# Projeto e Análise de Algoritmos

## Lista de exercícios A1

## Questões

1. Encontre $f(n)$ de forma que $T(n) = \theta(f(n))$ para cada função a seguir, provando seu resultado:

   (a) $T(n) = 4n^3 + n^2 + 3n$

   (b) $T(n) = 2n \log n + 2n + 7$

   (c) $T(n) = 100 \log(n^5) + n^2$

2. Prove que não existe uma função $T(n)$ que seja $O(n^a)$ e $\Omega\left(n^b\right)$ para $b > a$.

3. Prove que $T(n) = \log n$ é $O(n^a)$ para todo $a > 0$.

4. Encontre a complexidade assintótica do seguinte algoritmo:

```c
int f(int *array, int lenght) {
    int count = 0;
    for (int i = 0; i < lenght; i++) {
        count += i;
    }

    int result = 0;
    for (int i = 0; i < count; i++) {
        for (int j = 0; j*j < lenght; j++) {
            result += array[j*j];
        }
    }
    return result;
}
```

5. Para cada item a seguir, resolva a recorrência para encontrar a função $f(n)$ tal que $T(n) = O(f(n))$. Em pelo menos uma das questões, resolva a recorrência usando os quatro métodos vistos em aula.

   (a) $T(n) = \begin{cases} \theta(1) & se\ n = 1 \\ T(n-1) + 2n^2 & se\ n > 1 \end{cases}$

   (b) $T(n) = \begin{cases} \theta(1) & se\ n = 1 \\ 2T\left(\frac{n}{3}\right) + 4n^3 & se\ n > 1 \end{cases}$

   (c) $T(n) = \begin{cases} \theta(1) & se\ n = 1 \\ 4T\left(\frac{n}{4}\right) + 9\sqrt{n} & se\ n > 1 \end{cases}$

(d) $T(n) = \begin{cases} \theta(1) & se\ n = 1 \\ 6T\left(\frac{n}{5}\right) + n\log(n) + 3n & se\ n > 1 \end{cases}$

(e) $T(n) = \begin{cases} \theta(1) & se\ n = 1 \\ 2T\left(\frac{n}{4}\right) + T\left(\frac{n}{2}\right) + n & se\ n > 1 \end{cases}$

6. Considerando `pow(a,b)` como uma função que calcula $a^b$ em $\theta(1)$, encontre a complexidade assintótica do seguinte algoritmo:

```
int f(int *array, int start, int end) {
    int size = end - start;
    if (size <= 0) {
        return 0;
    }

    // Divisão inteira
    int mid = (start + end) / 2;
    int leftCount = f(array, start, mid);
    int rightCount = f(array, mid + 1, end);
    int count = leftCount + rightCount;

    for (int i = 0; pow(i, 2) < size; i++) {
        for (int j = 0; pow(2, j) < size; j++) {
            count++;
        }
    }
    return count;
}
```

Para os exercícios a seguir apresente um algoritmo capaz de solucionar o problema e apresente a sua análise. O algoritmo pode ser escrito em pseudocódigo, em alguma linguagem de programação, ou através de uma descrição textual sem ambiguidades.

7. Dado um inteiro $k$ e uma lista de inteiros $A$ de tamanho $n$ contendo $m$ números diferentes ($m \geq k$), projete um algoritmo que retorne o $k$-ésimo inteiro que mais se repete na lista. O algoritmo deve ter complexidade $O(n)$ no pior caso.

8. Dado um valor $x$ e uma lista $A$ que contém $n$ inteiros, projete um algoritmo que encontre um par de inteiros em $A$ cuja soma seja $x$, caso existam. O algoritmo deve ter complexidade $O(n)$ no pior caso.

9. Dada uma lista $A$ que contém $n$ listas com $m$ inteiros cada, projete um algoritmo que retorne o número de listas não contém elementos em comum com outras listas. O algoritmo deve ter complexidade $O(n \cdot m)$ no pior caso.

10. Dada uma lista $A$ de $n$ inteiros não negativos, projete um algoritmo que retorne o maior inteiro $x$ que esteja em $A$ e que tenha pelo menos $x$ inteiros em $A$ que sejam maiores ou iguais a ele (incluindo ele próprio), ou -1 se não existir. O algoritmo deve ter complexidade $O(n)$.

11. Dada uma árvore binária de busca (BST) $T$ com $n$ nós contendo inteiros, projete um algoritmo que encontre a menor diferença absoluta entre dois nós diferentes da árvore. O algoritmo deve ter complexidade $O(n)$ no pior caso.

12. Dado dois inteiros $x$ e $k$ e uma lista $A$ contendo $n$ inteiros, projete um algoritmo que retorne os $k$ elementos da lista mais próximos de $x$, em ordem crescente de proximidade. O algoritmo deve ter complexidade $O(n \log k)$ no pior caso.

13. Dado um número $x > 0$ e uma lista $A$ contendo $n$ inteiros, projete um algoritmo que retorne a lista com o maior número de elementos de $A$ de forma que a maior diferença entre quaisquer dois deles seja menor ou igual a $x$. O algoritmo deve ter complexidade $O(n \log n)$ no pior caso.

14. Dado uma lista $A$ contendo $n$ inteiros, projete um algoritmo **utilizando o método guloso** que retorne o par de índices $(i, j)$ de forma que o produto $|i-j| \cdot \min\{A[i], A[j]\}$ seja máximo. O algoritmo deve ter complexidade $O(n)$ no pior caso.

15. Dado uma lista $A$ contendo $n$ inteiros, projete um algoritmo **utilizando o paradigma de dividir e conquistar** que retorne uma árvore binária de busca (BST) **balanceada** contendo todos elementos de $A$. O algoritmo deve ter complexidade $O(n \log n)$ no pior caso.