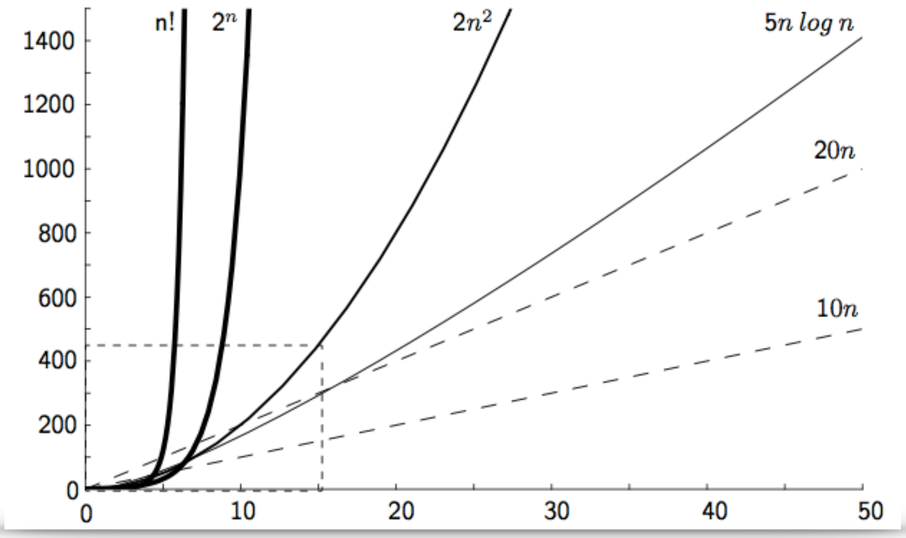
Profa. Dra. Raquel C. de Melo-Minardi
Departamento de Ciência da Computação
Instituto de Ciências Exatas
Universidade Federal de Minas Gerais



MÓDULO 3
COMPLEXIDADE DE ALGORITMOS
Função de complexidade – Parte III

ALGORITMO MAIORMENOR3

1. Vamos comparar os elementos aos pares, separando-os sempre em dois subconjuntos: o dos que foram maiores e o dos que foram menores na comparação de que participaram. Note que isso nos renderia no máximo ***n/2*** **comparações** (considerando um número par de elementos na lista)

2. Obteremos então o maior elemento na metade da lista que tem os elementos que venceram as comparações como um custo ótimo de **no máximo *n/2-1* comparações** (usando o algoritmo "maior" já apresentado anteriormente que é ótimo)

3. Obteremos o menor elemento de forma semelhante à obtenção do maior elemento a um custo também de **no máximo *n/2-1***

- Somando essas três funções esse algoritmo teria a seguinte função de complexidade

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

ALGORITMO MAIORMENOR3

```
def maiorMenor3(lista):
        if (len(lista)%2 == 1):          # Se lista tem tamanho ímpar, faz com
                lista.append(lista[0]) # que se torne par duplicando um valor
                                       # qualquer (no caso o primeiro)

        if (lista[0] < lista[1]):
                menor = lista[0]
                maior = lista[1]
        else:
                menor = lista[1]
                maior = lista[0]
```

ALGORITMO MAIORMENOR3

- A função “maiorMenor3” recebe uma lista como argumento
- Declaramos as variáveis `menor` e `maior` que armazenarão o menor e o maior valores ao longo da varredura da lista
- A variável `i` será usada para percorrer a lista

ALGORITMO MAIORMENOR3

```
def maiorMenor3(lista):
(I)     if (len(lista)%2 == 1):          # Se lista tem tamanho ímpar, faz com
                lista.append(lista[0]) # que se torne par duplicando um valor
                                       # qualquer (no caso o primeiro)

(II)    if (lista[0] < lista[1]):
                menor = lista[0]
                maior = lista[1]
        else:
                menor = lista[1]
                maior = lista[0]
```

ALGORITMO MAIORMENOR3

- O comando condicional mostrado em **I** serve para tratar os casos em que a lista tem tamanho ímpar
- Considere o exemplo no qual temos a lista 4 6 8 3 5 (5 elementos)
  - O menor elemento será 3 e o maior o 8
- Duplicamos o elemento `lista[0]` (que nesse exemplo é 4) colocando-o no final da lista através do comando `lista.append(lista[0])` resultando em 4 6 8 3 5 **4**
- **Não há prejuízo à corretude** do resultado final
- **Não** temos que **tratar nenhuma excepcionalidade** no código

ALGORITMO MAIORMENOR3

```
def maiorMenor3(lista):
(I)     if (len(lista)%2 == 1):          # Se lista tem tamanho ímpar, faz com
                lista.append(lista[0]) # que se torne par duplicando um valor
                                       # qualquer (no caso o primeiro)

(II)    if (lista[0] < lista[1]):
                menor = lista[0]
                maior = lista[1]
        else:
                menor = lista[1]
                maior = lista[0]
```

ALGORITMO MAIORMENOR3

- O condicional destacado em **II** inicializa as variáveis `maior` e `menor`

- Como trabalharemos sempre em pares, optamos por pegar o primeiro par de elementos `lista[0]` e `lista[1]` usando o menor deles para inicializar `menor` e o maior deles para inicializar `maior`

ALGORITMO MAIORMENOR3

```
        i=2;
(III)   while (i < len(lista)):
(IV)            if lista[i] > lista[i+1]:
(V)                     if lista[i] > maior:
                                maior = lista[i]
(VI)                    if lista[i+1] < menor:
                                menor = lista[i+1]
                else:
                        if lista[i+1] > maior:
                                maior = lista[i+1]
                        if lista[i] < menor:
                                menor = lista[i]
                i+=2

        tupla = (menor, maior)
        return tupla
```

ALGORITMO MAIORMENOR3

- O código **III** é um laço que inicia com `i=2`, ou seja, avaliando a lista a partir do seu terceiro elemento e o trabalhará em pares

- A idéia desse laço é implementar o passo (1) do algoritmo explicado previamente em alto nível "*Vamos comparar os elementos aos pares, separando-os sempre em dois subconjuntos: o dos que foram maiores e o dos que foram menores na comparação de que participaram.*"

- Perceba que o último comando o laço é `i+=2` que incrementa o indexador em duas posições passando a avaliar o próximo par

ALGORITMO MAIORMENOR3

```
        i=2;
(III)   while (i < len(lista)):
(IV)            if lista[i] > lista[i+1]:
(V)                     if lista[i] > maior:
                                maior = lista[i]
(VI)                    if lista[i+1] < menor:
                                menor = lista[i+1]
                else:
                        if lista[i+1] > maior:
                                maior = lista[i+1]
                        if lista[i] < menor:
                                menor = lista[i]
                i+=2

        tupla = (menor, maior)
        return tupla
```

ALGORITMO MAIORMENOR3

- O condicional **IV** testa qual dos elementos do par (`lista[i]` e `lista[i+1]` é o maior simulando como se a lista estivesse sendo dividida em duas metades como explicado também no passo (1): "*(...) separando-os sempre em dois subconjuntos: o dos que foram maiores e o dos que foram menores na comparação de que participaram.*"

- Essa divisão da lista não acontece em termos de estrutura de dados mas apenas do código

ALGORITMO MAIORMENOR3

```
        i=2;
(III)   while (i < len(lista)):
(IV)            if lista[i] > lista[i+1]:
(V)                     if lista[i] > maior:
                                maior = lista[i]
(VI)                    if lista[i+1] < menor:
                                menor = lista[i+1]
                else:
                        if lista[i+1] > maior:
                                maior = lista[i+1]
                        if lista[i] < menor:
                                menor = lista[i]
                i+=2

        tupla = (menor, maior)
        return tupla
```

ALGORITMO MAIORMENOR3

- Dentro desse condicional **IV** há o outro condicional **V** que resolve essa questão de avaliar se algum dos elementos do par é o novo menor ou maior elementos conforme explicado nos passos (2) e (3) do algoritmo "*Obteremos então o maior elemento na metade da lista que tem os elementos que venceram as comparações como um custo ótimo de no máximo n/2-1 comparações (usando o algoritmo "maior" já apresentado anteriormente que é ótimo)*" e "*Obteremos o menor elemento de forma semelhante à obtenção do maior elemento a um custo também de no máximo n/2-1.*"

- A divisão da lista não ocorre então na realidade em termos de quebrar uma lista em duas mas sim no código por meio dos `ifs`

ALGORITMO MAIORMENOR3

```
        i=2;
(III)   while (i < len(lista)):
(IV)            if lista[i] > lista[i+1]:
(V)                     if lista[i] > maior:
                                maior = lista[i]
(VI)                    if lista[i+1] < menor:
                                menor = lista[i+1]
                else:
                        if lista[i+1] > maior:
                                maior = lista[i+1]
                        if lista[i] < menor:
                                menor = lista[i]
                i+=2

        tupla = (menor, maior)
        return tupla
```

ALGORITMO MAIORMENOR3

- Por fim, observe em **VII** que essa função retorna uma lista de duas posições a saber:
  - a primeira é o **menor** elemento da lista
  - a segunda é o **maior** elemento da lista

CONCLUSÃO

- Com isso finalizamos a explicação do algoritmo que, como provado em [Ziviani, 2004], é o ótimo para o problema de se obter o menor e o maior elementos em um arranjo

- Finalizamos também um exemplo prático da necessidade de se avaliar a complexidade de um algoritmo em termos dos seus melhor caso, pior caso e caso médio

- A tabela a seguir resume o comparativo entre as três funções apresentadas

f(n)			
Algoritmos	Melhor caso	Pior caso	Caso médio
maiorMenor1	2(n-1)	2(n-1)	2(n-1)
maiorMenor2	n-1	2(n-1)	3n/2 - 3/2
maiorMenor3	3n/2-2	3n/2-2	3n/2-2