Aula 5 - Análise da Complexidade de Algoritmos

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

**1 –** Considere uma sequência (*array*) de **n** **valores reais**. Pretende-se determinar se os elementos da sequência são sucessivos termos de uma **progressão geométrica**:

**r = a[ 1 ] / a[ 0 ]** e **a[ i ] = r × a[ i – 1 ], i > 1.**

* Implemente uma função **eficiente** (utilize um algoritmo em lógica negativa) e **eficaz** que verifique se os n elementos (n > 2) de uma sequência de valores reais são sucessivos termos de uma progressão geométrica. A função deverá devolver 1 ou 0, consoante a sequência verificar ou não essa propriedade.

**Depois de validar o algoritmo apresente a função no verso da folha.**

* Pretende-se determinar experimentalmente a **ordem de** **complexidade do número de multiplicações e divisões** efetuadas pelo algoritmo e envolvendo elementos da sequência.
* Considere as seguintes sequências de 10 elementos, que cobrem as distintas situações possíveis de execução do algoritmo. Determine, para cada uma delas, se satisfazem a propriedade e qual o número de operações de multiplicação e de divisão efetuadas pelo algoritmo.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |  | Resultado | 0 |  | Nº de operações | 2 |
| 1 | 2 | 4 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |  | Resultado | 0 |  | Nº de operações | 3 |
| 1 | 2 | 4 | 8 | 5 | 6 | 7 | 8 | 9 | 10 |  | Resultado | 0 |  | Nº de operações | 4 |
| 1 | 2 | 4 | 8 | 16 | 6 | 7 | 8 | 9 | 10 |  | Resultado | 0 |  | Nº de operações | 5 |
| 1 | 2 | 4 | 8 | 16 | 32 | 7 | 8 | 9 | 10 |  | Resultado | 0 |  | Nº de operações | 6 |
| 1 | 2 | 4 | 8 | 16 | 32 | 64 | 8 | 9 | 10 |  | Resultado | 0 |  | Nº de operações | 7 |
| 1 | 2 | 4 | 8 | 16 | 32 | 64 | 128 | 9 | 10 |  | Resultado | 0 |  | Nº de operações | 8 |
| 1 | 2 | 4 | 8 | 16 | 32 | 64 | 128 | 256 | 10 |  | Resultado | 0 |  | Nº de operações | 9 |
| 1 | 2 | 4 | 8 | 16 | 32 | 64 | 128 | 256 | 512 |  | Resultado | 1 |  | Nº de operações | 9 |

**Depois da execução do algoritmo responda às seguintes questões:**

* Qual é a sequência (ou as sequências) que corresponde(m) ao melhor caso do algoritmo?

|  |
| --- |
| 1ª sequência {1, 2, 3, 4, 5, 6, 7, 8, 9, 10} porque o 3º elemento não verifica logo a primeira condição (array[3] = array[2] \* r) |

* Qual é a sequência (ou as sequências) que corresponde(m) ao pior caso do algoritmo?

|  |
| --- |
| São as duas últimas sequências, {1, 2, 4, 8, 16, 32, 64, 128, 256, 10} e {1, 2, 4, 8, 16, 32, 64, 128, 256, 512}, pois é verificado todos os elementos de cada array. Neste exemplo, a última sequência corresponde a uma progressão geométrica, enquanto na penúltima a condição não se verifica no último elemento do array. |

* Determine o número de operações efetuadas no caso médio do algoritmo (**para n = 10**).

|  |
| --- |
| Uma vez que as sequências são equiprováveis, o caso médio de operações é:  A(n) = (2+3+4+5+6+7+8+9+9) / 9 = 5,8(8) ≅ 6 operações |

* Qual é a ordem de complexidade do algoritmo?

|  |
| --- |
| O algoritmo tem complexidade linear, O(n), pois verifica-se que W(n) = n – 1. |

* **Determine formalmente a ordem de complexidade do algoritmo nas situações do melhor caso, do pior caso e do caso médio, considerando uma sequência de tamanho n.** Deve obter expressões matemáticas exatas e simplificadas. **Faça essas análises no verso da folha.**

Função

|  |
| --- |
| double check(float\* array, int n) {   assert (n > 2);  int i;  float r;  r = array[1] / array[0];  numOps++;   for ( i = 2 ; i < n ; i++) {  numOps++;  if(array[i] != r \* array[i-1]){  return 0;  }  }  return 1; } |

Análise Formal do Algoritmo

|  |
| --- |
| Worst Case:  Todos os elementos são comparados  Best Case:  Acontece quando o primeiro elemento não verifica a condição  Average Case: ) =  Nota: Para todas as sequências é feita uma operação para calcular a razão, pelo que, sendo assim, é adicionado 1 a todos os somatórios. |

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

|  |
| --- |
| Worst Case = 10 – 1 = 9  Best Case = 2  Average Case = 5,8(8)  Conclui-se que os resultados obtidos experimentalmente coincidem com os valores calculados |

**2 –** Considere uma sequência (array), possivelmente não ordenada, de n elementos inteiros e positivos. Pretende-se **eliminar os elementos da sequência que sejam iguais ou múltiplos ou submúltiplos de algum dos seus predecessores**, sem fazer a sua ordenação e sem alterar a posição relativa dos elementos.

Por exemplo, a sequência {2, 2, 2, 3, 3, 4, 5, 8, 8, 9 } com 10 elementos será transformada na sequência { 2, 3, 5 } com 3 elementos; e a sequência { 7, 8, 2, 2, 3, 3, 3, 8, 8, 9 } com 10 elementos será transformada na sequência { 7, 8, 3, } com 3 elementos.

* Implemente uma função **eficiente** e **eficaz** que elimina os elementos iguais ou múltiplos ou submúltiplos de algum dos seus predecessores numa sequência com n elementos (n > 1). **A função deverá ser *void* e alterar o valor do parâmetro indicador do número de elementos efetivamente armazenados na sequência (que deve ser passado por referência)**.

**Depois de validar o algoritmo apresente a função no verso da folha.**

* Pretende-se determinar experimentalmente a **ordem de** **complexidade do número de comparações** e **do número de deslocamentos** (i.e., cópias) efetuados pelo algoritmo e envolvendo elementos da sequência.
* Considere as sequências anteriormente indicadas de 10 elementos e outras à sua escolha. Determine, para cada uma delas, a sua configuração final, bem como o número de comparações e de deslocamentos efetuados.

**Depois da execução do algoritmo responda às seguintes questões:**

* Indique uma sequência inicial com 10 elementos que conduza ao **melhor caso do número de comparações** efetuadas. Qual é a sequência final obtida? Qual é o número de comparações efetuadas? Qual é o número de deslocamentos (i.e., cópias) de elementos efetuados? Justifique.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Inicial: | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |  | Nº de comparações | 9 |
| Final: | 1 |  |  |  |  |  |  |  |  |  |  | Nº de cópias | 36 |

|  |
| --- |
| Uma vez que nesta sequência todos os elementos são iguais, apenas é comparado o primeiro elemento com os outros restantes, sendo que vão sendo eliminados após essa comparação. |

* Indique uma sequência inicial com 10 elementos que conduza ao **pior caso do número de comparações** efetuadas. Qual é a sequência final obtida? Qual é o número de comparações efetuadas? Qual é o número de deslocamentos (i.e., cópias) de elementos efetuados? Justifique.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Inicial: | 3 | 5 | 7 | 11 | 13 | 17 | 23 | 29 | 31 | 33 |  | Nº de comparações | 46 |
| Final: | 3 | 5 | 7 | 11 | 13 | 17 | 23 | 29 | 31 | 33 |  | Nº de cópias | 0 |

|  |
| --- |
| Qualquer sequência que só contém números primos conduz ao pior caso do nº de comparações, pois todos os elementos terão que ser comparados com os seus predecessores, não havendo nenhuma “eliminação”. |

* Determine formalmente a ordem de complexidade do algoritmo nas situações do **melhor caso** e do **pior caso**, considerando uma sequência de tamanho n. Deve obter expressões matemáticas exatas e simplificadas. **Faça essas análises no verso da folha.**

Função

|  |
| --- |
| void checkArray(int \*array , int \*size) {   int i;  int j;  int k;   assert(\*size > 1);   for (i = 0 ; i < \*size ; i++) {  for (j = i + 1; j < \*size; j++) {   numComps++;   if (array[i] == array[j] || array[i] % array[j] == 0 || array[j] % array[i] == 0 ) {   (\*size)--;  for ( k = j ; k < \*size ; k++ ) {  numShifts++;  array[k] = array[k+1];  }  j--;  }  }  } } |

Análise Formal do Algoritmo – Comparações – Melhor Caso – Pior Caso

|  |
| --- |
| Best Case B(n) = n – 1  (Todos os elementos do array são iguais)  Worst Case W(n) =  (Todos os elementos são nºs primos) |

Análise Formal do Algoritmo – Deslocamentos – Melhor Caso – Pior Caso

|  |
| --- |
| Best Case B(n) = 0  (Todos os elementos são primos e diferentes, não é preciso deslocar nenhum)  Worst Case W(n) = |