Aula 4 – Análise da Complexidade de Algoritmos

**\*\*\* Entregue, num ficheiro ZIP, este guião preenchido e o código desenvolvido \*\*\***

**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 |  | Resultado | 0 |  | Nº de operações | 8 |
| 1 | 2 | 1 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |  | Resultado | 1 |  | Nº de operações | 8 |
| 1 | 2 | 1 | 3 | 2 | 6 | 7 | 8 | 9 | 10 |  | Resultado | 2 |  | Nº de operações | 8 |
| 0 | 2 | 2 | 0 | 3 | 3 | 0 | 4 | 4 | 0 |  | Resultado | 6 |  | Nº de operações | 8 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |  | Resultado | 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?

|  |
| --- |
| Para cada valor de n, o algoritmo compara todos os elementos do array, com o seu anterior e o seu posterior, fazendo, portanto, sempre n – 2 comparações.  Após os testes experimentais podemos concluir que estamos perante um algoritmo com caso sistemático, uma vez que, variando as sequências e, consequentemente, o resultado, o número de operações não é alterado, sendo sempre 8 (neste caso).  Sendo assim, não existe o melhor nem pior caso, uma vez que se faz sempre o mesmo número de comparações. |

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

|  |
| --- |
| A ordem de complexidade do algoritmo é linear, O(n), pois é “processado” todos os elementos do array. |

* 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.

|  |
| --- |
| Podemos verificar que o número de comparações obtido experimentalmente, para n = 10 elementos, foi sempre 8. |

Função

// numOps -> nº de comparações efetuadas

int check ( int\* array , int n) {  
 int i;  
 int result = 0;  
 for ( i = 1 ; i < n - 1 ; i++) {  
 if (array[i] == array[i-1] + array[i+1]) {  
 result++;  
 }  
 numOps++;  
 }  
 return result;  
}

Análise Formal do Algoritmo

|  |
| --- |
| T(n) =  Como estamos na presença de um algoritmo com caso sistemático é importante referir que: |

**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.

**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 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?

|  |
| --- |
| Após os testes experimentais podemos concluir que estamos perante um algoritmo com caso sistemático, uma vez que, variando as sequências e, consequentemente, o resultado, o número de operações não é alterado, sendo, por exemplo sempre 120 (para n = 10).  Sendo assim, não existe o melhor nem pior caso, uma vez que se faz sempre o mesmo número de comparações. |

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

|  |
| --- |
| A ordem de complexidade do algoritmo é cúbica,  (Através do código da função, deparamo-nos com 3 ciclos For) |

* 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 (10) = n.(n-1).(n-2) / 6 = 120  Podemos verificar que o número de comparações obtido experimentalmente, para n = 10 elementos, foi sempre 120. |

Função

// numOps -> nº de comparações efetuadas

int check ( int\* array , int n) {  
 int i;  
 int j;  
 int k;  
 int result = 0;  
 for ( k = 2 ; k < n ; k++) {  
 for (j = 1; j < k ; j++) {  
 for (i = 0; i < j ; i++) {  
 if (array[k] == array[i] + array[j]) {  
 result++;  
 }  
 numOps++;  
 }  
 }  
 }  
 return result;  
}

Análise Formal do Algoritmo

|  |
| --- |
| T(n) =        T(n) =    Como estamos na presença de um algoritmo sistemático é importante referir que: |