Contagem de Operações em Algoritmos

Exemplos práticos (f1 e f2)

Profa. Andréa Aparecida Konzen

Função f1(n)

```
int f1(n) {
    r = 0;
    for (i = 1; i < n; i++) {
        r = r + 1;
    }
    return r;
}
```

Contagem de Operações

O laço for executa de i = 1 até i < n, ou seja, **(n - 1)** vezes.
Dentro do laço, há uma operação de soma simples: r = r + 1.

Total de operações:

Operações de atribuição (r = r + 1): **(n - 1)**
A complexidade cresce linearmente com n.

Complexidade:

Classe de complexidade: O(n) (linear) – vamos trabalhar nas próximas aulas

Função f2(n)

```
int f2(n) {
   r = 0;
   for (i = 1; i < n; i++) {
      for (j = i + 1; j < n; j++) {
         r = r + 2;
      }
   }
   return r;
}
```

Contagem de Operações

Vamos analisar quantas vezes o comando r = r + 2 será executado.

Para cada i de 1 até n - 1, j vai de i + 1 até n - 1.

Número de iterações internas para cada i:

Quando i = 1, j = 2 a n - 1 ⇒ **(n - 2)** vezes

Quando i = 2, j = 3 a n - 1 ⇒ **(n - 3)** vezes

...

Quando i = n - 2, j = n - 1 ⇒ **1** vez

Quando i = n - 1, j = n → laço não executa

Total de operações:

Soma de uma PA

Como r = r + 2 é constante, podemos apenas contar quantas vezes é executado:
≈ $(n^2 - 3n + 2)/2$ vezes

Complexidade:

Classe de complexidade: $O(n^2)$ (quadrática) – vamos trabalhar nas próximas aulas

Por que contar operações só dentro dos laços?

```
int f1(n) {
   r = 0; // operação 1
   for (i = 1; i < n; i++) {// configuração do laço (contamos só se necessário)
      r = r + 1; // operação principal
   }
   return r; // operação final (constante)
}
```

A parte que **mais contribui para o custo total** (especialmente quando n é grande) é o corpo do for, ou seja, o r = r + 1.

Por isso:

Operações fora de laços (como r = 0 ou return r) são **executadas uma única vez** → custo constante, não cresce com n.

Já a operação **dentro do laço** é executada **(n - 1) vezes**, ou seja, cresce conforme n.

Por que contar operações só dentro dos laços?

E o próprio for? Ele não conta?

O for tem três partes:

```
for (i = 1; i < n; i++) {
  // corpo do laço
}
```

i = 1: executado **1 vez** (inicialização)

i < n: testado **n vezes** (inclusive uma última que falha)

i++: executado **(n - 1) vezes**

Essas também são operações, mas em análise assintótica (como Big-O) – vamos estudar nas próximas aulas, **não nos importamos com pequenas diferenças**, como contar 3 operações por iteração em vez de 1.

Estamos interessados em saber **como o custo cresce**, então focamos **na operação dominante** — ou seja, o que acontece **dentro do laço**.

Por que contar operações só dentro dos laços?

Regra prática

- Ignore operações **fora dos laços** (constantes).
- Dentro dos laços, **conte as repetições das operações que mais se repetem**.
- Se houver laços aninhados (for dentro de for), o número de repetições cresce mais rápido → atenção a isso!