# **Trabalho 2 – Análise de Complexidade e Recorrência**

**1.**

*sequentialSearch(S,k):*

*//Input: sequencia 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 numero 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 medio 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 *𝛀(1)* para o limite inferior.

**3.**

Uma imagem com mesa

Descrição gerada automaticamente

**4.**

*binarySearch(S,L,H,K):*

*//Input: sequência ordenada S, indice de inicio de procura L, indice de fim de procura H e um elemento, K*

*//Output: Boolean*

*M = (H+L)/2*

*if L==H:*

*return false*

*if K<S[M]:*

*binarySerach(S,L,M,K)*

*else if K>S[M]:*

*binarySerach(S,M,H,k)*

*else:*

*return true*

Explicação:

A estratégia utilizada na resolução deste problema é a "*Divide and Conquer*".

Este tipo de estratégia consiste numa divisão da sequência ordenada em duas subsequências. Depois, mediante uma avaliação dos indices L e H (indices de inicio e fim da sequência em que ocorreu a pesquisa) e

dos valores do indice médio, entre L e H, e o valor procurado é feita uma paragem ou uma recursividade.

**5.**

Uma imagem com texto, recibo, documento

Descrição gerada automaticamente

**6.**

*binarySearchIndex(S,L,H)*

*//Input: sequência ordenada S, indice de inicio de procura L, indice de fim de procura H e um elemento, K*

*//Output: Boolean*

*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;*

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 |