# Ficha 1 – Análise de Complexidade de Algoritmos

A analise de complexidade é:

* Independente do hardware
* Focada no **número de operações** relevantes para resolver o problema, e principalmente das operações dominantes
* Operação dominante é a que **demora** **mais tempo a executar**, não necessariamente por ser a mais exigente, mas **por ser realizada mais vezes**

soma = 0

for (int j = 0; j < n; j++)

soma += tab[j] // o que está dentro realiza-se n vezes

operação dominante => soma += tab[j]

Ignoram operações dependentes do hardware e outros fatores, como:

* Instruções de leitura e escrita
* Tempo de acesso a memória
* Chamadas de subprogramas

Contabilizam-se:

A cada uma destas instruções atribui-se uma unidade de tempo e soma-se o numero de instruções

* Comparações
* Atribuições
* Trocas
* Deslocamentos
* Operações aritméticas

O mais importante num algoritmo é saber como se comportar quando a dimensão dos dados aumentar muito.

**Complexidade espacial:** quanta memória vai precisar?

**Complexidade temporal:** quanto tempo vai precisar?

Normalmente preocupamo-nos apenas com a ordem de grandeza das operações dominantes.

Exemplo:

* Se houver uma operação de complexidade quadrática linear pode ser ignorada.
* Para **n** suficientemente elevado a quadrática sobrepõe-se e a linear pode ser desprezada.

Nota:

Vamos tentar evitar algoritmos quadráticos desde que haja um algoritmo linear ou logarítmico.

# Notação O-Grande (Big-O), teta-grande, ómega-grande

* O: Ordem de grandeza melhor ou igual (o O é como o “limite superior”)
* : ordem de grandeza media
* : limite inferior

: complexidade exatamente linear

:: complexidade linear ou melhor

: complexidade exatamente quadrática

: complexidade quadrática ou melhor

# Ordens de complexidade típicas

: complexidade constante, não aumenta com n

: logarítmica

: linear

: logarítmica linear

, , ... quadrática, cúbica, ...

Constante

* Verificar se o primeiro valor de um array é igual a uma chave
* É só uma operação, independentemente do tamanho do array

Linear

* Procurar um valor num array não ordenado de dimensão n (um ciclo)
* É necessário verificar todos os números do array

Quadrático

* Procurar um valor numa matriz bidimensional não ordenada com lado n (um ciclo dentro de outro ciclo)
* Por cada linha a mais tem de se percorrer todas as colunas, e vice-versa

# Comportamento Assintótico

* Algoritmo de ordem inferior != mais rápido
* Qualquer algoritmo de ordem superior irá tornar-se mais lento do que um algoritmo de ordem inferior “para **n** suficientemente grande”
* Para **n** pequeno o algoritmo mais eficiente pode ser diferente de **n** grande
* Num algoritmo de complexidade linear, duplicam os dados, o tempo de execução duplica
* Num algoritmo de complexidade quadrática, se os dados duplicam o tempo de execução quadruplica

# Ficha 1

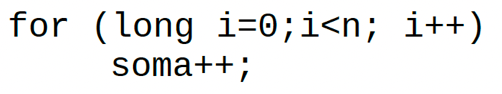
1. Para cada um dos programas seguintes:
   1. Efetue a análise de complexidade
   2. Calcule analiticamente quanto o tempo de execução que deverá aumentar cada a dimensão de N seja aumentada 4x

Uma imagem com texto

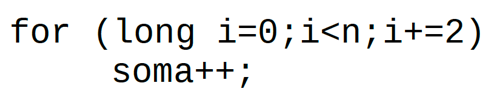
Descrição gerada automaticamente

soma++, quantas vezes é executada esta linha?

* A instrução soma++ é executada n\*n vezes (ciclo exterior \* ciclo interior)
* O programa é de complexidade quadrática -
* Se o n for aumentado para 4n, o soma++ vai ser executado 4n\*4n vezes. Portanto, o tempo de execução aumenta 16 vezes.



* A instrução soma++ é executada n vezes
* O programa é de complexidade linear
* Se o n for aumentado para 4n, o soma++ vai ser executado 4n vezes. Portanto, o tempo de execução aumenta 4 vezes.



* A instrução soma++ é executada n/2 vezes
* Programa de complexidade linear -
* Se o n for aumentado para , a instrução soma++ vai ser executada 2n vezes. Portanto o tempo de execução aumenta em 2 vezes.

Uma imagem com texto

Descrição gerada automaticamente

* Aqui no pior dos casos o ciclo externo executa 1000 vezes. (1000 \* algo linear) também é linear.
* O facto de serem dois ciclos encadeados não implica que seja um algoritmo quadrático.
* A instrução soma++ é executada 1000\*n vezes (ciclo exterior \* ciclo interior), portanto o ciclo exterior atua como uma constante.
* Programa de complexidade linear
* Se o n for aumentado para 4n, soma++ vai ser executado 1000\*4n vezes. Portanto, o tempo de execução aumenta 4 vezes.

Uma imagem com texto

Descrição gerada automaticamente

* O tempo de execução é n+n = 2n, a instrução é executada 2n vezes
* O dobro de algo linear é também linear
* Programa de complexidade linear -
* Se o n for aumentado para 2\*4n, soma++ vai ser executada 8n vezes. Portanto, o tempo de execução aumenta 4 vezes.

Uma imagem com texto

Descrição gerada automaticamente

* **if(n > 20000)**: operação de complexidade constante O(1)
* Ao aumento de n, no limite, não interfere com o tempo de execução, porque n é truncado em 20000
* Os ciclos encadeados têm um comportamento aparentemente quadrático. No entanto, o valor de n é limitado a 20000 pela primeira linha. Isso significa que o ciclo irá ter, sempre 20000\*20000 iteração de soma++, para valores suficientemente elevados de n (neste caso, valores de >= 20000)
* Assim, o código tem o mesmo tempo de execução à medida que **n** aumenta para valores de n elevados, e dessa forma é de **complexidade constante**.
* Respostas como “É para n < 20000 e para n >= 20000” não fazem muito sentido dado que, por definição, a notação de O descreve apenas oque acontece para valores de N suficientemente elevado.
* Dois ciclos encadeados não resultam necessariamente em complexidade quadrática – a complexidade pode ser maior ou menor.

Uma imagem com texto

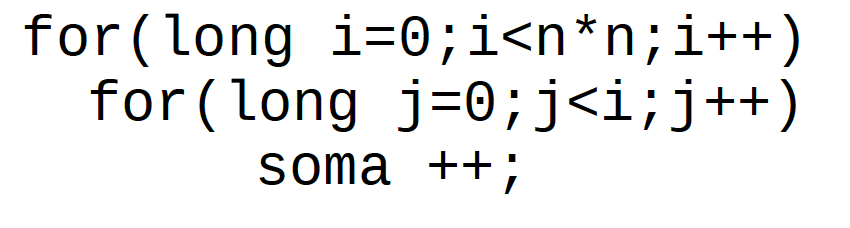
Descrição gerada automaticamente

* A instrução soma++ é executada n\*n\*n vezes
* O programa é de complexidade cúbica -
* Se o tempo de execução for aumentado em 4 vezes, a instrução será repetida 4n\*4n\*4n (64\*n) vezes, ou seja, o tempo de execução aumenta em 64 vezes.

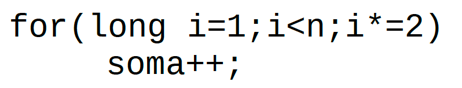
Uma imagem com texto

Descrição gerada automaticamente

* A instrução soma++ é executada vezes (ciclo exterior \* ciclo interior)
* O programa é de complexidade quadrática –
* Graficamente a soma de todos os números entre 1 e N pode ser representada sendo a área de um triângulo retângulo (da esquerda para a direita, uma coluna de altura 1, seguida de uma coluna de altura 2, etc.). Dessa forma, torna-se claro que a soma (área) é igual aproximadamente metade da área do quadrado (). Em vez de uma matriz, processa-se ”meia matriz”.
* O termo linear que pode ser observado na fórmula torna-se irrelevante para valores elevados de N, quando comparado com o termo quadrático.
* Quando n aumenta 4 vezes, o tempo de execução aumenta na mesma 16 vezes.



* O ciclo interno vai iterar
* A expressão matemática para este somatório é , pelo que o resultado é de ordem cúbica - .
* Quando n aumenta 4 vezes, o tempo de execução aumenta = 64 vezes



* O i duplica a cada iteração. Exemplo de ordem logarítmica, .
* Um aumento de 4 vezes de n obriga a mais duas iterações
* Na matemática, um logaritmo é o ”inverso” do expoente: log2(n) = x ⬄2^x = n
* É o numero de bits necessários para representar o inteiro N
  + Log2(N) é o número de vezes que é necessário dividir N por 2 para chegar a 1
    - (esta interpretação será importante quando for abordada a pesquisa binária).
  + Log2(n) é o numero de vezes que é preciso multiplicar por 2 (começando em 1) até chegar a N. Como no exemplo de código desta alínea.
* Quantas iterações adicionais são necessárias se o valor de N aumenta para o dobro?
  + 1 iteração
* Quando N quadruplica, são realizadas mais duas iterações.
  + A base do logaritmo não importa para a complexidade, porque o logaritmo de N numa determinada base é proporcional ao logaritmo do mesmo valor noutra base, logB(n).
  + logX(B). Ex ln(N). = log2(N) / log2(e). O valor 1/log2(e) é constante e independente em relação a N. Graficamente a curva mantém a "mesma forma" pelo que a característica de complexidade é a mesma.