# Lista - Algoritmos e Estrutura de Dados

*Aluno: J Carlos Viana F.*

1. (resposta)

   $f(n) = 2n^2 + 3n + 4$ é $\mathcal{O}(n^2)$; logo $g(n) = n^2$

   $2n^2 + 3n + 4 \leq cn^2$
   $2n^2 + 3n + 4 \leq 4n^2$

   $f(n) = \mathcal{O}(n^2)$,
   $c = 4$ e $n_0 = 3$

2. (resposta)

   $f(n) = n^3$ é $\mathcal{O}(n^2)$; $g(n) = n^2$

   $n^3 \leq cn^2$

   não há $c$ e $n_0$ tal que $n^3 \leq cn^2$
   Logo $f(n)$ não é $\mathcal{O}(n^2)$

3. (resposta)

   $f(n) = 2^{n+1}$ é $\mathcal{O}(2^n)$; $g(n) = 2^n$

   $2^{n+1} \leq c2^n$
   $2^{n+1} \leq 2.2^n$
   $2^{n+1} \leq 2^{n+1}$

   $f(n) = \mathcal{O}(2^n)$,
   $c = 2$ e $n_0 = 1$

4. (resposta)

   $T(n) = 1+T(n-1) = 1+1+T(n-2) = 2+T(n-2) = 2+1+T(n-3)$
   $= 3 + T(n-3)$

   $T(n) = k + T(n-k)$
   para $k = n - 1$ temos
   $T(n) = n - 1 + T(1)$
   Logo, a busca linear/sequencial é $\mathcal{O}(n)$

5. (resposta)

   $T(n) = 1+T(\frac{n}{2}) = 1+1+T(\frac{n}{4}) = 2+T(\frac{n}{4}) = 2+1+T(\frac{n}{8}) = 3+T(\frac{n}{8})$

   $T(n) = k + T(\frac{n}{2^k})$
   $2^k = n \Rightarrow \log_2 2^k = \log_2 n \Rightarrow k = \log_2 n$
   $T(n) = \log_2 n + T(1)$
   Logo, a busca binária é $\mathcal{O}(\log n)$

6. (resposta)

   $T(n) = n + 2T(\frac{n}{2}) = n + 2[\frac{n}{2} + 2T(\frac{n}{4})] = n + n + 4T(\frac{n}{4}) = 2n + 4T(\frac{n}{4}) = 2n + 4[\frac{n}{4} + 2T(\frac{n}{8})] = 3n + 8T(\frac{n}{8})$

   $T(n) = kn + 2^k T(\frac{n}{2^k})$
   $2^k = n \Rightarrow \log_2 2^k = \log_2 n \Rightarrow k = \log_2 n$
   $T(n) = n \log_2 n + nT(1)$ De modo que o *quicksort* é $\mathcal{O}(n \log n)$

7. (resposta)

   $f(n) + g(n) = \mathcal{O}(max\{f(n), g(n)\})$

   $f(n) \leq c_1 h(n)$
   $g(n) \leq c_2 h(n)$

   $f(n) + g(n) \leq c_1 h(n) + c_2 h(n)$
   $f(n) + g(n) \leq (c_1 + c_2) h(n)$
   $f(n) + g(n) \leq d.h(n)$, sendo $d = c_1 + c_2$
   $f(n) + g(n) \leq d.\mathcal{O}(max\{f(n), g(n)\})$

   $f(n) + g(n) = \mathcal{O}(max\{f(n), g(n)\})$ para $d = c_1 + c_2$ e um dado $n_0$

8. (resposta)

   Para o *merge*:

```
void merge(int *v, int begin, int mid, int end){
    int n1 = mid - begin + 1;
    int n2 = end - mid;

    int left[n1+1];
    int right[n2+1];

    int i,j,k;

    for(i=0;i<n1;i++)
        left[i] = v[begin+i];

    for(i=0;i<n2;i++)
        right[i] = v[mid+i+1];


    left[n1]=INT_MIN; // ao invés de INT_MAX
    right[n2]=INT_MIN; // ao invés de INT_MAX

```

```
    i = 0;
    j = 0;
    for(k=begin;k<=end;k++){
        if(left[i]>=right[j]){ // ao invés de <=
            v[k]=left[i];
            i++;
        }
        else{
            v[k]=right[j];
            j++;
        }
    }
}
```

Para o *mergesort* (não é alterado):

```
void mergesort(int *v, int begin, int end){

    if(begin < end){
        int mid = (begin+end)/2;

        printf("begin: %d, mid: %d, end: %d\n",begin
                ,mid,end);
        mergesort(v,begin,mid);
        mergesort(v,mid+1,end);
        merge(v,begin,mid,end);
    }

}
```

9. (resposta)

```
void bubblesort(int *v, int size) {

    int i,j,ord = 0;

    for (j = size - 1; j >= 0; j--) {
        ord = 0;
        for (i = 0; i < j; i++) {
            if (v[i] > v[i+1])
```

```
                swap(v+i, v+i+1);
            else
                ord++;
        }
        if(ord==j){
            printf("ordenado em j=%d\n",j);
            break;
        }
    }
}
```

10. (resposta)

```
somalista::[Int]->Int
somalista [] = 0
somalista (a:as) = a + (somalista as)
```

11. (resposta)

```
produtolista :: [Int] -> Int
produtolista [] = 1
produtolista (a:as) = a * (produtolista as)
```

12. (resposta)

```
potencia :: Int -> Int -> Int
potencia a 1 = a
potencia a b = a * (potencia a (b-1))
```

13. (resposta)

```
listsize :: [Int] -> Int
listsize [] = 0
listsize (a:as) = 1 + (listsize as)
```