|  |  |  |  |
| --- | --- | --- | --- |
| **Nome:** | **Raquel Resende Milheiro Pinto** | **N.º Mec:** | **92948** |

**Aula 4 - Análise da Complexidade de Algoritmos**

**1 –** Considere uma sequência (*array*) de n elementos inteiros, ordenada por **ordem não decrescente**. Pretende-se determinar se a sequência é uma **progressão aritmética de razão 1**, i.e., a[i+1] – a[i] = 1.

* Implemente uma função **eficiente** (utilize um algoritmo em lógica negativa) e **eficaz** que verifique se uma sequência com n elementos (n > 1) define uma sequência contínua de números. A função deverá devolver 1 ou 0, consoante a sequência verificar ou não essa propriedade. **Depois de validar o algoritmo apresente-o no verso da folha.**
* Determine experimentalmente a **ordem de** **complexidade do número de adições/subtrações** efetuadas pelo algoritmo e envolvendo elementos da sequência. Considere as seguintes 10 sequências de 10 elementos inteiros, todas diferentes, e que cobrem as distintas situações possíveis de execução do algoritmo. Determine, para cada uma delas, se satisfaz a propriedade e qual o número de operações de adição/subtração efetuadas pelo algoritmo.

|  |  |  |
| --- | --- | --- |
| **Sequência** | **Resultado** | **N.º de operações** |
| {1, 3, 4, 5, 5, 6, 7, 7, 8, 9} | 0 | 1 |
| {1, 2, 4, 5, 5, 6, 7, 8, 8, 9} | 0 | 2 |
| {1, 2, 3, 6, 8, 8, 8, 9, 9, 9} | 0 | 3 |
| {1, 2, 3, 4, 6, 7, 7, 8, 8, 9} | 0 | 4 |
| {1, 2, 3, 4, 5, 7, 7, 8, 8, 9} | 0 | 5 |
| {1, 2, 3, 4, 5, 6, 8, 8, 9, 9} | 0 | 6 |
| {1, 2, 3, 4, 5, 6, 7, 9, 9, 9} | 0 | 7 |
| {1, 2, 3, 4, 5, 6, 7, 8, 8, 9} | 0 | 8 |
| {1, 2, 3, 4, 5, 6, 7, 8, 9, 9} | 0 | 9 |
| {1, 2, 3, 4, 5, 6, 7, 8, 9, 10} | 1 | 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?

|  |
| --- |
| Das sequências dadas a que corresponde ao melhor caso é {1, 3, 4, 5, 5, 6, 7, 7, 8, 9}.  Em geral, será qualquer sequência cuja diferença entre o segundo e o primeiro seja diferente de um, ou seja, quando a condição if(a[1] - a[0] != 1) for falsa. Com isto, só fazemos uma operação. |

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

|  |
| --- |
| Das sequências dadas as que correspondem ao pior caso são (2): {1, 2, 3, 4, 5, 6, 7, 8, 9, 9} e {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}.  Em geral, será qualquer sequência cuja diferença seja sempre um ou cuja diferença apenas seja 0 entre os últimos dois elementos do array. Assim fazemos n – 1 comparações, para este caso temos nove comparações. |

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

|  |
| --- |
| Caso Médio: A(10) =  R: O número de adições efetuadas para n=10 no caso medio são 5. |

* Qual é a ordem de complexidade do algoritmo?

|  |
| --- |
| A ordem de complexidade deste algoritmo é O(n), ou seja, é um algoritmo com ordem de complexidade linear. |

* 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. Tenha em atenção que deve obter expressões matemáticas exatas e simplificadas.

|  |
| --- |
| **Análise Formal do Algoritmo** |
| **Melhor Caso - B(n) =** 1, pois a condição if(a[1]-a[0]!=1) é verdade. |
| **Pior Caso - W(n) =** |
| **Caso Médio - A(n) =**  , para grandes valores de N e considerando que os acontecimentos são equiprováveis. |

Efetuando a analise formar deste algoritmo, podemos ver que tem ordem de

complexidade linear – O(N).

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

|  |
| --- |
| Melhor Caso: B(10) = 1  Pior Caso: W(10)= 10-1 = 9  Caso Médio: A(10) =  Experimentalmente a sequência para o melhor caso é {1, 3, 4, 5, 5, 6, 7, 7, 8, 9},  B(10) = 1, tal como o valor calculado.  Para o pior caso as sequências eram {1, 2, 3, 4, 5, 6, 7, 8, 9, 9} e {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}, com o nove operações, tal como o valor calculado.  Quanto ao caso médio será aproximadamente 5, tanto obtido pela expressão como experimentalmente. |

|  |
| --- |
| **Apresentação do Algoritmo** |
| static int count = 0; // numero de operacoes  int isArithProg(int a[] , int n){  assert (n>1);  count = 0; // numero de operacoes  for(int i = 0;i<n-1;i++){  count++;  if(a[i+1]-a[i]!=1){  return 0;  }  }  return 1;  } |

**2 –** Considere uma sequência (array) não ordenada de n elementos inteiros. Pretende-se eliminar os elementos repetidos existentes na sequência, sem fazer uma pré-ordenação e sem alterar a posição relativa dos elementos. Por exemplo, a sequência { 1, 2, 2, 2, 3, 3, 4, 5, 8, 8 } com 10 elementos será transformada na sequência { 1, 2, 3, 4, 5, 8 } com apenas 6 elementos. Por exemplo, a sequência { 1, 2, 2, 2, 3, 3, 3, 3, 8, 8 } com 10 elementos será transformada na sequência { 1, 2, 3, 8 } com apenas 4 elementos. Por exemplo, a sequência { 1, 2, 3, 2, 1, 3, 4 } com 7 elementos será transformada na sequência { 1, 2, 3, 4 } com apenas 4 elementos. Mas, a sequência { 1, 2, 5, 4, 7, 0, 3, 9, 6, 8 } permanece inalterada.

* Implemente uma função **eficiente** e **eficaz** que elimina os elementos repetidos 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-o no verso da folha.**

* Determine experimentalmente a **ordem de** **complexidade do número de comparações** e **do número de deslocamentos** 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.

|  |  |  |  |
| --- | --- | --- | --- |
| **Sequência inicial** | **Número de cópias/ deslocamentos** | **Número de comparações** | **Sequência final** |
| {1, 2, 2, 2, 3, 3, 4, 5, 8, 8} | 17 | 28 | { 1, 2, 3, 4, 5, 8 } |
| {1, 2, 2, 2, 3, 3, 3, 3, 8, 8} | 22 | 23 | { 1, 2, 3, 8 } |
| {1, 2, 3, 2, 1, 3, 4, 4, 2, 1} | 18 | 23 | { 1, 2, 3, 4 } |
| {1, 1, 1, 1, 1, 1, 1, 1, 1, 1} | 36 | 9 | {1} |
| {3, 8, 5, 4, 7, 5, 4, 7, 2, 8} | 9 | 33 | {3, 8, 5, 4, 7, 2} |
| {9, 4, 8, 4, 8, 4, 8, 2, 5, 2} | 18 | 24 | {9, 4, 8, 2, 5} |
| {10, 5, 8, 4, 7, 5, 4, 10, 9 , 8} | 9 | 32 | {10, 5, 8, 4, 7, 9} |
| {4, 5, 5, 5, 6, 6, 6, 8, 8, 8} | 21 | 24 | {4, 5, 6, 8} |
| {1, 2, 5, 4, 7, 0, 3, 9, 6, 8} | 0 | 45 | {1, 2, 5, 4, 7, 0, 3, 9, 6, 8} |

Para a sequência de indicada em cima com sete elementos:

{1, 2, 3, 2, 1, 3, 4} tem 6 copias e 13 comparações, sendo {1, 2, 3, 4} a sequência final.

Podemos reparar que quando temos menos deslocamentos temos mais comparações e quando temos mais deslocamentos temos menos comparações.

**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?

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Inicial: | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |  | N.º de comparações: | 9 |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Final: | 1 |  |  |  |  |  |  |  |  |  |  | N.º de Cópias: | 36 |

Justifique a sua resposta:

|  |
| --- |
| O que acontece com um array que tem os elementos todos iguais é que o algoritmo fixa o primeiro elemento comparando-o com o segundo, remove o segundo (faz deslocamento), depois compara o primeiro com o segundo elemento (antigo terceiro antes do deslocamento) removendo-o também e assim sucessivamente. Assim, neste caso, o array fica com menos um elemento a cada comparação. Conclui-se que para este caso o número de comparações é igual ao número de elementos menos um. |

* 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?

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Inicial: | 1 | 2 | 5 | 4 | 7 | 0 | 3 | 9 | 6 | 8 |  | N.º de comparações: | 45 |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Final: | 1 | 2 | 5 | 4 | 7 | 0 | 3 | 9 | 6 | 8 |  | N.º de Cópias: | 0 |

Justifique a sua resposta:

|  |
| --- |
| O que acontece com um array que tem os elementos todos diferentes é que o algoritmo compara o segundo elemento com o primeiro, não removendo (não faz deslocamento), depois compara o terceiro elemento com o primeiro e o segundo, também não remove o terceiro elemento e assim sucessivamente até ao final do array. Ou seja, o array não fica mais pequeno isso faz com que existem mais comparações. |

* Determine formalmente a ordem de complexidade do algoritmo nas situações do **melhor caso** e do **pior caso**, considerando uma sequência de tamanho n. Tenha em atenção que deve obter expressões matemáticas exatas e simplificadas.

|  |
| --- |
| **Análise Formal do Algoritmo - Número de Comparações** |
| **Melhor Caso - B(n) =**  O melhor caso acontece quando j é sempre igual a 0, ou seja, quando os elementos do array são todos iguais. |
| **Pior Caso - W(n) = = = =**  O pior caso, como vimos anteriormente, acontece quando os elementos do array são todos diferentes. |

Efetuando a analise formar deste algoritmo usando o número de comparações, podemos ver que tem ordem de complexidade quadrática – O().

|  |
| --- |
| **Análise Formal do Algoritmo - Número de Deslocamentos de Elementos** |
| **Melhor Caso - B(n) =**  O melhor caso acontece quando os elementos do array são todos diferentes, ou seja, quando a condição if(a[i]==a[j]) é sempre falsa. Para o melhor caso do número de deslocamentos de elementos, o algoritmo tem complexidade constante → O(1). |
| **Pior Caso - W(n) =**  O pior caso dos deslocamentos é quando o array tem os elementos todos iguais, logo ao executar o algotimo vamos deslocar sucessivamente menos elementos, ou seja, alteramos o número de n à medida que avançamos no algoritmo. Sendo assim alteramos as condições do ciclo (número de elementos do array).  Experimentando com números diferentes de elementos do array (N), podemos ver que:  Para N = 1 ou 2 não existe deslocamentos  N = 3 → 1 deslocamento N = 4 → 3 deslocamentos N = 5 → 6 deslocamentos N = 6 → 10 deslocamentos N = 7 → 15 deslocamentos N = 8 → 21 deslocamentos N = 9 → 28 deslocamentos N = 10 → 36 deslocamentos  que corresponde a expressão |

Efetuando a analise formar deste algoritmo quando ao nível do número de deslocamentos de elementos, podemos ver que tem ordem de complexidade quadrática – O().

|  |
| --- |
| **Apresentação do Algoritmo** |
| static int ncopias = 0; // numero de copias -> numero de deslocamentos  static int ncomparacoes = 0; // numero de comparacoes  void RetiraRepetidos(int\* a, int\* n){ // vetor original e numero de elementos  assert (\*n>1);  ncomparacoes=0; // numero de comparacoes  ncopias=0; // numero de copias -> numero de deslocamentos  for(int i = 1; i<\*n;i++){  for(int j=0;j<i;j++){  ncomparacoes++;  if(a[i]==a[j]){  for(int k = i;k<\*n-1;k++){ //deslocamentos\copias  ncopias++;  a[k] = a[k+1];  }  i--;  (\*n)--;  }  }  }  } |