|  |  |  |  |
| --- | --- | --- | --- |
| **NMec:** | **93313** | **Nome:** | **Daniel Baptista Andrade** |

**Aula 3 – Análise da Complexidade de Algoritmos**

**1 -** Seja uma dada sequência (*array*) de n elementos inteiros. Pretende-se determinar quantos elementos da sequência são diferentes do seu elemento anterior. Ou seja:

array [i] ≠ array [i–1], para i > 0

* Implemente uma **função** **eficiente** e **eficaz** que determine quantos elementos (resultado da função) de uma sequência com n elementos (sendo n > 1) respeitam esta propriedade.

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

* Determine experimentalmente a **ordem de complexidade do número de comparações** 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, o número de elementos que obedecem à condição e o número de comparações efetuadas.

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

**Depois da execução do algoritmo 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?
* R: Não existe um melhor/pior caso na execução deste algoritmo. Desta forma, o número de operações é sempre igual a n-1, sendo n o número de elementos do array.
* Qual é a ordem de complexidade do algoritmo?
* R: O algoritmo apresenta uma ordem de complexidade linear ( O(n) ).
* 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.

**Apresentação do Algoritmo**

|  |
| --- |
| unsigned static int op,sz = 0;  int counter(int array[]){  op = 0;  int result=0;  for(int i =1; i<sz; i++){  if(array [i] != array [i-1]){  result++;  }  op++;  }  return result;  }  int main(void){  ...  sz = sizeof(array)/sizeof(int);  ...  } |

**Análise Formal do Algoritmo**

**E(n) =**

**2 -** Seja uma dada sequência (*array*) de n elementos inteiros e não ordenada. Pretende-se determinar qual é o primeiro elemento da sequência que tem mais elementos menores do que ele atrás de si, e indicar a posição (índice do *array*) onde esse elemento se encontra.

Por exemplo, na sequência { 1, 9, 2, 8, 3, 4, 5, 3, 7, 2 } o elemento 7, que está na posição de índice 8 da sequência, é maior do que 6 elementos seus predecessores. Na sequência { 1, 7, 4, 6, 5, 2, 3, 2, 1, 0 } o elemento 6, que está na posição de índice 3 da sequência, é maior do que 2 elementos seus predecessores. Mas, na sequência { 2, 2, 2, 2, 2, 2, 2, 2, 2, 2 } nenhum elemento é maior do que qualquer um dos seus predecessores, pelo que deve ser devolvido -1 como resultado.

* Implemente uma **função** **eficiente** e **eficaz** que determine o índice do primeiro elemento (resultado da função) de uma sequência com n elementos (sendo n > 1) que tem o maior número de predecessores menores do que ele.

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

* Determine experimentalmente a **ordem de complexidade do número de comparações** efetuadas envolvendo elementos da sequência. Considere as sequências anteriormente indicadas de 10 elementos inteiros e outras sequências diferentes à sua escolha. Determine, para cada uma delas, o índice do elemento procurado e o número de comparações efetuadas.

**Depois da execução do algoritmo 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?
* R1 → Default Version: Não existe melhor nem pior caso;
* R2 → Optimized Version: Pior Caso ocorre quando todos os elementos são iguais e/ou o array se encontra ordenado por ordem decrescente; Melhor Caso ocorre quando a última posição do array contém o número de numeros mais pequenos igual ao indice da sua posição.
* Qual é a ordem de complexidade do algoritmo?
* 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.

**Apresentação do Algoritmo**

|  |
| --- |
| \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_DEFAULT VERSION\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  unsigned int size, index, iteracoes = 0;  int counter(int array[]){  int index = 0;  int count = 0;  int amnt = -1;  iteracoes = 0;  for(int i=0;i<size;i++){  count = 0;  for(int j=0;j<i;j++){  iteracoes++;  if(array[j]<array[i]){  count++;  }  }  if(count>=amnt){  amnt=count;  index = i;  }  }  if(amnt == 0){  return -1;  }  return index;  }  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_OPTIMIZED VERSION\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  unsigned int size, index, iteracoes = 0;  int counter(int array[]){  iteracoes = 0;  index = 0;  int amnt = -1;  int i = size-1;  for(i; i>0; i--){ //indice  int count = 0;  for(int j =0; j< i; j++){ //valores  if(amnt > i){ //qntdd de nmrs menores de um ind. for > que o ind. a verificar  return index;  }  iteracoes++;  if(array[j]<array[i]){  count++;  }  }  if(count >= amnt){  amnt = count;  index = i;  }  }  if(amnt == 0){  return -1;  }  return 0;  }  int main(void){  …  size = sizeof(array)/sizeof(int);  …  } |

**Análise Formal do Algoritmo**

**E(n) =**