

# ANÁLISE DE COMPLEXIDADE



**AEDS II**


**VICTOR DIAS FROTA**



# ALGORITIMOS

são processos computacionais bem definidos  
que podem receber entrada, processar os dados  
e produzir um resultado





Existem diversos problemas que podem ser resolvidos usando algoritmos e diversos algoritmos que podem ser feitos para resolver um mesmo problema.

Como definir o melhor caminho, ou o caminho mais eficiente, para a solução de um problema ?



# A IMPORTÂNCIA DA ANÁLISE MATEMÁTICA



Comparação de algoritmos não é tão direta como simplesmente comparar a velocidade de processamento.

Diferenças em linguagens e hardware podem ser bastante significativas no tempo de execução.

Para isso devemos analisar o número de computações que um algoritmo faz, para que assim possamos determinar qual o algoritmo mais eficiente.

# CONTANDO INSTRUÇÕES DE UM ALGORITMO



Para começar análise de complexidade de um algoritmo devemos saber identificar superficialmente o número de instruções que ele possui. Dado o seguinte exemplo:

```
maior = lista[0]
for (int i = 0; i < n; i++){
    if (lista[i] > maior){
        maior = lista[i]
    }
}
```

```
maior = lista[0]
for (int i = 0; i < n; i++){
    if (lista[i] > maior){
        maior = lista[i]
    }
}
```



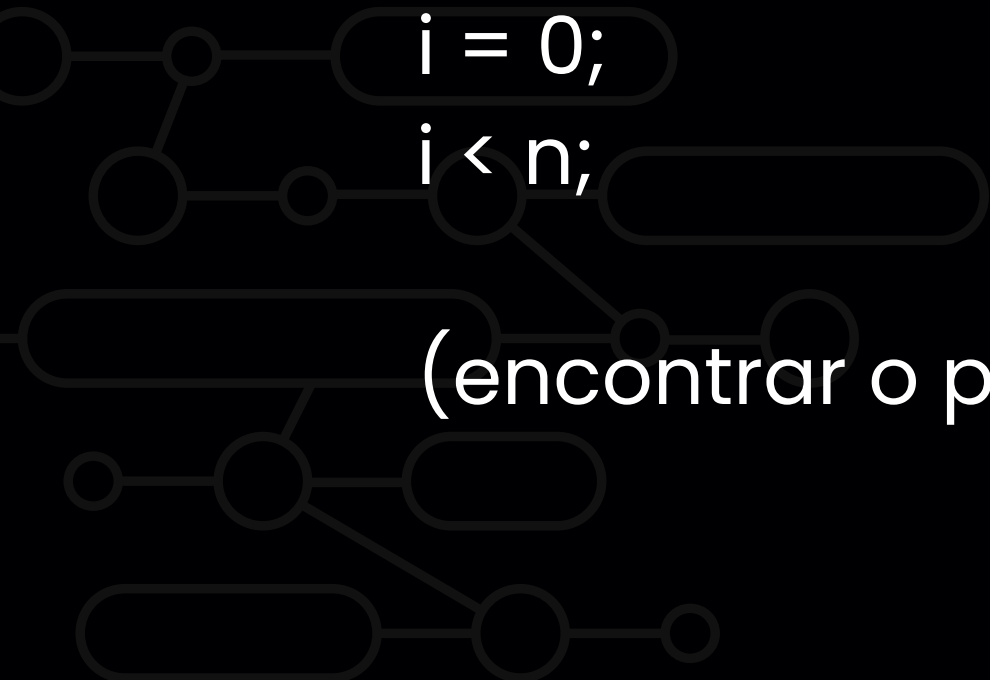
Então para cada execução do algoritmo são executadas 4 instruções fundamentais (que são feitas independente da entrada):

```
maior = lista[0];
```

```
i = 0;
```

```
i < n;
```

(encontrar o primeiro elemento da lista também conta como uma instrução)



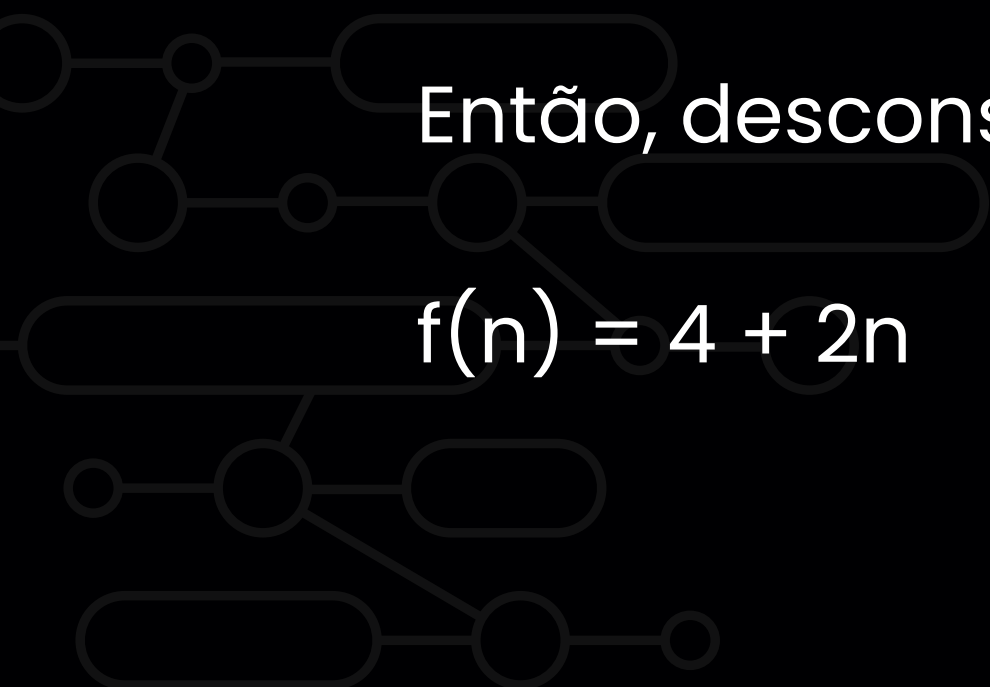
```
maior = lista[0]
for (int i = 0; i < n; i++){
    if (lista[i] > maior){
        maior = lista[i]
    }
}
```

E para cada “n” são feitas mais duas:

```
i > n;
i++;
```

Então, desconsiderando o loop, podemos chegar à função básica:

$$f(n) = 4 + 2n$$



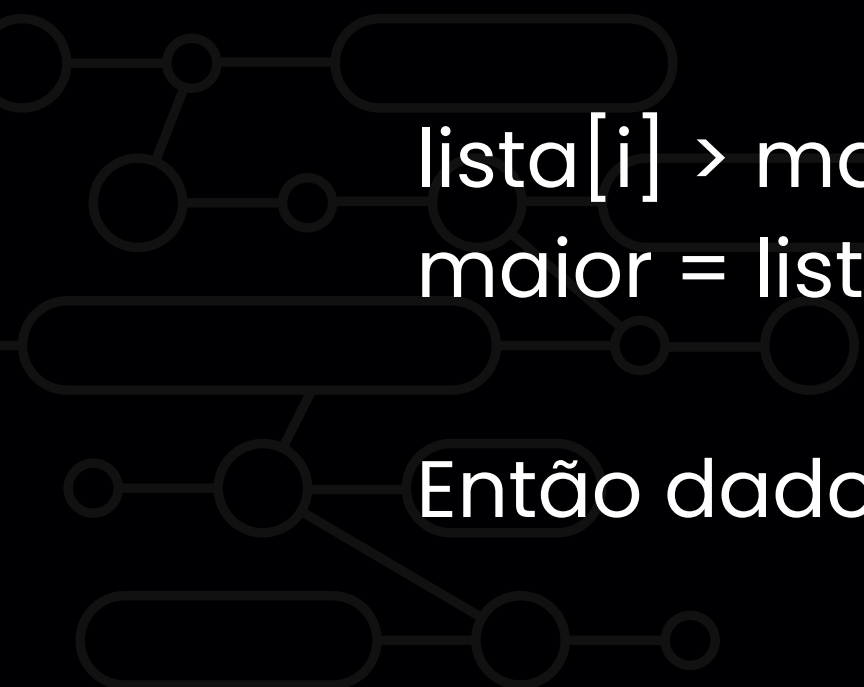
```
maior = lista[0]
for (int i = 0; i < n; i++){
    if (lista[i] > maior){
        maior = lista[i]
    }
}
```



Porém, considerando o loop e o tipo de lista, ele pode executar mais instruções. Quando analisamos um algoritmo é necessário pensar no pior caso em que ele pode ser aplicado, no exemplo, esse caso seria se a lista estivesse em ordem crescente, ou seja, o comando if seria executado todo loop, então seriam atribuídas mais 4 instruções por n.

```
lista[i] > maior;
maior = lista[i];
```

Então dado a análise do pior caso temos:  $f(n) = 4 + 6n$





# COMPORTAMENTO ASSINTÓTICO



Porém, como sabemos algoritmos podem ter centenas de instruções diferentes, então seria inviável contar instrução por instrução, por isso, para a análise de complexidade, levamos em conta apenas o termo que cresce mais rapidamente de acordo com a entrada.

No caso de  $f(n) = 4 + 6n$ , 4 é independente da entrada e 6 é uma constante, então a função que importa fica:

$$f(n) = n.$$



Outros exemplos:

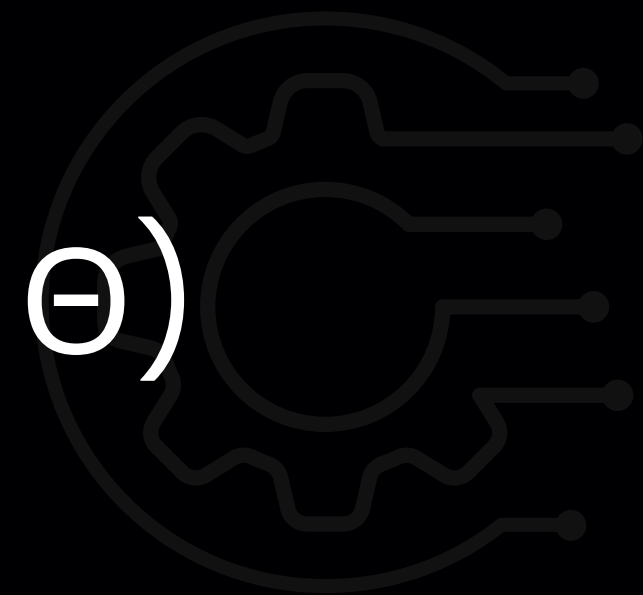
$$f(n) = 915 \text{ ----- } f(n) = 1$$

$$f(n) = 5n + 12 \text{ ----- } f(n) = n$$

$$f(n) = n^2 + 2n + 300 \text{ ----- } f(n) = n^2$$

$$f(n) = n^2 + 2000n + 5929 \text{ ----- } f(n) = n^2$$


# TIPOS DE ANÁLISE ASSINTÓTICA ( $\Omega$ , $O$ , $\Theta$ )



Esse tipo de análise que somente considera a taxa de crescimento de  $n$ , ou seja, tendendo  $n$  a valores enormes, é chamada de assintótica.

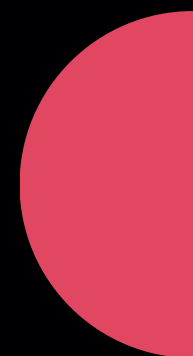
E pode ser representada em 3 diferentes ordens:

- $O$  (big  $O$ )
- $\Omega$  (ômega)
- $\Theta$  (Theta)



## Definição da Ordem O:

Dadas funções assintoticamente não-negativas  $f$  e  $g$ , dizemos que  $f$  está na ordem O de  $g$  e escrevemos  $f = O(g)$  se existe um número positivo  $c$  tal que  $f(n) \leq cg(n)$  para todo  $n$  suficientemente grande.

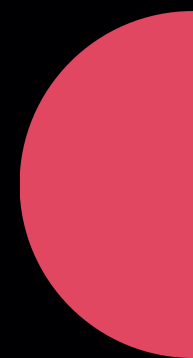




## Definição da Ordem Ômega:

Dadas funções assintoticamente não-negativas  $f$  e  $g$ , dizemos que  $f$  está na ordem  $\Omega$  de  $g$  e escrevemos  $f = \Omega(g)$  se existe um número positivo  $c$  tal que  $f(n) \geq cg(n)$  para todo  $n$  suficientemente grande.

Então é possível dizer que:  $f = O(g)$  se e somente se  $g = \Omega(f)$

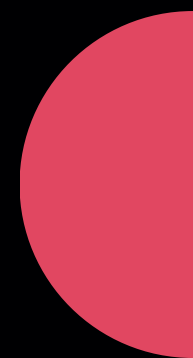
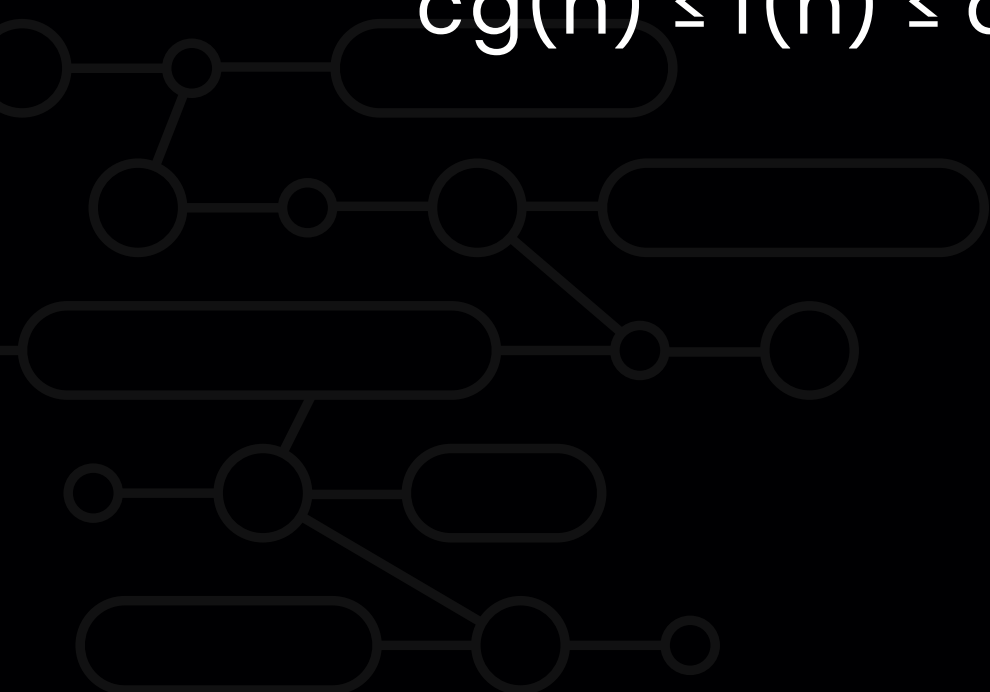




# Definição da Ordem Theta:

Dizemos que duas funções assintoticamente não negativas  $f$  e  $g$  são da mesma ordem e escrevemos  $f = \Theta(g)$  se  $f = O(g)$  e  $f = \Omega(g)$ .

Ou seja,  $f = \Theta(g)$  significa que existe constantes positivas  $c$  e  $d$  tais que  $cg(n) \leq f(n) \leq dg(n)$  para todo  $n$  suficientemente grande.




# CLASSES DE PROBLEMAS





Podemos classificar os problemas computacionais em 3 tipos:


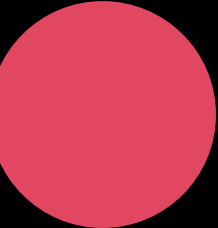
- Problemas de otimização: que pedem o mínimo ou máximo de alguma coisa.
- Problemas de busca: que pede que encontre certas propriedades.
- Problemas de decisão: são os que apenas aceitam dois tipos de resultado, sim ou não.



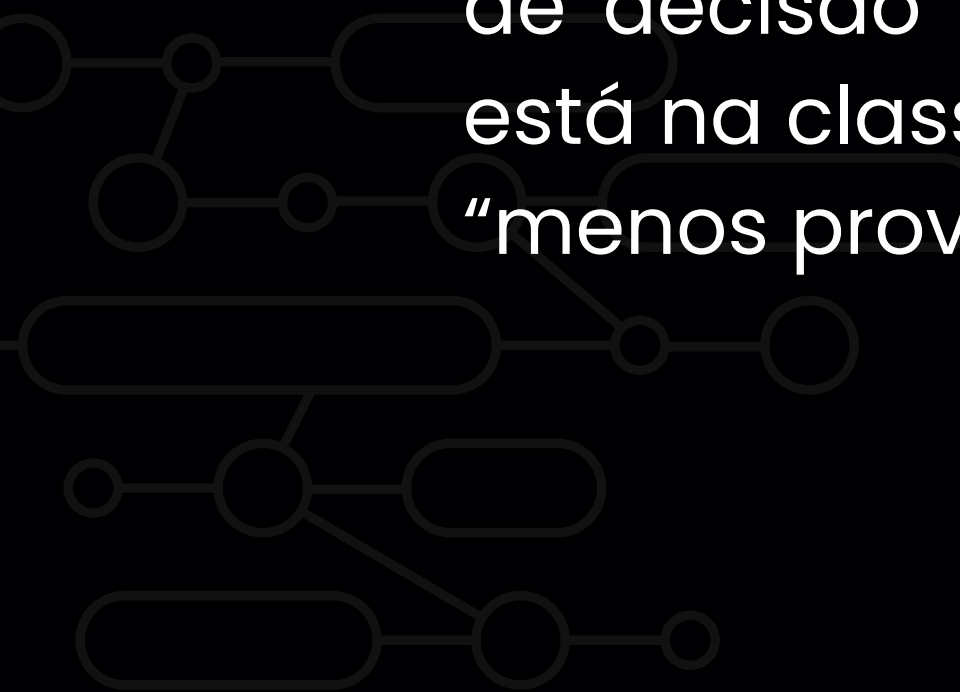
Um algoritmo que resolve um dado problema é polinomial se seu consumo de tempo no pior caso é limitado por uma função polinomial do tamanho de entradas ao problema.

- A classe P de problemas é o conjunto de todos os problemas de decisão que são polinomiais.
  - A classe NP de problemas é o conjunto de todos os problemas de decisão que são polinomialmente verificáveis.
- 
- 






Para cada problema da classe NP, gostaríamos de saber se o problema é fácil (polinomial), ou difícil (não polinomial). Para não enfrentar esse gigantesco desafio de frente, podemos tentar, ao menos, entender a complexidade relativa dos problemas.



A complexidade de muitos problemas computacionais de interesse prático é desconhecida. Para muitos desses problemas, não sabemos se um algoritmo polinomial existe. Em particular, para muitos problemas de decisão polinomialmente verificáveis não sabemos se o problema está na classe P. Resta-nos tentar entender quais desses problemas são “menos provavelmente” polinomiais.



# CONCLUSÃO



A implementação de códigos de maneira eficiente é de extrema importância para construir aplicações escaláveis e que recebem uma grande quantidade de dados. Cada vez mais os hardwares ficam mais eficientes e mais baratos, então no futuro tempo de processamento vai depender cada vez mais da eficiência do algoritmo. Por isso o estudo da complexidade é de grande importância e apresenta como um grande diferencial nas maiores empresas de tecnologia do mundo.

OBRIGADO

