INSTITUTO FEDERAL DE SÃO PAULO

BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

FERNANDA MARTINS DA SILVA

ANÁLISE DE CORRETUDE, FINITUDE E COMPLEXIBILIDADE DE TEMPO

Exercício 03 listex 02 - Projeto e Análise de Algoritmos

São João da Boa Vista/SP

Corretude

Como mostra a FIGURA 1, a função *imprimeArray* contém um *for* que é inicializado em zero, também é menor que o tamanho *N* (variável que guarda o tamanho do *array*, fornecido pelo usuário), e o incremento (*i++)*. *N* é o limitante da função, quando *i* atinge o valor de *N*, a função é encerrada.

Ainda na FIGURA 1 temos a função *buscaArray*, que segue os mesmos princípios anteriormente comentados sobre o *for*, a diferença é a variável *v*, fornecida pelo usuário. A função varre o *array*, procurando pelo mesmo valor *v,* o comparando em um *if* (se o valor na posição *i* do *array* for igual ao número *v*), retorna suas propriedades/ele mesmo (valor e posição), sendo sua condição de parada, caso contrário, retorna NULL caso o valor *v* não apareça no *array*.

Finitude

Analisando a função *imprimeArray* presente na FIGURA 1, no conceito de finitude, observa-se que sua sequência (ordem) e se dão com base em *N.*

Já na função *buscaArray,* também presente na FIGURA 1, vemos o *for* varrer o *array* em função de *v*, que, na condição *if* verdadeira, encerra o laço retornando que a condição existe - e que foi encontrada. Caso falso, após varrer todo o *array* e fazer todas as comparações do *if*, retorna NULL.

Complexibilidade de tempo

A FIGURA 2 traz as mesmas linhas de código, agora, em um tom claro de roxo, uma contagem simplificada de complexibilidade da função.

Analisando a função *imprimeArray*, percebe-se que sua complexibilidade é linear (n + 1), sendo *n* vezes para o *for* e *1* vez para o *printf.* Seu tempo de execução também é linear com base no tamanho do *array*.

Já a função *buscaArray* tem uma complexibilidade de n+n+1+1 (ou 2n+2), ou seja, também apresenta linearidade. *N* vezes para o *for*, *n* vezes para o *if*, duas vezes para ambos os *return* (uma para cada). Seu tempo de execução também é linear com base no tamanho do *array*.

```c
void imprimeArray(int N, int array[]){

    printf("Array Gerado: ");
    for(int i = 0; i < N; i++){
        printf(" %d ", array[i]);
    }

}

int *buscaArray(int N, int array[], int v){

    for(int i = 0; i < N; i++){
        if(array[i] == v){
            return &array[i];
        }
    }

    return (int*)NULL;

}
```

FIGURA 1

```c
void imprimeArray(int N, int array[]){

    printf("Array Gerado: "); // 1x
    for(int i = 0; i < N; i++){ //Nx
        printf(" %d ", array[i]);
    }

}

int *buscaArray(int N, int array[], int v){

    for(int i = 0; i < N; i++){ //Nx
        if(array[i] == v){ //Nx
            return &array[i]; //1x
        }
    }

    return (int*)NULL; //1x

}
```

FIGURA 2