INSTITUTO UNIVERSITÁRIO DE LISBOA

Trabalho 2 – Análise de Complexidade e Recorrência

1.

```
sequentialSearch(S,k):
    //Input: sequência desordenada S, e um elemento, k
    //Output: Boolean

    for i= 1...len(S):
        if S[i]==k:
            return true
    return false
```

Explicação:

A estratégia utilizada na resolução deste problema é a *"Decrease and Conquer".* Este tipo de estratégia consiste na iteração sequencial sobre os elementos da sequência, avaliando em cada iteração se cada um destes corresponde ao elemento procurado.
No caso positivo, termina a procura e é retornado um valor true. Por outro lado, se o elemento não estiver na sequência, após iterar sobre todos os elementos, é retornado um valor *false*.

2.

Em cada iteração do algoritmo acima ocorre uma operação de soma (iterador "i" do ciclo for), um acesso (acesso ao elemento de índice "i" da sequência) e uma comparação.
O trabalho, em número de passos elementares, é dado por:
$T(n)=3(t-1+1)=3t$ onde t é o número de iterações ocorridas do ciclo for, $1<=t<=n$, e 'n' o tamanho da sequência ($n=len(S)$).

No melhor caso o elemento procurado corresponde ao elemento na primeira posição da sequência, isto é, ocorre apenas uma iteração no ciclo for:
$t=1$
$B(n)=3, O(3)=O(1)$ a ordem de complexidade é constante para qualquer tamanho de sequência

No pior caso o elemento de procura não se encontra na sequência, isto é, ocorrem tantas iterações quanto o tamanho da sequência:
$t=n$
$W(n)=3n,\ O(3n)=O(n)$ a ordem de complexidade é linear e depende do tamanho da sequência

No caso médio o elemento procurado corresponde ao elemento na posição $n/2$ da sequência, isto é, ocorrem $n/2$ iterações no ciclo for:
$t=n/2$
$A(n)=3(n/2), O(3n/2) = O(n)$ a ordem de complexidade é linear e depende do tamanho da sequência

Logo, no geral, a ordem de complexidade desta pesquisa é $O(n)$ para o limite superior e $\Omega(1)$ para o limite inferior.

3.

$$T(n) = \begin{cases} T(n-1) + 3, & n > 1 \\ 0, & n = 0 \end{cases}$$

$$\begin{aligned} T(n) &= T(n-1) + 3 \\ &= T(n-2) + 2*3 \\ &\;\;\ldots \\ &= T(n-k) + 3k \\ &= T(0) + 3n \\ &= 0 + 3n = 3n = O(n) \end{aligned}$$

índice de paragem : $k = n$

4.

```
binarySearch(S,L,H,K):
    //Input: sequência ordenada S, índice de inicio de procura L, índice de fim
de procura H e um elemento K
    //Output: Boolean

    M = (H+L)/2
    if L==H:
        return false
    if K<S[M]:
        binarySearch(S,L,M,K)
    else if K>S[M]:
        binarySearch(S,M,H,k)
    else:
        return true
```

Explicação:

A estratégia utilizada na resolução deste problema é a *"Decrease and Conquer".* Este tipo de estratégia, aplicada neste algoritmo, consiste numa divisão da sequência ordenada em duas subsequências. Depois, mediante uma avaliação dos índices L e H (índices de início e fim da sequência em que ocorreu a pesquisa) e dos valores do índice médio, entre L e H, e o valor procurado é feita uma paragem ou uma recursividade em apenas uma das metades resultantes.

5.

$$T(n) = \begin{cases} T\left(\frac{n}{2}\right) + f(n),\ n > 1,\ f(n) = O(7) \\ 7,\ n = 1 \end{cases}$$

$$\begin{aligned}
T(n) &= T\left(\frac{n}{2^1}\right) + f\left(\frac{n}{2^0}\right) \\
&= T\left(\frac{n}{2^2}\right) + f\left(\frac{n}{2^0}\right) + f\left(\frac{n}{2^1}\right) \\
&\ \ . \ . \ . \\
&= T\left(\frac{n}{2^k}\right) + \sum_{i=0}^{k-1} f\left(\frac{n}{2^i}\right) \qquad f(n) \leqslant 7,\ \textit{no caso limite vem que } f(n) = 7 \\
&= T\left(\frac{n}{2^k}\right) + 7(k - 1 + 1) \qquad \textit{índice de paragem}: \ k = log(n) \\
&= T(1) + 7log(n) \\
&= 7(1 + log(n)) = O(log(n))
\end{aligned}$$

6.

```
binarySearchIndex(S,L,H)
    //Input: sequência ordenada S, índice de inicio de procura L, índice de fim
de procura H e um elemento K
    //Output: Integer

    M = (L+H)/2

    if M == S[M]:
        return M;
    if L>=H:
        return -1;
    if S[M]>M:
        return binarySearchIndex(S,L,M-1);
    else if S[M]<M:
        return binarySearchIndex(S,M+1,H);
    return -1;
```

Explicação:

A estratégia utilizada na resolução deste problema é a *"Decrease and Conquer".* O algoritmo para a resolução do problema tem como principal funcionalidade descobrir se existe um número com valor igual ao índice, identificar se o mesmo se encontra na parte inicial ou na parte final da sequência. Repetido recursivamente a mesma estratégia até chegar a uma solução. Sendo assim, este algoritmo divide a sequência em metade e verifica se o valor encontrado no meio tem o índice igual ao seu valor. Caso o seu valor seja superior ao índice, sendo a sequência ordenada, se existir um valor igual ao seu índice terá de se encontrar na primeira parte da sequência, logo chama se recursivamente para a primeira parte da sequência. Caso contrário chama-se recursivamente para a segunda metade da sequência. Na hipótese de não existir nenhum elemento com o valor igual ao seu índice o algoritmo devolve -1.

$$T(n) = \begin{cases} T\left(\frac{n}{2}\right) + f(n),\ n > 1,\ f(n) = O(10) \\ 10,\ n = 1 \end{cases}$$

$$\begin{aligned} T(n) &= T\left(\frac{n}{2^1}\right) + f\left(\frac{n}{2^0}\right) \\ &= T\left(\frac{n}{2^2}\right) + f\left(\frac{n}{2^0}\right) + f\left(\frac{n}{2^1}\right) \\ &\ldots \\ &= T\left(\frac{n}{2^k}\right) + \sum_{i=0}^{k-1} f\left(\frac{n}{2^i}\right) \qquad f(n) \le 10,\ \textit{no caso limite vem que}\ f(n) = 10 \\ &= T\left(\frac{n}{2^k}\right) + 10(k-1+1) \\ &= T(1) + 10log(n) \qquad \textit{índice de paragem}:\ k = log(n) \\ &= 10(1 + log(n)) = O(log(n)) \end{aligned}$$

Este algoritmo tem como trabalho *T(n) = O(log(n)*. Pelo o Teorema Principal, sendo *a=1, b=2 e d=0*. Temos *T(n) = O(n^0*log(n)) = O(log(n)).*

Autores e Contribuições:

Contribuição. (%)	João Correia	Miguel Valadares	Fabian Gobet
Elaboração de pseudo-código	33.3	33.3	33.3
Alínea (2)	33.3	33.3	33.3
Alínea (3) e (4)	33.3	33.3	33.3
Alínea (5)	33.3	33.3	33.3
Alínea (6)	33.3	33.3	33.3