento: iremos dividir essa análise em alguns casos:

a) Para um vetor com n elementos (n > 1), o melhor caso será aquele em que o índice devolvido pelo algoritmo Particao sempre cair no meio da sequência. Assim, temos o tempo da função Particao, que é $\theta(n)$ mais o tempo de uma das chamadas recursivas, ou seja, teremos $T(n) = T\left(\frac{n}{2}\right) + \theta(n) \in \theta(n)$ (pelo Teorema Mestre).

b) Também para um vetor com n elementos (n > 1), o pior caso será aquele em que o índice de partição é sempre 0 ou n - 1. Nesse caso, $T(n) = \frac{n(n+1)}{2} \approx \frac{n^2}{2} \in \theta(n^2)$.

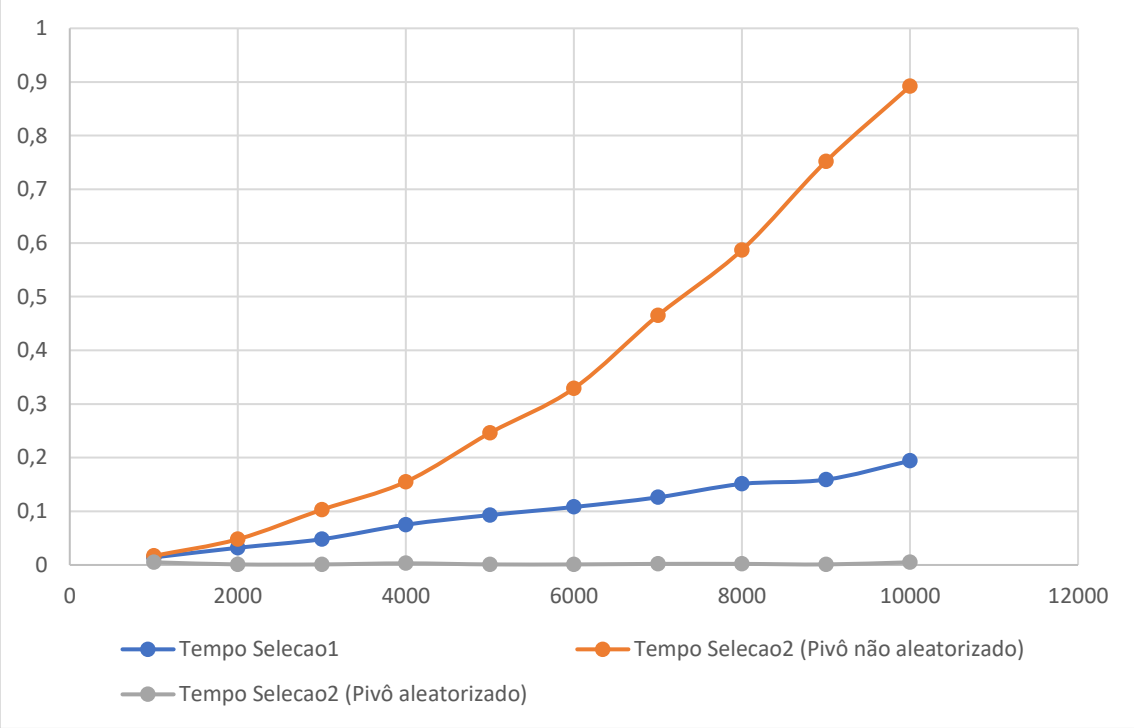
OBJETIVO E RESULTADO

Após testarmos o tempo de processamento para entradas de tamanhos diferentes, segue abaixo uma tabela onde consta a comparação dos tempos obtidos para cada um dos algoritmos:

Tabela 1: comparação dos tempos entre os dois algoritmos (tempo em segundos)

Quantidade de valores	Tempo Selecao1	Tempo Selecao2 (Pivô não aleatorizado)	Tempo Selecao2 (Pivô aleatorizado)
1000	0,014	0,017	0,005
2000	0,032	0,048	0,001
3000	0,048	0,103	0,001
4000	0,075	0,155	0,003
5000	0,093	0,246	0,001
6000	0,108	0,329	0,001
7000	0,126	0,465	0,002
8000	0,151	0,587	0,002
9000	0,159	0,752	0,001
10000	0,194	0,892	0,005

Gráfico 1: comparação dos tempos entre os dois algoritmos (eixo vertical com o tempo em segundos e o eixo horizontal com a quantidade de valores do vetor)



CONCLUSÃO

Conforme visto anteriormente, é possível termos um tempo de processamento $\theta(n)$para o algoritmo Selecao2, o que seria mais vantajoso que um tempo $\theta(n\log(n))$. Porém também vimos que para termos um tempo $\theta(n)$, é necessário que sejam escolhidos bons pivôs, ou seja, valores que dividam o vetor de forma homogênea. Portanto, levando em consideração a tabela de comparação exposta na seção “Objetivo e Resultado”, o melhor algoritmo é o Selecao2, desde que possamos aleatorizar o pivô, para que tenhamos um tempo de processamento $\theta(n)$.

BIBLIOGRAFIA

Cormen et al. **Introduction to Algorithms**. Massachusetts: The MIT Press, 2009. " Medians and Order Statistics".