Estruturas Discretas 2017.2

Marco Molinaro

Indução Forte

Indução Forte X Indução Fraca

Para provar *Propriedade* para todo inteiro $n \geq n_0$ basta mostrar:

Indução (fraca)

- **(Caso base)** Que *Propriedade* vale para n_0
- **(Passo indutivo)** Assume que *Propriedade* **vale para** n, mostra que vale para $n+1$

Indução Forte X Indução Fraca

Para provar *Propriedade* para todo inteiro $n \geq n_0$ basta mostrar:

Indução (fraca)

- **(Caso base)** Que *Propriedade* vale para n_0
- **(Passo indutivo)** Assume que *Propriedade* **vale para** n, mostra que vale para $n + 1$

Na verdade, no passo indutivo podemos utilizar que a **propriedade vale para todo caso menor ou igual a** n para provar que vale para $n + 1$

Indução Forte X Indução Fraca

Para provar *Propriedade* para todo inteiro $n \geq n_0$ basta mostrar:

Indução (fraca)

- **(Caso base)** Que *Propriedade* vale para n_0
- **(Passo indutivo)** Assume que *Propriedade* **vale para** n, mostra que vale para $n + 1$

Na verdade, no passo indutivo podemos utilizar que a **propriedade vale para todo caso menor ou igual a** n para provar que vale para $n + 1$

Indução forte

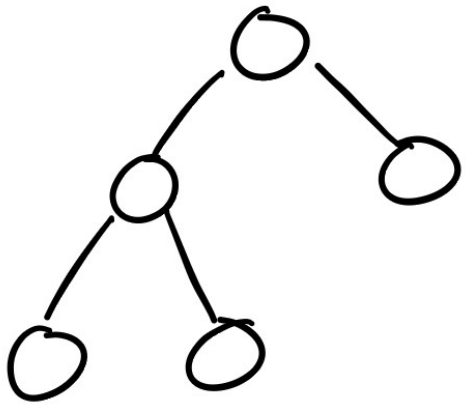
- **(Caso base)** Que *Propriedade* vale para n_0
- **(Passo indutivo)** Assume que propriedade vale para **todos os números** $\{n_0, n_0 + 1, \ldots, n\}$, mostra que vale para n + 1

Exemplos

Definição (Informal)

Uma ***árvore binária enraizada*** *é*

(i) um conjunto vazio ou
(ii) um nó denominado raíz com uma subárvore a esquerda e outra a direita

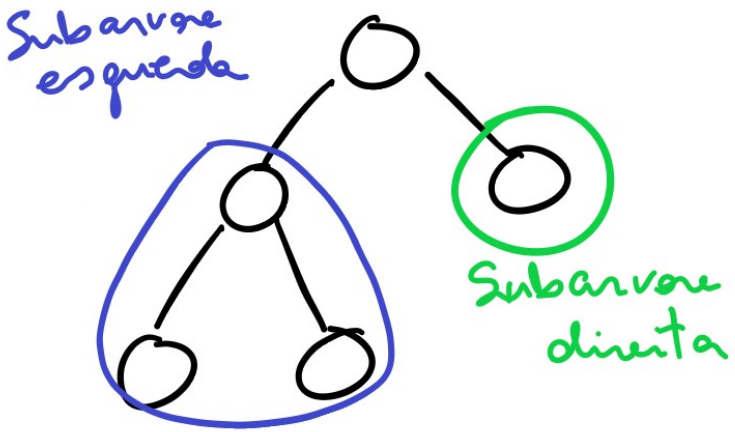


Exemplos

Definição (Informal)

Uma ***árvore binária enraizada*** *é*

(i) um conjunto vazio ou
(ii) um nó denominado raíz com uma subárvore a esquerda e outra a direita

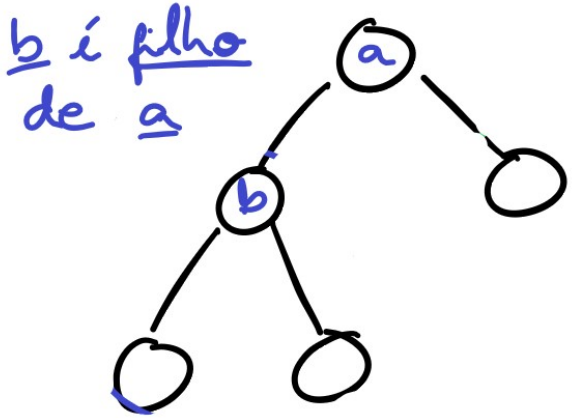


Exemplos

Definição (Informal)

Uma ***árvore binária enraizada*** *é*

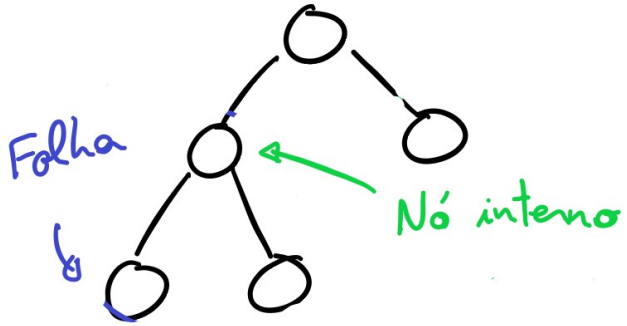
(i) um conjunto vazio ou
(ii) um nó denominado raíz com uma subárvore a esquerda e outra a direita



Definição (Informal)

Dada uma árvore binária enraizada:

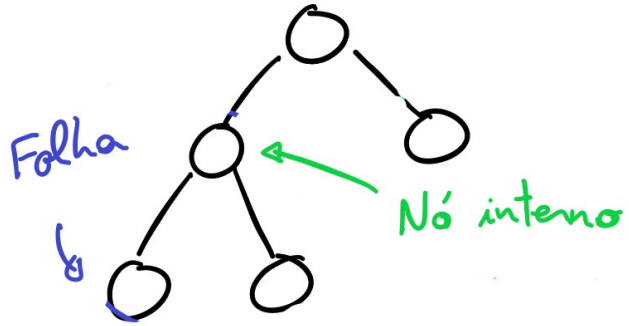
- *Uma* ***folha*** *é um nó com 0 filhos*
- *Um* ***nó interno*** *é um nó com pelo menos um filho*



Definição (Informal)

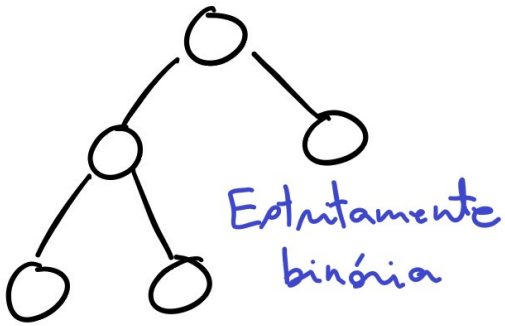
Dada uma árvore binária enraizada:

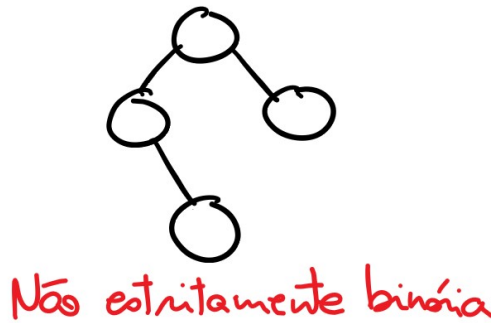
- *Uma **folha** é um nó com 0 filhos*
- *Um **nó interno** é um nó com pelo menos um filho*



Definição

*Uma árvore é dita **estritamente binária** se todo nó possui 0 ou 2 filhos*





Exemplos

Exemplo

Prove por indução no número de nós internos que para toda árvore estritamente binária T temos

$$folhas(T) - internos(T) = 1,$$

onde

$$folhas(T) = número\ de\ folhas$$
$$internos(T) = número\ de\ nós\ internos$$

Corretude de Algoritmos

Corretude de Algoritmos

Lembre a definição de fatorial: $n! = n \cdot (n-1) \cdot (n-2) \cdot \ldots \cdot 2 \cdot 1$

Exemplo (Calculando o fatorial)

Considere o código abaixo.

Fatorial
Entrada: n (inteiro positivo)
Função *Fat*(n)
If $n \leq 1$
Return n
End if
Return $n \cdot Fat(n-1)$
Fim Função

Mostre que a saída de $Fat(n)$ *é* $n!$*, para todo* $n \geq 1$.

Corretude de Algoritmos

Prova: (por indução fraca)
Caso Base: $n = 1$**.** Por causa do "**If**", $Fat(1) = 1 = 1!$

Corretude de Algoritmos

Prova: (por indução fraca)
Caso Base: $n = 1$**.** Por causa do "**If**", $Fat(1) = 1 = 1!$

Passo Indutivo. Suponha que a propriedade valha para n, ou seja $Fat(n) = n!$. Precisamos provar que vale para $n + 1$

Corretude de Algoritmos

Prova: (por indução fraca)
Caso Base: $n = 1$**.** Por causa do "**If**", $Fat(1) = 1 = 1!$

Passo Indutivo. Suponha que a propriedade valha para n, ou seja $Fat(n) = n!$. Precisamos provar que vale para $n + 1$

Pelo código, $Fat(n + 1) = (n + 1) \cdot Fat(n)$

Corretude de Algoritmos

Prova: (por indução fraca)
Caso Base: $n = 1$**.** Por causa do "**If**", $Fat(1) = 1 = 1!$

Passo Indutivo. Suponha que a propriedade valha para n, ou seja $Fat(n) = n!$. Precisamos provar que vale para $n + 1$

Pelo código, $Fat(n + 1) = (n + 1) \cdot Fat(n)$

Pela hipótese indutiva $Fat(n) = n!$, então

Corretude de Algoritmos

Prova: (por indução fraca)
Caso Base: $n = 1$**.** Por causa do "**If**", $Fat(1) = 1 = 1!$

Passo Indutivo. Suponha que a propriedade valha para n, ou seja $Fat(n) = n!$. Precisamos provar que vale para $n + 1$

Pelo código, $Fat(n + 1) = (n + 1) \cdot Fat(n)$

Pela hipótese indutiva $Fat(n) = n!$, então

$$\begin{aligned} Fat(n+1) &= (n+1) \cdot Fat(n) \\ &= (n+1) \cdot n! \\ &= (n+1) \cdot n \cdot (n-1) \cdot \ldots \cdot 2 \cdot 1 \qquad \text{(Pela definição de } n!) \\ &= (n+1)! \end{aligned}$$

Corretude de Algoritmos

Prova: (por indução fraca)
Caso Base: $n = 1$**.** Por causa do "**If**", $Fat(1) = 1 = 1!$

Passo Indutivo. Suponha que a propriedade valha para n, ou seja $Fat(n) = n!$. Precisamos provar que vale para $n + 1$

Pelo código, $Fat(n + 1) = (n + 1) \cdot Fat(n)$

Pela hipótese indutiva $Fat(n) = n!$, então

$$\begin{aligned} Fat(n+1) &= (n+1) \cdot Fat(n) \\ &= (n+1) \cdot n! \\ &= (n+1) \cdot n \cdot (n-1) \cdot \ldots \cdot 2 \cdot 1 \qquad \text{(Pela definição de } n!) \\ &= (n+1)! \end{aligned}$$

Isso conclui o paso indutivo. Fim de prova.

Corretude de Algoritmos

Vamos agora considerar o problema de **ordenação** de uma lista A de números

Corretude de Algoritmos

Vamos agora considerar o problema de **ordenação** de uma lista A de números

Vamos provar a corretude do algoritmo **Quicksort**

Corretude de Algoritmos

Vamos agora considerar o problema de **ordenação** de uma lista A de números

Vamos provar a corretude do algoritmo **Quicksort**

Para simplificar, vamos assumir que os números na lista A são distintos

Quicksort

Quicksort(A,p,r)
Entrada: vetor A, posição de "início" p posição de "termino" r
Saída: vetor A com posições $A[p], A[p+1], \ldots, A[r]$ ordenadas

Se $p < r$ então
    $q \leftarrow$ Partition(A,p,r)
    Quicksort(A,p,q-1)
    Quicksort(A,q+1,r)
Fim Se

Quicksort

Quicksort(A,p,r)
Entrada: vetor A, posição de "início" p posição de "termino" r
Saída: vetor A com posições $A[p], A[p+1], \ldots, A[r]$ ordenadas

Se $p < r$ então
    $q \leftarrow$ Partition(A,p,r)
    Quicksort(A,p,q-1)
    Quicksort(A,q+1,r)
Fim Se

Partition(A,p,r) é uma rotina de particionamento que funciona da seguinte maneira:

- Seja j o número de elementos no subvetor $A[p, r]$ menores ou iguais ao valor $x = A[p]$

Quicksort

Quicksort(A,p,r)
Entrada: vetor A, posição de "início" p posição de "termino" r
Saída: vetor A com posições $A[p], A[p+1], \ldots, A[r]$ ordenadas

Se $p < r$ então
    $q \leftarrow$ Partition(A,p,r)
    Quicksort(A,p,q-1)
    Quicksort(A,q+1,r)
Fim Se

Partition(A,p,r) é uma rotina de particionamento que funciona da seguinte maneira:

- Seja j o número de elementos no subvetor $A[p, r]$ menores ou iguais ao valor $x = A[p]$
- **Partition** coloca os elementos de $A[p, r]$ menores a x no início, ou seja, nas posições $A[p], \ldots, A[p + j - 1]$ (em qualquer ordem)

Quicksort

Quicksort(A,p,r)
Entrada: vetor A, posição de "início" p posição de "termino" r
Saída: vetor A com posições $A[p], A[p+1], \ldots, A[r]$ ordenadas

Se $p < r$ então
    $q \leftarrow$ Partition(A,p,r)
    Quicksort(A,p,q-1)
    Quicksort(A,q+1,r)
Fim Se

Partition(A,p,r) é uma rotina de particionamento que funciona da seguinte maneira:

- Seja j o número de elementos no subvetor $A[p, r]$ menores ou iguais ao valor $x = A[p]$
- **Partition** coloca os elementos de $A[p, r]$ menores a x no início, ou seja, nas posições $A[p], \ldots, A[p + j - 1]$ (em qualquer ordem)
- coloca valor x logo após, na posição $A[p + j]$

Quicksort

Quicksort(A,p,r)
Entrada: vetor A, posição de "início" p posição de "termino" r
Saída: vetor A com posições $A[p], A[p+1], \ldots, A[r]$ ordenadas

Se $p < r$ então
    $q \leftarrow$ Partition(A,p,r)
    Quicksort(A,p,q-1)
    Quicksort(A,q+1,r)
Fim Se

Partition(A,p,r) é uma rotina de particionamento que funciona da seguinte maneira:

- Seja j o número de elementos no subvetor $A[p, r]$ menores ou iguais ao valor $x = A[p]$
- **Partition** coloca os elementos de $A[p, r]$ menores a x no início, ou seja, nas posições $A[p], \ldots, A[p + j - 1]$ (em qualquer ordem)
- coloca valor x logo após, na posição $A[p + j]$
- coloca elementos maiores que x nas outras posições, ou seja $A[p + j + 1], \ldots, A[r]$

Quicksort

Quicksort(A,p,r)
Entrada: vetor A, posição de "início" p posição de "termino" r
Saída: vetor A com posições $A[p], A[p+1], \ldots, A[r]$ ordenadas

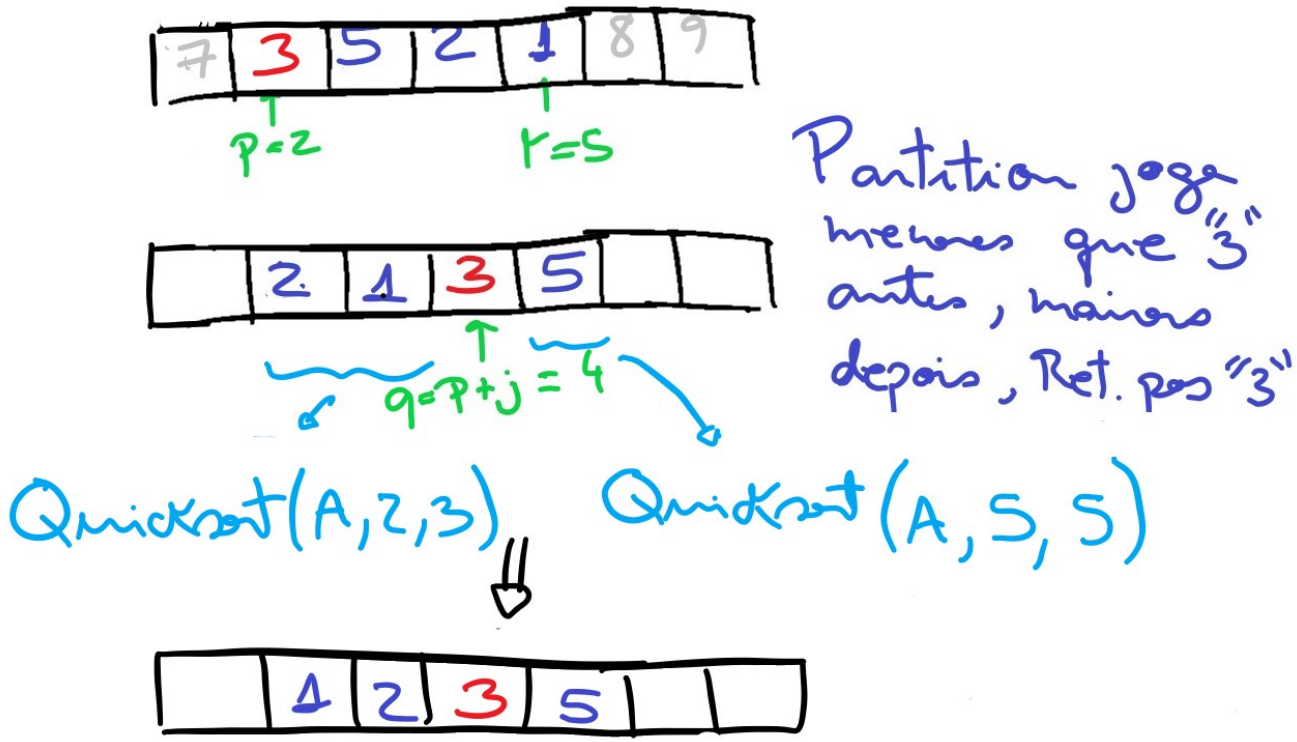
Se $p < r$ então
    $q \leftarrow$ Partition(A,p,r)
    Quicksort(A,p,q-1)
    Quicksort(A,q+1,r)
Fim Se

Partition(A,p,r) é uma rotina de particionamento que funciona da seguinte maneira:

- Seja j o número de elementos no subvetor $A[p, r]$ menores ou iguais ao valor $x = A[p]$
- **Partition** coloca os elementos de $A[p, r]$ menores a x no início, ou seja, nas posições $A[p], \ldots, A[p + j - 1]$ (em qualquer ordem)
- coloca valor x logo após, na posição $A[p + j]$
- coloca elementos maiores que x nas outras posições, ou seja $A[p + j + 1], \ldots, A[r]$
- e retorna posição do x, ou seja, $p + j$

Quicksort

Exemplo de **Quicksort**(A,2,5)



Quicksort

Exemplo (Quicksort)

Assumindo que a rotina Partition funciona da maneira especificada, mostre que Quicksort(A,1,n) ordena o vetor $A[1, n]$.

Quicksort

Exemplo (Quicksort)

Assumindo que a rotina Partition funciona da maneira especificada, mostre que Quicksort(A,1,n) ordena o vetor $A[1, n]$.

Prova: Vamos provar que pra todo $p \leq r$, Quicksort(A, p,r) ordena $A[p, r]$

Quicksort

Exemplo (Quicksort)

Assumindo que a rotina Partition funciona da maneira especificada, mostre que Quicksort(A,1,n) ordena o vetor $A[1, n]$.

Prova: Vamos provar que pra todo $p \leq r$, Quicksort(A, p,r) ordena $A[p, r]$

Prova por indução forte em $n = p - r + 1$ (tamanho to subvetor que queremos ordenar)

Quicksort

Exemplo (Quicksort)

Assumindo que a rotina Partition funciona da maneira especificada, mostre que Quicksort(A,1,n) ordena o vetor $A[1, n]$.

Prova: Vamos provar que pra todo $p \leq r$, Quicksort(A, p,r) ordena $A[p, r]$

Prova por indução forte em $n = p - r + 1$ (tamanho to subvetor que queremos ordenar)

Casos base: $p - r + 1 = 0$ **e** $p - r + 1 = 1$**.** Em ambos os casos Quicksort(A,p,r) não faz nada

Quicksort

Exemplo (Quicksort)

Assumindo que a rotina Partition funciona da maneira especificada, mostre que Quicksort(A,1,n) ordena o vetor $A[1, n]$.

Prova: Vamos provar que pra todo $p \leq r$, Quicksort(A, p,r) ordena $A[p, r]$

Prova por indução forte em $n = p - r + 1$ (tamanho to subvetor que queremos ordenar)

Casos base: $p - r + 1 = 0$ **e** $p - r + 1 = 1$**.** Em ambos os casos Quicksort(A,p,r) não faz nada

Ok, pois todo subvetor de tamanho 0 ou 1 já está ordenado

Quicksort

Passo indutivo. Suponha que a Quicksort retorna $A[p, r]$ ordenato desde que $p - r + 1 \leq n$ (indução forte)

Quicksort

Passo indutivo. Suponha que a Quicksort retorna $A[p, r]$ ordenato desde que $p - r + 1 \leq n$ (indução forte)

Precisamos provar que vale quando $p - r + 1 = n + 1$

Quicksort

Passo indutivo. Suponha que a Quicksort retorna $A[p, r]$ ordenato desde que $p - r + 1 \leq n$ (indução forte)

Precisamos provar que vale quando $p - r + 1 = n + 1$

Primeiro, Quicksort(A,p,r) executa Partition(A,p,r)

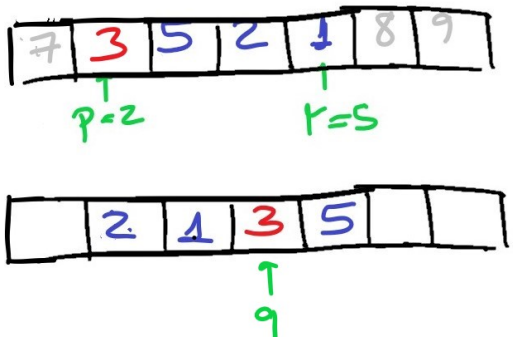
Quicksort

Passo indutivo. Suponha que a Quicksort retorna $A[p, r]$ ordenato desde que $p - r + 1 \leq n$ (indução forte)

Precisamos provar que vale quando $p - r + 1 = n + 1$

Primeiro, Quicksort(A,p,r) executa Partition(A,p,r)

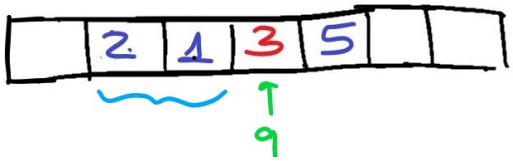
Partition coloca todos os elementos menores que valor $x = A[p]$ à esquerda, e os maiores a direita



$\Rightarrow$ só falta ordenar os subvetores a esquerda e a direita do valor x

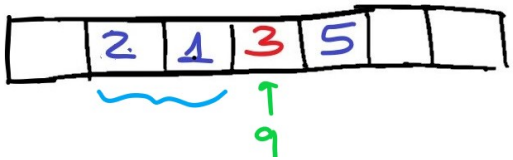
Quicksort

Depois, Quicksort executa a recursão Quicksort(A,p,q-1) (subvetor a esquerda de x)



Quicksort

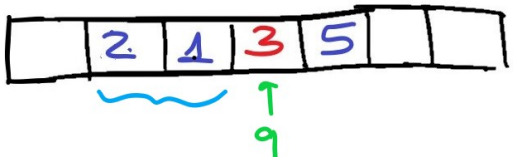
Depois, Quicksort executa a recursão Quicksort(A,p,q-1) (subvetor a esquerda de x)



Essa recursão opera em subvetor de tamanho $r - (q-1) + 1 = r - q$, estritamente menor que $r - p + 1 = n + 1$

Quicksort

Depois, Quicksort executa a recursão Quicksort(A,p,q-1) (subvetor a esquerda de x)

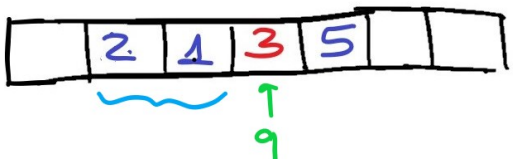


Essa recursão opera em subvetor de tamanho $r - (q-1) + 1 = r - q$, estritamente menor que $r - p + 1 = n + 1$

$\Rightarrow$ tamanho do subvetor é $\leq n$

Quicksort

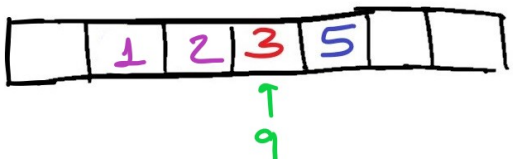
Depois, Quicksort executa a recursão Quicksort(A,p,q-1) (subvetor a esquerda de x)



Essa recursão opera em subvetor de tamanho $r - (q-1) + 1 = r - q$, estritamente menor que $r - p + 1 = n + 1$

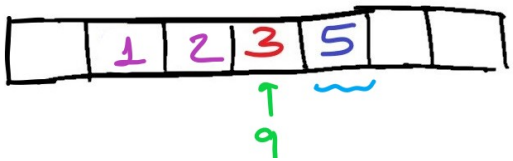
$\Rightarrow$ tamanho do subvetor é $\leq n$

Pela **hipótese indutiva** essa chamada recursiva ordena o subvetor esquerdo $A[p, q-1]$ de forma correta



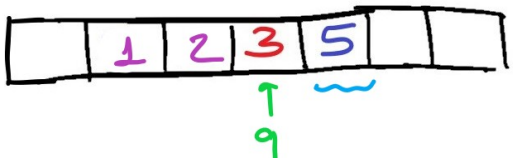
Quicksort

Finalmente, Quicksort executa a recursão Quicksort(A,q+1,r)
(subvetor a direita de x)



Quicksort

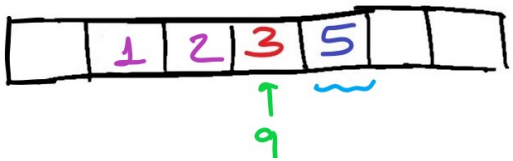
Finalmente, Quicksort executa a recursão Quicksort(A,q+1,r)
(subvetor a direita de x)



Essa recursão opera em subvetor de tamanho $r - (q + 1) + 1 = r - q$, estritamente menor que $r - p + 1 = n + 1$

Quicksort

Finalmente, Quicksort executa a recursão Quicksort(A,q+1,r)
(subvetor a direita de x)

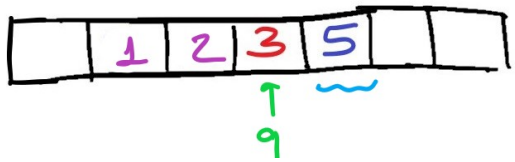


Essa recursão opera em subvetor de tamanho $r - (q + 1) + 1 = r - q$, estritamente menor que $r - p + 1 = n + 1$

$\Rightarrow$ tamanho do subvetor é $\leq n$

Quicksort

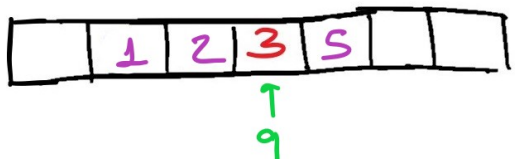
Finalmente, Quicksort executa a recursão Quicksort(A,q+1,r)
(subvetor a direita de x)



Essa recursão opera em subvetor de tamanho $r - (q + 1) + 1 = r - q$, estritamente menor que $r - p + 1 = n + 1$

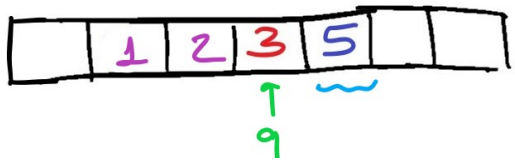
$\Rightarrow$ tamanho do subvetor é $\leq n$

Pela **hipótese indutiva** essa chamada recursiva ordena o subvetor direito $A[q + 1, r]$ de forma correta



Quicksort

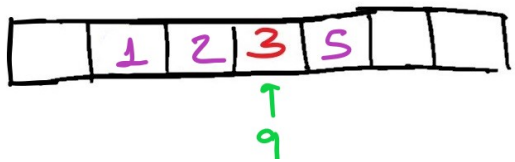
Finalmente, Quicksort executa a recursão Quicksort(A,q+1,r) (subvetor a direita de x)



Essa recursão opera em subvetor de tamanho $r - (q + 1) + 1 = r - q$, estritamente menor que $r - p + 1 = n + 1$

$\Rightarrow$ tamanho do subvetor é $\leq n$

Pela **hipótese indutiva** essa chamada recursiva ordena o subvetor direito $A[q + 1, r]$ de forma correta



Quicksort retorna. Note que vetor $A[p, r]$ está ordenado

Exercicios

Exercicio: Considere o seguinte procedimento

Prog2(int n)
If n $<$ 3
    Imprima('Oi') 4 vezes
    Return
Else
    Prog2(n-1)
    Prog2(n-2)
End if

Seja $T(n)$ o número de vezes que a palavra "OI" é impressa quando Prog2 é chamado com parâmetro n

Prove por indução (forte) que $T(n) \leq 4 \cdot (7/4)^n$ pra todo $n \geq 1$