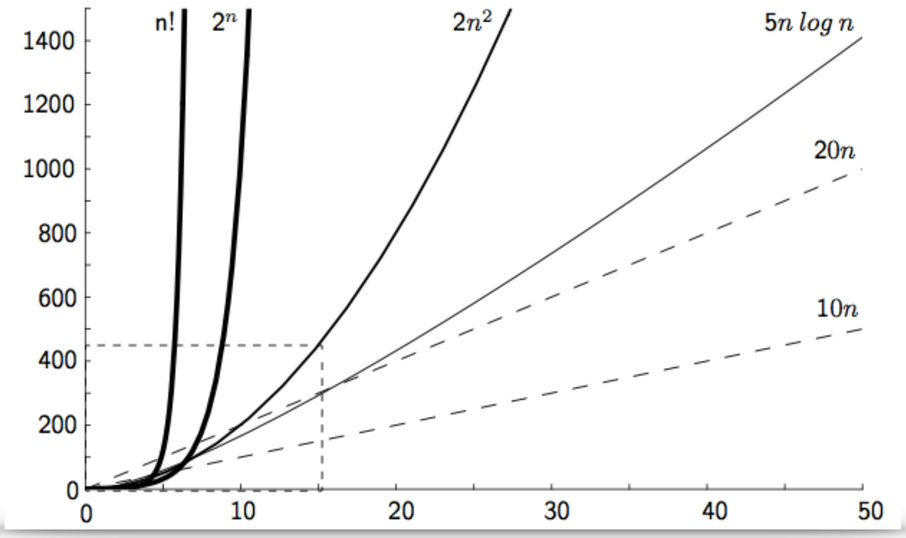
Profa. Dra. Raquel C. de Melo-Minardi
Departamento de Ciência da Computação
Instituto de Ciências Exatas
Universidade Federal de Minas Gerais



MÓDULO 3
COMPLEXIDADE DE ALGORITMOS
Função de complexidade – Parte II

ALGORITMO MAIORMENOR2

- Apresentamos a seguir uma nova versão desse código com uma pequena melhoria que trará um certo ganho de desempenho

```
def maiorMenor2(lista):
        menor = maior = lista[0]
        for i in range(1, len(lista)):
                if (menor > lista[i]):
                        menor = lista[i]
                elif (maior < lista[i]): # Troca de um “if“ por um “elif”
                        maior = lista[i]
        tupla = (menor, maior)
        return tupla
```

ALGORITMO MAIORMENOR2

- O que você observou de diferente em relação às funções “maiorMenor1“ e “maiorMenor2”?
  - Elas são quase idênticas mas há uma **pequena diferença** na codificação que pode evitar inúmeras comparações desnecessárias

- **Trocamos um `if` por um `elsif`** simplesmente porque quando um valor da lista avaliada for maior que o maior valor, ele não tem como ser menor que o menor valor, ou seja, estávamos fazendo inúmeras comparações inúteis

- Vamos então pensar em qual seria agora nossa função de complexidade
  - Será que ela mudou muito?
  - Qual seria essa nova função?
  - Pense um pouco sobre ela

MELHOR CASO, CASO MÉDIO E PIOR CASO

- Se você passou um tempo refletindo sobre essa nova função, deve ter percebido que ela não é tão simples de ser definida pois ela se tornou **probabilística** ou passou a depender não só do tamanho da entrada como também dos valores que estarão preenchendo a nossa lista

  - Como devemos proceder para analisar a complexidade do nosso algoritmo já que não temos como antever que dados estarão na lista?

MELHOR CASO, CASO MÉDIO E PIOR CASO

- Para esse tipo de análise, precisamos distinguir **três cenários**
  - **Melhor caso**: corresponde ao **menor tempo de execução sobre todas as possíveis execuções**, ou seja, consiste em identificar em que cenário o seu algoritmo trabalhará menos
  - **Pior caso**: corresponde ao **maior tempo de execução sobre todas as possíveis entradas** de tamanho *n*, ou seja, identificar quando o algoritmo trabalhará mais. Trata-se de de um limite de tempo superior que nunca será sobrepassado
  - **Caso médio**: corresponde à **média dos tempos de execução para todas as possíveis entradas** de tamanho *n*, ou seja, seria o mais próximo do caso esperado na realidade
- Qual das três análises você considera mais interessante ou mais apropriada?

LIMITE SUPERIOR

- A análise de melhor caso é excessivamente otimista e que provavelmente não será frequente na prática
- O mesmo vale para o pior caso que é a análise mais pessimista, mas também pouco provável
- Então caso médio é o mais interessante por ser o mais realista?

LIMITE SUPERIOR

- Para se calcular o caso médio, precisamos saber a **distribuição de probabilidades** sobre o conjunto de possíveis entradas de tamanho *n*, o que normalmente não temos
- É comum supor uma distribuição de probabilidades em que todas as entradas sejam **equiprováveis** para possibilitar os cálculos mas note que, dessa forma, o que era mais realista, pode passar a não o ser
- Isso torna a análise de caso médio **complicada** e **nem tão realista**
- Ela só é realizada em casos que façam sentido essas suposições de equiprobabilidade

- Normalmente, a análise mais importante é a de **pior caso** pois nos dá um **limite superior** ou uma **garantia** de que tempo maior não pode ocorrer

ALGORITMO MAIORMENOR2

- Voltando ao nosso problema de análise de complexidade da função “maiorMenor2“, como proceder então?
  - Devemos analisar o comportamento do algoritmo nos três cenários previamente definidos
    - **Melhor caso**: precisamos encontrar o tipo de entrada que levaria a nossa construção `if-elif` a **trabalhar o mínimo possível**
      - O `if` sempre será executado, então o melhor caso só poderia ser o caso em que o `elif` nunca fosse executado
      - Quando isso aconteceria?
      - Quando a lista estivesse em ordem decrescente

$$f(n) = n-1$$

ALGORITMO MAIORMENOR2

- Voltando ao nosso problema de análise de complexidade da função “maiorMenor2“, como proceder então?
  - Devemos analisar o comportamento do algoritmo nos três cenários previamente definidos
    - **Pior caso**: quando nosso algoritmo **trabalhará o máximo possível**?
      - Em todas as iterações tanto o `if` quanto o `elsif` seriam executados
      - Quando isso aconteceria?
      - Lista em ordem crescente de forma que a cada iteração um novo maior elemento é encontrado

$$f(n) = 2(n-1)$$

ALGORITMO MAIORMENOR2

- Voltando ao nosso problema de análise de complexidade da função “maiorMenor2“, como proceder então?
  - Devemos analisar o comportamento do algoritmo nos três cenários previamente definidos
    - **Caso médio**: como o algoritmo se comportará em média?
      - Para essa análise, temos que supor *0.5* de probabilidade de o primeiro `if` ser verdadeiro de modo o segundo `elsif` execute em *50%* das iterações
      - Nossa função de complexidade deveria ser então a média da função em melhor caso *f(n) = n-1* e em pior caso *f(n) = 2(n-1)*

$$f(n) = 3n/2 - 3/2$$

ALGORITMO MAIORMENOR2

- Observe que no pior caso a função “maiorMenor2” tem o mesmo desempenho que “maiorMenor1“ mas no melhor caso, seu desempenho é muito melhor e no caso médio também é um pouco melhor

- Seria a função “maiorMenor2” ótima?

  - A resposta é não ja que podemos obter um algoritmo ainda melhor

  - Como?

Desafio

1. Pense um pouco a respeito de como esse algoritmo poderia ser melhorado
2. Implemente esse algoritmo
3. Qual sua função de complexidade?