Departamento de Ciência de Computadores FCUP

Desenho e Análise de Algoritmos CC2001 2023/2024

Folha 2

Escrita de algoritmos em pseudo-código. Caraterização da sua complexidade assintótica. Análise e demonstração de correção para programas simples.

Ordens de grandeza / Complexidade Assintótica

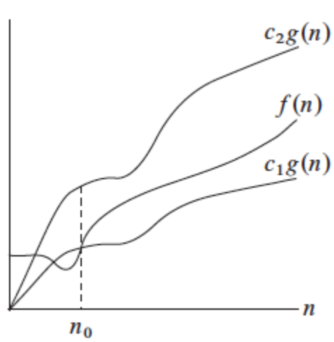
Para recordar:

As ordens de grandeza O, Θ e Ω são assim definidas:

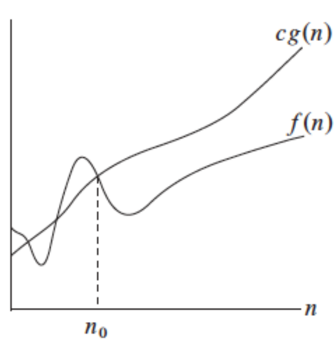
$$O(g(n)) = \{f(n) \mid \text{existem } c > 0 \text{ e } n_0 > 0 \text{ tais que } f(n) \leq cg(n) \text{ para todo } n \geq n_0\}$$
$$\Omega(g(n)) = \{f(n) \mid \text{existem } c > 0 \text{ e } n_0 > 0 \text{ tais que } f(n) \geq cg(n) \text{ para todo } n \geq n_0\}$$
$$\Theta(g(n)) = \{f(n) \mid \text{existem } c_1 > 0,\ c_2 > 0 \text{ e } n_0 > 0 \text{ tais que } c_1 g(n) \leq f(n) \leq c_2 g(n) \text{ para todo } n \geq n_0\}$$

As notações $f(n)$ e $g(n)$ estão a ser usadas com as duas interpretações. Habitualmente, $f(n)$ designa a imagem de n pela função f, mas, nesta definição, $f(n)$ designa também a função $f : \mathbb{N} \to \mathbb{R}_0^+$ que a cada n associa $f(n)$. Do mesmo modo, $g(n)$ designa a função $g : \mathbb{N} \to \mathbb{R}_0^+$ que a cada n associa $g(n)$. Assim, em $f(n) \in O(g(n))$ estamos a classificar as funções f e g e em $f(n) \leq cg(n)$ estamos a referir as imagens de n por f e por g.

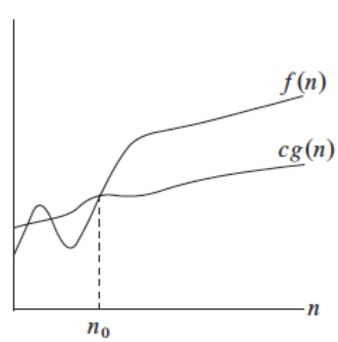
- $f(n) \in O(g(n))$ sse $f(n)$ é **majorada** por $cg(n)$ para alguma constante $c > 0$, a partir de uma certa ordem (definida por n_0).
- $f(n) \in \Omega(g(n))$ sse $f(n)$ é **minorada** por $cg(n)$ para alguma constante $c > 0$, a partir de uma certa ordem (definida por n_0).
- $f(n) \in \Theta(g(n))$ sse $f(n)$ é **majorada** por $c_2 g(n)$ e **minorada** por $c_1 g(n)$ para algum $c_1 > 0$ e algum $c_2 > 0$, a partir de uma certa ordem (definida por n_0).



$f(n) \in \Theta(g(n))$



$f(n) \in O(g(n))$



$f(n) \in \Omega(g(n))$

Exemplos

$3n^2 + 100 \in \Omega(n^2)$?

Verdade, porque $\exists_{c\in\mathbb{R}^+}\ \exists_{n_0\in\mathbb{Z}^+}\ \forall_{n\geq n_0}\ \ 3n^2 + 100 \geq cn^2$. Por exemplo, $c = 1$ e $n_0 = 1$.

Pois, $3n^2 + 100 \geq n^2$, para todo $n \geq 1$

$3n^2 + 100 \in O(n^2)$?

Verdade, porque $\exists_{c\in\mathbb{R}^+}\ \exists_{n_0\in\mathbb{Z}^+}\ \forall_{n\geq n_0}\ \ 3n^2 + 100 \leq cn^2$. Por exemplo, $c = 103$ e $n_0 = 1$.

Pois, $3n^2 + 100 \leq 3n^2 + 100n^2 = 103n^2$, para todo $n \geq 1$

$3n^2 + 100 \in \Theta(n^2)$?

Verdade, porque $\exists_{c_1\in\mathbb{R}^+}\ \exists_{c_2\in\mathbb{R}^+}\ \exists_{n_0\in\mathbb{Z}^+}\ \forall_{n\geq n_0}\ \ c_1 n^2 \leq 3n^2 + 100 \leq c_2 n^2$. Por exemplo, $c_1 = 1, c_2 = 103$ e $n_0 = 1$.

Pois, $n^2 \leq 3n^2 + 100 \leq 103n^2$, para todo $n \geq 1$

$100n + 3n\log_2 n \in \Theta(n\log_2 n)$?

Verdade, pois $\exists_{c_1\in\mathbb{R}^+}\exists_{c_2\in\mathbb{R}^+}\exists_{n_0\in\mathbb{Z}^+}\forall_{n\geq n_0}\ \ c_1(n\log_2 n) \leq 100n + 3n\log_2 n \leq c_2(n\log_2 n)$. Por exemplo, $c_1 = 1$, $c_2 = 103$ e $n_0 = 2$.

Notar que $100n + 3n\log_2 n \leq 103(n\log_2 n)$, pois $n \leq n\log_2 n$, para todo $n \geq n_0 = 2$. Se $n_0 = 1$, a justificação estaria incorreta (se $n = 1$ então $n > n\log_2 n$, pois $\log_2 1 = 0$).

$f(n) \in O(f(n))$, $f(n) \in \Omega(f(n))$, e $f(n) \in \Theta(f(n))$, qualquer que seja a função $f(n)$?

Verdade. Basta tomar $c = 1$, $c_1 = c_2 = 1$ e $n_0 = 1$ nas definições de $O(f(n))$, $\Omega(f(n))$ e $\Theta(f(n))$.

$\frac{3}{5}n^2 - \frac{1}{2} \in \Theta(n^2)$? Verdade porque:

- $\frac{3}{5}n^2 - \frac{1}{2} \in \Omega(n^2)$ porque $\exists_{c\in\mathbb{R}^+}\ \exists_{n_0\in\mathbb{Z}^+}\ \forall_{n\geq n_0}\ \ \frac{3}{5}n^2 - \frac{1}{2} \geq cn^2$. Por exemplo, $c = \frac{1}{1000}$ e $n_0 = 1$.
- $\frac{3}{5}n^2 - \frac{1}{2} \in O(n^2)$ porque $\exists_{c\in\mathbb{R}^+}\ \exists_{n_0\in\mathbb{Z}^+}\ \forall_{n\geq n_0}\ \ \frac{3}{5}n^2 - \frac{1}{2} \leq cn^2$. Por exemplo, $c = 1$ e $n_0 = 1$.

$O(n) \subseteq O(n^p)$, para todo $p \in \mathbb{Z}^+$?

Verdade porque $c \in \mathbb{R}^+$ e $p \in \mathbb{Z}^+$. Assim, qualquer que seja $f(n) \in O(n)$, tem-se $f(n) \in O(n^p)$, pois **se** $\exists c \in \mathbb{R}^+\ \exists n_0 \in \mathbb{Z}^+\ \forall n \geq n_0\ \ f(n) \leq cn$ **então** $\exists c \in \mathbb{R}^+\ \exists n_0 \in \mathbb{Z}^+\ \forall n \geq n_0\ \ f(n) \leq cn^p$.

$\Theta(\log_a n) = \Theta(\log_b n)$, para todo $a, b \in \mathbb{R}^+$, com $a, b > 1$?

Verdade porque $\log_a n = \log_a b\ \times\ \log_b n$.

Por isso, em vez de $\Theta(\log_2 n)$, podemos escrever $\Theta(\log n)$, sem especificar a base.

$O(a^n) = O(b^n)$, quaisquer que sejam as constantes $a, b \in \mathbb{R}^+$ com $1 < a < b$?

Falso pois, para $f(n) = b^n$ tem-se $f(n) \notin O(a^n)$ e, trivialmente, $f(n) \in O(b^n)$.

De facto, sabemos que, se $1 < a < b$ então $\lim_{n\to+\infty} \frac{b^n}{a^n} = \lim_{n\to+\infty} \left(\frac{b}{a}\right)^n = +\infty$.

Por definição de limite, isto significa que $\forall M \in \mathbb{R}^+\ \exists L \in \mathbb{Z}^+\ \forall n \geq L \quad \frac{b^n}{a^n} > M$.

Esta condição implica que $\forall M \in \mathbb{R}^+\ \forall n_0 \in \mathbb{Z}^+\ \exists n \geq n_0 \quad b^n > Ma^n$, ou seja, que $b^n \notin O(a^n)$, já que, dado $M \in \mathbb{R}^+$ e n_0, se tomarmos $n = \max(n_0, L) + 1$, satisfazemos $n \geq n_0$ e $\frac{b^n}{a^n} > M$ (ou seja, $b^n > Ma^n$).

Em **algoritmos exponenciais**, a base é importante. $\Theta(a^n) \neq \Theta(b^n)$, se $1 < a < b$.
https://jeffe.cs.illinois.edu/teaching/algorithms/notes/B-fastexpo.pdf

$\Omega(n \log_2 n) \cup \Theta(n^2) = O(n^2)$?

Falso, pois $O(1) \subset O(n^2)$ e nenhuma função de $O(1)$ pertence a $\Theta(n^2)$ nem a $\Omega(n \log_2 n)$.
De facto, por definição de $O(1)$, tem-se $\forall f \in O(1)\ \ \exists c \in \mathbb{R}^+\ \ \exists n_0 \in \mathbb{Z}^+\ \ \forall n \geq n_0\ \ f(n) \leq c$. Logo:

- para o mesmo n_0 e mesmo c, tem-se $f(n) \leq c \leq cn^2$, para todo $n \geq n_0$, pois $1 \leq n^2$ implica $c \leq cn^2$. Assim, $O(1) \subset O(n^2)$, sendo a inclusão estrita, pois $n^2 \in O(n^2)$ mas $n^2 \notin O(1)$.
- se $f(n) \in O(1)$ então $f(n) \notin \Omega(n^2)$, e portanto, $f(n) \notin \Theta(n^2)$, já que $\Theta(n^2) = \Omega(n^2) \cap O(n^2)$. Notar que, se $f(n) \in \Omega(n^2)$ então existia $c' \in \mathbb{R}^+$ tal que $f(n) \geq c'n^2$, para todo $n \geq n_0'$. Mas, como $c'n^2 > c$ para, por exemplo, todo $n > \lceil c/c' \rceil$, não podia existir n_0 nas condições acima referidas. Analogamente, se conclui que $f(n) \notin \Omega(n \log_2 n)$.

Prova alternativa: Falso porque $n^4 \in \Omega(n \log_2 n)$ pois $\lim_{n\to\infty} \frac{n^4}{n \log_2 n} = \infty$ e, portanto, por definição de união de conjuntos, $n^4 \in \Omega(n \log_2 n) \cup \Theta(n^2)$. Mas, $n^4 \notin O(n^2)$, porque $\lim_{n\to\infty} \frac{n^4}{n^2} = \infty$.

$O(an) = O(bn)$, quaisquer que sejam as constantes $a, b \in \mathbb{R}^+$?
Verdade pois:

- $O(an) \subseteq O(bn)$

  Por definição de $O(an)$, tem-se $f(n) \in O(an)$ então $\exists c \in \mathbb{R}^+\ \exists n_0 \in \mathbb{Z}^+\ \forall n \geq n_0\ \ f(n) \leq c(an)$.

  Para concluir que se $f(n) \in O(an)$ então $f(n) \in O(bn)$, note-se que $c(an) = c'(bn)$ se $c' = \frac{ca}{b}$.

  $\therefore\ \exists c \in \mathbb{R}^+\ \exists n_0 \in \mathbb{Z}^+\ \forall n \geq n_0\ f(n) \leq c(an) \ \Rightarrow\ \exists c' \in \mathbb{R}^+\ \exists n_0' \in \mathbb{Z}^+\ \forall n \geq n_0'\ f(n) \leq c'(bn)$.

  Tomamos $c' = \frac{ca}{b}$ e $n_0' = n_0$.

- $O(bn) \subseteq O(an)$. A prova é análoga.

  Qualquer que seja a função $f(n)$, se $f(n) \in O(bn)$ então $f(n) \in O(an)$, porque

  $$\exists c' \in \mathbb{R}^+\ \exists n_0' \in \mathbb{Z}^+\ \forall n \geq n_0'\ f(n) \leq c'(bn) \ \Rightarrow\ \exists c \in \mathbb{R}^+\ \exists n_0 \in \mathbb{Z}^+\ \forall n \geq n_0\ f(n) \leq c(an)$$

  já que podemos tomar $c = \frac{c'b}{a}$ e $n_0 = n_0'$.

Exercícios

1. Assuma que v e w são *arrays* que contêm uma sequência de inteiros não negativos que termina por 0 (e não tem outros zeros). A função COMPLEX(v, w) retorna um inteiro positivo, negativo, ou 0, se a sequência dada por v for lexicograficamente maior, menor, ou igual à sequência dada por w, respetivamente. Os elementos dos *arrays* são indexados a partir de 1. Sejam $a_1, a_2, \ldots, a_m, 0$ e $b_1, b_2, \ldots, b_n, 0$ os estados de $v[\cdot]$ e $w[\cdot]$, na linha 1, com $m \geq 1$ e $n \geq 1$. Recorde que os a_i's e b_i's são inteiros positivos.

```
COMPLEX(v, w)
1  j ← 1;
2  Enquanto (w[j] = v[j]  ∧  v[j] ≠ 0) fazer
3      j ← j + 1;
4  retorna v[j] − w[j];
```

a) Escreva o **invariante do ciclo 2-3** que nos permite concluir que a função está correta. A resposta deve começar por *"Quando estamos a testar a condição na linha 2 pela k-ésima vez…"*

b) Explique como é que o **invariante é mantido ao longo do ciclo** e como permite concluir que a **função retorna o valor correto** na linha 4.

c) Caraterize a **complexidade temporal assintótica da função**, em termos de m e n (pode usar $\min(m, n)$ ou $\max(m, n)$ na expressão, se necessário) . Comece por descrever instâncias que conduzam ao pior caso e ao melhor caso. Justifique a resposta: apresente um **modelo de custos** e as **expressões do tempo total de execução** segundo esse modelo, no melhor caso e pior caso. Na linha 2, atribua um tempo a cada uma das condições, incluindo nesses tempos a transferência de controlo.

2. O algoritmo apresentado à esquerda resolve o problema de imprimir a soma dos dois últimos valores de uma sequência de valores dada pelo utilizador, sendo lidos pelo menos dois valores além do valor -1, o qual indica que a sequência terminou. À direita, apresentamos um modelo de custos para as instruções indicadas em cada linha, sendo as constantes c_1, c_2, c_3 e c_4 números reais positivos.

ler($penultimo$);	c_1: leitura e atribuição do valor
ler($ultimo$);	c_1: leitura e atribuição do valor
ler($novo$);	c_1: leitura e atribuição do valor
Enquanto ($novo \neq -1$) fazer	c_2: avaliação da condição e transferências de controlo
$penultimo \leftarrow ultimo$;	c_3: atribuição do valor de uma variável simples a outra
$ultimo \leftarrow novo$;	c_3: atribuição do valor de uma variável simples a outra
ler($novo$);	c_1: leitura e atribuição do valor
escrever($penultimo + ultimo$);	c_4: avaliação da expressão e escrita do valor

a) Para o **modelo de custos** indicado, determine **a expressão que define o tempo de execução** do algoritmo quando aplicado a instâncias de tamanho n (onde n denota **o número <u>total</u> de inteiros lidos**. Justifique se trata de um algoritmo $\Theta(n)$.

b) Justifique que o algoritmo é **assintoticamente ótimo**, ou seja, que qualquer algoritmo que resolva o problema tem a mesma complexidade assintótica ou pior.

3. Considere o algoritmo seguinte, supondo que serão lidos pelo menos dois valores e todos são inteiros.

```
resposta ← 0;
ler(primeiro);
ler(x);
Enquanto (x ≠ −1) fazer
   Se (x = primeiro) então
      resposta ← resposta + 1;
   ler(x);
escrever(resposta);
```

a) Defina um modelo de custos.

b) Determine as expressões que definem o tempo de execução no **melhor caso** e no **pior caso**, supondo que serão lidos exatamente n valores, com $n \geq 2$. Comece por explicar o que distingue o melhor caso do pior caso, neste algoritmo.

c) Caraterize a sua complexidade temporal assintótica.

4. Seja $T_{(v,n,x)}(n)$ o tempo que a função seguinte requer quando aplicada a uma instância (v, n, x), supondo que $n \geq 1$ e $v[0]$ designa o primeiro elemento de v.

```
PROCURA(v, n, x)
   i ← 0;
   Enquanto (i < n ∧ v[i] ≠ x) fazer
      i ← i + 1;
   Se (i < n) então retorna i;
   retorna −1;
```

a) Justifique que, no melhor caso, $T_{(v,n,x)}(n) \in O(1)$.
b) Justifique que, no pior caso, $T_{(v,n,x)}(n) \in \Omega(n)$.
c) Averigue a veracidade de:

- $T_{(v,n,x)}(n) \in \Theta(n)$, qualquer que seja (v, n, x).
- O tempo máximo que a função demora em instâncias de tamanho n é $O(n)$ e, de facto, $\Theta(n)$.

5. Os algoritmos seguintes imprimem o número de valores dados antes de 0 pelo utilizador, supondo que dará uma sequência de inteiros (embora o Algoritmo B seja mais natural).

```
//Algoritmo B
c ← 0;
ler(x);
Enquanto (x ≠ 0) fazer
   c ← c + 1;
   ler(x);
escrever(c);
```

```
//Algoritmo D
c ← −1;
Repita
   ler(x);
   c ← c + 1;
até (x = 0);
escrever(c);
```

a) Defina um modelo de custos para os algoritmos apresentados.

b) Para cada um, determine a expressão que define o tempo de execução se forem lidos n valores, com $n \geq 1$.

c) Conclua que a complexidade temporal assintótica de ambos é $\Theta(n)$.

6. Admita que M uma matriz, com pelo menos n linhas e n colunas, indexadas a partir de 0. Considere o programa seguinte para averiguar se se trata de uma matriz simétrica, retornando 1 se for e 0 se não for.

```
SIMETRICA(M, n)
   i ← 1;
   Enquanto(i < n) fazer
      j ← 0;
      Enquanto(j < i) fazer
         Se (M[i][j] ≠ M[j][i]) então
            retorna 0;
         j ← j + 1;
      i ← i + 1;
   retorna 1;
```

a) Determine o tempo de execução da função no melhor caso e no pior caso, e indique a sua ordem de grandeza. Comece por definir o modelo de custos e a propriedade que carateriza as instâncias nas duas situações.

b) Indique os invariantes de ciclo que permitem demonstrar a correção do programa para o problema indicado (para o ciclo interno, admita i fixo). Demonstre-os.

7. **Usando a definição matemática das ordens de grandeza**, prove a veracidade ou falsidade de:

a) $50n^3 - 2n + 100 \in \Theta(\frac{1}{100}n^3)$.

b) $3n^2 - n + 10 \in \Omega(20n^2)$.

c) $10000n^3 - 2n^2 + 5 \notin O(n^3 + 4n)$.

d) $10000n^3 - 2n^2 + 5 \in \Omega(n^3 + 4n^2)$.

e) $10000n^3 - 2n^2 + 5 \in \Theta(50000n^3 + 4n^2 + 1000)$.

f) $\Theta(n^3) \neq \Theta(n^3 + 4n + 1000000)$.

g) $\Omega(1000 + \log_2 n) \cap O(\frac{1}{20000}\log_2 n) = \{\ \}$.

h) $\frac{1}{5}n + 30\log_2 n \notin \Omega(n^2)$.

i) $cn^4 \in O(n^3)$, para algum $c \in \mathbb{R}^+$ fixo (i.e., constante).

j) Qualquer que seja a função $f(n)$ tal que $f(n) \in O(1)$, o valor de $f(n)$ é constante.

k) $n \notin \Omega(2^{10}\log_2 n)$ pois $n < 2^{10}\log_2(n)$ para $1 < n \leq 2^{13}$ (de facto, para $1 < n \leq 14115$).

l) $\Omega(n^4) \cap \Theta(n^2 \log_2 n) = \{\ \}$.

m) $\Omega(n \log_2 n) \subseteq O(n^2)$.

n) $\Theta(n^2) \subset \Omega(n \log_2 n)$.

Os problemas 8, 9, e 10 são de testes/exames de 2018/19

8. Considere a função REMOVE(v, x, n) para retirar todos os valores maiores que v de um *array* x de n inteiros, compactando-o. O resultado ficará em $x[1], \ldots, x[t]$ e a função retorna t.

```
REMOVE(v, x, n)
  t ← 0
  p ← 1;
  Enquanto (p ≤ n) fazer
      Se x[p] ≤ v então
          t ← t + 1;
          x[t] ← x[p];
      p ← p + 1;
  retornar t;
```

a) Apresente um modelo de custos e a expressão que define o tempo de execução da função, no pior caso e no melhor caso.

b) Na continuação de **a)**, usando a definição de $\Theta(n)$, prove que o tempo de execução satisfaz $T(n) \in \Theta(n)$, para toda a instância com n elementos.

c) Indique um **invariante de ciclo** que permita provar a correção da função. Diga ainda como é se conclui que a função está correta usando esse invariante.

d) Prove o invariante que indicou, por indução matemática.

9. (**) Seja v um *array* em que cada elemento é um par $(valor, codigo)$, ambos inteiros positivos, com $1 \leq v[i].valor \leq k$, para $1 \leq i \leq n$. Considere a função seguinte supondo que COPIA(v, i, w, j) copia o conteúdo de $v[i]$ para $w[j]$ e tem complexidade $O(1)$, e p é um *array* auxiliar.

ORDENAR(v, n, k, w)

1 Para $i \leftarrow 1$ até k fazer
2   $p[i] \leftarrow 0;$
3 Para $i \leftarrow 1$ até n fazer
4   $p[v[i].valor] \leftarrow p[v[i].valor] + 1;$
5 Para $i \leftarrow 2$ até k fazer
6   $p[i] \leftarrow p[i] + p[i-1];$
7 $i \leftarrow n;$
8 Enquanto $(i \geq 1)$ fazer
9   COPIA$(v, i, w, p[v[i].valor]);$
10   $p[v[i].valor] \leftarrow p[v[i].valor] - 1;$
11   $i \leftarrow i - 1;$

a) Indique o estado do *array* p após a execução dos blocos 1–4 e 1–6 se $k = 7$, $n = 11$ e v for

5	1	4	3	4	1	5	6	4	5	3	$valor$
3	7	18	12	15	19	1	5	16	8	10	$codigo$

Qual é o estado final do *array* w para tal instância?

b) No caso geral, o que contém $p[i]$, para $1 \leq i \leq k$, após a execução do bloco 1–6?

c) À saída da função, o *array* w deve conter os elementos de v ordenados por **valor**. Indique o invariante do ciclo 8-11 que o permite concluir (deve caraterizar o estado de i e dos *arrays* v, w e p, quando a condição na linha 8 é testada pela q-ésima vez, para $q \geq 1$).

d) Caraterize a complexidade de ORDENAR(v, n, k, w) no melhor caso ______ e no pior caso ______, como função de k e n. Quando será vantajoso usar esta abordagem para ordenação em vez de MERGESORT?

10. Considere a função apresentada que determina os valores que se repetem consecutivamente num *array* v com n elementos, indicando-os em $r[\,]$ por ordem de ocorrência, juntamente com o número de vezes que se repetem. Retorna o número de valores nessas condições. Os arrays v e r são indexados a partir de 1.

REPETICOES(v, n, r)

1 $k \leftarrow 1; \quad j \leftarrow 0;$
2 Enquanto $(k < n)$ fazer
3   $c \leftarrow 0; \quad k \leftarrow k + 1;$
4   Enquanto $(k \leq n \ \wedge \ v[k] = v[k-1])$ fazer
5     $c \leftarrow c + 1; \quad k \leftarrow k + 1;$
6   Se $c \geq 1$ então
7     $r[j+1] \leftarrow v[k-1]; \quad r[j+2] \leftarrow c;$
8     $j \leftarrow j + 2;$
9 retornar $j/2;$

a) Indique o estado de c, k, j e dos *arrays* quando se executa instrução 6 pela 5ª vez se $v =$ `[1,1,3,4,3,3,6,6,6,6,2,2,5,6]`, com $n = 14$ (se for desconhecido, use `??`).

b) Caraterize a complexidade temporal da função no melhor caso ______ e no pior caso ______. No caso geral, para que valores de $p \in \mathbb{Z}_0^+$, a complexidade é $\Theta(n^p)$ ______, $\Omega(n^p)$ ______ ou $O(n^p)$ ______?

c) (**) Em geral, designe por $[a_1, a_2, \ldots, a_n]$, o estado de $v[\,]$ à entrada da função. Caraterize o estado das variáveis do programa quando se está a testar a condição na **linha 2** pela i-ésima vez. Caraterize ainda o estado para o t-ésimo teste da condição na **linha 4 na i-ésima iteração** do bloco 3-8 (para i-fixo). Como se podem usar essas condições para concluir que o programa está correto?