CENTRO UNIVERSITÁRIO
FEI MESTRADO ENGENHARIA ELÉTRICA
ALGORITMOS COMPUTACIONAIS – PEL201

RELATÓRIO 2
ALGORITMOS DE ORDENAÇÃO
BUBBLE SORT E QUICK SORT

GUSTAVO APARECIDO DE SOUZA VIANA – 120110-2
SÃO BERNARDO DO CAMPO
2020

O código fonte está disponível em:
https://github.com/apparecidoo/master-computational-algorithms

1)

BubbleSort é um algoritmo de ordenação onde itera N x N vezes o vetor de tamanho N, durante a iteração ele verifica se o valor na posição atual é maior que a próxima posição, caso positivo é feita a troca, e assim sucessivamente.

```
template<class T>
inline void Sort<T>::BubbleSort(T arr[], int size)
{
    for (int i = 0; i < size; i++)
    {
        for (int j = 0; j < size - 1; j++)
        {
            if (arr[j] > arr[j + 1]) {
                Swap(arr, j, j + 1);
            }
        }
    }
}
```

2)

O QuickSort usa o conceito de divisão e conquista, o processo de ordenação é recursiva e consiste na escolha de um pivô e assim colocar todos os número menores para seu lado esquerdo e todos maiores para o lado direito, e depois achar um novo pivô que será passado recursivamente para a mesma função QuickSort, ou seja, esse processo faz com que dívida em vetores menores e aplique a ordenação nesses menores.

```
template<class T>
inline void Sort<T>::QuickSort(T arr[], int left, int right)
{
    if (left < right) {
        int pivot = QuickSortDivide(arr, left, right);
        QuickSort(arr, left, pivot - 1);
        QuickSort(arr, pivot + 1, right);
    }
}

template<class T>
inline void Sort<T>::QuickSortRandom(T arr[], int left, int right)
{
    if (left < right) {
        int pivot = QuickSortDivide(arr, left, right, true);
        QuickSort(arr, left, pivot - 1);
        QuickSort(arr, pivot + 1, right);
    }
}

template<class T>
inline int Sort<T>::QuickSortDivide(T arr[], int left, int right, bool pivotRandom)
{
    int i = left + 1;
    int j = right;
    int pivot = pivotRandom ? arr[Random().generate(left, right)] : arr[left];
    while (i <= j)
    {
        if (arr[i] <= pivot) {
            i++;
        }
        else {
            if (arr[j] > pivot) {
                j--;
            }
            else {
                if (i <= j) {
                    Swap(arr, i, j);
                    i++;
                    j--;
                }
            }
        }
    }

    Swap(arr, left, j);

    return j;
}
```

3)

Para os experimentos de desempenho foram gerados 20 vetores com 3000 números aleatórios entre 0 a 100000. A seguir a tabela que mostra o tempo de execução em microssegundos para cada algoritmo e o gráfico correspondente.

Podemos ver que o BubbleSort é bem mais lento e que usar um pivô aleatório inicial não é tão vantajoso, mas se usar um pivô inicial que tenha uma heurística por trás pode ser que seja mais vantajoso.

Attempts	BubbleSort	QuickSort	QuickSortRandom
1	308085	1187	1261
2	220711	1104	1713
3	268381	1160	1229
4	188429	1571	2666
5	269322	1254	16250
6	241054	1122	1083
7	208828	1245	908
8	181042	967	1399
9	186672	860	766
10	151794	826	825
11	110845	861	658
12	108419	914	912
13	127166	698	672
14	137102	975	931
15	142995	982	921
16	168505	939	1077
17	177794	938	941
18	198709	1097	866
19	196212	1068	1067
20	197543	1063	1054