AULA 4 – ANÁLISE DA COMPLEXIDADE DE ALGORITMOS

1 - Seja uma dada sequência (*array*) de n elementos inteiros e não ordenada. Pretende-se determinar quantos elementos da sequência respeitam a seguinte propriedade:

array [i] = array [i – 1] + array [i + 1], para 0 < i < (n – 1)

- Implemente uma **função eficiente** e **eficaz** que determine quantos elementos (resultado da função) de uma sequência com n elementos (sendo n > 2) respeitam esta propriedade.
  Depois de validar o algoritmo apresente a função no verso da folha.

- Pretende-se determinar experimentalmente a **ordem de complexidade do número de comparações** efetuadas pelo algoritmo e envolvendo elementos da sequência.

- Considere as seguintes sequências de 10 elementos inteiros, que cobrem algumas situações possíveis de execução do algoritmo.
  Determine, para cada uma delas, o número de elementos que obedecem à condição e o número de comparações efetuadas, envolvendo elementos da sequência.

1	2	3	4	5	6	7	8	9	10
1	2	1	4	5	6	7	8	9	10
1	2	1	3	2	6	7	8	9	10
0	2	2	0	3	3	0	4	4	0
0	0	0	0	0	0	0	0	0	0

Resultado	0
Resultado	1
Resultado	2
Resultado	6
Resultado	8

Nº de operações	8
Nº de operações	8
Nº de operações	8
Nº de operações	8
Nº de operações	8

Depois dos testes experimentais responda às seguintes questões:

- Em termos do número de comparações efetuadas, podemos distinguir alguma variação na execução do algoritmo? Ou seja, existe a situação de melhor caso e de pior caso, ou estamos perante um algoritmo com caso sistemático?

Estamos perante um algoritmo com caso sistemático, o número de comparações não varia com diferentes dados, apenas com quantidades diferentes de dados introduzidas. Não sendo possível reconhecer um melhor ou pior caso.

- Com base nos resultados experimentais, qual é a ordem de complexidade do algoritmo? Justifique.

A ordem de complexidade é linear (n) , para conjuntos de valores experimentais:
T(2n)/T(n) = ~2 .
Exemplos:
n = 5 -> T(10)/T(5) = 8/3 = 2,6(6) ; n= 10 -> T (20)/T(10) = 18/8 = 2,25 ;
n = 20 -> T(40)/T(20) = 38/18 = 2,1(1) ; n = 40 -> T(80)/T(40) = 78/38 = 2,05 ;

- Determine formalmente a ordem de complexidade do algoritmo. Tenha em atenção que deve obter uma expressão matemática exata e simplificada.
  Faça a análise no verso da folha.

- Calcule o valor da expressão para **n = 10** e compare-o com os resultados obtidos experimentalmente.

T(n) = n – 2
n = 10 -> T(10) = 10 -2 = 8 -> Este resultado vai de encontro ao obtido experimentalmente.

NOME: JOÃO EDUARDO DA SILVA FIGUEIREDO **Nº MEC: 93250**

FUNÇÃO

```
int condFind(int *array, int n)
{
   assert (n > 2);
   int oc = 0; // Número de Ocurrências da condição:
            //array [i] = array [i – 1] + array [i + 1], para 0 < i < (n – 1)
   nComp = 0;

   for (int i = 1; i < (n - 1); i++)
   {
      nComp++; //O Número de comparações é incrementado a cada comparação
      if (array[i] == array[i-1] + array[i+1])
      {
         oc++; // Caso a condição se verifique o número de ocorrências é incrementado.
      }
   }
   return oc;

}
```

ANÁLISE FORMAL DO ALGORITMO

T(n) = $\sum_{n=1}^{(n-1)-1} 1 = 1 * [(n-1)-1] = n-2$

O algoritmo é de complexidade linear.

2 - Seja uma dada sequência (*array*) de n elementos inteiros e não ordenada. Pretende-se determinar quantos ternos **(i, j, k)** de índices da sequência respeitam a seguinte propriedade:

array [k] = array [i] + array [j], para i < j < k

- Implemente uma **função eficiente** e **eficaz** que determine quantos ternos **(i, j, k)** de índices (resultado da função) de uma sequência com n elementos (sendo n > 2) respeitam esta propriedade.
  <u>Depois de validar o algoritmo apresente a função no verso da folha.</u>

- Pretende-se determinar experimentalmente a **ordem de complexidade do número de comparações** efetuadas pelo algoritmo e envolvendo elementos da sequência.

- Considere as sequências anteriormente indicadas de 10 elementos inteiros e outras sequências diferentes à sua escolha**; use sequências com 5, 10, 20, 30 e 40 elementos**. Determine, para cada uma delas, quantos ternos **(i, j, k)** de índices respeitam propriedade e o número de comparações efetuadas.

Depois dos testes experimentais responda às seguintes questões:

- Em termos do número de comparações efetuadas, podemos distinguir alguma variação na execução do algoritmo? Ou seja, existe a situação de melhor caso e de pior caso, ou estamos perante um algoritmo com caso sistemático?

Estamos perante um algoritmo com caso sistemático, o número de comparações não varia com diferentes dados, apenas com quantidades diferentes de dados introduzidas. Não sendo possível reconhecer um melhor ou pior caso.

- Com base nos resultados experimentais, qual é a ordem de complexidade do algoritmo? Justifique.

A ordem de complexidade é cúbica (n^3), para conjuntos de valores experimentais:
T(2n)/T(n) a tender para 8 , T(2n)/T(n) = 8
Exemplos:
n = 5 -> T(10)/T(5) = 120/10 = 12 ;
n= 10 -> T (20)/T(10) = 1140/120 = 9,5 ;
n = 15 -> T(30)/T(15) = 4060/455 = 8,92 ;
n = 20 -> T(40)/T(20) = 9880/ 1140 = 8,6(6) ;

- Determine formalmente a ordem de complexidade do algoritmo. Tenha em atenção que deve obter uma expressão matemática exata e simplificada.
  <u>Faça a análise no verso da folha.</u>

- Calcule o valor da expressão para **n = 10** e compare-o com os resultados obtidos experimentalmente.

$$T(N) = \frac{n^3}{6} - \frac{n^2}{2} + \frac{n}{3}$$
$$N = 10 \rightarrow T(10) = \frac{10^3}{6} - \frac{10^2}{2} + \frac{10}{3} = 120$$

Este resultado vai de encontro ao obtido experimentalmente.

FUNÇÃO

```
int condFind(int *array, int n)
{
   assert (n > 2);
   int oc = 0; // Número de Ocurrências da condição:
            //array [k] = array [i] + array [j], para i < j < k
   nComp = 0;

   for (int k = 2; k < n; k++)
   {
      for (int j = 1; j < k; j++)
      {
         for (int i = 0; i < j; i++)
         {
            nComp++; //O Número de comparações é incrementado a cada comparação
            if (array [k] == array [i] + array [j])
            {
               oc++; // Caso a condição se verifique o número de ocorrências é incrementado.
            }
         }
      }
   }

   return oc;

}
```

ANÁLISE FORMAL DO ALGORITMO

$$
\begin{aligned}
T(N) &= \sum_{k=2}^{n-1}\left[\sum_{j=1}^{k-1}\left(\sum_{i=0}^{j-1} 1\right)\right] = \sum_{k=2}^{n-1}\left(\sum_{j=1}^{k-1} j\right) = \sum_{k=2}^{n-1}\frac{1}{2}k\,(\,n-1\,) \\
&= \sum_{k=1}^{n-1}\frac{1}{2}k\,(\,k-1\,) - \sum_{k=1}^{2-1}\frac{1}{2}k\,(\,k-1\,) = \sum_{k=1}^{n-1}\frac{k^2}{2} - \sum_{k=1}^{n-1}\frac{k}{2} - \sum_{k=1}^{1}\frac{k^2-k}{2} \\
&= \frac{1}{2}*\frac{1}{6}\,(n-1)\big((n-1)+1\big)(2(n-1)+1) - \frac{1}{2}*\frac{1}{2}\,(n-1)(n-1+1) - 0 \\
&= \frac{1}{12}(n-1)n(2n-1) - \frac{1}{4}(n-1)n = \frac{2n^3-6n^2+4n}{12} = \frac{n^3}{6} - \frac{n^2}{2} + \frac{n}{3}
\end{aligned}
$$

O algoritmo é de complexidade cúbica.