ECM-306 – TÓPICOS AVANÇADOS EM ESTRUTURAS DE DADOS

AULA 08 – TAREFAS

Amanda Carolina Ambrizzi Ramin; 22.00721-0

Exercício 1

```
 static void merge (int arr [], int l, int m, int r)
{
    int n1 = m - l + 1;
    int n2 = r - m;

    int L [] = new int[n1];
    int R [] = new int[n2];

    for (int i = 0; i < n1; ++i)
        L[i] = arr [l + i];
    for (int j = 0; j < n2; ++j)
        R[j] = arr [m + 1 + j];

    int i = 0, j = 0;

    int k = l;
    while (i < n1 && j < n2) {
        if (L[i] <= R[j]) {
            arr[k] = L[i];
            i++;
        }
        else {
            arr[k] = R[j];
            j++;
        }
        k++;
    }

    while (i < n1) {
        arr[k] = L[i];
        i++;
        k++;
    }

    while (j < n2) {
        arr[k] = R[j];
        j++;
        k++;
    }
}
```

```
static void sort (int arr [], int l, int r)
{
    if (l < r) {
        int m =l+ (r-l)/2;

        sort (arr, l, m);
        sort (arr, m + 1, r);

        merge (arr, l, m, r);
    }
}
```

Implementação tirado do site: https://www.sortvisualizer.com/mergesort/

A função de complexidade do merge-sort é O (n log n).

Exercício 2

```
private static void binSearch (int item, int begin, int end) {
        int metade = (begin + end)/2;

        if (begin > end) {
            indice = -1;
            nComparacoes++;
            return;
        }

        if(A[metade] == item) {
            indice = metade;
            nComparacoes++;
            return;
        }

        if(A[metade] < item) {
            nComparacoes++;
            binSearch (item, metade+1, end);
        }
        else {
            nComparacoes++;
            binSearch (item, begin, metade);
        }
    }
```

Fonte: slides da aula

A ordem de complexidade é O (log n).

Fazendo a equação recorrência, tem-se:

$$T(n) = \begin{cases} 1 + T\left(\frac{n}{2}\right), & n > 1 \\ 1, & n = 1 \end{cases}$$

Truncando o resultado de $\frac{n}{2}$:

$$T(n) = \begin{cases} 1 + \left\lfloor T\left(\frac{n}{2}\right) \right\rfloor, & n > 1 \\ 1, & n = 1 \end{cases}$$

Fechando a equação e assumindo $n = 2^k$:

$$T(n) = \begin{cases} 1 + T\left(\frac{n}{2}\right), & n > 1 \text{ e } n = 2^k, k_{inteiro} > 0 \\ 1, & n = 1 \end{cases}$$

Aplicando o Método da Substituição, tem-se:

$$T(n) = k + T\left(\frac{n}{2^k}\right)$$

Isolando T(n), tem-se:

$$T(n) = \log_2 n + 1, \text{ para } n = 2^k, k_{inteiro} > 0$$

Dessa forma, percebe-se que a função complexidade é O(log n) .