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?

|  |
| --- |
| O número de comparações depende apenas do número de elementos que o array tem e não depende do conteúdo dos elementos em si, ou seja, da sua ordenação. Não existe melhor ou pior caso, todos os arrays de n elementos irão ter o mesmo número de operações. |

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

|  |
| --- |
| Com base nos resultados a ordem de complexidade é de O(n), pois para um arrays com 10 elementos obtemos 8 comparações, para arrays com 4 elementos obtemos 2 comparações e arrays com 6 elementos obtemos 4 comparações. Chegamos a uma tendência de nComp = n – 2 onde n é o número de 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.

|  |
| --- |
| Usando a expressão obtida na Análise Formal do Algoritmo, para um n = 10 => nops = 10 – 2 = 8, valor que está de acordo com os valores experimentais obtidos |

Função

|  |
| --- |
| int func(int array[], int size)  {      assert(size > 2 && "The array must have 3 elements or more.");      int res = 0;    // Number of elements that verify the condition      // 0 < i < (n –1)      for (int i = 1; i < size - 1; i++)      {          nComp++;  // Compares every element that has a neighbor          // Elem = sum of neighbors          if(array[i] == (array[i - 1] + array[i + 1]))          {              res++;          }      }      return res;  } |

Análise Formal do Algoritmo

|  |
| --- |
| nops -> Número de operações  arrayX -> Array de X elementos  nops(array10) = 8  nops(array6) = 4  nops(array4) = 2  nops(array3) = 1  Este algoritmo compara todos os elementos que têm vizinhos, isto é, desde o elemento de índice 1 até ao elemento de índice n - 2, onde n é o tamanho do array. Visto que todos os elementos só têm 1 par de vizinhos a operação é executada apenas uma vez para cada elemento que tem vizinhos. Daqui retira-se  Daqui retira-se 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?

|  |
| --- |
| Tal como visto no algoritmo anterior, também conseguimos ver neste algoritmo que a disposição dos elementos no array, ou seja, a sua ordenação não altera os valores de comparações. Sendo assim este algoritmo também não tem melhor e pior caso, o número do comparações depende apenas do número de elementos que o array contém, independentemente da sua ordenação |

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

|  |
| --- |
| nops -> Número de operações arrayX -> Array de X elementos  nops(array5) = 10 nops(array10) = 120  nops(array20) = 1140 nops(array30) = 4060 nops(array40) = 9880  nops(array10)/nops(array5) = 12 nops(array20)/nops(array10) = 9,5 nops(array40)/nops(array20) = 8,6  Podemos ver que a ordem de operações está a tender para o valor 8. Sendo assim podemos dizer que o algoritmo é cúbico, ou seja, O(n3) |

* 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.

|  |
| --- |
| Usando a expressão obtida na Análise Formal do Algoritmo, para um n = 10 => nops =  , valor que está de acordo com os valores experimentais obtidos |

Função

|  |
| --- |
| int func(int array[], int size)  {      assert(size > 2 && "The array must have 3 elements or more.");      int res = 0;    // Number of elements that verify the condition        // i < j < k      for (int k = 2; k < size ; k++)      {          for (int j = 1; j < k; j++)          {              for (int i = 0; i < j; i++)              {                  nComp++;                  if(array[k] == (array[i] + array[j]))                  {                      res++;                  }              }          }      }      return res;  } |

Análise Formal do Algoritmo

|  |
| --- |
| Seguindo o mesmo raciocínio do exercício 1, passamos os *for loops* para somatórios onde k vai de 2 até size – 1 (inclusive), j vai de 1 até k – 1 (inclusive) e i vai de 0 até j – 1 (inclusive).  De seguida metemos um “1” dentro do somatório mais interno porque a comparação é feita um única vez cada vez que o *for loop* mais interno é executado. |