Estruturas de Dados

Análise de Complexidade

2025/2026

Análise Temporal

- Quanto tempo demora esta função?

```
int contaPar(int m[])
{
int contador=0;
for(int i=0;i<m.length;i++){
        int base=m[i];
        for(int j=i+1;j<m.length;j++)
            if(base==m[j])
                    contador++;
        }
return contador;
}
```

Análise Temporal

- Quanto tempo demora esta função?
  - Vamos pressupor que sabemos quanto tempo demora cada linha…

	Custo (tempo) de Execução
int contaPar(int m[])	
{	
int contador=0;	*C1*
for(int i=0;i<m.length;i++){	*C2*
int base=m[i];	*C3*
for(int j=i+1;j<m.length;j++)	*C4*
if(base==m[j])	*C5*
contador++;}	*C6*
return contador;	*C7*
}	

Análise Temporal

- Quanto tempo demora esta função?

	Custo de Execução	*Número de Execuções (N=m.length)*
int contaPar(int m[])		
{		
int contador=0;	*C1*	*1*
for(int i=0;i<m.length;i++){	*C2*	*N+1*
int base=m[i];	*C3*	*N*
for(int j=i+1;j<m.length;j++)	*C4*	$1+\sum_{i=1}^{N}\sum_{j=i+1}^{N}1$
if(base==m[j])	*C5*	$\sum_{i=1}^{N}\sum_{j=i+1}^{N}1$
contador++;}	*C6*	*0..* $\sum_{i=1}^{N}\sum_{j=i+1}^{N}1$
return contador;	*C7*	*1*
}		

Análise Temporal

O número de execuções da 5ª linha é dado por:

$$\sum_{i=1}^{N}\sum_{j=i+1}^{N}1 = \overbrace{1+2+3+\ldots+N-1}^{\text{i=N-1}} = \sum_{i=1}^{N-1} i$$

Para obter uma formula fechada para a expressão, vamos recorrer à igualdade conhecida:

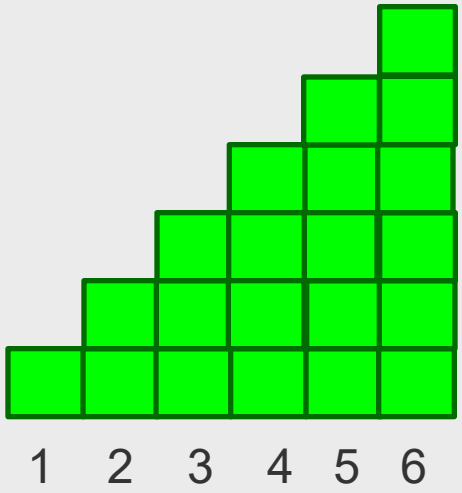
$$\sum_{i=1}^{N} i = 1+2+3+\ldots+N = \frac{1}{2}N(N+1)$$

Análise Temporal

- Esta última relação aparece frequentemente porque está associada a ciclos encadeados.

$$\sum_{i=1}^{N} i = 1+2+3+...+N = \frac{1}{2}N(N+1)$$

- Uma forma de visualizar esta relação é a seguinte:



Exemplo para N=6.

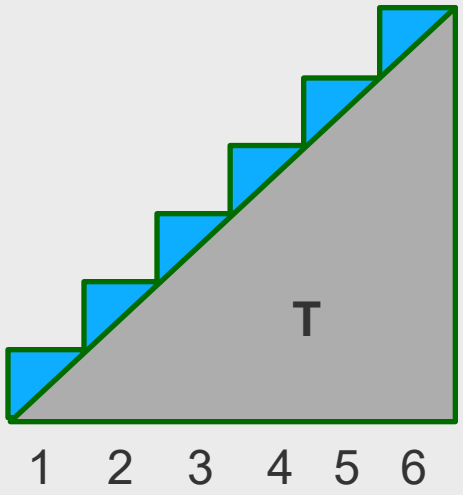
A soma dos valores 1 a 6 é igual à àrea da figura aqui apresentada.

Análise Temporal

- Esta última relação aparece frequentemente porque está associada a ciclos encadeados.

$$\sum_{i=1}^{N} i = 1+2+3+\ldots+N = \frac{1}{2}N(N+1)$$

- Uma forma de visualizar esta relação é a seguinte:



Exemplo para N=6.

A soma dos valores 1 a 6 é igual à àrea da figura aqui apresentada.

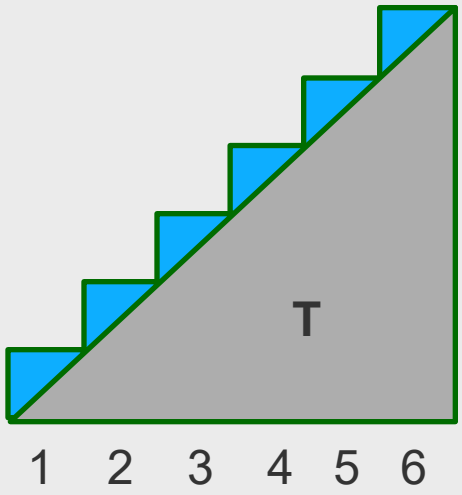
Essa àrea total pode ser calculada como a soma do triangulo T com a àrea dos seis triangulos mais pequenos (azuis).

Análise Temporal

- Esta última relação aparece frequentemente porque está associada a ciclos encadeados.

$$\sum_{i=1}^{N} i = 1+2+3+\ldots+N = \frac{1}{2}N(N+1)$$

- Uma forma de visualizar esta relação é a seguinte:



Exemplo para N=6.
A àrea de um triangulo é metade da base vezes a altura.

*Base de T é 6 (n) e altura é 6 (n) por isso tem àrea ½ *6*6 (½ n^2 no caso geral).*

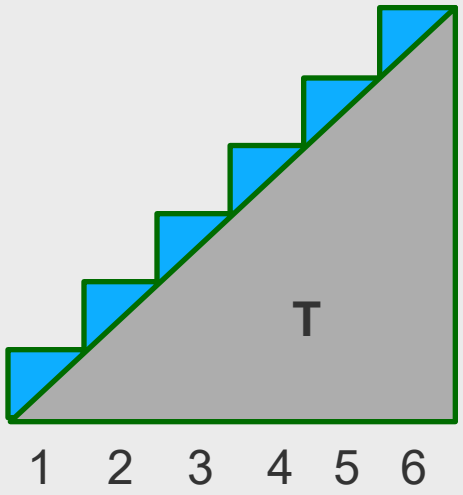
*Cada um dos 6 (n) triangulos azuis tem altura 1 e base 1, por isso tem àrea ½*1*1=1/2.*

Análise Temporal

- Esta última relação aparece frequentemente porque está associada a ciclos encadeados.

$$\sum_{i=1}^{N} i = 1+2+3+\ldots+N = \frac{1}{2}N(N+1)$$

- Uma forma de visualizar esta relação é a seguinte:



Exemplo para N=6.

Sendo assim, a àrea total é a àrea de T mais seis vezes a àrea de cada um dos triangulos azuis.

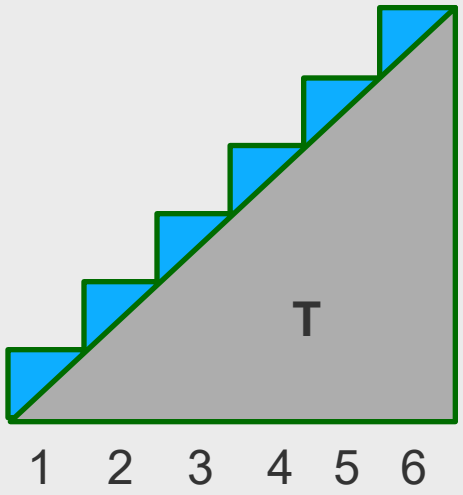
$$\sum_{i=1}^{6} i = \underbrace{\frac{1}{2}\cdot 6\cdot 6}_{\text{àrea de T}} + \underbrace{6\cdot\frac{1}{2}}_{\substack{\text{àrea de 6}\\ \text{triangulos}\\ \text{azuis}}}$$

Análise Temporal

- Esta última relação aparece frequentemente porque está associada a ciclos encadeados.

$$\sum_{i=1}^{N} i = 1+2+3+\ldots+N = \frac{1}{2}N(N+1)$$

- Uma forma de visualizar esta relação é a seguinte:



Exemplo para N=6.

Sendo assim, a àrea total é a área de T mais seis vezes a àrea de cada um dos triangulos azuis.

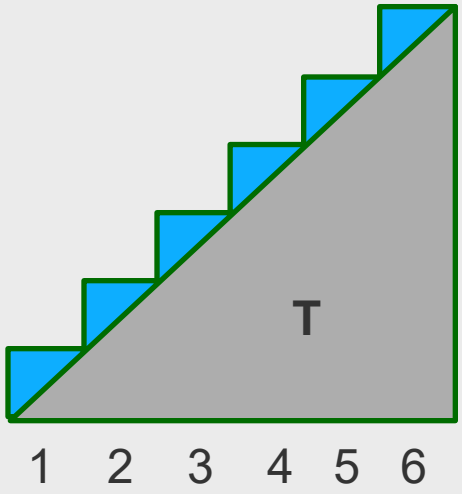
$$\sum_{i=1}^{6} i = \frac{1}{2}\cdot 6\cdot 6 + 6\cdot\frac{1}{2}$$

Análise Temporal

- Esta última relação aparece frequentemente porque está associada a ciclos encadeados.

$$\sum_{i=1}^{N} i = 1+2+3+\ldots+N = \frac{1}{2}N(N+1)$$

- Uma forma de visualizar esta relação é a seguinte:



Generalizando para um valor N

$$\sum_{i=1}^{N} i = \frac{1}{2}\cdot N\cdot N+\frac{1}{2}N = \frac{1}{2}\cdot N\cdot(N+1)$$

Análise Temporal

- **Voltando ao problema:**
- O tempo total de execução pode ser obtido calculando a soma do produto dos termos atrás calculados:

$$T=C1*1+C2*(N+1)+\ldots$$

- *Quais são os obstáculos que encontramos quando queremos aplicar esta formula na prática?*

Análise Temporal

- Tempo de execução de cada linha (C1,C2,…) é:
  - *Dependente do hardware.*
  - *Dependente do compilador*.
  - *Potencialmente variável.*
- O número de execuções de cada linha depende da entrada fornecida à função:
  - *Dimensão do array*
  - *Valores do array*

Análise Temporal

- É preciso clarificar exactamente o que é que se está a tentar contabilizar:
  - *Tempo máximo, tempo mínimo, tempo "médio"?*
- A comparação entre dois algoritmos é mais útil se for independente da plataforma de *Hardware* e compilador utilizado.

Análise dos Algoritmos

- Regressando à questão original...
- ***Quanto tempo demora um algoritmo?***
  - Um cálculo preciso pode ser dificil:
    - *Depende da natureza do algoritmo.*
    - *Depende dos dados que processa.*
    - *Depende de Hardware.*
    - *Depende de compilador*.
  - **mas…** felizmente não é necessário calcular o tempo exacto para comparar dois algoritmos diferentes.

Complexidade de um Algoritmo

- Em vez de considerar o tempo de execução, vamos socorrer-nos de um novo conceito: a complexidade de um algoritmo
- A complexidade de um algoritmo indica de que forma o seu tempo de execução varia com tamanho da entrada:
  - Exemplo: Num algoritmo com complexidade linear, o tempo de execução é, essencialmente, proporcional à dimensão da entrada:
    - Puxar um ficheiro da internet….

Complexidade

- Como representar a complexidade?

Nomenclatura (Notação O-grande)

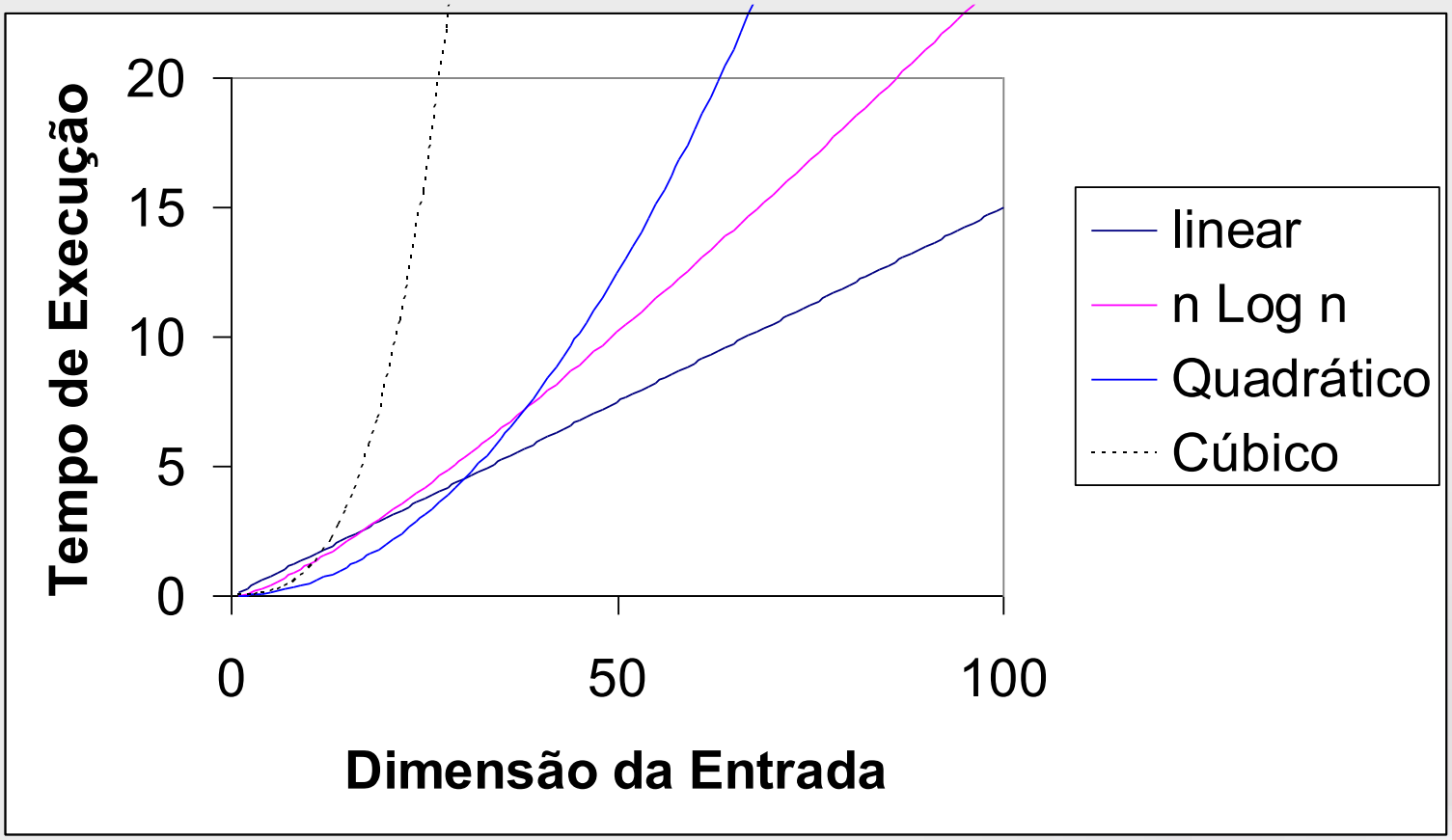
O(n) – Linear ou melhor

Θ(n) – Exactamente linear

Complexidade

- Classes de complexidade comuns:
  - **$\Theta(1)$, $O(1)$** –Constante
  - **$\Theta(N)$, $O(N)$** –Linear
  - **$\Theta(N^2)$, $\Theta(N^3)$, …, $O(N^2)$,** …–Quadratica, cubica, …
  - **$\Theta(\log(N))$, $O(\log(N))$** – Logaritmica
  - **$\Theta(N*\log(N))$, $O(N*\log(N))$** – N-Log

Crescimento de Funções



Notação O-Grande

- A notação O-Grande indica como é que uma função cresce para valores suficientemente elevados da entrada.
  - Exemplos:

$$2n^3 + 20n \longrightarrow O(n^3)$$

$$\frac{1}{2}n^2 + 200n - n\log(n) \longrightarrow O(n^2)$$

Notação O-Grande

- **Porque é que a notação O-Grande é adequada para descrever a complexidade de um algoritmo?**
  1. O valor de uma função é essencialmente dependente do termo dominante, para valores elevados de *N*.
  2. O tempo de execução para valores pequenos de *N* é inconsequente.
  3. O valor exacto da constante multiplicativa do termo dominante é dependente do *hardware,* pelo que não tem relevância para termos efeitos de comparação entre algoritmos.

Exemplo

- **Qual é a complexidade dos seguintes problemas?**
  - Procurar o menor elemento num array de N elementos.
  - Procurar os dois pontos (x,y) mais próximos num plano num conjunto de N pontos.
  - Verificar se, num conjunto de N pontos, há três que sejam colineares (i.e: formem uma linha).

Exemplo: Menor Elemento

Algoritmo

1. *Inicializar uma variável* min *que mantém o valor mínimo encontrado até ao momento com o primeiro elemento do array.*
2. *Percorrer o array e actualizar* min *conforme necessário.*

- Dado que é necessário realizar a mesma quantidade de trabalho para todos os elementos do array, a processo é linear : $\Theta(N)$

Exemplo: Menor distância entre pontos

Algoritmo

1. *Calcular a distância entre cada par de pontos.*
2. *Procurar a menor distância.*

- Existem *N(N-1)/2* pares de pontos… Logo o algoritmo é de ordem quadrática.
  - No entanto, …, existem algoritmos, menos óbvios, de ordem subquadrática para este problema!

Exemplo: Colinearidade

Algoritmo

1. *Identificar cada conjunto distinto de 3 pontos, e verificar se é ou não colinear.*

- Existem *N(N-1)(N-2)/6* conjuntos de 3 pontos, pelo que a complexidade é de ordem cúbica…
  - De novo, existem algoritmos (menos óbvios) de complexidade quadrática…

Exemplo

- Qual é a complexidade dos seguintes problemas?
  - Procurar o menor elemento num array de N elementos não ordenados. **Θ(N)**
  - Procurar os dois pontos (x,y) mais próximos num plano num conjunto de N pontos. **Θ(N^2)**
  - Verificar se, num conjunto de N pontos, há três que sejam colineares (i.e: formem uma linha). **O(N^3)**

É *exatamente* linear, porque é necessário percorrer todos os **N** elementos

É *exatamente* quadrática, porque é necessário considerar todos os **N(N-1)/2** pares de pontos.

É no *pior caso* cúbica. Há **N(N-1)(N-2)/6** trios de pontos, mas o algoritmo termina imediatamente quando se encontrar um deles que seja colinear.