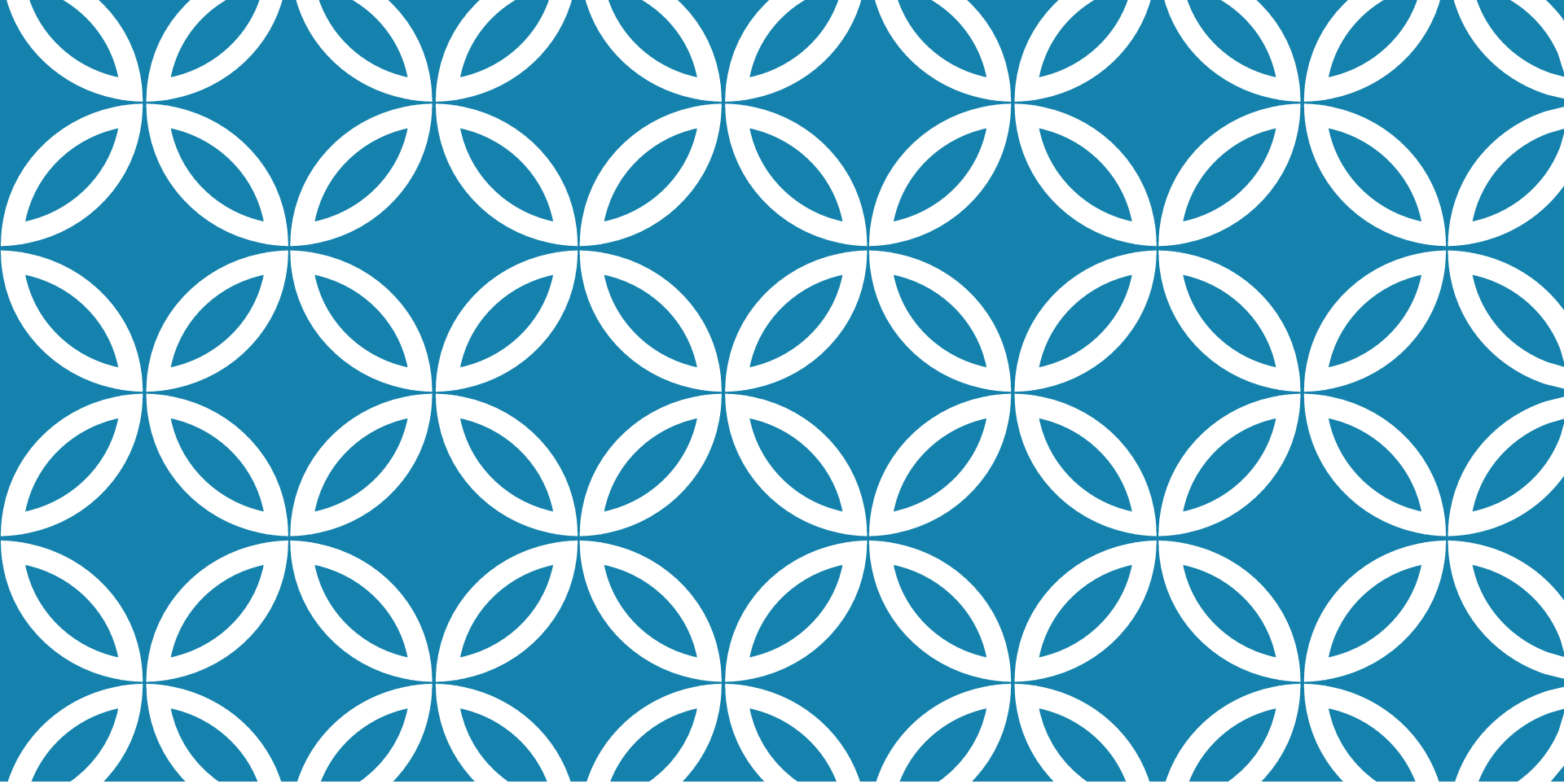


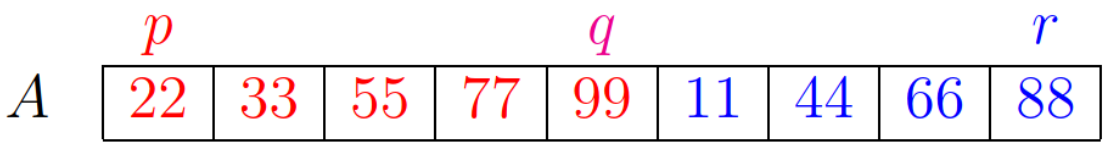
COMPLEXIDADE DE TEMPO DE ALGORITMOS RECURSIVOS

Prof. André Vignatti

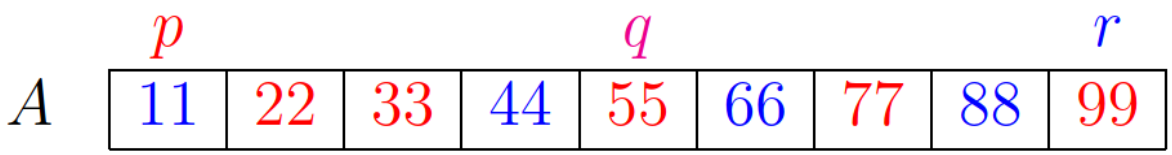
INTERCALAÇÃO

Problema: Dados $A[p \ldots q]$ e $A[q+1 \ldots r]$ crescentes, rearranjar $A[p \ldots r]$ de modo que ele fique em ordem crescente.

Entrada:



Saída:



```
INTERCALA(A, p, q, r)
1    para i ← p até q faça
2        B[i] ← A[i]
3    para j ← q + 1 até r faça
4        B[r + q + 1 − j] ← A[j]
5    i ← p
6    j ← r
7    para k ← p até r faça
8        se B[i] ≤ B[j]
9            então A[k] ← B[i]
10                 i ← i + 1
11           senão A[k] ← B[j]
12                 j ← j − 1
```

```
INTERCALA(A, p, q, r)
1    para i ← p até q faça
2        B[i] ← A[i]
3    para j ← q + 1 até r faça
4        B[r + q + 1 − j] ← A[j]
5    i ← p
6    j ← r
7    para k ← p até r faça
8        se B[i] ≤ B[j]
9            então A[k] ← B[i]
10                 i ← i + 1
11           senão A[k] ← B[j]
12                 j ← j − 1
```

Complexidade de Intercala

Tamanho da entrada: $n = r - p + 1$

Consumo de tempo: $\Theta(n)$

Corretude do Intercala

Invariantes do Intercala: no começo de cada iteração do laço das linhas 7–12:

1. $A[p \ldots k-1]$ está ordenado,
2. $A[p \ldots k-1]$ contém todos os elementos de $B[p \ldots i-1]$ e de $B[j+1 \ldots r]$,
3. $B[i] \geq A[k-1]$ e $B[j] \geq A[k-1]$.

```
INTERCALA(A, p, q, r)
1   para i ← p até q faça
2       B[i] ← A[i]
3   para j ← q + 1 até r faça
4       B[r + q + 1 − j] ← A[j]
5   i ← p
6   j ← r
7   para k ← p até r faça
8       se B[i] ≤ B[j]
9           então A[k] ← B[i]
10                    i ← i + 1
11          senão A[k] ← B[j]
12                    j ← j − 1
```

Corretude do Intercala

Invariantes do Intercala: no começo de cada iteração do laço das linhas 7–12:

1. $A[p \ldots k-1]$ está ordenado,
2. $A[p \ldots k-1]$ contém todos os elementos de $B[p \ldots i-1]$ e de $B[j+1 \ldots r]$,
3. $B[i] \geq A[k-1]$ e $B[j] \geq A[k-1]$.

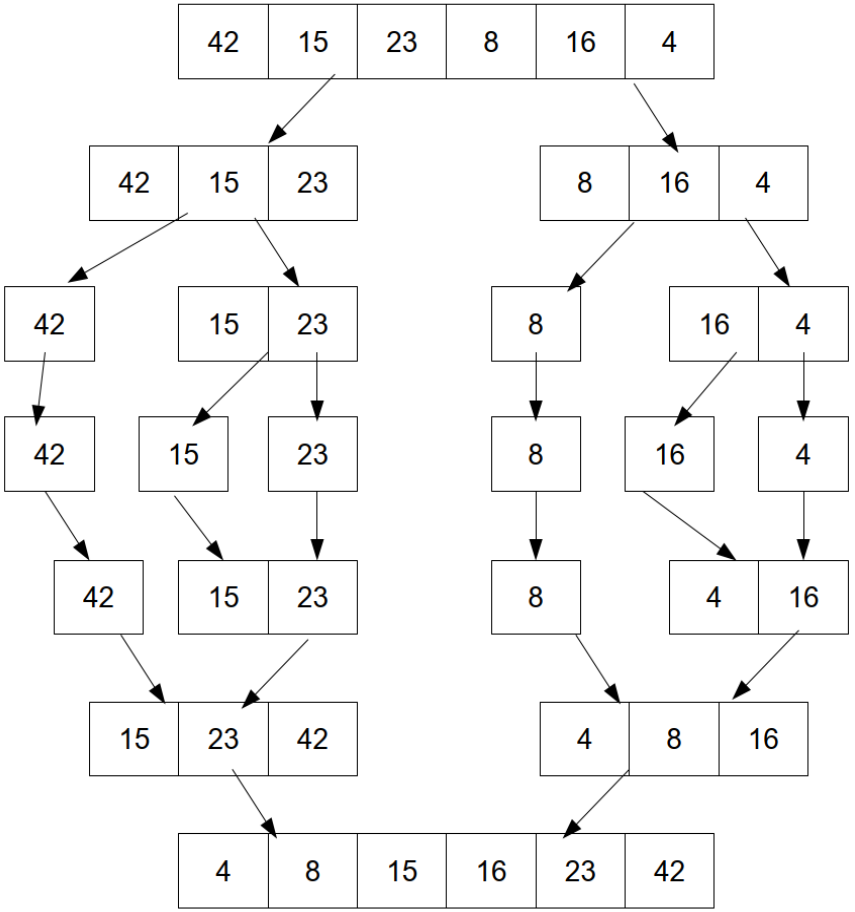
```
INTERCALA(A, p, q, r)
1   para i ← p até q faça
2       B[i] ← A[i]
3   para j ← q + 1 até r faça
4       B[r + q + 1 − j] ← A[j]
5   i ← p
6   j ← r
7   para k ← p até r faça
8       se B[i] ≤ B[j]
9           então A[k] ← B[i]
10                    i ← i + 1
11          senão A[k] ← B[j]
12                    j ← j − 1
```

Exercício. Prove que a afirmação acima é de fato um invariante de INTERCALA.

Exercício. (fácil) Mostre usando o invariante acima que INTERCALA é correto

MERGESORT

```
MERGESORT(A, p, r)
1   se p < r
2       então q ← ⌊(p + r)/2⌋
3       MERGESORT(A, p, q)
4       MERGESORT(A, q + 1, r)
5       INTERCALA(A, p, q, r)
```



DIVISÃO E CONQUISTA

O MergeSort é projetado com a técnica de **divisão e conquista**.

DIVISÃO E CONQUISTA

O MergeSort é projetado com a técnica de **divisão e conquista**.

Algoritmos de divisão-e-conquista possuem (geralmente) três etapas:

DIVISÃO E CONQUISTA

O MergeSort é projetado com a técnica de **divisão e conquista**.

Algoritmos de divisão-e-conquista possuem (geralmente) três etapas:

1. **Divisão:** a instância do problema é dividida em instâncias de tamanho menor, gerando subproblemas.

DIVISÃO E CONQUISTA

O MergeSort é projetado com a técnica de **divisão e conquista**.

Algoritmos de divisão-e-conquista possuem (geralmente) três etapas:

1. **Divisão:** a instância do problema é dividida em instâncias de tamanho menor, gerando subproblemas.

2. **Conquista:** cada subproblema é resolvido recursivamente.

DIVISÃO E CONQUISTA

O MergeSort é projetado com a técnica de **divisão e conquista**.

Algoritmos de divisão-e-conquista possuem (geralmente) três etapas:

1. **Divisão:** a instância do problema é dividida em instâncias de tamanho menor, gerando subproblemas.

2. **Conquista:** cada subproblema é resolvido recursivamente.

3. **Combinação:** as soluções dos subproblemas são combinadas para obter uma solução do problema original.

No caso do MergeSort:

1. **Divisão**: divida o vetor com n elementos em dois subvetores de tamanho $\lfloor n/2 \rfloor$ e $\lceil n/2 \rceil$

No caso do MergeSort:

1. **Divisão**: divida o vetor com n elementos em dois subvetores de tamanho $\lfloor n/2 \rfloor$ e $\lceil n/2 \rceil$

2. **Conquista**: ordene os dois subvetores **recursivamente** usando o Merge-sort

No caso do MergeSort:

1. **Divisão**: divida o vetor com n elementos em dois subvetores de tamanho $\lfloor n/2 \rfloor$ e $\lceil n/2 \rceil$

2. **Conquista**: ordene os dois subvetores **recursivamente** usando o Merge-sort

3. **Combinação**: intercale os dois subvetores para obter um vetor ordenado usando o algoritmo Intercala

ANÁLISE DE ALGORITMOS RECURSIVOS

- Como mostrar a corretude de um algoritmo recursivo?

ANÁLISE DE ALGORITMOS RECURSIVOS

- Como mostrar a corretude de um algoritmo recursivo?
- Como analisar o tempo de um algoritmo recursivo?

ANÁLISE DE ALGORITMOS RECURSIVOS

- Como mostrar a corretude de um algoritmo recursivo?
- Como analisar o tempo de um algoritmo recursivo?
  - O que é uma **relação de recorrência**?

ANÁLISE DE ALGORITMOS RECURSIVOS

- Como mostrar a corretude de um algoritmo recursivo?
- Como analisar o tempo de um algoritmo recursivo?
  - O que é uma **relação de recorrência**?
  - O que significa *resolver* uma relação de recorrência?

Corretude

A corretude do Mergesort apoia-se na corretude do Intercala, e segue por indução em $n = r - p + 1$.

```
MERGESORT(A, p, r)
1   se p < r
2      então q ← ⌊(p + r)/2⌋
3      MERGESORT(A, p, q)
4      MERGESORT(A, q + 1, r)
5      INTERCALA(A, p, q, r)
```

Corretude

A corretude do Mergesort apoia-se na corretude do Intercala, e segue por indução em $n = r - p + 1$.

```
MERGESORT(A, p, r)
1   se p < r
2      então q ← ⌊(p + r)/2⌋
3      MERGESORT(A, p, q)
4      MERGESORT(A, q + 1, r)
5      INTERCALA(A, p, q, r)
```

Complexidade

Seja $T(n)$ o consumo de tempo em função de $n = r - p + 1$

linha	consumo de tempo
1	$\Theta(1)$
2	$\Theta(1)$
3	$T(\lceil n/2 \rceil)$
4	$T(\lfloor n/2 \rfloor)$
5	$\Theta(n)$

Obtemos uma **relação de recorrência**:

$$T(n) = \begin{cases} \Theta(1) & \text{se } n \leq 1 \\ T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + \Theta(n) & \text{se } n > 1 \end{cases}$$

Obtemos uma **relação de recorrência**:

$$T(n) = \begin{cases} \Theta(1) & \text{se } n \leq 1 \\ T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + \Theta(n) & \text{se } n > 1 \end{cases}$$

Em geral, a análise de algoritmos recursivos levam a uma relação de recorrência

Obtemos uma **relação de recorrência**:

$$T(n) = \begin{cases} \Theta(1) & \text{se } n \leq 1 \\ T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + \Theta(n) & \text{se } n > 1 \end{cases}$$

Em geral, a análise de algoritmos recursivos levam a uma relação de recorrência

- É necessário então **resolver** a recorrência!

Obtemos uma **relação de recorrência**:

$$T(n) = \begin{cases} \Theta(1) & \text{se } n \leq 1 \\ T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + \Theta(n) & \text{se } n > 1 \end{cases}$$

Em geral, a análise de algoritmos recursivos levam a uma relação de recorrência

- É necessário então **resolver** a recorrência!
- Ou seja, obter uma "fórmula não-recursiva" ("fórmula fechada") para $T(n)$

Obtemos uma **relação de recorrência**:

$$T(n) = \begin{cases} \Theta(1) & \text{se } n \leq 1 \\ T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + \Theta(n) & \text{se } n > 1 \end{cases}$$

Em geral, a análise de algoritmos recursivos levam a uma relação de recorrência

- É necessário então **resolver** a recorrência!
- Ou seja, obter uma "fórmula não-recursiva" ("fórmula fechada") para $T(n)$

Nas próximas aulas, veremos que $T(n) = \Theta(n \lg n)$ (no pior ou melhor caso?)