Complexidade de Códigos

3 Passos para calcular a complexidade:

1 – Levar em consideração apenas as repetições do código

2 – Verificar a complexidade das funções/métodos próprios da linguagem (se utilizado)

3 – Ignorar as constantes e utilizar o termo de maior grau

Exemplo 1:

```
boo1 exemplo1(vector<int> v, int X){
        int tamanho = v.size();
        for (int i=0; i<tamanho; i++){ // O(n)
                if(v[i] == X) return true;
        }
        return false;
}
```

1 – Existe apenas uma repetição nesse código, naquele for onde ele repete tamanho vezes , onde tamanho é o tamanho, portando , essa repetição é um O(n)

2 – A funções/métodos que tem complexidades são:

int tamanho = v.size(); // O(1)

if(v[i] == X) return true; // O(1)

return false; // O(1)

A primeira linha citada, o vetor é constante

A segunda linha apenas por ser comparação também é constante

A terceira linha também retorna constante

3 -

OBS: Tudo que é constante nesse código: O(1) é ignorado.

Portanto, sobra apenas o O(n) no código e conclui-se que a complexidade do código é O(n).

Exemplo 2:

```
boo1 exemplo2(vector<int> v){
        int tamanho = v.size(); // O(1)
        for (int i=0; i<tamanho; i++){ // O(n)
                for(int j=0; j<tamanho; j++){ // O(n)
                        if( != j && v[i] == v[j]) // O(1)
                                return true; // O(1)
                }
        }
        return false;
}
```

A partir desse exemplo, fazemos os mesmos esquemas citados,

Encontra a complexidade de cada linha e depois ignora as constantes, sobrando apenas as seguintes linhas de códigos:

```
for (int i=0; i<tamanho; i++){ // O(n)
        for(int j=0; j<tamanho; j++){ // O(n)
```

Depois disso como sobrou dois O(n), é preciso multiplica-los

$O(n) * O(n) \rightarrow O(n^2)$

Portanto, a complexidade desse código é $O(n^2)$

Exemplo 3:

```
int exemplo3(vector <int> v){
    int tamanho = v.size();
    int bla = 0;
    for(int i=0; i<tamanho; i++){ // O(n)
        for (int j=0; j<tamanho; j++){ //O(n)
            if(v[i] == v[j]) bla++
        }
    }

int ble = 0;
for (int i=0; i<tamanho; i++){ // O(n)
    if(v[i] == 10){
        ble = 2*ble;
    }
```

```
}

int bli = 0;
for(int i=0; i<tamanho; i++){ // O(n)
        if(v[i] == 5){
                bli += 5;
        }
}
return bla+ble+bli;
}
```

Para ficar mais fácil, pegamos apenas os O(n) em cada **for** e nem botamos a constantes, é coletado neste código:

O(n) * O(n) + O(n) + O(n)

Transforma em: $O(n^2) + 2O(n)$

Ignorando as constantes: $O(n^2) + O(n)$

Utilizar o termo de maior grau: $O(n^2)$ <- resposta final

Exemplo 4:

```
bool exemplo4(vector<int> v, vector<int> w){
        int tamanho = v.size();
        int tamanho2 = w.size();
```

```
    for(int i=0; i<tamanho; i++){ // O(n)
        for(int j=0; j<tamanho2; j++){ // O(m)
            if(v[i] == v[j])
                return true;
        }
    }
    return false;
}
```

Os dois for não vão ser dois O(n) pois eles chamam dois valores diferentes nas suas linhas de código, por isso são O(n) e O(m)

O resultado final seria: O(n) * O(m)

Exemplo 5:

```
bool exemplo5(vector<int> idades){
    int tamanho = idades.size();
    int menor_idade = 200;
    for(int i=0; i<tamanho; i++){ // O(n)
        if(idades[i] < menor_idade){
            menor_idade = idades[i];
        }
    }
    Int cont = 0;
```

```
        for(int i=0; i<tamanho; i++){ // O(n)
                if(v[i] == menor_idade){
                        cont++;
                }
        }
        return cont > 1;
}
```

O(n) + O(n)

2O(n)

O(n) <- complexidade do código acima

```
boo1 exemplo6(vector<int> idades){
        sort(idades.begin(), idades.end()); // O(NLogN)
        return idades[0] == idades[1];
}
```

O código acima não tem repetição, mas tem um método sort que vale O(NLogN)

Comparando, os dois códigos que fazem a mesma coisa, mesmo o segundo tendo poucas linhas, ele acaba sendo pior.

Exemplo 7:

```
bool exemplo7(set<int> s, vector<int> v){
	int tamanho = v.size();
	for(int i=0; i<tamanho; i++){ // O(n)
		if(s.count(v[i])) return true; // O(LogN)
	}
	return false;
}

// O(n) * O(LogN)
// O(NLogN)
```

Neste código tem uma repetição que vale O(n), e uma um método count da linguagem que vale LogN, as duas linhas s e compõem por isso são multiplicadas.