

INSTITUTO FEDERAL DE SÃO PAULO
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

FERNANDA MARTINS DA SILVA

TEMPO DE EXECUÇÃO ($T(n)$)

Exercício 02 listex 02 - Projeto e Análise de Algoritmos

São João da Boa Vista/SP

MELHOR e PIOR caso

Na perspectiva da FIGURA 1 e da FIGURA 2:

O pior caso ocorre quando o elemento máximo é encontrado no final do *array* e o elemento mínimo é encontrado no início. Isso resulta em um número máximo de iterações do loop. O melhor caso ocorre quando o elemento máximo é encontrado no início do *array*, e o elemento mínimo é encontrado logo em seguida. Isso significa que o *loop* precisa ser executado apenas uma vez.

Para calcular $T(n)$, onde n é o tamanho do array, precisamos considerar o número de operações realizadas. Cada iteração do *loop* executa duas comparações (para verificar o máximo e o mínimo), portanto, o número total de operações é proporcional ao número de elementos no *array*, n .

Na perspectiva da FIGURA 3:

O melhor caso ocorre quando o menor e o maior elementos estão nos primeiros índices do *array*. Além disso, o tamanho do *array* é ímpar (não havendo necessidade de realizar a condição no *if* inicial para tornar o tamanho do *array* par). O pior caso ocorre quando o menor e o maior elementos estão nos últimos índices do *array*. Além disso, o tamanho do *array* é par (portanto, é necessário adicionar um elemento ao *array* no *if* inicial).

obs: está sendo desconsiderado do cálculo todos os printf presentes no final de todas as funções.

FIGURA 1

Começando pela FIGURA 1, iremos analisar o PIOR e o MELHOR caso possível.

Para o PIOR caso temos um $T(n) = 3 + (n-2) + 3(n-1)$, simplificando, $4n-2$. Já para o MELHOR caso temos $T(n) = 3 + (n-2) + 2(n-1)$, simplificando, $3n-1$.

FIGURA 2

Analisando a FIGURA 2 para o MELHOR e PIOR caso, obtém-se:

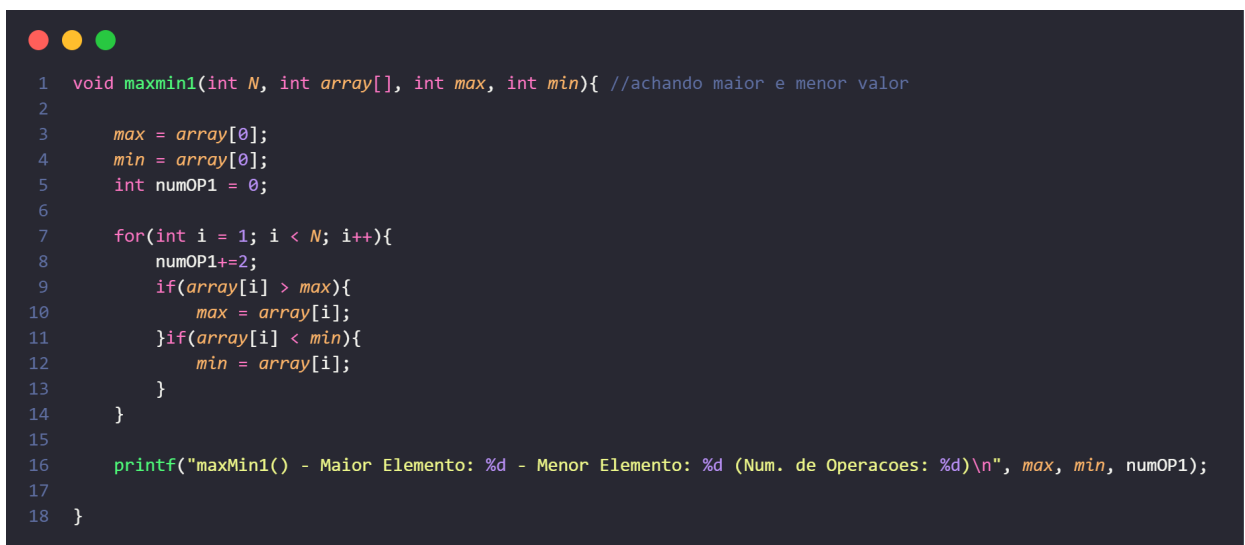
Para o PIOR caso $T(n) = 3 + 3(n-1)$, simplificando, $3n$. Para o MELHOR caso, $T(n) = 3 + 2(n-1)$, que simplificado, $1+2n$.

FIGURA 3

Agora, analisando a FIGURA 3 para o MELHOR e PIOR caso:

No pior caso, o número de operações aumenta linearmente com o tamanho do *array*. Isso ocorre porque todas as operações fora do *loop* são executadas uma vez e o *loop* é executado $n/2$ vezes, mas dentro do *loop*, há um aumento do número de operações quando o tamanho do *array* é par (já que o *if* inicial duplica o array). $T(n)$ então seria $9 + (n-3) + 5((n-2)/2)$, simplificando, $(2+7n)/2$.

No melhor caso, o número de operações é constante, independentemente do tamanho do *array*. Isso ocorre porque todas as operações fora do *loop* são executadas uma vez e o *loop* é executado $n/2$ vezes (já que o *loop* é incrementado de 2 em 2). Neste caso $T(n) = 5 + (n-3) + 3((n-2)/2)$, que simplificando dá o resultado de $5n/2$.

FIGURAS


```

1 void maxmin1(int N, int array[], int max, int min){ //achando maior e menor valor
2
3     max = array[0];
4     min = array[0];
5     int numOP1 = 0;
6
7     for(int i = 1; i < N; i++){
8         numOP1+=2;
9         if(array[i] > max){
10             max = array[i];
11         }if(array[i] < min){
12             min = array[i];
13         }
14     }
15
16     printf("maxMin1() - Maior Elemento: %d - Menor Elemento: %d (Num. de Operacoes: %d)\n", max, min, numOP1);
17
18 }

```

FIGURA 1

```

1 void maxmin2(int N, int array[], int max, int min){ //achando maior e menor valor pela segunda vez, mas de maneira diferente
2
3     max = array[0];
4     min = array[0];
5     int numOP2 = 0;
6
7     for(int i = 1; i < N; i++){
8         numOP2++;
9         if(array[i] > max){
10             max = array[i];
11         }else if(array[i] < min){
12             min = array[i];
13         }
14     }
15
16     printf("maxMin2() - Maior Elemento: %d - Menor Elemento: %d (Num. de Operacoes: %d)\n", max, min, numOP2);
17
18 }

```

FIGURA 2

```

1 void maxmin3(int N, int array[], int max, int min){ //achando maior e menor valor pela terceira vez, novamente de
2
3     int numOP3 = 0;
4
5     if(N%2 != 0){
6         array[N+1] = array[N];
7         N = N + 1;
8     }
9
10    max = array[0];
11    min = array[1];
12
13    if(array[0] < array[1]){
14        max = array[1];
15        min = array[0];
16    }
17
18    for(int i = 2; i < N; i += 2){
19        numOP3+=3;
20        if(array[i] > array[i+1]){
21            if(array[i] > max){
22                max = array[i];
23            }
24        }else{
25            if(array[i] < min){
26                min = array[i];
27                numOP3++;
28            }if(array[i+1] > max){
29                max = array[i+1];
30                numOP3++;
31            }
32        }
33    }
34
35    printf("maxMin3() - Maior Elemento: %d - Menor Elemento: %d (Num. de Operacoes: %d)\n", max, min, numOP3);
36
37 }

```

FIGURA 3