

# *Noções de Complexidade*

Unidade I: Análise de Algoritmos

# Agenda

- Exercícios Iniciais
- Contagem de Operações
- Funções de Complexidade
- Noção sobre a Notação  $\Theta$
- Exercícios

# Exercícios Iniciais

# Exercício Resolvido (1): Resolva as Equações

a)  $2^0 =$

d)  $2^3 =$

g)  $2^6 =$

j)  $2^9 =$

b)  $2^1 =$

e)  $2^4 =$

h)  $2^7 =$

k)  $2^{10} =$

c)  $2^2 =$

f)  $2^5 =$

i)  $2^8 =$

l)  $2^{11} =$

# Exercício Resolvido (1): Resolva as Equações

a)  $2^0 =$

d)  $2^3 =$

g)  $2^6 =$

j)  $2^9 =$

b)  $2^1 =$

e)  $2^4 =$

h)  $2^7 =$

k)  $2^{10} =$

c)  $2^2 =$

f)  $2^5 =$

i)  $2^8 =$

l)  $2^{11} =$

**Pause!**

# Exercício Resolvido (1): Resolva as Equações

a)  $2^0 = 1$

d)  $2^3 = 8$

g)  $2^6 = 64$

j)  $2^9 = 512$



b)  $2^1 = 2$

e)  $2^4 = 16$

h)  $2^7 = 128$

k)  $2^{10} = 1024$

c)  $2^2 = 4$

f)  $2^5 = 32$

i)  $2^8 = 256$

l)  $2^{11} = 2048$

# Exercício Resolvido (2): Resolva as Equações

a)  $\lg(2048) =$

d)  $\lg(256) =$

g)  $\lg(32) =$

j)  $\lg(4) =$

b)  $\lg(1024) =$

e)  $\lg(128) =$

h)  $\lg(16) =$

k)  $\lg(2) =$

c)  $\lg(512) =$

f)  $\lg(64) =$

i)  $\lg(8) =$

l)  $\lg(1) =$

# Exercício Resolvido (2): Resolva as Equações

a)  $\lg(2048) =$

d)  $\lg(256) =$

g)  $\lg(32) =$

j)  $\lg(4) =$

b)  $\lg(1024) =$

e)  $\lg(128) =$

h)  $\lg(16) =$

k)  $\lg(2) =$

c)  $\lg(512) =$

f)  $\lg(64) =$

i)  $\lg(8) =$

l)  $\lg(1) =$

**Pause!**



# Exercício Resolvido (2): Resolva as Equações

a)  $\lg(2048) = 11$    d)  $\lg(256) = 8$    g)  $\lg(32) = 5$    j)  $\lg(4) = 2$

b)  $\lg(1024) = 10$    e)  $\lg(128) = 7$    h)  $\lg(16) = 4$    k)  $\lg(2) = 1$

c)  $\lg(512) = 9$    f)  $\lg(64) = 6$    i)  $\lg(8) = 3$    l)  $\lg(1) = 0$



# Exercício Resolvido (3): Resolva as Equações

$$a) \sqrt{4,01} =$$

$$d) \sqrt{4,99} =$$

$$g) \lg(17) =$$

$$j) \lg(15) =$$

$$b) \sqrt{4,01} =$$

$$e) \sqrt{\lg(16)} =$$

$$h) \sqrt{\lg(17)} =$$

$$k) \sqrt{\lg(15)} =$$

$$c) \sqrt{4,99} =$$

$$f) \sqrt{\lg(16)} =$$

$$i) \sqrt{\lg(17)} =$$

$$l) \sqrt{\lg(15)} =$$

# Exercício Resolvido (3): Resolva as Equações

a)  $\sqrt{4,01} =$

d)  $\sqrt{4,99} =$

g)  $\lg(17) =$

j)  $\lg(15) =$

b)  $\sqrt{4,01} =$

e)  $|\lg(16)| =$

h)  $|\lg(17)| =$

k)  $|\lg(15)| =$

c)  $\sqrt{4,99} =$

f)  $|\lg(16)| =$

i)  $|\lg(17)| =$

l)  $|\lg(15)| =$

**Pause!**

# Exercício Resolvido (3): Resolva as Equações

$$\text{a) } \overline{4,01} = 5$$

$$\text{d) } \overline{4,99} = 4$$

$$\text{g) } \lg(17) = 4,087 \quad \text{j) } \lg(15) = 3,907$$



$$\text{b) } \overline{4,01} = 4$$

$$\text{e) } \overline{\lg(16)} = 4$$

$$\text{h) } \overline{\lg(17)} = 5$$

$$\text{k) } \overline{\lg(15)} = 4$$

$$\text{c) } \overline{4,99} = 5$$

$$\text{f) } \overline{\lg(16)} = 4$$

$$\text{i) } \overline{\lg(17)} = 4$$

$$\text{l) } \overline{\lg(15)} = 3$$

## Exercício Resolvido (4): Plote os Gráficos

a)  $f(n) = n^3$

b)  $f(n) = n^2$

c)  $f(n) = n \times \lg(n)$

d)  $f(n) = n$

e)  $f(n) = \text{sqrt}(n)$

f)  $f(n) = \lg(n)$

## Exercício Resolvido (4): Plote os Gráficos

a)  $f(n) = n^3$

b)  $f(n) = n^2$

c)  $f(n) = n \times \lg(n)$

d)  $f(n) = n$

e)  $f(n) = \text{sqrt}(n)$

f)  $f(n) = \lg(n)$

**Pause!**

## Exercício Resolvido (4): Plote os Gráficos

a)  $f(n) = n^3$

1,25E+9

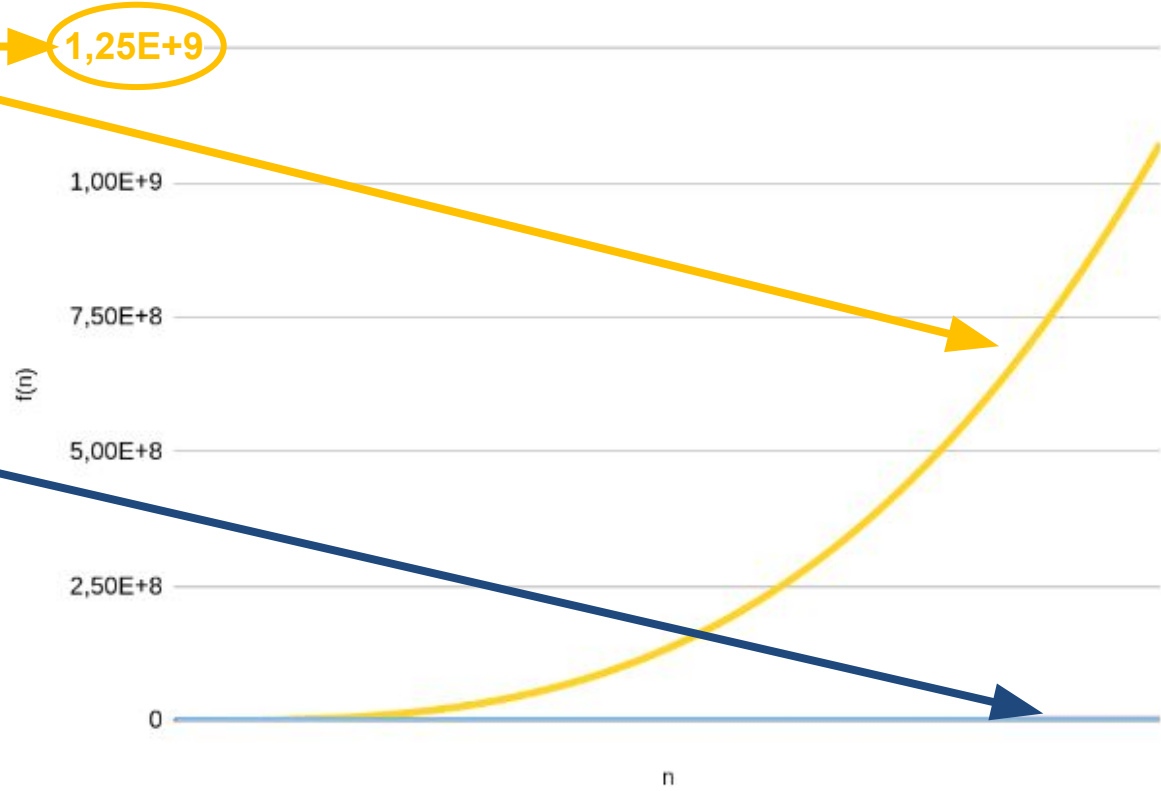
b)  $f(n) = n^2$

c)  $f(n) = n \times \lg(n)$

d)  $f(n) = n$

e)  $f(n) = \text{sqrt}(n)$

f)  $f(n) = \lg(n)$



## Exercício Resolvido (4): Plote os Gráficos

a)  $f(n) = n^3$

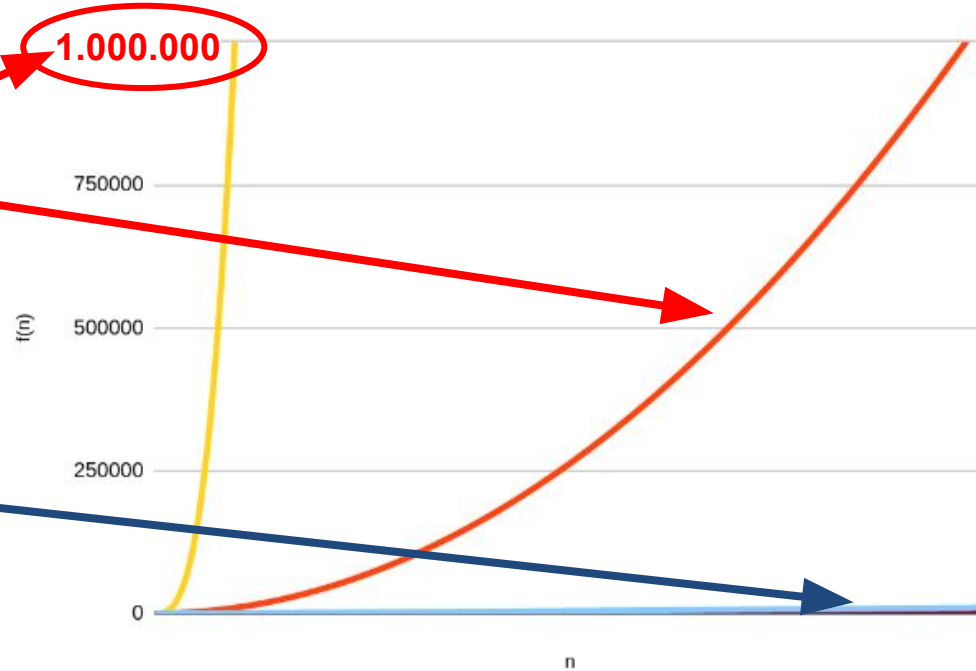
b)  $f(n) = n^2$

c)  $f(n) = n \times \lg(n)$

d)  $f(n) = n$

e)  $f(n) = \text{sqrt}(n)$

f)  $f(n) = \lg(n)$





## Exercício Resolvido (4): Plote os Gráficos

a)  $f(n) = n^3$

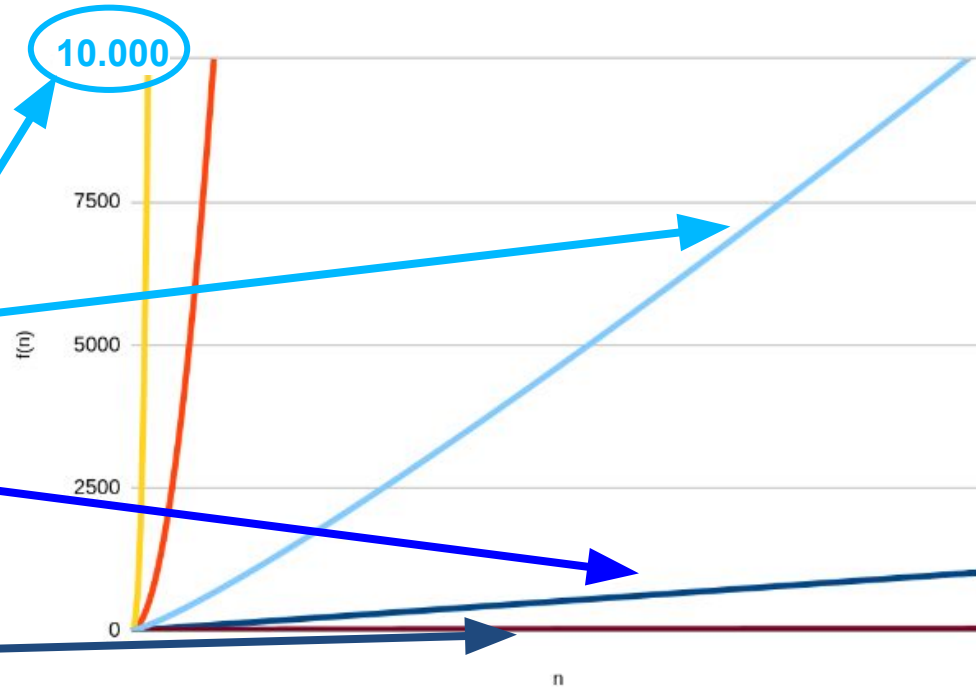
b)  $f(n) = n^2$

c)  $f(n) = n \times \lg(n)$

d)  $f(n) = n$

e)  $f(n) = \text{sqrt}(n)$

f)  $f(n) = \lg(n)$



## Exercício Resolvido (4): Plote os Gráficos

a)  $f(n) = n^3$

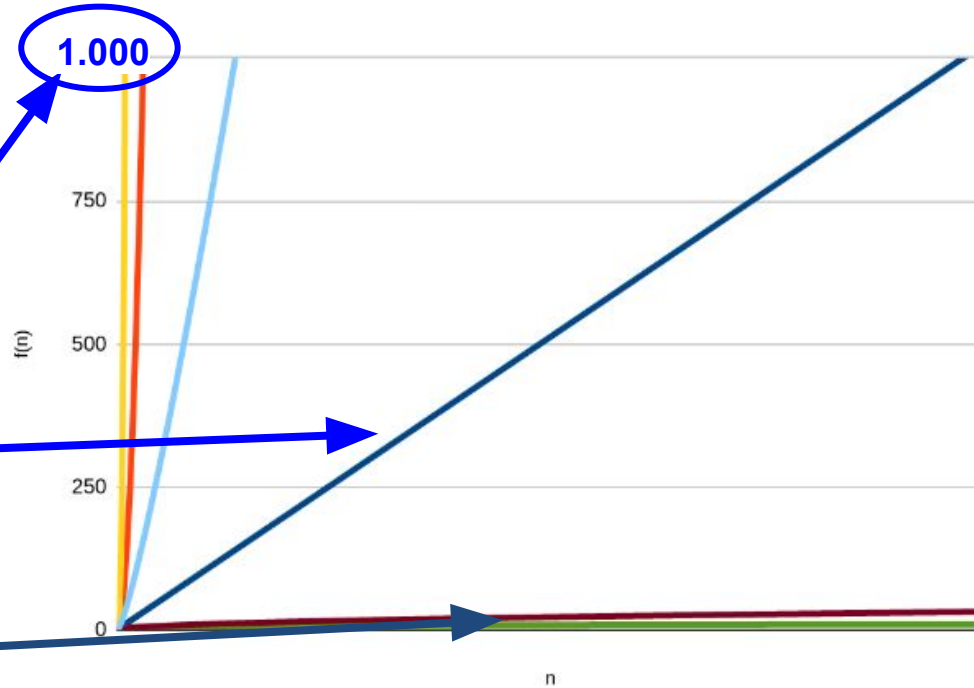
b)  $f(n) = n^2$

c)  $f(n) = n \times \lg(n)$

d)  $f(n) = n$

e)  $f(n) = \text{sqrt}(n)$

f)  $f(n) = \lg(n)$



## Exercício Resolvido (4): Plote os Gráficos

a)  $f(n) = n^3$

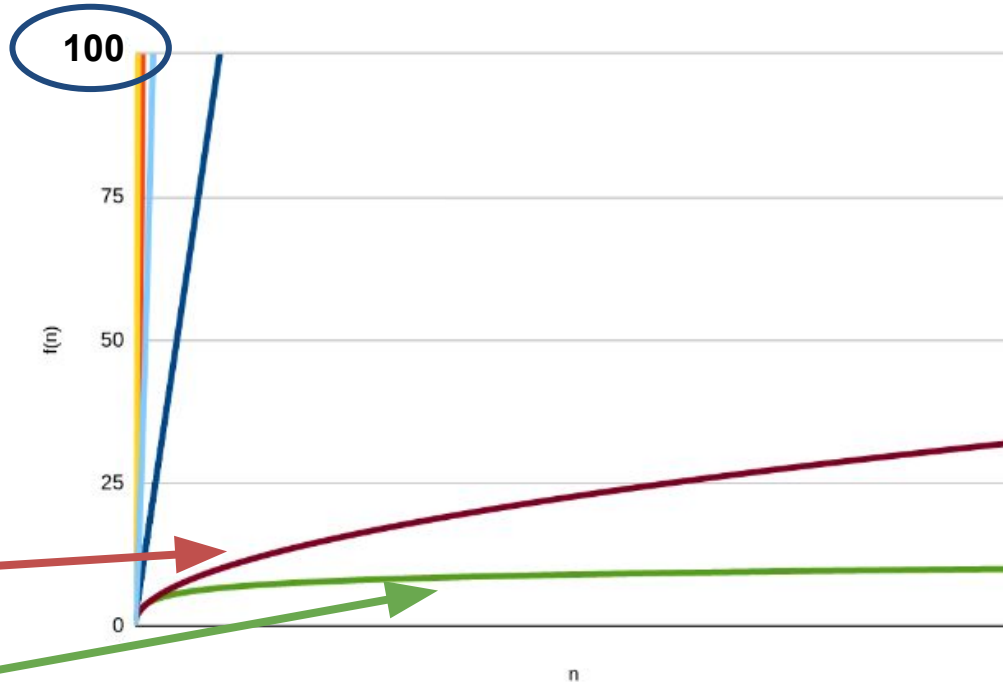
b)  $f(n) = n^2$

c)  $f(n) = n \times \lg(n)$

d)  $f(n) = n$

e)  $f(n) = \text{sqrt}(n)$

f)  $f(n) = \lg(n)$



## Exercício Resolvido (5): Plote os Gráficos

a)  $f(n) = n^2$

b)  $f(n) = 2 \times | - n^2 |$

c)  $f(n) = 3n^2 + 5n - 3$

## Exercício Resolvido (5): Plote os Gráficos

a)  $f(n) = n^2$

b)  $f(n) = 2 \times | - n^2 |$

c)  $f(n) = 3n^2 + 5n - 3$

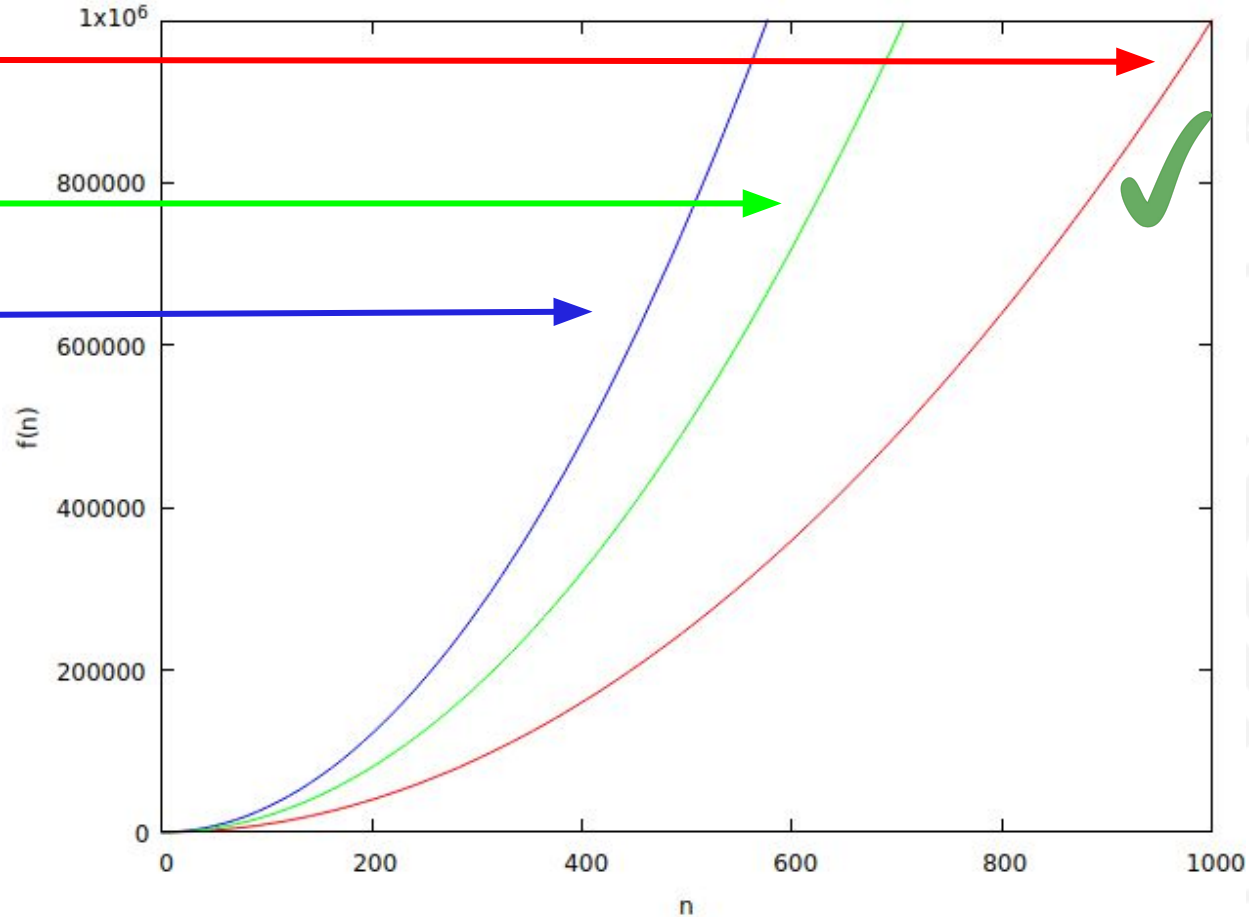
**Pause!**

## Exercício Resolvido (5): Plote os Gráficos

a)  $f(n) = n^2$

b)  $f(n) = 2 \times | -n^2 |$

c)  $f(n) = 3n^2 + 5n - 3$



# Contagem de Operações

# Agenda

- Exercícios Iniciais
  - **Contagem de Operações**
  - Funções de Complexidade
  - Noção sobre a Notação  $\Theta$
- **Estrutura sequencial e condicional**
  - Estrutura de repetição simples
  - Estrutura de repetição dupla
  - Estrutura de repetição com custo logarítmico
  - Mais exercícios resolvidos



# Exercício Resolvido (6)

- Calcule o **número de subtrações** que o código abaixo realiza:

```
...  
a--;  
a -= 3;  
a = a - 2;
```

# Exercício Resolvido (6)

- Calcule o **número de subtrações** que o código abaixo realiza:

```
...  
a--;  
a -= 3;  
a = a - 2;
```

Pause!

# Exercício Resolvido (6)

- Calcule o **número de subtrações** que o código abaixo realiza:

```
...  
a--;  
a -= 3;  
a = a - 2; //três subtrações
```



# Exercício Resolvido (7)

- Calcule o **número de subtrações** que o código abaixo realiza:

```
...  
if (a - 5 < b - 3){  
    i--;  
    --b;  
    a -= 3;  
} else {  
    j--;  
}
```

# Exercício Resolvido (7)

- Calcule o **número de subtrações** que o código abaixo realiza:

```
...  
if (a - 5 < b - 3){  
    i--;  
    --b;  
    a -= 3;  
} else {  
    j--;  
}
```

Pause!

# Exercício Resolvido (7)

- Calcule o **número de subtrações** que o código abaixo realiza:

```
... //Melhor caso
if (a - 5 < b - 3){ //2
    i--;
    --b;
    a -= 3;
} else {
    j--; //1
}
```



# Exercício Resolvido (7)

- Calcule o **número de subtrações** que o código abaixo realiza:

```
... //Pior caso
if (a - 5 < b - 3){ //2
    i--;
    --b; //3
    a -= 3;
} else {
    j--;
}
```



# Cenários Possíveis

- **Melhor caso**: menor “tempo de execução” para todas entradas possíveis de tamanho  $n$
- **Pior caso**: maior “tempo de execução” para todas entradas possíveis
- **Caso médio (ou esperado)<sup>(1)</sup>**: média dos tempos de execução para todas as entradas possíveis

(1) Não abordado neste material



# Contagem com Condicional

- Corresponde ao custo da condição mais o de uma das listas (verdadeira/falsa)

```
if ( condição() ){  
    listaVerdadeiro();  
} else {  
    listaFalso();  
}
```

**Melhor caso:**  $\text{condição()} + \text{mínimo}(\text{listaVerdadeiro}(), \text{listaFalso}())$

**Pior caso:**  $\text{condição()} + \text{máximo}(\text{listaVerdadeiro}(), \text{listaFalso}())$

# Exercício Resolvido (8)

- Calcule o **número de subtrações** que o código abaixo realiza:

```
...  
if (a - 5 < b - 3 || c - 1 < d - 3){  
    i--;  
    --b;  
    a -= 3;  
} else {  
    j--;  
}
```

# Exercício Resolvido (8)

- Calcule o **número de subtrações** que o código abaixo realiza:


```
...  
if (a - 5 < b - 3 || c - 1 < d - 3){  
    i--;  
    --b;  
    a -= 3;  
} else {  
    j--;  
}
```

Pause!

# Exercício Resolvido (8)

- Calcule o **número de subtrações** que o código abaixo realiza:

```
...  
if (a - 5 < b - 3 || c - 1 < d - 3){  
    i--;  
    --b;  
    a -= 3;  
} else {  
    j--;  
}
```



A	B	OR
F	x	x
T	x	T

**Pior caso (7 subtrações):** acontece quando a primeira condição do if é falsa e a segunda, verdadeira. Se a primeira é verdadeira, o *C-like* nem executa a segunda condição


**Melhor caso (5 subtrações):** acontece quando a primeira condição do if é verdadeira ou ambas são falsas

# Exercício Resolvido (8)

- Calcule o **número de subtrações** que o código abaixo realiza:

```
...  
if (a - 5 < b - 3 || c - 1 < d - 3){  
    i--;  
    --b;  
    a -= 3;  
} else {  
    j--;  
}
```

**Operador AND:** Se a primeira condição é falsa, o *C-like* nem executa a segunda



A	B	OR
F	x	x
T	x	T

A	B	AND
F	x	F
T	x	x

# Agenda

- Exercícios Iniciais
  - **Contagem de Operações**
  - Funções de Complexidade
  - Noção sobre a Notação  $\Theta$
- Estrutura sequencial e condicional
  - **Estrutura de repetição simples**
  - Estrutura de repetição dupla
  - Estrutura de repetição com custo logarítmico
  - Mais exercícios resolvidos

# Exercício Resolvido (9)

- Calcule o **número de subtrações** que o código abaixo realiza:

```
...  
for (int i = 0; i < 4; i++){  
    a--;  
}
```

# Exercício Resolvido (9)

- Calcule o **número de subtrações** que o código abaixo realiza:

```
...  
for (int i = 0; i < 4; i++){  
    a--;  
}
```

Pause!



# Exercício Resolvido (9)

- Calcule o **número de subtrações** que o código abaixo realiza:

```
...  
for (int i = 0; i < 4; i++){ // Faremos as subtrações quando i vale  
    a--; // 0, 1, 2, 3  
}
```



# Contagem com Repetição

- Temos  $n$  iterações quando o contador da estrutura de repetição começa com zero, repete enquanto menor que  $n$  e é incrementado em uma unidade

```
for (int  $i = 0$ ;  $i < n$ ;  $i++$ ){  
    lista();  
}
```

# Exercício Resolvido (10)

- Calcule o **número de subtrações** que o código abaixo realiza:

```
...  
for (int i = 0; i < n; i++){  
    a--;  
    b--;  
}
```

**Observação:** Sua resposta deve ser em função de  $n$

# Exercício Resolvido (10)

- Calcule o **número de subtrações** que o código abaixo realiza:

```
...  
for (int i = 0; i < n; i++){  
    a--;  
    b--;  
}
```

**Observação:** Sua resposta deve ser em função de  $n$

Pause!

# Exercício Resolvido (10)

- Calcule o **número de subtrações** que o código abaixo realiza:

```
...  
for (int i = 0; i < n; i++){  
    a--;  
    b--;  
} // 2n subtrações
```

**Observação:** Sua resposta deve ser em função de  $n$



# Exercício Resolvido (11)

- Calcule o **número de subtrações** que o código abaixo realiza:

```
...  
for (int i = 3; i < n; i++){  
    a--;  
}
```

# Exercício Resolvido (11)

- Calcule o **número de subtrações** que o código abaixo realiza:

```
...  
for (int i = 3; i < n; i++){  
    a--;  
}
```

Pause!

# Exercício Resolvido (11)

- Calcule o **número de subtrações** que o código abaixo realiza:

```
...  
for (int i = 3; i < n; i++){  
    a--;  
} // (n - 3) subtrações
```

Por exemplo:

Se  $n = 6$ , temos  $6 - 3 = 3$  subtrações quando  $i$  vale 3, 4, 5

$n = 7$        $7 - 3 = 4$       3, 4, 5, 6





# Contagem com Repetição

- Temos  $(n-a)$  iterações quando o contador da estrutura de repetição começa com  $a$ , repete enquanto menor que  $n$  e é incrementado em uma unidade

```
for (int i = a; i < n ; i++){  
    lista();  
}
```

# Exercício Resolvido (12)

- Calcule o **número de subtrações** que o código abaixo realiza:

```
...  
int i = 0, b = 10;  
  
while (i < 3){  
    i++;  
    b--;  
}
```

# Exercício Resolvido (12)

- Calcule o **número de subtrações** que o código abaixo realiza:

```
...  
int i = 0, b = 10;  
  
while (i < 3){  
    i++;  
    b--;  
}
```

Pause!

# Exercício Resolvido (12)

- Calcule o **número de subtrações** que o código abaixo realiza:

```
...
int i = 0, b = 10;
while (i < 3){
    i++;
    b--;
}
```

// Faremos subtrações quando  
// o valor de i igual a 0, 1 e 2,  
// totalizando três subtrações



# Contagem com Repetição

- Corresponde ao custo da condição mais o número de iterações multiplicado pela soma dos custos da condição e da lista a ser repetida

```
while ( condição() ){  
    lista();  
}
```

**Custo:**  $\text{condição}() + n \times (\text{lista}() + \text{condição}())$ , onde  $n$  é o número de vezes que o laço será repetido

# Exercício Resolvido (13)

- Calcule o **número de subtrações** que o código abaixo realiza:

```
...  
int i = 0, b = 10;  
  
do {  
    i++;  
    b--;  
} while (i < 3);
```



# Exercício Resolvido (13)

- Calcule o **número de subtrações** que o código abaixo realiza:

```
...  
int i = 0, b = 10;  
  
do {  
    i++;  
    b--;  
} while (i < 3);
```



Pause!

# Exercício Resolvido (13)

- Calcule o **número de subtrações** que o código abaixo realiza:

```
...  
int i = 0, b = 10;           // Faremos subtrações quando  
                             // o valor de i igual a 0, 1 e 2,  
do {                         // totalizando três subtrações  
    i++;  
    b--;  
} while (i < 3);
```





# Contagem com Repetição: *do-while*

- Corresponde ao número de iterações multiplicado pela soma dos custos da lista a ser repetida e da condição

```
do {  
    lista();  
} while ( condição() );
```

**Custo:**  $n \times (\text{lista}() + \text{condição}())$ , onde  $n$  é o número de vezes que o laço será repetido

# Agenda

- Exercícios Iniciais
  - **Contagem de Operações**
  - Funções de Complexidade
  - Noção sobre a Notação  $\Theta$
- Estrutura sequencial e condicional
  - Estrutura de repetição simples
  - **Estrutura de repetição dupla**
  - Estrutura de repetição com custo logarítmico
  - Mais exercícios resolvidos

# Exercício Resolvido (14)

- Calcule o **número de subtrações** que o código abaixo realiza:

```
...  
for (int i = 0; i < 3; i++){  
    for (int j = 0; j < 2; j++){  
        a--;  
    }  
}
```

# Exercício Resolvido (14)

- Calcule o **número de subtrações** que o código abaixo realiza:

```
...  
for (int i = 0; i < 3; i++){  
    for (int j = 0; j < 2; j++){  
        a--;  
    }  
}
```

Pause!

# Exercício Resolvido (14)

- Calcule o **número de subtrações** que o código abaixo realiza:

```
...  
for (int i = 0; i < 3; i++){  
    for (int j = 0; j < 2; j++){  
        a--;  
    }  
}
```

Solução fácil: **3 x 2 x 1**

# Exercício Resolvido (14)

- Calcule o **número de subtrações** que o código abaixo realiza:

```
...  
for (int i = 0; i < 3; i++){  
    for (int j = 0; j < 2; j++){  
        a--;  
    }  
}
```

Solução fácil: **3 x 2 x 1**

# Exercício Resolvido (14)

- Calcule o **número de subtrações** que o código abaixo realiza:

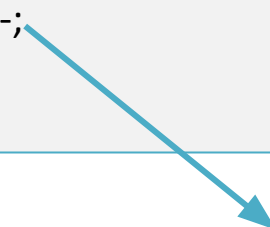
```
...  
for (int i = 0; i < 3; i++){  
    for (int j = 0; j < 2; j++){  
        a--;  
    }  
}
```

Solução fácil:  $3 \times 2 \times 1$

# Exercício Resolvido (14)

- Calcule o **número de subtrações** que o código abaixo realiza:

```
...  
for (int i = 0; i < 3; i++){  
    for (int j = 0; j < 2; j++){  
        a--;  
    }  
}
```



Solução fácil: **3 x 2 x 1**



# Exercício Resolvido (14)

- Calcule o **número de subtrações** que o código abaixo realiza:

```
...  
for (int i = 0; i < 3; i++){  
    for (int j = 0; j < 2; j++){  
        a--;  
    }  
}
```

Solução difícil:

i	j	sub

# Exercício Resolvido (14)

- Calcule o **número de subtrações** que o código abaixo realiza:

```
...  
for (int i = 0; i < 3; i++){  
    for (int j = 0; j < 2; j++){  
        a--;  
    }  
}
```

Solução difícil:

i	j	sub
0		

# Exercício Resolvido (14)

- Calcule o **número de subtrações** que o código abaixo realiza:

```
...
for (int i = 0; i < 3; i++){           true
    for (int j = 0; j < 2; j++){
        a--;
    }
}
```

Solução difícil:

i	j	sub
0		

# Exercício Resolvido (14)

- Calcule o **número de subtrações** que o código abaixo realiza:

```
...  
for (int i = 0; i < 3; i++){  
    for (int j = 0; j < 2; j++){  
        a--;  
    }  
}
```

Solução difícil:

i	j	sub
0	0	

# Exercício Resolvido (14)

- Calcule o **número de subtrações** que o código abaixo realiza:

```
...  
for (int i = 0; i < 3; i++){           true  
    for (int j = 0; j < 2; j++){  
        a--;  
    }  
}
```

Solução difícil:

i	j	sub
0	0	

# Exercício Resolvido (14)

- Calcule o **número de subtrações** que o código abaixo realiza:

```
...  
for (int i = 0; i < 3; i++){  
    for (int j = 0; j < 2; j++){  
        a--;  
    }  
}
```

Solução difícil:

i	j	sub
0	0	1

# Exercício Resolvido (14)

- Calcule o **número de subtrações** que o código abaixo realiza:

```
...  
for (int i = 0; i < 3; i++){  
    for (int j = 0; j < 2; j++){  
        a--;  
    }  
}
```

Solução difícil:

i	j	sub
0	1	1

# Exercício Resolvido (14)

- Calcule o **número de subtrações** que o código abaixo realiza:

```
...  
for (int i = 0; i < 3; i++){           true  
    for (int j = 0; j < 2; j++){  
        a--;  
    }  
}
```

Solução difícil:

i	j	sub
0	1	1



# Exercício Resolvido (14)

- Calcule o **número de subtrações** que o código abaixo realiza:

```
...  
for (int i = 0; i < 3; i++){  
    for (int j = 0; j < 2; j++){  
        a--;  
    }  
}
```

Solução difícil:

i	j	sub
0	1	2

# Exercício Resolvido (14)

- Calcule o **número de subtrações** que o código abaixo realiza:

```
...  
for (int i = 0; i < 3; i++){  
    for (int j = 0; j < 2; j++){  
        a--;  
    }  
}
```

Solução difícil:

i	j	sub
0	2	2

# Exercício Resolvido (14)

- Calcule o **número de subtrações** que o código abaixo realiza:

```
...  
for (int i = 0; i < 3; i++){           false  
    for (int j = 0; j < 2; j++){  
        a--;  
    }  
}
```

Solução difícil:

i	j	sub
0	2	2

# Exercício Resolvido (14)

- Calcule o **número de subtrações** que o código abaixo realiza:

```
...  
for (int i = 0; i < 3; i++){  
    for (int j = 0; j < 2; j++){  
        a--;  
    }  
}
```

Solução difícil:

i	j	sub
1		2

# Exercício Resolvido (14)

- Calcule o **número de subtrações** que o código abaixo realiza:

```
...  
for (int i = 0; i < 3; i++){           true  
    for (int j = 0; j < 2; j++){  
        a--;  
    }  
}
```

Solução difícil:

i	j	sub
1		2

# Exercício Resolvido (14)

- Calcule o **número de subtrações** que o código abaixo realiza:

```
...  
for (int i = 0; i < 3; i++){  
    for (int j = 0; j < 2; j++){  
        a--;  
    }  
}
```

Solução difícil:

i	j	sub
1	0	2

# Exercício Resolvido (14)

- Calcule o **número de subtrações** que o código abaixo realiza:

```
...  
for (int i = 0; i < 3; i++){           true  
    for (int j = 0; j < 2; j++){  
        a--;  
    }  
}
```

Solução difícil:

i	j	sub
1	0	2

# Exercício Resolvido (14)

- Calcule o **número de subtrações** que o código abaixo realiza:

```
...  
for (int i = 0; i < 3; i++){  
    for (int j = 0; j < 2; j++){  
        a--;  
    }  
}
```

Solução difícil:

i	j	sub
1	0	3



# Exercício Resolvido (14)

- Calcule o **número de subtrações** que o código abaixo realiza:

```
...  
for (int i = 0; i < 3; i++){  
    for (int j = 0; j < 2; j++){  
        a--;  
    }  
}
```

Solução difícil:

i	j	sub
1	1	3

# Exercício Resolvido (14)

- Calcule o **número de subtrações** que o código abaixo realiza:

```
...  
for (int i = 0; i < 3; i++){           true  
    for (int j = 0; j < 2; j++){  
        a--;  
    }  
}
```

Solução difícil:

i	j	sub
1	1	3

# Exercício Resolvido (14)

- Calcule o **número de subtrações** que o código abaixo realiza:

```
...  
for (int i = 0; i < 3; i++){  
    for (int j = 0; j < 2; j++){  
        a--;  
    }  
}
```

Solução difícil:

i	j	sub
1	1	4

# Exercício Resolvido (14)

- Calcule o **número de subtrações** que o código abaixo realiza:

```
...  
for (int i = 0; i < 3; i++){  
    for (int j = 0; j < 2; j++){  
        a--;  
    }  
}
```

Solução difícil:

i	j	sub
1	2	4

# Exercício Resolvido (14)

- Calcule o **número de subtrações** que o código abaixo realiza:

```
...  
for (int i = 0; i < 3; i++){           false  
    for (int j = 0; j < 2; j++){  
        a--;  
    }  
}
```

Solução difícil:

i	j	sub
1	2	4

# Exercício Resolvido (14)

- Calcule o **número de subtrações** que o código abaixo realiza:

```
...  
for (int i = 0; i < 3; i++){  
    for (int j = 0; j < 2; j++){  
        a--;  
    }  
}
```

Solução difícil:

i	j	sub
2		4

# Exercício Resolvido (14)

- Calcule o **número de subtrações** que o código abaixo realiza:

```
...  
for (int i = 0; i < 3; i++){           true  
    for (int j = 0; j < 2; j++){  
        a--;  
    }  
}
```

Solução difícil:

i	j	sub
2		4

# Exercício Resolvido (14)

- Calcule o **número de subtrações** que o código abaixo realiza:

```
...  
for (int i = 0; i < 3; i++){  
    for (int j = 0; j < 2; j++){  
        a--;  
    }  
}
```

Solução difícil:

i	j	sub
2	0	4



# Exercício Resolvido (14)

- Calcule o **número de subtrações** que o código abaixo realiza:

```
...  
for (int i = 0; i < 3; i++){           true  
    for (int j = 0; j < 2; j++){  
        a--;  
    }  
}
```

Solução difícil:

i	j	sub
2	0	4

# Exercício Resolvido (14)

- Calcule o **número de subtrações** que o código abaixo realiza:

```
...  
for (int i = 0; i < 3; i++){  
    for (int j = 0; j < 2; j++){  
        a--;  
    }  
}
```

Solução difícil:

i	j	sub
2	0	5

# Exercício Resolvido (14)

- Calcule o **número de subtrações** que o código abaixo realiza:

```
...  
for (int i = 0; i < 3; i++){  
    for (int j = 0; j < 2; j++){  
        a--;  
    }  
}
```

Solução difícil:

i	j	sub
2	1	5

# Exercício Resolvido (14)

- Calcule o **número de subtrações** que o código abaixo realiza:

```
...  
for (int i = 0; i < 3; i++){           true  
    for (int j = 0; j < 2; j++){  
        a--;  
    }  
}
```

Solução difícil:

i	j	sub
2	1	5

# Exercício Resolvido (14)

- Calcule o **número de subtrações** que o código abaixo realiza:

```
...  
for (int i = 0; i < 3; i++){  
    for (int j = 0; j < 2; j++){  
        a--;  
    }  
}
```

Solução difícil:

i	j	sub
2	1	6

# Exercício Resolvido (14)

- Calcule o **número de subtrações** que o código abaixo realiza:

```
...  
for (int i = 0; i < 3; i++){  
    for (int j = 0; j < 2; j++){  
        a--;  
    }  
}
```

Solução difícil:

i	j	sub
2	2	6

# Exercício Resolvido (14)

- Calcule o **número de subtrações** que o código abaixo realiza:

```
...  
for (int i = 0; i < 3; i++){           false  
    for (int j = 0; j < 2; j++){  
        a--;  
    }  
}
```

Solução difícil:

i	j	sub
2	2	6

# Exercício Resolvido (14)

- Calcule o **número de subtrações** que o código abaixo realiza:

```
...  
for (int i = 0; i < 3; i++){  
    for (int j = 0; j < 2; j++){  
        a--;  
    }  
}
```

Solução difícil:

i	j	sub
3		6



# Exercício Resolvido (14)

- Calcule o **número de subtrações** que o código abaixo realiza:

```
...  
for (int i = 0; i < 3; i++){           false  
    for (int j = 0; j < 2; j++){  
        a--;  
    }  
}
```

Solução difícil:

i	j	sub
3		6

# Exercício Resolvido (14)

- Calcule o **número de subtrações** que o código abaixo realiza:

```
...  
for (int i = 0; i < 3; i++){  
    for (int j = 0; j < 2; j++){  
        a--;  
    }  
}
```

Solução difícil:

i	j	sub
		6

# Agenda

- Exercícios Iniciais
  - **Contagem de Operações**
  - Funções de Complexidade
  - Noção sobre a Notação  $\Theta$
- Estrutura sequencial e condicional
  - Estrutura de repetição simples
  - Estrutura de repetição dupla
  - **Estrutura de repetição com custo logarítmico**
  - Mais exercícios resolvidos

# Exercício Resolvido (15)

- Calcule o **número de multiplicações** que o código abaixo realiza:

```
...  
for (int i = n; i > 0; i /= 2){  
    a *= 2;  
}
```

# Exercício Resolvido (15)

- Calcule o **número de multiplicações** que o código abaixo realiza:

```
...  
for (int i = n; i > 0; i /= 2){  
    a *= 2;  
}
```

Pause!

# Exercício Resolvido (15)

- Calcule o **número de multiplicações** que o código abaixo realiza:

```
...  
for (int i = n; i > 0; i /= 2){  
    a *= 2;  
}
```



Supondo  $n = 2^5 = 32$ , temos:

i	mult
32	0

# Exercício Resolvido (15)

- Calcule o **número de multiplicações** que o código abaixo realiza:

```
...  
for (int i = n; i > 0; i /= 2){           true  
    a *= 2;  
}
```



Supondo  $n = 2^5 = 32$ , temos:

i	mult
32	0

# Exercício Resolvido (15)

- Calcule o **número de multiplicações** que o código abaixo realiza:

```
...  
for (int i = n; i > 0; i /= 2){  
    a *= 2;  
}
```



Supondo  $n = 2^5 = 32$ , temos:

i	mult
32	1



# Exercício Resolvido (15)

- Calcule o **número de multiplicações** que o código abaixo realiza:

```
...  
for (int i = n; i > 0; i /= 2){  
    a *= 2;  
}
```



Supondo  $n = 2^5 = 32$ , temos:

i	mult
16	1

# Exercício Resolvido (15)

- Calcule o **número de multiplicações** que o código abaixo realiza:

```
...  
for (int i = n; i > 0; i /= 2){           true  
    a *= 2;  
}
```



Supondo  $n = 2^5 = 32$ , temos:

i	mult
16	1

# Exercício Resolvido (15)

- Calcule o **número de multiplicações** que o código abaixo realiza:

```
...  
for (int i = n; i > 0; i /= 2){  
    a *= 2;  
}
```



Supondo  $n = 2^5 = 32$ , temos:

i	mult
16	2

# Exercício Resolvido (15)

- Calcule o **número de multiplicações** que o código abaixo realiza:

```
...  
for (int i = n; i > 0; i /= 2){  
    a *= 2;  
}
```



Supondo  $n = 2^5 = 32$ , temos:

i	mult
8	2

# Exercício Resolvido (15)

- Calcule o **número de multiplicações** que o código abaixo realiza:

```
...  
for (int i = n; i > 0; i /= 2){           true  
    a *= 2;  
}
```



Supondo  $n = 2^5 = 32$ , temos:

i	mult
8	2

# Exercício Resolvido (15)

- Calcule o **número de multiplicações** que o código abaixo realiza:

```
...  
for (int i = n; i > 0; i /= 2){  
    a *= 2;  
}
```



Supondo  $n = 2^5 = 32$ , temos:

i	mult
8	3

# Exercício Resolvido (15)

- Calcule o **número de multiplicações** que o código abaixo realiza:

```
...  
for (int i = n; i > 0; i /= 2){  
    a *= 2;  
}
```



Supondo  $n = 2^5 = 32$ , temos:

i	mult
4	3

# Exercício Resolvido (15)

- Calcule o **número de multiplicações** que o código abaixo realiza:

```
...  
for (int i = n; i > 0; i /= 2){           true  
    a *= 2;  
}
```



Supondo  $n = 2^5 = 32$ , temos:

i	mult
4	3



# Exercício Resolvido (15)

- Calcule o **número de multiplicações** que o código abaixo realiza:

```
...  
for (int i = n; i > 0; i /= 2){  
    a *= 2;  
}
```



Supondo  $n = 2^5 = 32$ , temos:

i	mult
4	4

# Exercício Resolvido (15)

- Calcule o **número de multiplicações** que o código abaixo realiza:

```
...  
for (int i = n; i > 0; i /= 2){  
    a *= 2;  
}
```



Supondo  $n = 2^5 = 32$ , temos:

i	mult
2	4

# Exercício Resolvido (15)

- Calcule o **número de multiplicações** que o código abaixo realiza:

```
...  
for (int i = n; i > 0; i /= 2){           true  
    a *= 2;  
}
```



Supondo  $n = 2^5 = 32$ , temos:

i	mult
2	4

# Exercício Resolvido (15)

- Calcule o **número de multiplicações** que o código abaixo realiza:

```
...  
for (int i = n; i > 0; i /= 2){  
    a *= 2;  
}
```



Supondo  $n = 2^5 = 32$ , temos:

i	mult
2	5

# Exercício Resolvido (15)

- Calcule o **número de multiplicações** que o código abaixo realiza:

```
...  
for (int i = n; i > 0; i /= 2){  
    a *= 2;  
}
```



Supondo  $n = 2^5 = 32$ , temos:

i	mult
1	5

# Exercício Resolvido (15)

- Calcule o **número de multiplicações** que o código abaixo realiza:

```
...  
for (int i = n; i > 0; i /= 2){           true  
    a *= 2;  
}
```



Supondo  $n = 2^5 = 32$ , temos:

i	mult
1	5

# Exercício Resolvido (15)

- Calcule o **número de multiplicações** que o código abaixo realiza:

```
...  
for (int i = n; i > 0; i /= 2){  
    a *= 2;  
}
```



Supondo  $n = 2^5 = 32$ , temos:

i	mult
1	6

# Exercício Resolvido (15)

- Calcule o **número de multiplicações** que o código abaixo realiza:

```
...  
for (int i = n; i > 0; i /= 2){  
    a *= 2;  
}
```



Supondo  $n = 2^5 = 32$ , temos:

i	mult
0	6



# Exercício Resolvido (15)

- Calcule o **número de multiplicações** que o código abaixo realiza:

```
...  
for (int i = n; i > 0; i /= 2){           false  
    a *= 2;  
}
```



Supondo  $n = 2^5 = 32$ , temos:

i	mult
0	6

$\lg(n) + 1$  vezes

# Exercício Resolvido (15)

- Calcule o **número de multiplicações** que o código abaixo realiza:

```
...  
for (int i = n; i > 0; i /= 2){           false  
    a *= 2;  
}
```



Supondo  $n = 2^5 = 32$ , temos:

i	mult
0	6

$\lg(n) + 1$  vezes

**Por exemplo:**

$n = 8 \Rightarrow 4$  vezes  
 $n = 16 \Rightarrow 5$  vezes  
 $n = 64 \Rightarrow 7$  vezes  
...

# Exercício Resolvido (15)

- Calcule o **número de multiplicações** que o código abaixo realiza:

```
...  
for (int i = n; i > 0; i /= 2){           false  
    a *= 2;  
}
```

O que acontece quando  $n$  não é uma potência de 2?



# Exercício Resolvido (15)

- Calcule o número de multiplicações

```

...
for (int i = n; i > 0; i /= 2){
    a *= 2;
}
    
```

O que acontece quando  $n$  não é uma potência de 2?

Temos  $\lfloor \lg(n) \rfloor + 1$  multiplicações

n	valores de i	multiplicações
16	16 8 4 2 1	5
17	17 8 4 2 1	5
18	18 9 4 2 1	5
19	19 9 4 2 1	5
20	20 10 5 2 1	5
21	21 10 5 2 1	5
22	22 11 5 2 1	5
23	23 11 5 2 1	5
24	24 12 6 3 1	5
25	25 12 6 3 1	5
26	26 13 6 3 1	5
27	27 13 6 3 1	5
28	28 14 7 3 1	5
29	29 14 7 3 1	5
30	30 15 7 3 1	5
31	31 15 7 3 1	5
32	32 16 8 4 2 1 0	6

n	valores de i	multiplicações
32	32 16 8 4 2 1	6
33	33 16 8 4 2 1	6
34	34 17 8 4 2 1	6
35	35 17 8 4 2 1	6
36	36 18 9 4 2 1	6
37	37 18 9 4 2 1	6
38	38 19 9 4 2 1	6
39	39 19 9 4 2 1	6
40	40 20 10 5 2 1	6
41	41 20 10 5 2 1	6
42	42 21 10 5 2 1	6
43	43 21 10 5 2 1	6
44	44 22 11 5 2 1	6
45	45 22 11 5 2 1	6
46	46 23 11 5 2 1	6
47	47 23 11 5 2 1	6
48	48 24 12 6 3 1	6
49	49 24 12 6 3 1	6
50	50 25 12 6 3 1	6
51	51 25 12 6 3 1	6
52	52 26 13 6 3 1	6
53	53 26 13 6 3 1	6
54	54 27 13 6 3 1	6
55	55 27 13 6 3 1	6
56	56 28 14 7 3 1	6
57	57 28 14 7 3 1	6
58	58 29 14 7 3 1	6
59	59 29 14 7 3 1	6
60	60 30 15 7 3 1	6
61	61 30 15 7 3 1	6
62	62 31 15 7 3 1	6
63	63 31 15 7 3 1	6
64	64 32 16 8 4 2 1	7

# Contagem de Operações com Repetição

- Temos um **custo logarítmico** quando a estrutura de repetição reduzir sistematicamente o escopo de busca pela metade

```
...  
for (int i = n; i > 0; i /= 2){  
    lista();  
}
```

# Agenda

- Exercícios Iniciais
  - **Contagem de Operações**
  - Funções de Complexidade
  - Noção sobre a Notação  $\Theta$
- Estrutura sequencial e condicional
  - Estrutura de repetição simples
  - Estrutura de repetição dupla
  - Estrutura de repetição com custo logarítmico
  - **Mais exercícios resolvidos**

# Exercício Resolvido (16)

- Faça um método que receba um número inteiro  $n$  e efetue o **número de subtrações** pedido em:
  - a)  $3n + 2n^2$
  - b)  $5n + 4n^3$
  - c)  $\lg(n) + n$
  - d)  $2n^3 + 5$
  - e)  $2n^4 + 2n^2 + n/2$
  - f)  $\lg(n) + 5 \lg(n)$

# Exercício Resolvido (16)

- Faça um método que receba um número inteiro  $n$  e efetue o **número de subtrações** pedido em:
  - a)  $3n + 2n^2$
  - b)  $5n + 4n^3$
  - c)  $\lg(n) + n$
  - d)  $2n^3 + 5$
  - e)  $2n^4 + 2n^2 + n/2$
  - f)  $\lg(n) + 5 \lg(n)$

Pause!



# Exercício Resolvido (16)

- Faça um método que receba um número inteiro  $n$  e efetue o **número de subtrações** pedido em:

a)  $3n + 2n^2$

b)  $5n + 4n^3$

c)  $\lg(n) + n$

d)  $2n^3 + 5$

e)  $2n^4 + 2n^2 + n/2$

f)  $\lg(n) + 5 \lg(n)$

```
...  
i = 0;  
while (i < n){  
    i++;  
    a--; b--; c--;  
}  
for (i = 0; i < n; i++){  
    for (j = 0; j < n; j++){  
        a--; b--;  
    }  
}
```

# Exercício Resolvido (16)

- Faça um método que receba um número inteiro  $n$  e efetue o **número de subtrações** pedido em:
  - a)  $3n + 2n^2$
  - b)  $5n + 4n^3$**
  - c)  $\lg(n) + n$
  - d)  $2n^3 + 5$
  - e)  $2n^4 + 2n^2 + n/2$
  - f)  $\lg(n) + 5 \lg(n)$

**Pause!**

# Exercício Resolvido (16)

- Faça um método que receba um número inteiro  $n$  e efetue o **número de subtrações** pedido em:

a)  $3n + 2n^2$

b)  $5n + 4n^3$

c)  $\lg(n) + n$

d)  $2n^3 + 5$

e)  $2n^4 + 2n^2 + n/2$

f)  $\lg(n) + 5 \lg(n)$

```
...
i = 0;
while (i < n){
    i++;
    a--; b--; c--; d--; e--;
}
for (i = 0; i < n; i++){
    for (j = 0; j < n; j++){
        for (k = 0; k < n; k++, a--, b--, c--, d--);
    }
}
```

# Exercício Resolvido (16)

- Faça um método que receba um número inteiro  $n$  e efetue o **número de subtrações** pedido em:
  - a)  $3n + 2n^2$
  - b)  $5n + 4n^3$
  - c)  $\lg(n) + n$
  - d)  $2n^3 + 5$
  - e)  $2n^4 + 2n^2 + n/2$
  - f)  $\lg(n) + 5 \lg(n)$

Pause!

# Exercício Resolvido (16)

- Faça um método que receba um número inteiro  $n$  e efetue o **número de subtrações** pedido em:

a)  $3n + 2n^2$

b)  $5n + 4n^3$

c)  $\lg(n) + n$

d)  $2n^3 + 5$

e)  $2n^4 + 2n^2 + n/2$

f)  $\lg(n) + 5 \lg(n)$

```
...  
i = 1;  
while (i < n){  
    i *= 2;  
    a--;  
}  
for (i = 0; i < n; i++){  
    a--;  
}
```

# Exercício Resolvido (16)

- Faça um método que receba um número inteiro  $n$  e efetue o **número de subtrações** pedido em:
  - a)  $3n + 2n^2$
  - b)  $5n + 4n^3$
  - c)  $\lg(n) + n$
  - d)  $2n^3 + 5$**
  - e)  $2n^4 + 2n^2 + n/2$
  - f)  $\lg(n) + 5 \lg(n)$

**Pause!**

# Exercício Resolvido (16)

- Faça um método que receba um número inteiro  $n$  e efetue o **número de subtrações** pedido em:

a)  $3n + 2n^2$

b)  $5n + 4n^3$

c)  $\lg(n) + n$

**d)  $2n^3 + 5$**

e)  $2n^4 + 2n^2 + n/2$

f)  $\lg(n) + 5 \lg(n)$

```
...
for (i = 0; i < n; i++){
    for (j = 0; j < n; j++){
        for (k = 0; k < n; k++, a--, b--);
    }
}

a--; b--; c--; d--; e--;
```

# Exercício Resolvido (16)

- Faça um método que receba um número inteiro  $n$  e efetue o **número de subtrações** pedido em:
  - a)  $3n + 2n^2$
  - b)  $5n + 4n^3$
  - c)  $\lg(n) + n$
  - d)  $2n^3 + 5$
  - e)  $2n^4 + 2n^2 + n/2$
  - f)  $\lg(n) + 5 \lg(n)$

Pause!



# Exercício Resolvido (16)

- Faça um método que receba um número inteiro  $n$  e efetue o **número de subtrações** pedido em:

a)  $3n + 2n^2$

b)  $5n + 4n^3$

c)  $\lg(n) + n$

d)  $2n^3 + 5$

e)  $2n^4 + 2n^2 + n/2$

f)  $\lg(n) + 5 \lg(n)$

```
...
for (i = 0; i < n; i++){
    for (j = 0; j < n; j++){
        a--; b--;
        for (k = 0; k < n; k++){
            for (l = 0; l < n; l++, c--; d--);
        }
    }
    if (i % 2 == 0) e--;
}
```

# Exercício Resolvido (16)

- Faça um método que receba um número inteiro  $n$  e efetue o **número de subtrações** pedido em:
  - a)  $3n + 2n^2$
  - b)  $5n + 4n^3$
  - c)  $\lg(n) + n$
  - d)  $2n^3 + 5$
  - e)  $2n^4 + 2n^2 + n/2$
  - f)  $\lg(n) + 5 \lg(n)$

Pause!

# Exercício Resolvido (16)

- Faça um método que receba um número inteiro  $n$  e efetue o **número de subtrações** pedido em:
  - a)  $3n + 2n^2$
  - b)  $5n + 4n^3$
  - c)  $\lg(n) + n$
  - d)  $2n^3 + 5$
  - e)  $2n^4 + 2n^2 + n/2$
  - f)  $\lg(n) + 5 \lg(n)$

```
...  
i = 1;  
while (i < n){  
    i *= 2;  
    a--; b--; c--; d--; e--; f--;  
}
```

# Funções de Complexidade

# Funções de Complexidade

- Resultam da contagem do número de operações
- Mensuram a quantidade de recursos (como tempo e espaço) que um algoritmo requer à medida que aumentamos o tamanho da sua entrada
- Podem ter como entrada, por exemplo:
  - Tamanho do *array* (pesquisa e ordenação)
  - Número de vértices (algoritmos em grafos)

# Exemplo: Algoritmo de Ordenação por Seleção

- Números de comparações e movimentações entre elementos do *array*

$$c(n) = \frac{n^2}{2} - \frac{n}{2}$$

$$m(n) = 3n - 3$$

- Por exemplo, quando  $n = 10$ , temos:

$$c(10) = \frac{10^2}{2} - \frac{10}{2} = 45$$

$$m(10) = 3 \times 10 - 3 = 27$$

- Por exemplo, quando  $n = 100$ , temos:

$$c(100) = \frac{100^2}{2} - \frac{100}{2} = 4950$$

$$m(100) = 3 \times 100 - 3 = 297$$

# Exemplo de Funções de Complexidade

- Considere os algoritmos abaixo e suas funções de complexidade para o **número de subtrações**

```
1  ...
2  a--;
3  a -= 3;
4  a = a - 2;
```

$f(n) = 3$ , em todos os casos

```
1  ...
2  if (a - 5 < b - 3){
3      i--;
4      --b;
5      a -= 3;
6  } else {
7      j--;
8  }
```

$f(n) = \begin{cases} 3, & \text{no melhor caso} \\ 5, & \text{no pior caso} \end{cases}$

```
1  ...
2  for (int i = 0; i < n; i++){
3      a--;
4      b--;
5  }
```

$f(n) = 2n$ , em todos os casos

# Algumas Funções de Complexidade

- **Função de complexidade de tempo** mensura o tempo (número de execuções da operação relevante) necessário na execução de um algoritmo
- **Função de complexidade de espaço** mensura a quantidade de memória necessária na execução de um algoritmo
- **Função de complexidade de energia** mensura a energia consumida na execução de um algoritmo



# Como Calcular a Complexidade de um Algoritmo



# Como Calcular a Complexidade de um Algoritmo

- Da mesma forma que calculamos o custo de um churrasco:
  - Carne: 400 gramas por pessoa (preço médio do kg R\$ 20,00 - picanha, asinha, coraçãozinho ...)
  - Cerveja: 1,2 litros por pessoa (litro R\$ 3,80)
  - Refrigerante: 1 litro por pessoa (Garrafa 2 litros R\$ 3,50)

**Exercício:** Monte a função de complexidade (ou custo) do nosso churrasco

# Como Calcular a Complexidade de um Algoritmo

- Da mesma forma que calculamos o custo de um churrasco:
  - Carne: 400 gramas por pessoa (preço médio do kg R\$ 20,00 - picanha, asinha, coraçãozinho ...)
  - Cerveja: 1,2 litros por pessoa (litro R\$ 3,80)
  - Refrigerante: 1 litro por pessoa (Garrafa 2 litros R\$ 3,50)

**Exercício:** Monte a função de complexidade (ou custo) do nosso churrasco

$$f(n) = n * \frac{400}{1000} * 20 + n * 1,2 * 3,8 + n * 1 * \frac{3,5}{2} = 14,31 * n$$

# Como Calcular a Complexidade de um Algoritmo

- Da mesma forma que calculamos o custo de uma viagem:
  - Passagem
  - Hotel
  - Saídas

# Cálculo de Complexidade

- Outros laços: sempre consideramos o limite superior
- Métodos: consideramos o custo do método
- Métodos recursivos: utilizamos equações de recorrência<sup>(1)</sup>

(1) Não abordado neste material

# Algoritmo Ótimo

- Algoritmo cujo custo é igual ao menor custo possível

# Exercício Resolvido (17)

- Encontre o menor valor em um *array* de inteiros

```
int min = array[0];  
  
for (int i = 1; i < n; i++){  
    if (min > array[i]){  
        min = array[i];  
    }  
}
```

# Exercício Resolvido (17)

- Encontre o menor valor em um *array* de inteiros

```
int min = array[0];  
  
for (int i = 1; i < n; i++){  
    if (min > array[i]){  
        min = array[i];  
    }  
}
```

1º) Qual é a operação relevante?

2º) Quantas vezes ela será executada?



# Exercício Resolvido (17)

- Encontre o menor valor em um *array* de inteiros

```
int min = array[0];  
  
for (int i = 1; i < n; i++){  
    if (min > array[i]){  
        min = array[i];  
    }  
}
```

1º) Qual é a operação relevante?

R: Comparação entre elementos do *array*

2º) Quantas vezes ela será executada?

# Exercício Resolvido (17)

- Encontre o menor valor em um *array* de inteiros

```
int min = array[0];  
  
for (int i = 1; i < n; i++){  
    if (min > array[i]){  
        min = array[i];  
    }  
}
```

**Pause!**

1º) Qual é a operação relevante?

R: Comparação entre elementos do *array*

2º) Quantas vezes ela será executada?

# Exercício Resolvido (17)

- Encontre o menor valor em um *array* de inteiros

```
int min = array[0];  
  
for (int i = 1; i < n; i++){  
    if (min > array[i]){  
        min = array[i];  
    }  
}
```

1º) Qual é a operação relevante?

R: Comparação entre elementos do *array*

2º) Quantas vezes ela será executada?

R: Se tivermos  $n$  elementos:  $T(n) = n - 1$

# Exercício Resolvido (17)

- Encontre o menor valor em um *array* de inteiros

```
int min = array[0];  
  
for (int i = 1; i < n; i++){  
    if (min > array[i]){  
        min = array[i];  
    }  
}
```

1º) Qual é a operação relevante?

R: Comparação entre elementos do *array*

2º) Quantas vezes ela será executada?

R: Se tivermos  $n$  elementos:  $T(n) = n - 1$

3º) O nosso  $T(n) = n - 1$  é para qual dos três casos?

# Exercício Resolvido (17)

- Encontre o menor valor em um *array* de inteiros

```
int min = array[0];  
  
for (int i = 1; i < n; i++){  
    if (min > array[i]){  
        min = array[i];  
    }  
}
```

Pause!

1º) Qual é a operação relevante?

R: Comparação entre elementos do *array*

2º) Quantas vezes ela será executada?

R: Se tivermos  $n$  elementos:  $T(n) = n - 1$

3º) O nosso  $T(n) = n - 1$  é para qual dos três casos?

# Exercício Resolvido (17)

- Encontre o menor valor em um *array* de inteiros

```
int min = array[0];  
  
for (int i = 1; i < n; i++){  
    if (min > array[i]){  
        min = array[i];  
    }  
}
```

1º) Qual é a operação relevante?

R: Comparação entre elementos do *array*

2º) Quantas vezes ela será executada?

R: Se tivermos  $n$  elementos:  $T(n) = n - 1$

3º) O nosso  $T(n) = n - 1$  é para qual dos três casos?

R: Em todos os casos

# Exercício Resolvido (17)

- Encontre o menor valor em um *array* de inteiros

```
int min = array[0];  
  
for (int i = 1; i < n; i++){  
    if (min > array[i]){  
        min = array[i];  
    }  
}
```

4º) O nosso algoritmo é ótimo? Por que?

# Exercício Resolvido (17)

- Encontre o menor valor em um *array* de inteiros

```
int min = array[0];  
  
for (int i = 1; i < n; i++){  
    if (min > array[i]){  
        min = array[i];  
    }  
}
```

4º) O nosso algoritmo é ótimo? Por que?

**Pause!**



# Exercício Resolvido (17)

- Encontre o menor valor em um *array* de inteiros

```
int min = array[0];  
  
for (int i = 1; i < n; i++){  
    if (min > array[i]){  
        min = array[i];  
    }  
}
```

4º) O nosso algoritmo é ótimo? Por que?  
R: Sim porque temos que testar todos os elementos para garantir nossa resposta

# Exercício Resolvido (18)

- Apresente a função de complexidade de tempo (número de comparações entre elementos do *array*) da pesquisa sequencial no melhor e no pior caso

```
boolean resp = false;

for (int i = 0; i < n; i++){
    if (array[i] == x){
        resp = true;
        i = n;
    }
}
```

# Exercício Resolvido (18)

- Apresente a função de complexidade de tempo (número de comparações entre elementos do *array*) da pesquisa sequencial no melhor e no pior caso

```
boolean resp = false;

for (int i = 0; i < n; i++){
    if (array[i] == x){
        resp = true;
        i = n;
    }
}
```

Pause!

# Exercício Resolvido (18)

- Apresente a função de complexidade de tempo (número de comparações entre elementos do array) da pesquisa sequencial no melhor e no pior caso

```
boolean resp = false;

for (int i = 0; i < n; i++){
    if (array[i] == x){
        resp = true;
        i = n;
    }
}
```

Melhor caso: elemento desejado na primeira posição  
 $t(n) = 1$

Pior caso: elemento desejado não está no array ou está na última posição  
 $t(n) = n$

## Exercício Resolvido (19)

- Apresente a função de complexidade de tempo (número de comparações entre elementos do *array*) da pesquisa binária no melhor e no pior caso

```
boolean resp = false;
int dir = n - 1, esq = 0, meio, diferença;
while (esq <= dir) {
    meio = (esq + dir) / 2;
    diferença = (x - array[meio]);
    if (diferença == 0){
        resp = true;
        esq = n;
    } else if (diferença > 0){
        esq = meio + 1;
    } else {
        dir = meio - 1;
    }
}
```

## Exercício Resolvido (19)

- Apresente a função de complexidade de tempo (número de comparações entre elementos do *array*) da pesquisa binária no melhor e no pior caso

```
boolean resp = false;
int dir = n - 1, esq = 0, meio, diferenca;
while (esq <= dir) {
    meio = (esq + dir) / 2;
    diferenca = (x - array[meio]);
    if (diferenca == 0){
        resp = true;
        esq = n;
    } else if (diferenca > 0){
        esq = meio + 1;
    } else {
        dir = meio - 1;
    }
}
```

Pause!

## Exercício Resolvido (19)

- Apresente a função de complexidade de tempo (número de comparações entre elementos do array) da pesquisa binária no melhor e no pior caso

```
boolean resp = false;
int dir = n - 1, esq = 0, meio, diferença;
while (esq <= dir) {
    meio = (esq + dir) / 2;
    diferença = (x - array[meio]);
    if (diferença == 0){
        resp = true;
        esq = n;
    } else if (diferença > 0){
        esq = meio + 1;
    } else {
        dir = meio - 1;
    }
}
```

Melhor caso: elemento desejado está na posição  $[(esq+dir)/2]$

$$t(n) = 1$$

Pior caso: elemento desejado não está no array ou está na última posição procurada; contudo, lembrando que cada iteração reduz o espaço de busca pela metade

$$t(n) = \lg(n)$$

# Exercício Resolvido (20)

- Explique porque o Algoritmo de Seleção realiza  $m(n) = 3n - 3$  movimentações de registros

```
for (int i = 0; i < (n - 1); i++) {  
    int menor = i;  
    for (int j = (i + 1); j < n; j++){  
        if (array[menor] > array[j]){  
            menor = j;  
        }  
    }  
    swap(menor, i);  
}
```

```
void swap(int a, int b) {  
    int temp = array[a];  
    array[a] = array[b];  
    array[b] = temp;  
}
```



# Exercício Resolvido (20)

- Explique porque o Algoritmo de Seleção realiza  $m(n) = 3n - 3$  movimentações de registros

```
for (int i = 0; i < (n - 1); i++) {  
    int menor = i;  
    for (int j = (i + 1); j < n; j++){  
        if (array[menor] > array[j]){  
            menor = j;  
        }  
    }  
    swap(menor, i);  
}
```

Pause!

```
void swap(int a, int b) {  
    int temp = array[a];  
    array[a] = array[b];  
    array[b] = temp;  
}
```

# Exercício Resolvido (20)

- Explique porque o Algoritmo de Seleção realiza  $m(n) = 3n - 3$  movimentações de registros

```
for (int i = 0; i < (n - 1); i++) {  
    int menor = i;  
    for (int j = (i + 1); j < n; j++){  
        if (array[menor] > array[j]){  
            menor = j;  
        }  
    }  
    swap(menor, i);  
}
```

Todos os casos: o laço repete (n-1) vezes sendo que cada repetição chama o swap que faz 3 movimentações, resultando em:

$$t(n) = 3 \times (n-1) = 3n - 3$$

```
void swap(int a, int b) {  
    int temp = array[a];  
    array[a] = array[b];  
    array[b] = temp;  
}
```

## Exercício Resolvido (21)

- Modifique o código do Algoritmo de Seleção para que ele contabilize o número de movimentações de registros

```
for (int i = 0; i < (n - 1); i++) {  
    int menor = i;  
    for (int j = (i + 1); j < n; j++){  
        if (array[menor] > array[j]){  
            menor = j;  
        }  
    }  
    swap(menor, i);  
}
```

## Exercício Resolvido (21)

- Modifique o código do Algoritmo de Seleção para que ele contabilize o número de movimentações de registros

```
for (int i = 0; i < (n - 1); i++) {  
    int menor = i;  
    for (int j = (i + 1); j < n; j++){  
        if (array[menor] > array[j]){  
            menor = j;  
        }  
    }  
    swap(menor, i);  
}
```

Pause!

## Exercício Resolvido (21)

- Modifique o código do Algoritmo de Seleção para que ele contabilize o número de movimentações de registros

```
int mov = 0;
for (int i = 0; i < (n - 1); i++) {
    int menor = i;
    for (int j = (i + 1); j < n; j++){
        if (array[menor] > array[j]){
            menor = j;
        }
    }
    swap(menor, i);
    mov += 3;
}
printf("Teoria: " + (3*n - 3));
printf("Prática: " + mov);
```

# Exercício Resolvido (22)

- Explique porque o Algoritmo de Seleção realiza  $c(n) = \frac{n^2}{2} - \frac{n}{2}$  comparações entre registros

```
for (int i = 0; i < (n - 1); i++) {  
    int menor = i;  
    for (int j = (i + 1); j < n; j++){  
        if (array[menor] > array[j]){  
            menor = j;  
        }  
    }  
    swap(menor, i);  
}
```

# Exercício Resolvido (22)

- Explique porque o Algoritmo de Seleção realiza  $c(n) = \frac{n^2}{2} - \frac{n}{2}$  comparações entre registros

```
for (int i = 0; i < (n - 1); i++) {  
    int menor = i;  
    for (int j = (i + 1); j < n; j++){  
        if (array[menor] > array[j]){  
            menor = j;  
        }  
    }  
    swap(menor, i);  
}
```

Pause!

# Exercício Resolvido (22)

- Explique porque o Algoritmo de Seleção realiza  $c(n) = \frac{n^2}{2} - \frac{n}{2}$  comparações entre registros

```
for (int i = 0; i < (n - 1); i++) {  
    int menor = i;  
    for (int j = (i + 1); j < n; j++){  
        if (array[menor] > array[j]){  
            menor = j;  
        }  
    }  
    swap(menor, i);  
}
```

## CONSIDERAÇÕES INICIAIS:

- 1) Comparações desejadas: no if
- 2) Laço externo repete (n-1) vezes, para: 0, 1, 2, ..., n-2
- 3) Laço interno repete n - (i+1) vezes, para: i+1, i+2, i+3, ..., n-1



# Exercício Resolvido (22)

- Explique porque o Algoritmo de Seleção realiza  $c(n) = \frac{n^2}{2} - \frac{n}{2}$  comparações entre registros

```
for (int i = 0; i < (n - 1); i++) {  
    int menor = i;  
    for (int j = (i + 1); j < n; j++){  
        if (array[menor] > array[j]){  
            menor = j;  
        }  
    }  
    swap(menor, i);  
}
```

i	0	1	2	...	n-2
c(i) = (n - (i+1))	n-1	n-2	n-3	...	1

## CONSIDERAÇÕES INICIAIS:

- 1) Comparações desejadas: no if
- 2) Laço externo repete (n-1) vezes, para: 0, 1, 2, ..., n-2
- 3) Laço interno repete n - (i+1) vezes, para: i+1, i+2, i+3, ..., n-1

# Exercício Resolvido (22)

- Explique porque o Algoritmo de Seleção realiza  $c(n) = \frac{n^2}{2} - \frac{n}{2}$  comparações entre registros

```
for (int i = 0; i < (n - 1); i++) {  
    int menor = i;  
    for (int j = (i + 1); j < n; j++){  
        if (array[menor] > array[j]){  
            menor = j;  
        }  
    }  
    swap(menor, i);  
}
```

i	0	1	2	...	n-2
$c(i) = (n - (i+1))$	n-1	n-2	n-3	...	1

RESPOSTA:

$$c(n) = \sum_{i=0}^{n-2} (n - i - 1)$$

# Noção sobre a Notação $\Theta$

# Notação

- “Mão na roda” e pode ser lida como aproximadamente
- Abordada e detalhada posteriormente
- Neste ponto, apresentamos “somente” noções sobre a notação

# Notação

- Identifica a tendência de crescimento de uma função de complexidade (número de operações), assim um algoritmo que realiza:
  - $f(n) = 1$  operação é  $\Theta(1)$
  - $f(n) = n$  operações é  $\Theta(n)$
  - $f(n) = n^2$  operações é  $\Theta(n^2)$
  - $f(n) = \lg(n)$  operações é  $\Theta(\lg(n))$

# Notação

- Ignora as constantes, assim um algoritmo que realiza:
  - $f(n) = 2, 3$  ou  $5$  operações é  $\Theta(1)$
  - $f(n) = 2n, 3n$  ou  $5n$  operações é  $\Theta(n)$
  - $f(n) = 2n^2, 3n^2$  ou  $5n^2$  operações é  $\Theta(n^2)$
  - $f(n) = 2\lg(n), 3\lg(n)$  ou  $5\lg(n)$  operações é  $\Theta(\lg(n))$

# Notação

- Ignora os termos com menor crescimento das funções de complexidade, assim um algoritmo que realiza:
  - $f(n) = 3n + 2n^2$  operações é  $\Theta(n^2)$
  - $f(n) = 5n + 4n^3$  operações é  $\Theta(n^3)$
  - $f(n) = \lg(n) + n$  operações é  $\Theta(n)$
  - $f(n) = 2n^3 + 5$  operações é  $\Theta(n^3)$
  - $f(n) = 2n^4 + 2n^2 + n/2$  operações é  $\Theta(n^4)$
  - $f(n) = \lg(n) + 5 \lg(n)$  operações é  $\Theta(\lg(n))$

# Tipos de Funções

Ordem de complexidade	Tipo de função	Exemplos de funções
$\Theta(1)$	Constante	<ul style="list-style-type: none"><li>◦ <math>f(n) = 1</math></li><li>◦ <math>f(n) = 2</math></li><li>◦ <math>f(n) = 3</math></li><li>◦ <math>f(n) = 5</math></li></ul>
$\Theta(\lg n)$	Logarítmica	<ul style="list-style-type: none"><li>◦ <math>f(n) = \lg n</math></li><li>◦ <math>f(n) = 2 \times \lg n</math></li><li>◦ <math>f(n) = 3 \times \lg n</math></li><li>◦ <math>f(n) = 5 \times \lg n</math></li><li>◦ <math>f(n) = \lg(n) + 5\lg(n)</math></li><li>◦ <math>f(n) = \lg(n) + 5</math></li></ul>
$\Theta(n)$	Linear	<ul style="list-style-type: none"><li>◦ <math>f(n) = n</math></li><li>◦ <math>f(n) = 2n</math></li><li>◦ <math>f(n) = 3n</math></li><li>◦ <math>f(n) = 5n</math></li><li>◦ <math>f(n) = n + \lg(n)</math></li></ul>
$\Theta(n^2)$	Quadrática	<ul style="list-style-type: none"><li>◦ <math>f(n) = n^2</math></li><li>◦ <math>f(n) = 2n^2</math></li><li>◦ <math>f(n) = 3n^2</math></li><li>◦ <math>f(n) = 5n^2</math></li><li>◦ <math>f(n) = 3n + 2n^2</math></li></ul>
$\Theta(n^3)$	Cúbica	<ul style="list-style-type: none"><li>◦ <math>f(n) = 5n + 4n^3</math></li><li>◦ <math>f(n) = 2n^3 + 5</math></li><li>◦ <math>f(n) = 2n^3 + 4n^2 + 2n + 5</math></li></ul>
$\Theta(n^4)$	Quártica	<ul style="list-style-type: none"><li>◦ <math>f(n) = n^4 + 2n^3 + 4n^2 + 2n + 5</math></li><li>◦ <math>f(n) = 9n^4 + 5n^2 + \frac{n}{2}</math></li></ul>



# Exercício Resolvido (23)

- Calcule o **número de subtrações** que o código abaixo realiza:

```
...
for (int i = 0; i < n; i++){
    if (rand() % 2 == 0){
        a--;
        b--;
    } else {
        c--;
    }
}
```

# Exercício Resolvido (23)

- Calcule o **número de subtrações** que o código abaixo realiza:

```
...  
for (int i = 0; i < n; i++){  
    if (rand() % 2 == 0){  
        a--;  
        b--;  
    } else {  
        c--;  
    }  
}
```

Pause!

# Exercício Resolvido (23)

- Calcule o **número de subtrações** que o código abaixo realiza:

```
...
for (int i = 0; i < n; i++){
    if (rand() % 2 == 0){
        a--;
        b--;
    } else {
        c--;
    }
}
```

//Melhor caso:  $f(n) = n$ , logo,  $\Theta(n)$   
//Pior caso:  $f(n) = 2n$ , logo,  $\Theta(n)$



# Exercícios

# Exercício (1)

- Calcule o **número de subtrações** que o código abaixo realiza:

```
int i = 10;  
while (i >= 7){  
    i--;  
}
```

# Exercício (2)

- Calcule o **número de subtrações** que o código abaixo realiza:

```
...  
for (int i = 5; i >= 2; i--){  
    a--;  
}
```

# Exercício (3)

- Calcule o **número de subtrações** que o código abaixo realiza:

```
...  
for (int i = 0; i < 5; i++){  
    if (i % 2 == 0){  
        a--;  
        b--;  
    } else {  
        c--;  
    }  
}
```

# Exercício (4)

- Calcule o **número de subtrações** que o código abaixo realiza:

```
...  
int i = 10, b = 10;  
while (i > 0){  
    b--;  
    i = i >> 1;  
}  
i = 0;  
while (i < 15){  
    b--;  
    i += 2;  
}
```



# Exercício (5)

- Calcule o **número de multiplicações** que o código abaixo realiza:

```
...
for (int i = 0; i < n; i++){
    for (int j = 0; j < n - 3; j++){
        a *= 2;
    }
}
```

# Exercício (6)

- Calcule o **número de multiplicações** que o código abaixo realiza:

```
...  
for (int i = n - 7; i >= 1; i--){  
    for (int j = 0; j < n; j++){  
        a *= 2;  
    }  
}
```

# Exercício (7)

- Calcule o **número de multiplicações** que o código abaixo realiza:

```
...  
for (int i = n - 7; i >= 1; i--){  
    for (int j = n - 7; j >= 1; j--){  
        a *= 2;  
    }  
}
```

# Exercício (8)

- Calcule o **número de multiplicações** que o código abaixo realiza:

```
...  
for (int i = n; i > 1; i /= 2){  
    a *= 2;  
}
```

# Exercício (9)

- Calcule o **número de multiplicações** que o código abaixo realiza:

```
...  
for (int i = n + 1; i > 0; i /= 2)  
    a *= 2;  
}
```

# Exercício (10)

- Calcule o **número de multiplicações** que o código abaixo realiza:

```
...  
for (int i = 1; i < n; i *= 2)  
    a *= 2;  
}
```

# Exercício (11)

- Calcule o **número de multiplicações** que o código abaixo realiza:

```
...  
for (int i = 1; i <= n; i *= 2)  
    a *= 2;  
}
```

# Exercício (12)

- Calcule o **número de multiplicações** que o código abaixo realiza:

```
...  
for (int i = n+4; i > 0; i >>= 1){  
    a *= 2;  
}
```