Análise de Algoritmos - Trabalho 1

Professor: Marco Molinaro

Grupo: Alexandre Dias - 1413183

Para este trabalho, foi criado um git, e o código completo dele pode ser encontrado no link:
https://github.com/Tijuk/AnaliseDeAlgoritmos/tree/master/T1

Tarefa 1 : Seleção em tempo linear

1.1) No código, esta função é o método *findKthElement* da classe *LinearSelection*. Ela esta localizada no arquivo selections.js

```
findKthSmallest(array, k) {
        if(array.length == 1) {
            return array[0];
        }
        if(array.length < 1 || array.length < k) {
            throw message;
        }
        const pivot = this.MedOfMed(array);
        let nPivots = -1, pivots = [];
        const L = [], R = [];
        for(let i = 0; i < array.length; i++) { // 1 .. n
            if(array[i] < pivot) {
                L.push(array[i]);
            } else if(array[i] > pivot) {
                R.push(array[i]);
            } else {
                if(nPivots >= 0) {
                    pivots.push(array[i]);
                }
                nPivots++;
```

```
            }
        }
        let edge = k - nPivots;
        const L_size = L.length;
        if((L_size + nPivots) > k && L_size <= k) {
            return pivot;
        }
        if(L_size == edge) {
            return pivot;

        }
        else if(L_size > edge) {
            if(L_size <= 0) {
                return pivot;
            }
            return this.findKthSmallest(L, k);
        }
        else if(L_size < edge) {
            return this.findKthSmallest(R, k - (L_size) - nPivots - 1);
        } else {
            console.error('glitch in the matrix');
        }
    }
```

1.2) $T(n) \leq cst \cdot n + T(\frac{n}{5}) + T\left(\frac{7n}{10}\right)$

A parte com cst n nela é dada pela separação entre elementos a esquerda e a direita da mediana (que também realiza a quantidade de elementos iguais ao pivô. No código sendo essa parte:

```
let nPivots = -1, pivots = [];
const L = [],  R = [];
for(let i = 0; i < array.length; i++) { // 1 .. n
    if(array[i] < pivot) {
        L.push(array[i]);
```

```
    } else if(array[i] > pivot) {
        R.push(array[i]);
    } else {
        if(nPivots >= 0) {
            pivots.push(array[i]);
        }
        nPivots++;
    }
}
```

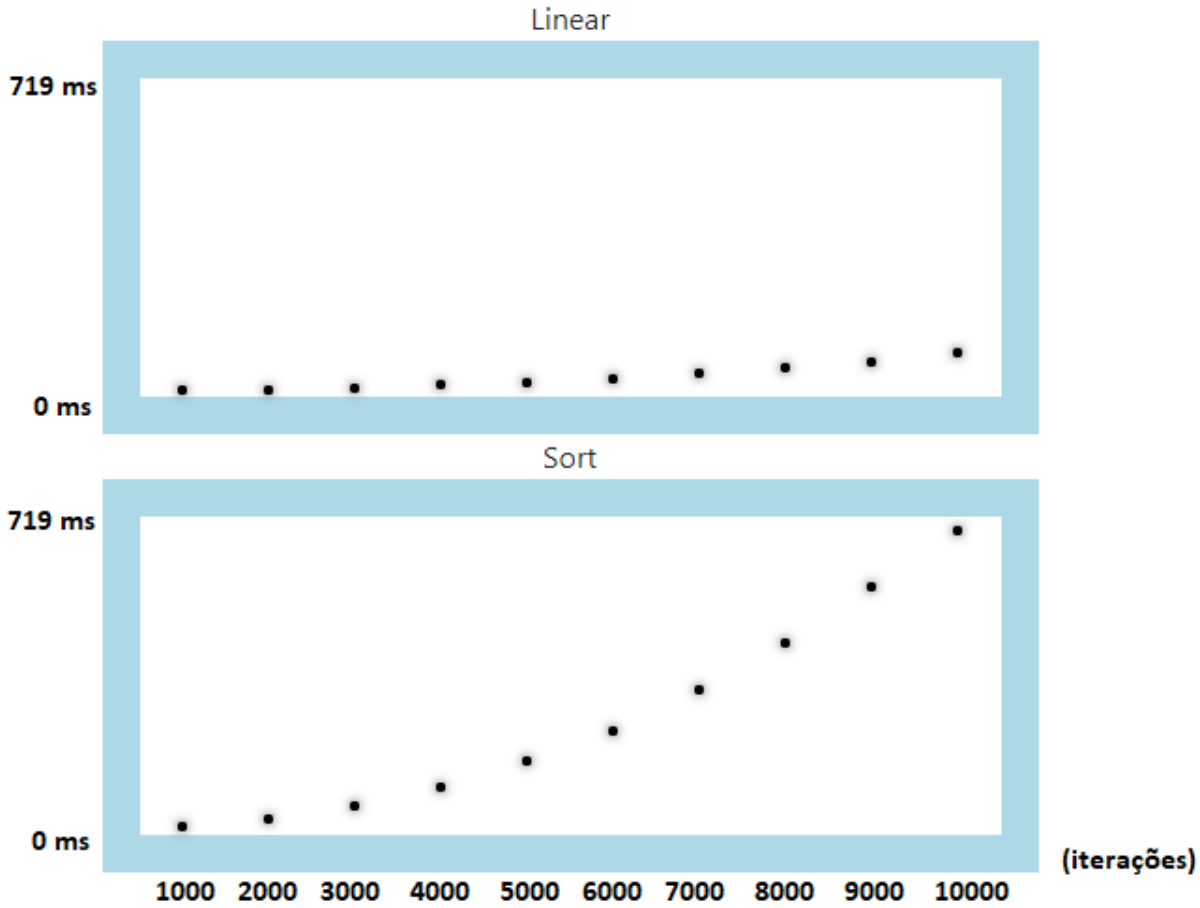
1.3) Para rodar o código basta rodar o arquivo **TrabalhoAlgoritmo.html** no navegador. Uma tela com 2 botões irão aparecer.
Para rodar inserindo uma entrada qualquer, clique no botão escrito **“Passo a Passo”**.
Insira a entrada na caixa de texto de cima no formato descrito pelo placeholder de:
[número][vírgula][número], etc...
Ex: 10, 3, -2, -5, 0, 3

- *Note que o uso de espacamento é opcional*
- *Espaço com 2 vírgulas consecutivas serão interpretadas como zero. Análogo a vírgula no final da entrada.*

A o segundo campo é o valor que deseja-se obter do array.

Tarefa 2 : Experimentos

2.1) Gráficos



Os valores finais foram:

Iterações:	**1000**	**2000**	**3000**	**4000**	**5000**	**6000**	**7000**	**8000**	**9000**	**10000**
Linear	2.41	5.16	11.07	16.73	23.52	32.40	46.17	56.81	74.01	93.37
Sort	6.9	27.33	58.67	104.99	165.87	240.07	337.80	447.76	583.67	718.44

** 2.41, 5.16, etc... são todos tempos em milissegundos*

2.2) Como podemos ver pelo gráfico, o algoritmo Sort cresce muito mais rápido com o aumento do tamanho do array em comparação ao Linear.
Isso se dá pela complexidade dos algoritmos, uma vez que a complexidade do Sort é de $O(n^2)$, enquanto a do Linear é de $O(n)$.