André Lyra Fernandes- BV303139X

BCC - IFSP - Listex2 - PDF 2

Análise de três algoritmos - maxMin1, maxMin2 e maxMin3 - todos desenvolvidos para encontrar o maior e o menor valor em um array. A análise considera apenas o pior cenário, onde o algoritmo se comporta diante de conjuntos de dados mais desfavoráveis possível.

Análise da função **maxMin1**:

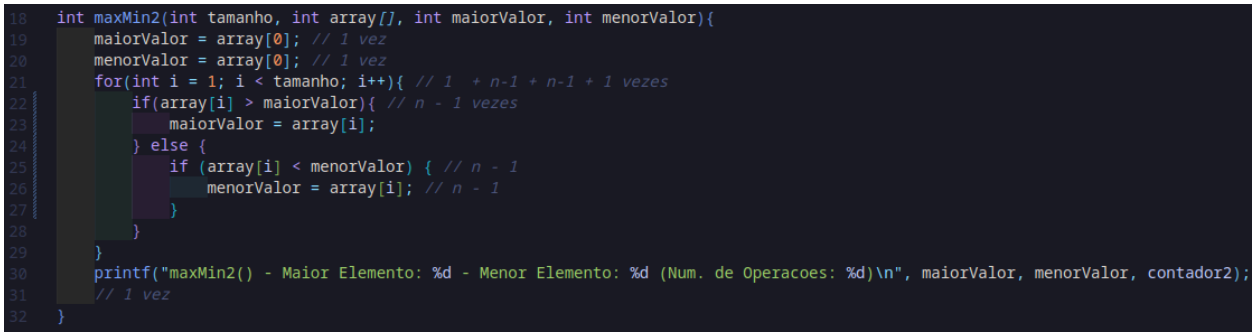
```
3  int maxMin1 (int tamanho, int array[], int maiorValor, int menorValor){
4      maiorValor = array[0]; // 1 vez
5      menorValor = array[0]; // 1 vez
6      for(int i = 1; i < tamanho; i++){  // 1 + (n-1) + (n-1) + 1
7          if(array[i] > maiorValor){ // n - 1 vezes
8              maiorValor = array[i]; // n - 1 vezes
9          }
10         if(array[i] < menorValor){ // n - 1 vezes
11             menorValor = array[i]; //
12         }
13     }
14     printf("maxMin1() - Maior Elemento: %d - Menor Elemento: %d (Num. de Operacoes: %d)\n", maiorValor, menorValor, contador1);
15     // 1 vez
16 }
```

1. Na linha 4 e 5, maiorValor e menorValor é atribuído ao primeiro elemento do array, contabilizando **2** execuções.
2. Na linha 6, a estrutura de repetição “for” é executada, contabilizando **1** execução para a inicialização, **2(n-1)** para a condição e incremento, e **1** execução para a atualização. Vale considerar que n-1 ocorre visto que o “for” é iniciado em 1.
3. Dentro do loop:
    - Na linha 7, dentro do primeiro “if”, é feita uma comparação **n-1** vezes, que se verdadeira, é executada **n-1** vezes.
    - Na linha 10, dentro do segundo “if”, é feita uma outra comparação. Como apenas um dos dois pode retornar verdadeiro, apenas faz a comparação e não é executada, assim, é executada **n-1 vezes**
4. Na linha 14, temos uma impressão formatada, contabilizando **1** execução.

Assim, a análise conclui que é um algoritmo linear, visto que:

$$T(n) = 1 + 1 + 1 + 2(n - 1) + 1 + 3(n - 1) + 1 = 5 + 2n - 2 + 3n - 3 = 5n$$

Análise da função **maxMin2**:



```
18  int maxMin2(int tamanho, int array[], int maiorValor, int menorValor){
19      maiorValor = array[0]; // 1 vez
20      menorValor = array[0]; // 1 vez
21      for(int i = 1; i < tamanho; i++){ // 1  + n-1 + n-1 + 1 vezes
22          if(array[i] > maiorValor){ // n - 1 vezes
23              maiorValor = array[i];
24          } else {
25              if (array[i] < menorValor) { // n - 1
26                  menorValor = array[i]; // n - 1
27              }
28          }
29      }
30      printf("maxMin2() - Maior Elemento: %d - Menor Elemento: %d (Num. de Operacoes: %d)\n", maiorValor, menorValor, contador2);
31      // 1 vez
32  }
```

1. Na linha 19 e 20, maiorValor e menorValor são atribuídos ao primeiro elemento do array[0], contabilizando **2** execuções.
2. Na linha 21, a estrutura de repetição “for” é executada, contabilizando **1** execução para a inicialização, **2(n-1)** para a condição e incremento, e **1** execução para a atualização. Vale considerar que n-1 ocorre visto que o “for” é iniciado em 1.
3. Dentro do loop:
    - Na linha 22, executa-se **n-1** para ver se a instrução é verdadeira. Por se tratar do pior caso, nenhum retorna verdadeiro e prossegue-se para o “else”.
    - No bloco “else”, é feita uma comparação com execução **n-1**, e quando for verdadeira (pior caso), retorna outro **n-1.**
4. Finalmente, na linha 30, temos uma impressão formatada, contabilizando 1 execução.

Assim, a análise conclui que é um algoritmo linear, visto que:

$T(n)= 1 + 1 + 1 + 2(n-1) + 1 + (n-1) + (n-1) + (n-1) + 1 = 5 + 2n - 2 + 3n - 3 = 5n$

Análise da função **maxMin3**:

```
34  int maxMin3 (int tamanho, int Array[], int maiorValor, int menorValor){
35      if (tamanho % 2 != 0) { // 1 vez
36          Array[tamanho + 1] = Array[tamanho]; // 1 vez
37          tamanho++; // 1 vez
38      }
39      maiorValor = Array[1]; // 1 vez
40      menorValor = Array[0]; // 1 vez
41
42      if (Array[0] > Array[1]) { // 1 vez
43          maiorValor = Array[0]; // 1 vez
44          menorValor = Array[1]; // 1 vez
45      }
46
47      for (int i = 2; i < tamanho; i += 2) { // 1 + 2(((n - 2)) / 2) + 1 vezes
48          if (Array[i] > Array[i + 1]) { // (n - 2) / 2 vezes
49              if (Array[i] > maiorValor) { // (n - 2) / 2 vezes
50                  maiorValor = Array[i]; // (n - 2) / 2 vezes
51              }
52              if (Array[i + 1] < menorValor) { // (n - 2) / 2 vezes
53                  menorValor = Array[i + 1]; // (n - 2) / 2 vezes
54              }
55          } else {
56              if (Array[i + 1] > maiorValor) {
57                  maiorValor = Array[i + 1];
58              }
59              if (Array[i] < menorValor) {
60                  menorValor = Array[i];
61              }
62          }
63      }
64      printf("maxMin3() - Maior Elemento: %d - Menor Elemento: %d (Num. de operacoes: %d)\n", maiorValor, menorValor, contador3);
65      // 1 vez
66  }
```

1. Na linha 35, verificamos se o tamanho n é ímpar, contabilizando **1** execução.
2. Se o tamanho for ímpar, na linha 36 e 37, ocorre uma atribuição e um incremento, contabilizando **2** execuções.
3. Na linha 39 e 40 ocorre duas atribuições, contabilizando **2** execuções.
4. Na linha 42 verifica-se se a o maior e menor valor estão invertidos, por se tratar do pior caso, contabiliza **3** execuções.
5. Na linha 9, verificamos se o primeiro elemento é maior que o segundo no array, contabilizando **1** execução.
6. Dentro do loop for:
    - Na linha 47, o loop é executado **1 + 2((n - 2)) / 2) + 1** (2x por iniciar int i = 2, e dividido por 2 por ter um incremento de 2 em 2).
    - Dentro do loop, na linha 48, temos um bloco “if” que é executado **(n - 2) / 2** vezes.
    - Com o primeiro bloco “if” retornando verdadeiro, caso os outros dois retornem verdadeiro (linha 49 e 52), **4((n-2)/2)** execuções ocorrem.
7. Na linha 63, tem-se a impressão formatada, contabilizando 1 execução.

Assim, a análise conclui que é um algoritmo linear, visto que:

$$T(n) = 11 + 2((n - 2) / 2) + 4((n - 2) / 2) = n + ((4n - 8) / 2) + 9 = 3n + 5$$