Projeto e Análise de Algoritmos

Parte 2: Quicksort

Eduardo Freire Nakamura
nakamura@dcc.ufam.edu.br

Departamento de Ciência da Computação (DCC)
Instituto de Ciências Exatas (ICE)
Universidade Federal do Amazonas (UFAM)

O quicksort

- Quicksort (ordenação rápida)
  - No pior caso é $\Theta(n^2)$
  - Na prática a eficiência na média é muito boa
  - O tempo de execução esperado $\Theta(n \log n)$
  - Os fatores ocultos na notação $\Theta(n \log n)$ são pequenos
  - Funciona bem em ambientes de memória virtual

Descrição

- Dividir
  - O arranjo $A[p..r]$ é particionado em dois subarranjos (possivelmente vazios) $A[p..q-1]$ e $A[q+1..r]$ tais que cada elemento de $A[p..q-1]$ é menor que $A[q]$ que, por sua vez, é igual ou menor a cada elemento de $A[q+1..r]$. O índice q é calculado como parte do procedimento
- Conquistar
  - Os dois subarranjos $A[p..q-1]$ e $A[q+1..r]$ são ordenados por chamadas recursivas
- Combinar
  - Os arranos são ordenados localmente, não há a necessidade de combinação

QUICKSORT

```
QUICKSORT(A,p,r)
1: if p < r then
2:    q ← PARTITION(A,p,r);
3:    QUICKSORT(A,p,q-1);
4:    QUICKSORT(A,q+1,r);
6: end if
```

- Para ordenar um arranjo *A*, a chamada inicial seria
  - `QUICKSORT(A,1,n)`
- Onde está o pulo do gato?

PARTITION

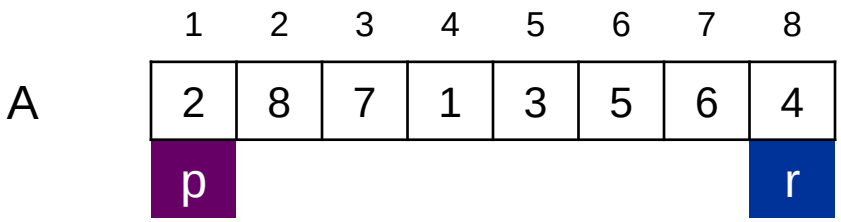
```
PARTITION(A,p,r)
 1: x ← A[r];
 2: i ← p-1;
 3: for j ← p to r-1 do
 4:   if A[j] ≤ x then
 6:     i ← i+1;
 7:     trocar(A[i],A[j]);
 8:   end if
 9: end for
10: trocar(A[i+1],A[r]);
11: return i+1;
```

- O PARTITION sempre seleciona um elemento $x=A[r]$ como um pivô
- O particionamento do arranjo $A[p..r]$ é feito em torno do pivô x

PARTITION

```
PARTITION(A,1,n)
```

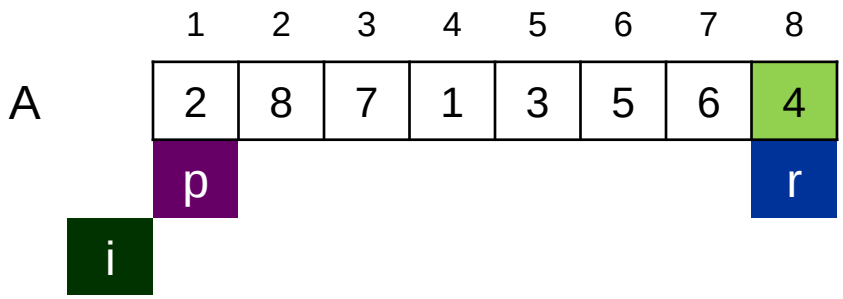
```
PARTITION(A,p,r)
 1: x ← A[r];
 2: i ← p-1;
 3: for j ← p to r-1 do
 4:    if A[j] ≤ x then
 6:       i ← i+1;
 7:       trocar(A[i],A[j]);
 8:    end if
 9: end for
10: trocar(A[i+1],A[r]);
11: return i+1;
```



PARTITION

```
PARTITION(A,1,n)
```

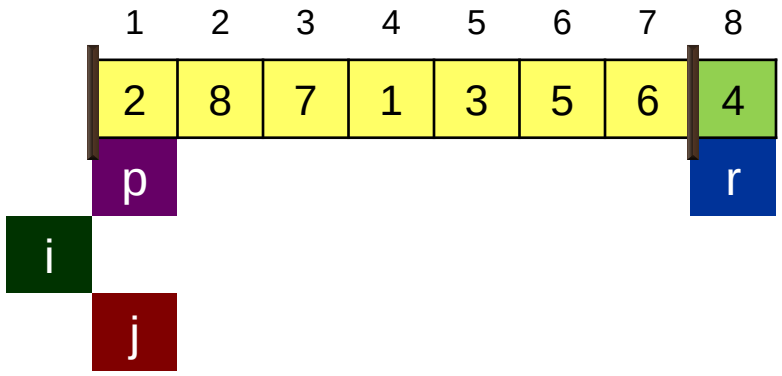
```
PARTITION(A,p,r)
 1: x ← A[r];
 2: i ← p-1;
 3: for j ← p to r-1 do
 4:    if A[j] ≤ x then
 6:       i ← i+1;
 7:       trocar(A[i],A[j]);
 8:    end if
 9: end for
10: trocar(A[i+1],A[r]);
11: return i+1;
```



PARTITION

```
      PARTITION(A,1,n)
```

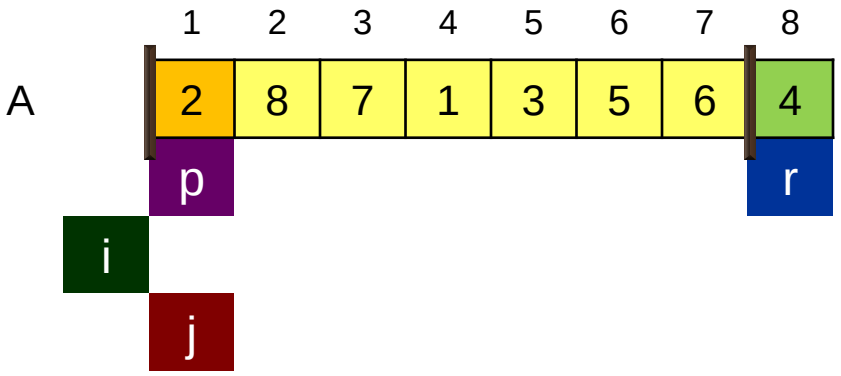
```
PARTITION(A,p,r)
 1: x ← A[r];
 2: i ← p-1;
 3: for j ← p to r-1 do
 4:    if A[j] ≤ x then
 6:       i ← i+1;
 7:       trocar(A[i],A[j]);
 8:    end if
 9: end for
10: trocar(A[i+1],A[r]);
11: return i+1;
```



PARTITION

```
      PARTITION(A,1,n)
```

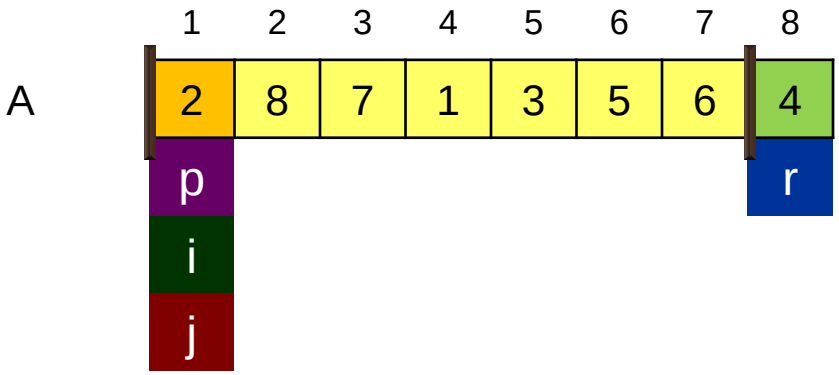
```
PARTITION(A,p,r)
 1: x ← A[r];
 2: i ← p-1;
 3: for j ← p to r-1 do
 4:    if A[j] ≤ x then
 6:       i ← i+1;
 7:       trocar(A[i],A[j]);
 8:    end if
 9: end for
10: trocar(A[i+1],A[r]);
11: return i+1;
```



PARTITION

```
      PARTITION(A,1,n)
```

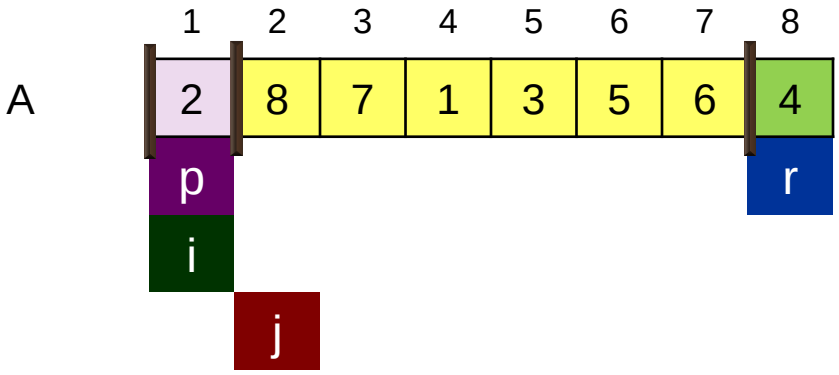
```
PARTITION(A,p,r)
 1: x ← A[r];
 2: i ← p-1;
 3: for j ← p to r-1 do
 4:    if A[j] ≤ x then
 6:       i ← i+1;
 7:       trocar(A[i],A[j]);
 8:    end if
 9: end for
10: trocar(A[i+1],A[r]);
11: return i+1;
```



PARTITION

```
      PARTITION(A,1,n)
```

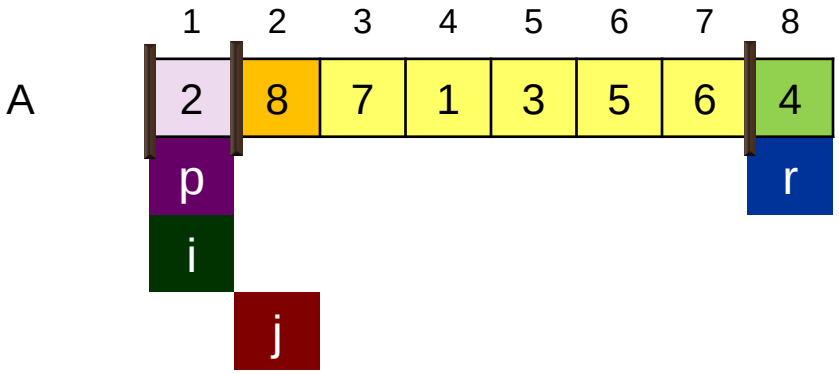
```
PARTITION(A,p,r)
 1: x ← A[r];
 2: i ← p-1;
 3: for j ← p to r-1 do
 4:    if A[j] ≤ x then
 6:       i ← i+1;
 7:       trocar(A[i],A[j]);
 8:    end if
 9: end for
10: trocar(A[i+1],A[r]);
11: return i+1;
```



PARTITION

```
      PARTITION(A,1,n)
```

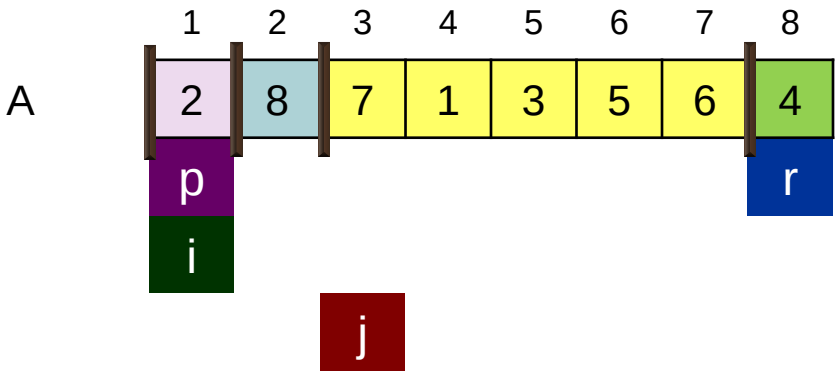
```
PARTITION(A,p,r)
 1: x ← A[r];
 2: i ← p-1;
 3: for j ← p to r-1 do
 4:    if A[j] ≤ x then
 6:       i ← i+1;
 7:       trocar(A[i],A[j]);
 8:    end if
 9: end for
10: trocar(A[i+1],A[r]);
11: return i+1;
```



PARTITION

```
      PARTITION(A,1,n)
```

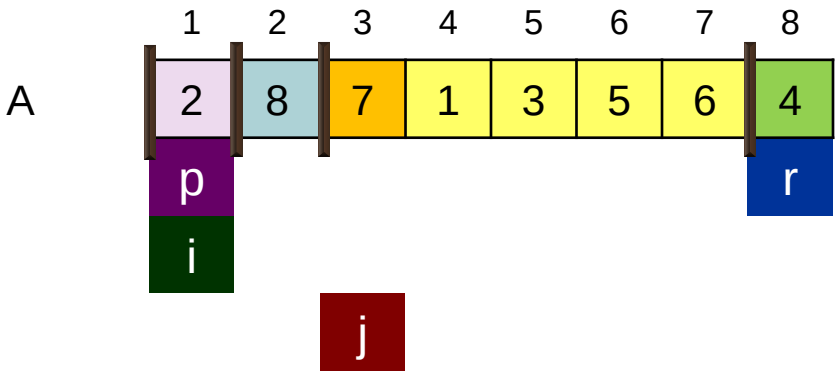
```
PARTITION(A,p,r)
 1: x ← A[r];
 2: i ← p-1;
 3: for j ← p to r-1 do
 4:    if A[j] ≤ x then
 6:       i ← i+1;
 7:       trocar(A[i],A[j]);
 8:    end if
 9: end for
10: trocar(A[i+1],A[r]);
11: return i+1;
```



PARTITION

```
      PARTITION(A,1,n)
```

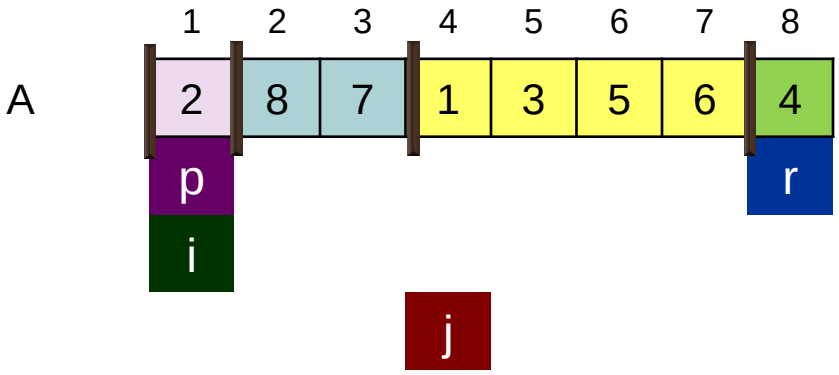
```
PARTITION(A,p,r)
 1: x ← A[r];
 2: i ← p-1;
 3: for j ← p to r-1 do
 4:    if A[j] ≤ x then
 6:       i ← i+1;
 7:       trocar(A[i],A[j]);
 8:    end if
 9: end for
10: trocar(A[i+1],A[r]);
11: return i+1;
```



PARTITION

```
      PARTITION(A,1,n)
```

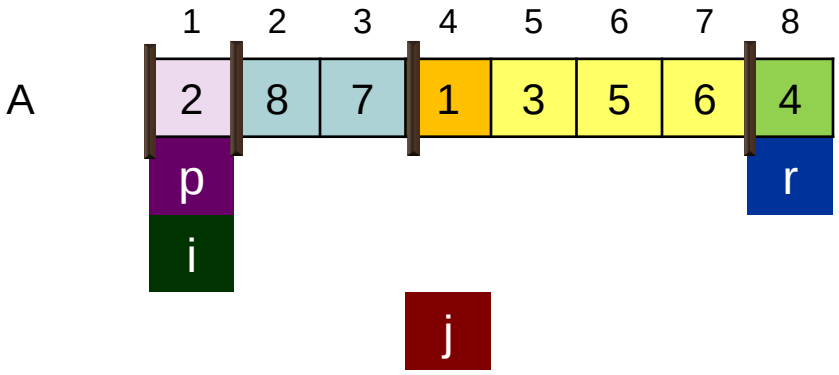
```
PARTITION(A,p,r)
 1: x ← A[r];
 2: i ← p-1;
 3: for j ← p to r-1 do
 4:    if A[j] ≤ x then
 6:       i ← i+1;
 7:       trocar(A[i],A[j]);
 8:    end if
 9: end for
10: trocar(A[i+1],A[r]);
11: return i+1;
```



PARTITION

```
      PARTITION(A,1,n)
```

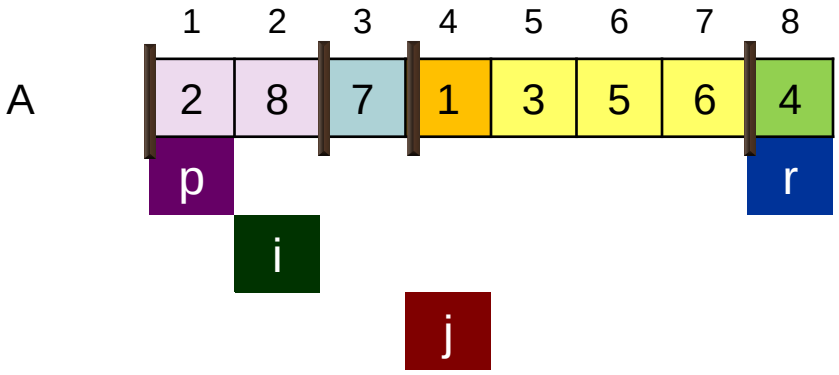
```
PARTITION(A,p,r)
 1: x ← A[r];
 2: i ← p-1;
 3: for j ← p to r-1 do
 4:    if A[j] ≤ x then
 6:       i ← i+1;
 7:       trocar(A[i],A[j]);
 8:    end if
 9: end for
10: trocar(A[i+1],A[r]);
11: return i+1;
```



PARTITION

```
      PARTITION(A,1,n)
```

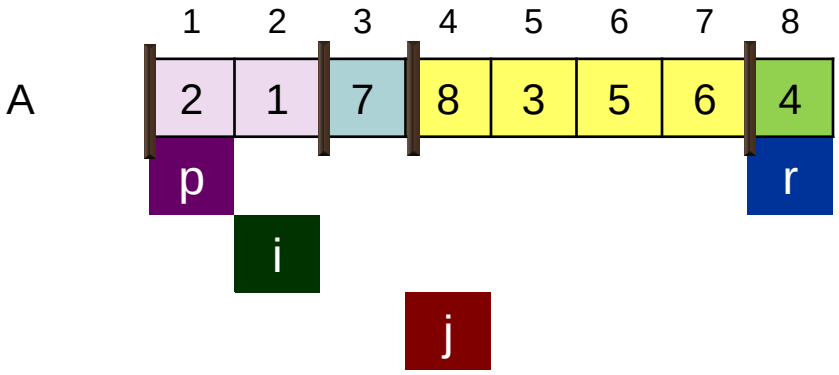
```
PARTITION(A,p,r)
 1: x ← A[r];
 2: i ← p-1;
 3: for j ← p to r-1 do
 4:    if A[j] ≤ x then
 6:       i ← i+1;
 7:       trocar(A[i],A[j]);
 8:    end if
 9: end for
10: trocar(A[i+1],A[r]);
11: return i+1;
```



PARTITION

```
PARTITION(A,1,n)
```

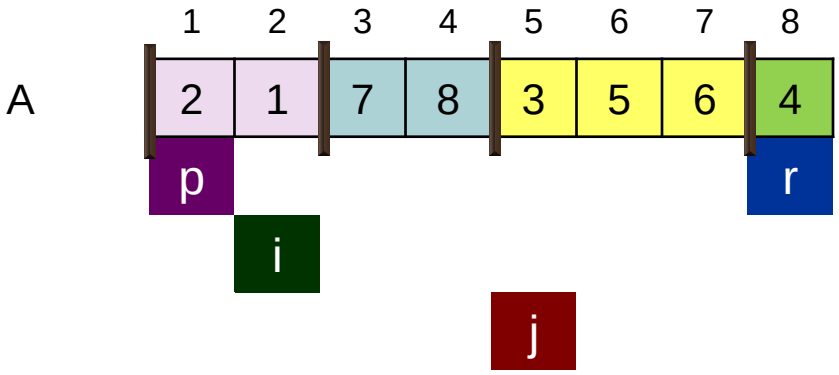
```
PARTITION(A,p,r)
 1: x ← A[r];
 2: i ← p-1;
 3: for j ← p to r-1 do
 4:    if A[j] ≤ x then
 6:       i ← i+1;
 7:       trocar(A[i],A[j]);
 8:    end if
 9: end for
10: trocar(A[i+1],A[r]);
11: return i+1;
```



PARTITION

```
      PARTITION(A,1,n)
```

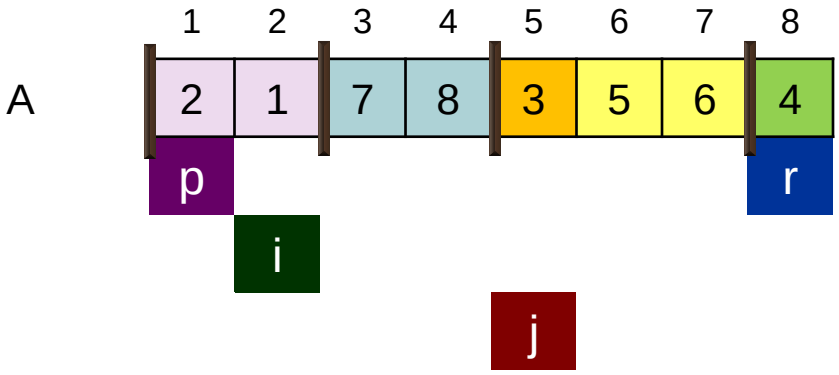
```
PARTITION(A,p,r)
 1: x ← A[r];
 2: i ← p-1;
 3: for j ← p to r-1 do
 4:    if A[j] ≤ x then
 6:       i ← i+1;
 7:       trocar(A[i],A[j]);
 8:    end if
 9: end for
10: trocar(A[i+1],A[r]);
11: return i+1;
```



PARTITION

```
      PARTITION(A,1,n)
```

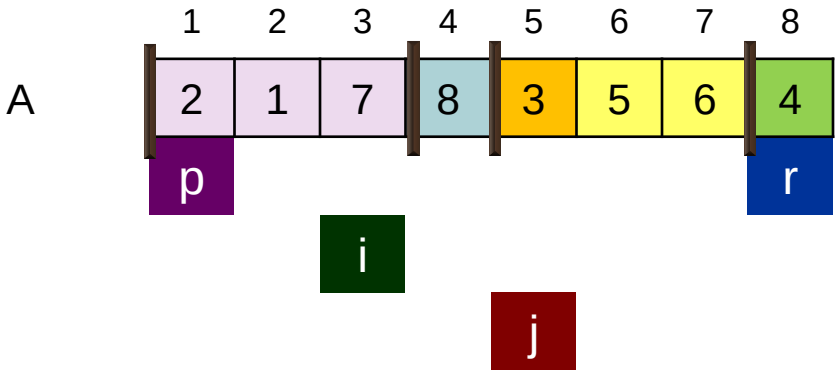
```
PARTITION(A,p,r)
 1: x ← A[r];
 2: i ← p-1;
 3: for j ← p to r-1 do
 4:    if A[j] ≤ x then
 6:       i ← i+1;
 7:       trocar(A[i],A[j]);
 8:    end if
 9: end for
10: trocar(A[i+1],A[r]);
11: return i+1;
```



PARTITION

```
      PARTITION(A,1,n)
```

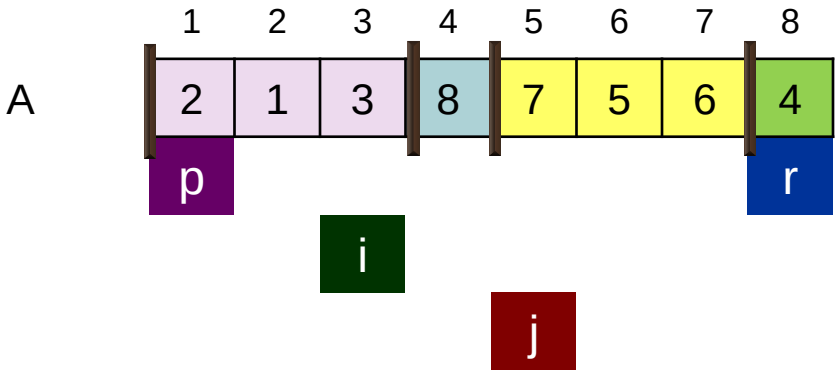
```
PARTITION(A,p,r)
 1: x ← A[r];
 2: i ← p-1;
 3: for j ← p to r-1 do
 4:    if A[j] ≤ x then
 6:       i ← i+1;
 7:       trocar(A[i],A[j]);
 8:    end if
 9: end for
10: trocar(A[i+1],A[r]);
11: return i+1;
```



PARTITION

```
      PARTITION(A,1,n)
```

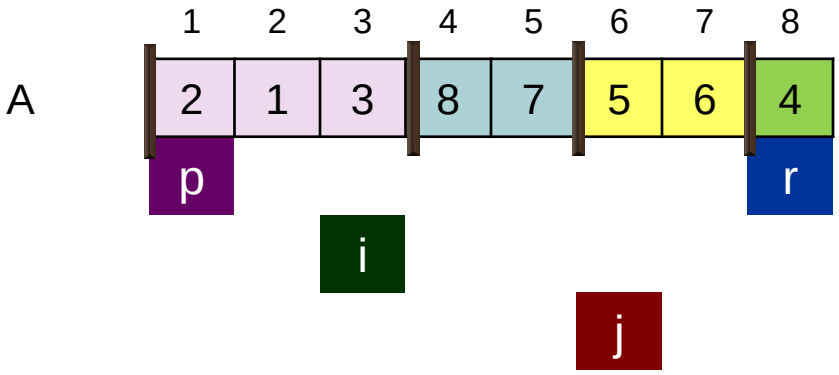
```
PARTITION(A,p,r)
 1: x ← A[r];
 2: i ← p-1;
 3: for j ← p to r-1 do
 4:    if A[j] ≤ x then
 6:       i ← i+1;
 7:       trocar(A[i],A[j]);
 8:    end if
 9: end for
10: trocar(A[i+1],A[r]);
11: return i+1;
```



PARTITION

```
      PARTITION(A,1,n)
```

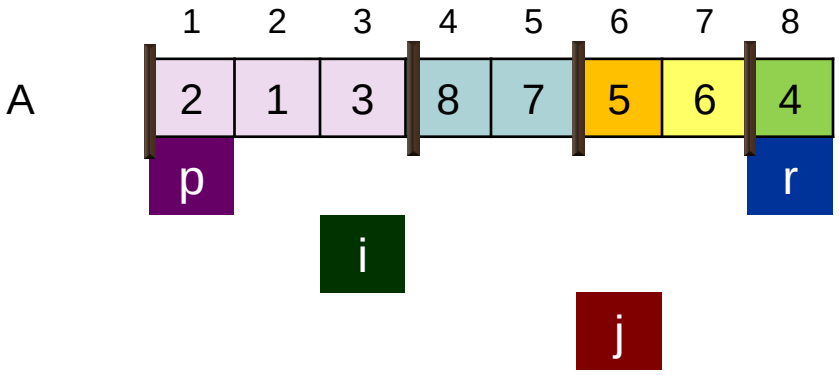
```
PARTITION(A,p,r)
 1: x ← A[r];
 2: i ← p-1;
 3: for j ← p to r-1 do
 4:    if A[j] ≤ x then
 6:       i ← i+1;
 7:       trocar(A[i],A[j]);
 8:    end if
 9: end for
10: trocar(A[i+1],A[r]);
11: return i+1;
```



PARTITION

```
      PARTITION(A,1,n)
```

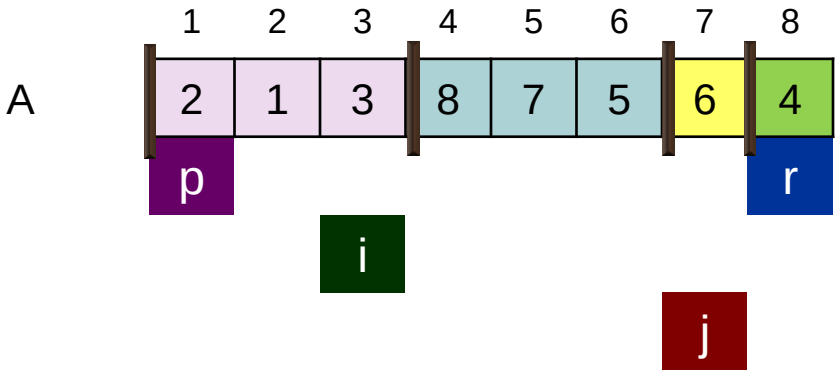
```
PARTITION(A,p,r)
 1: x ← A[r];
 2: i ← p-1;
 3: for j ← p to r-1 do
 4:    if A[j] ≤ x then
 6:       i ← i+1;
 7:       trocar(A[i],A[j]);
 8:    end if
 9: end for
10: trocar(A[i+1],A[r]);
11: return i+1;
```



PARTITION

```
      PARTITION(A,1,n)
```

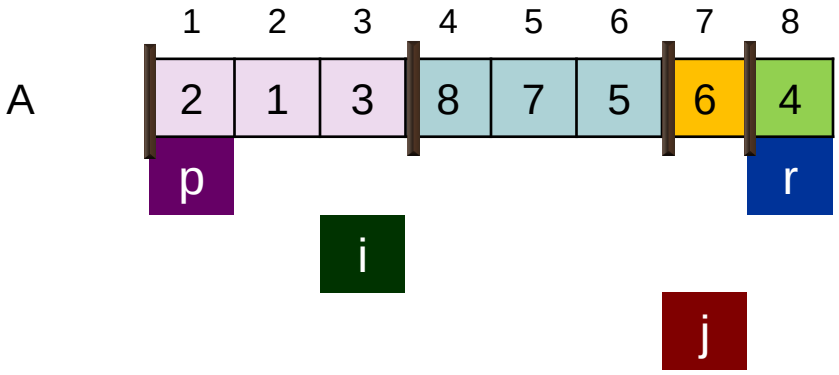
```
PARTITION(A,p,r)
 1: x ← A[r];
 2: i ← p-1;
 3: for j ← p to r-1 do
 4:    if A[j] ≤ x then
 6:       i ← i+1;
 7:       trocar(A[i],A[j]);
 8:    end if
 9: end for
10: trocar(A[i+1],A[r]);
11: return i+1;
```



PARTITION

```
      PARTITION(A,1,n)
```

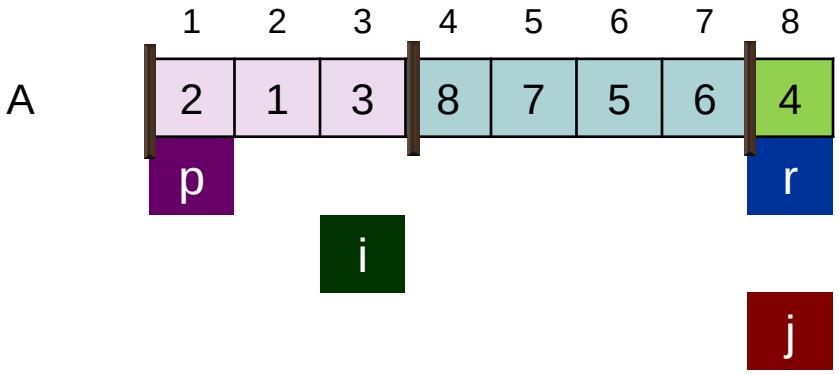
```
PARTITION(A,p,r)
 1: x ← A[r];
 2: i ← p-1;
 3: for j ← p to r-1 do
 4:    if A[j] ≤ x then
 6:       i ← i+1;
 7:       trocar(A[i],A[j]);
 8:    end if
 9: end for
10: trocar(A[i+1],A[r]);
11: return i+1;
```



PARTITION

```
      PARTITION(A,1,n)
```

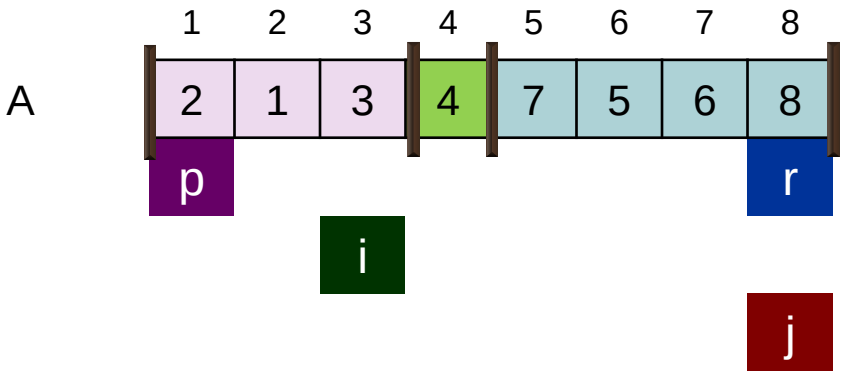
```
PARTITION(A,p,r)
 1: x ← A[r];
 2: i ← p-1;
 3: for j ← p to r-1 do
 4:    if A[j] ≤ x then
 6:       i ← i+1;
 7:       trocar(A[i],A[j]);
 8:    end if
 9: end for
10: trocar(A[i+1],A[r]);
11: return i+1;
```



PARTITION

```
      PARTITION(A,1,n)
```

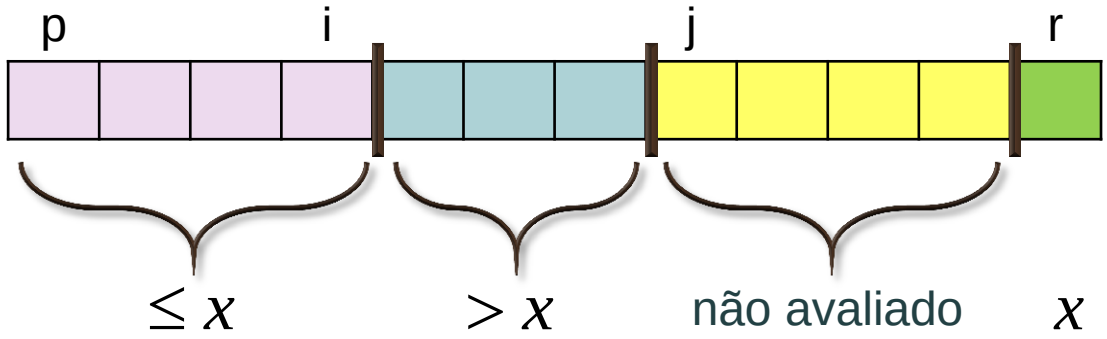
```
PARTITION(A,p,r)
 1: x ← A[r];
 2: i ← p-1;
 3: for j ← p to r-1 do
 4:    if A[j] ≤ x then
 6:       i ← i+1;
 7:       trocar(A[i],A[j]);
 8:    end if
 9: end for
10: trocar(A[i+1],A[r]);
11: return i+1;
```



PARTITION

```
PARTITION(A,p,r)
 1: x ← A[r];
 2: i ← p-1;
 3: for j ← p to r-1 do
 4:    if A[j] ≤ x then
 6:       i ← i+1;
 7:       trocar(A[i],A[j]);
 8:    end if
 9: end for
10: trocar(A[i+1],A[r]);
11: return i+1;
```

- A cada iteração do loop (linhas 3 – 8), para qualquer *k*
  - Se $p \leq k \leq i$, então $A[k] \leq x$
  - Se $i < k < j$, então $A[k] > x$
  - Se $k = r$, então $A[k] = x$
  - Se $j \leq k < r$, então $A[k]$ não foi avaliado ainda



Análise do PARTITION

```
                    PARTITION(A,p,r)
Θ(1)  {  1: x ← A[r];
         2: i ← p-1;
r-p = n-1 vezes ⟶  3: for j ← p to r-1 do
         4:    if A[j] ≤ x then
Θ(1)  {  6:       i ← i+1;
         7:       trocar(A[i],A[j]);
         8:    end if
         9: end for
Θ(1)  { 10: trocar(A[i+1],A[r]);
        11: return i+1;
```

$$T(n) = \Theta(1) + (n-1)\Theta(1) + \Theta(1) = \Theta(n) \quad \Rightarrow \quad T(n) = \Theta(n)$$

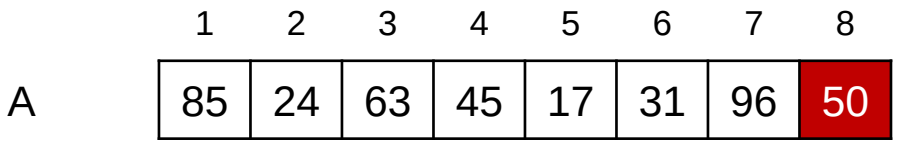
QUICKSORT

```
QUICKSORT(A,p,r)
1: if p < r then
2:    q ← PARTITION(A,p,r);
3:    QUICKSORT(A,p,q-1);
4:    QUICKSORT(A,q+1,r);
6: end if
```

	1	2	3	4	5	6	7	8
A	85	24	63	45	17	31	96	50

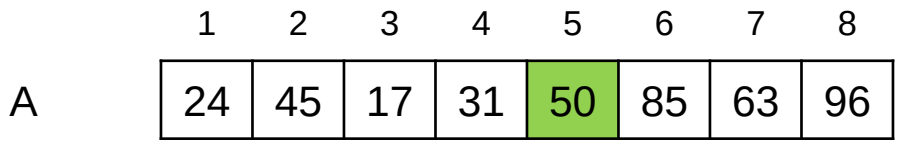
QUICKSORT

```
QUICKSORT(A,p,r)
1: if p < r then
2:    q ← PARTITION(A,p,r);
3:    QUICKSORT(A,p,q-1);
4:    QUICKSORT(A,q+1,r);
6: end if
```



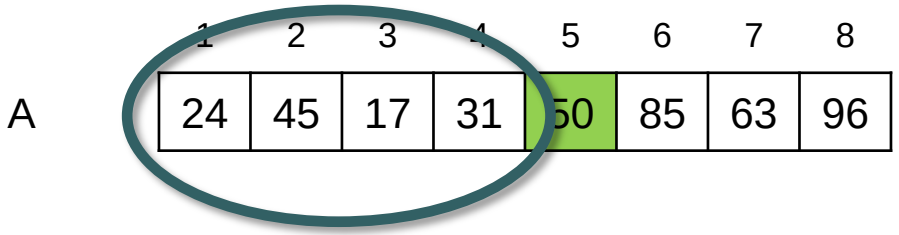
QUICKSORT

```
QUICKSORT(A,p,r)
1: if p < r then
2:    q ← PARTITION(A,p,r);
3:    QUICKSORT(A,p,q-1);
4:    QUICKSORT(A,q+1,r);
6: end if
```



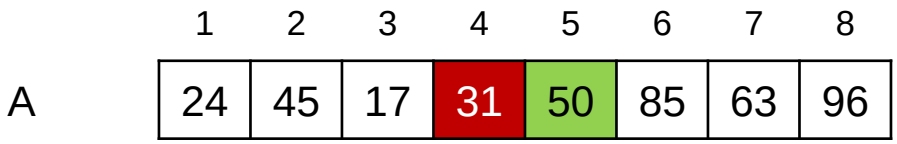
QUICKSORT

```
QUICKSORT(A,p,r)
1: if p < r then
2:    q ← PARTITION(A,p,r);
3:    QUICKSORT(A,p,q-1);
4:    QUICKSORT(A,q+1,r);
6: end if
```



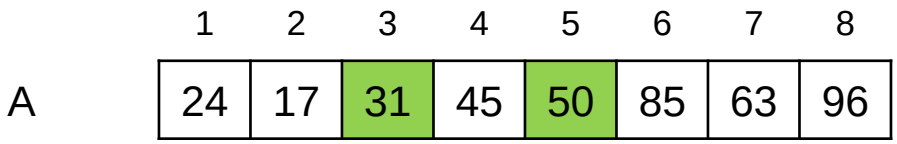
QUICKSORT

```
QUICKSORT(A,p,r)
1: if p < r then
2:    q ← PARTITION(A,p,r);
3:    QUICKSORT(A,p,q-1);
4:    QUICKSORT(A,q+1,r);
6: end if
```



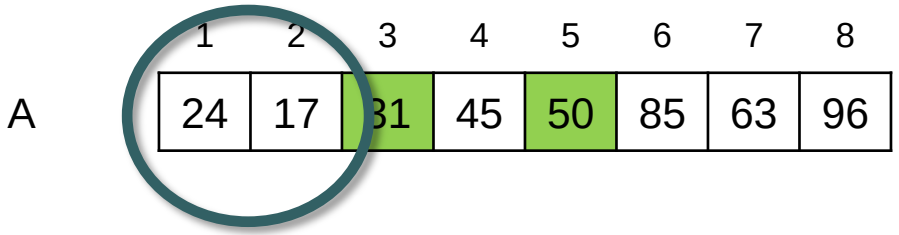
QUICKSORT

```
QUICKSORT(A,p,r)
1: if p < r then
2:    q ← PARTITION(A,p,r);
3:    QUICKSORT(A,p,q-1);
4:    QUICKSORT(A,q+1,r);
6: end if
```



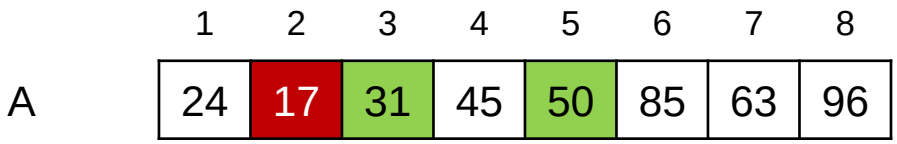
QUICKSORT

```
QUICKSORT(A,p,r)
1: if p < r then
2:    q ← PARTITION(A,p,r);
3:    QUICKSORT(A,p,q-1);
4:    QUICKSORT(A,q+1,r);
6: end if
```



QUICKSORT

```
QUICKSORT(A,p,r)
1: if p < r then
2:    q ← PARTITION(A,p,r);
3:    QUICKSORT(A,p,q-1);
4:    QUICKSORT(A,q+1,r);
6: end if
```



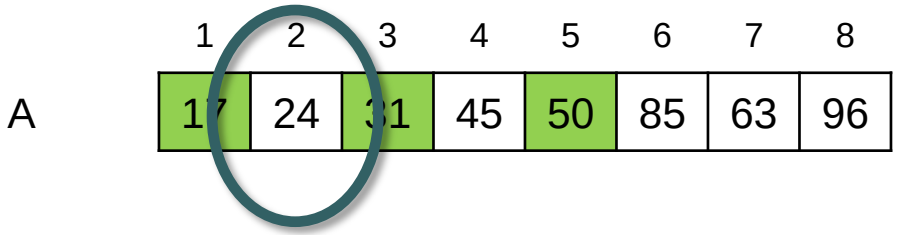
QUICKSORT

```
QUICKSORT(A,p,r)
1: if p < r then
2:    q ← PARTITION(A,p,r);
3:    QUICKSORT(A,p,q-1);
4:    QUICKSORT(A,q+1,r);
6: end if
```

	1	2	3	4	5	6	7	8
A	17	24	31	45	50	85	63	96

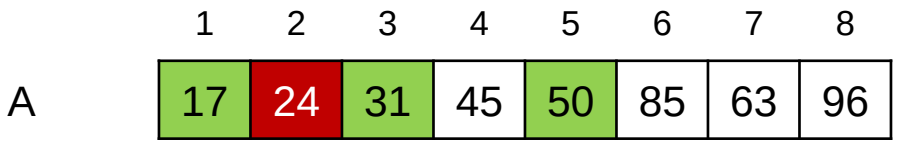
QUICKSORT

```
QUICKSORT(A,p,r)
1: if p < r then
2:    q ← PARTITION(A,p,r);
3:    QUICKSORT(A,p,q-1);
4:    QUICKSORT(A,q+1,r);
6: end if
```



QUICKSORT

```
QUICKSORT(A,p,r)
1: if p < r then
2:    q ← PARTITION(A,p,r);
3:    QUICKSORT(A,p,q-1);
4:    QUICKSORT(A,q+1,r);
6: end if
```



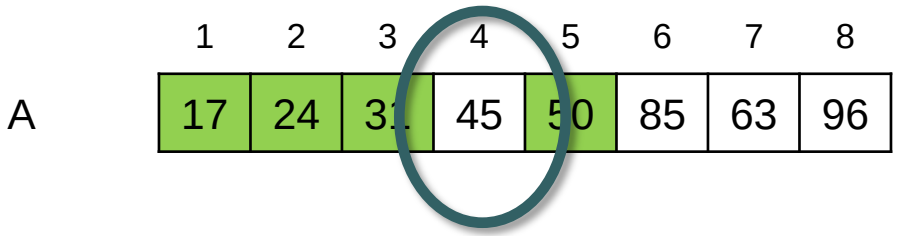
QUICKSORT

```
QUICKSORT(A,p,r)
1: if p < r then
2:    q ← PARTITION(A,p,r);
3:    QUICKSORT(A,p,q-1);
4:    QUICKSORT(A,q+1,r);
6: end if
```

	1	2	3	4	5	6	7	8
A	17	24	31	45	50	85	63	96

QUICKSORT

```
QUICKSORT(A,p,r)
1: if p < r then
2:    q ← PARTITION(A,p,r);
3:    QUICKSORT(A,p,q-1);
4:    QUICKSORT(A,q+1,r);
6: end if
```



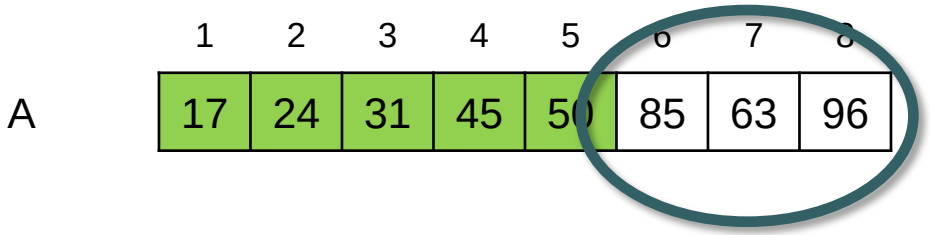
QUICKSORT

```
QUICKSORT(A,p,r)
1: if p < r then
2:    q ← PARTITION(A,p,r);
3:    QUICKSORT(A,p,q-1);
4:    QUICKSORT(A,q+1,r);
6: end if
```

	1	2	3	4	5	6	7	8
A	17	24	31	45	50	85	63	96

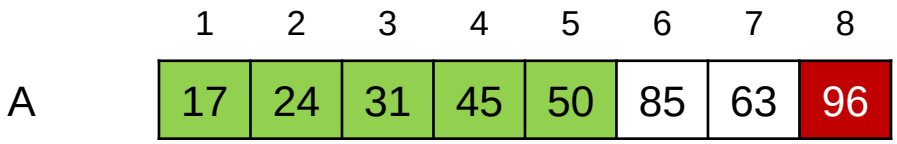
QUICKSORT

```
QUICKSORT(A,p,r)
1: if p < r then
2:    q ← PARTITION(A,p,r);
3:    QUICKSORT(A,p,q-1);
4:    QUICKSORT(A,q+1,r);
6: end if
```



QUICKSORT

```
QUICKSORT(A,p,r)
1: if p < r then
2:    q ← PARTITION(A,p,r);
3:    QUICKSORT(A,p,q-1);
4:    QUICKSORT(A,q+1,r);
6: end if
```



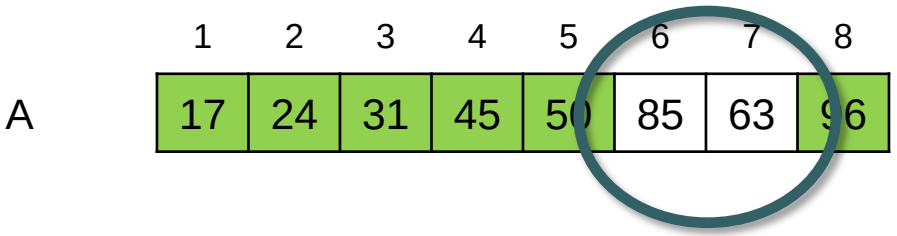
QUICKSORT

```
QUICKSORT(A,p,r)
1: if p < r then
2:    q ← PARTITION(A,p,r);
3:    QUICKSORT(A,p,q-1);
4:    QUICKSORT(A,q+1,r);
6: end if
```

	1	2	3	4	5	6	7	8
A	17	24	31	45	50	85	63	96

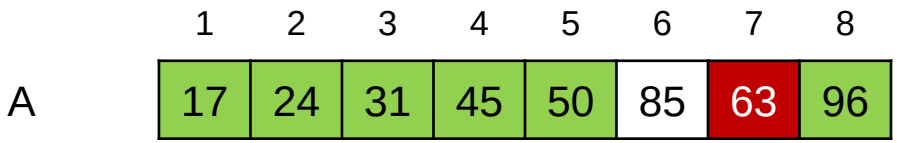
QUICKSORT

```
QUICKSORT(A,p,r)
1: if p < r then
2:    q ← PARTITION(A,p,r);
3:    QUICKSORT(A,p,q-1);
4:    QUICKSORT(A,q+1,r);
6: end if
```



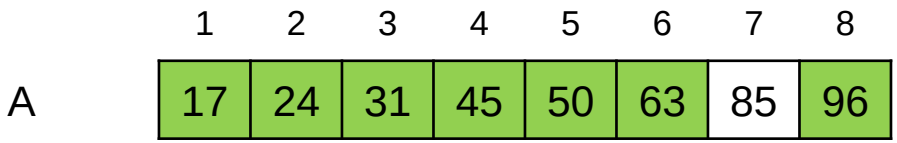
QUICKSORT

```
QUICKSORT(A,p,r)
1: if p < r then
2:    q ← PARTITION(A,p,r);
3:    QUICKSORT(A,p,q-1);
4:    QUICKSORT(A,q+1,r);
6: end if
```



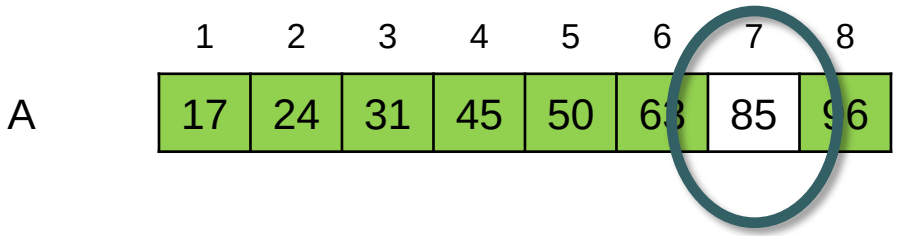
QUICKSORT

```
QUICKSORT(A,p,r)
1: if p < r then
2:    q ← PARTITION(A,p,r);
3:    QUICKSORT(A,p,q-1);
4:    QUICKSORT(A,q+1,r);
6: end if
```



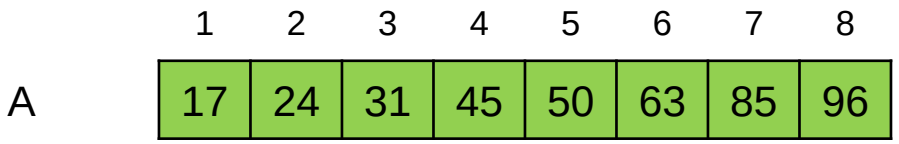
QUICKSORT

```
QUICKSORT(A,p,r)
1: if p < r then
2:    q ← PARTITION(A,p,r);
3:    QUICKSORT(A,p,q-1);
4:    QUICKSORT(A,q+1,r);
6: end if
```



QUICKSORT

```
QUICKSORT(A,p,r)
1: if p < r then
2:    q ← PARTITION(A,p,r);
3:    QUICKSORT(A,p,q-1);
4:    QUICKSORT(A,q+1,r);
6: end if
```

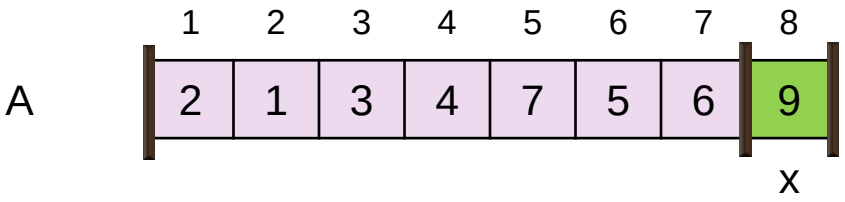


Análise do QUICKSORT

- O desempenho do Quicksort depende da qualidade do particionamento
  - Pior caso: totalmente desbalanceado
  - Melhor caso: totalmente balanceado
- Particionamento balanceado
  - Custo de mesma ordem que o Mergesort
- Particionamento desbalanceado
  - Custo de mesma ordem que o Insertionsort

Pior caso

- O pior balanceamento é quando o particionamento produz
  - Subproblema de *n*-1 elementos
  - Outro com 0 elementos
  - Ocorre quando o pivô é o maior (ou o menor) elemento do arranjo



Pior caso

- Quando isso ocorre sempre (recursivamente), temos

```
QUICKSORT(A,p,r)
1: if p < r then
2:    q ← PARTITION(A,p,r);
3:    QUICKSORT(A,p,q-1);
4:    QUICKSORT(A,q+1,r);
6: end if
```

$$T(n) = T(n-1) + T(0) + \Theta(n)$$
$$= T(n-1) + \Theta(1) + \Theta(n)$$
$$= T(n-1) + \Theta(n)$$

$$T(n) = \Theta\left(n^2\right)$$

Melhor caso

- Na divisão mais balanceada possível, o PARTITION produz dois subproblemas de tamanho igual

```
QUICKSORT(A,p,r)
1: if p < r then
2:    q ← PARTITION(A,p,r);
3:    QUICKSORT(A,p,q-1);
4:    QUICKSORT(A,q+1,r);
6: end if
```

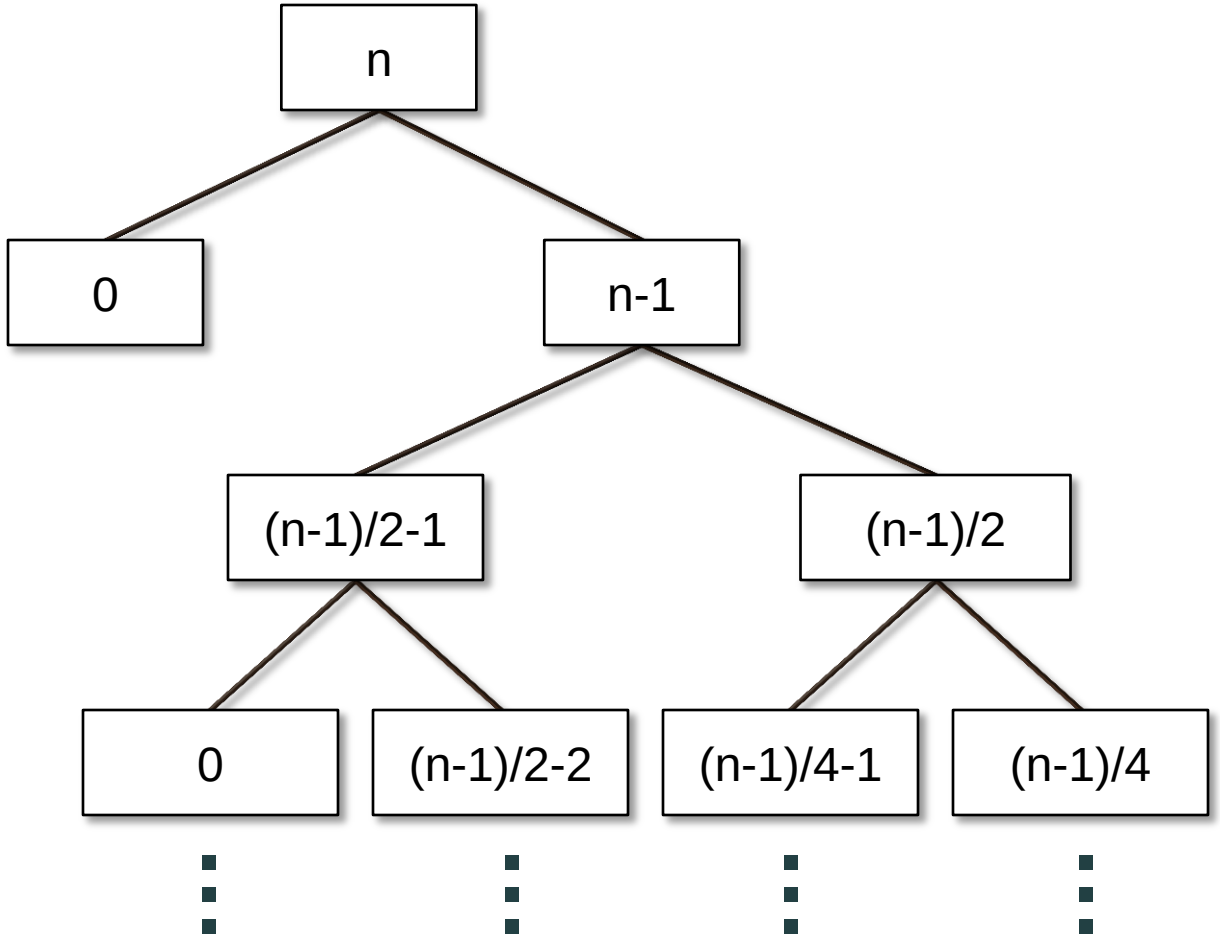
$$T(n) \leq 2T(n/2) + \Theta(n)$$

$$T(n) = O(n \log n)$$

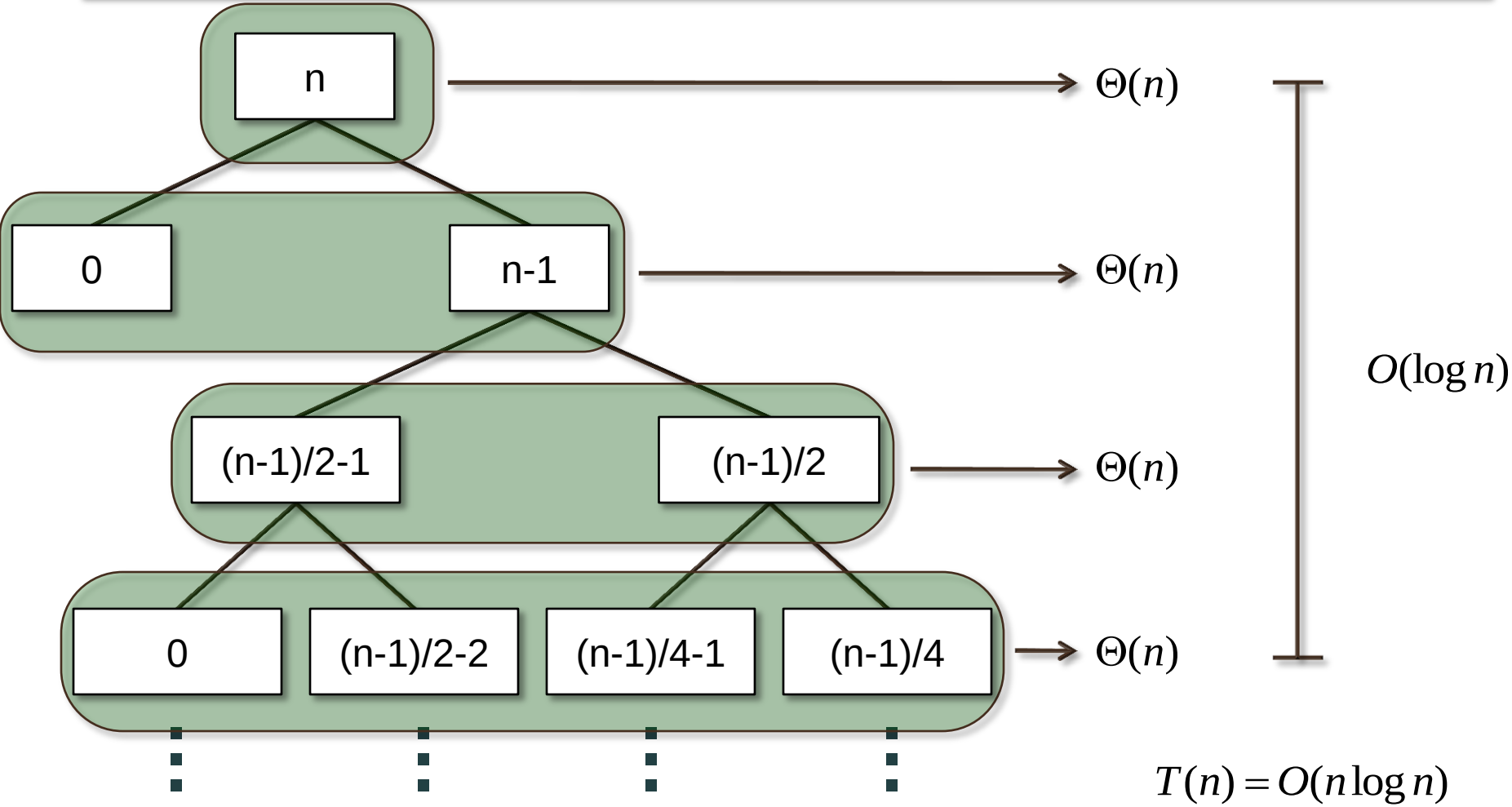
Intuição para o caso médio

- No caso médio, o PARTITION produz uma mistura de divisões BOAS e RUINS
  - As divisões boas e ruins são divididas aleatoriamente
- Para efeitos de intuição
  - Suponha que as divisões boas sejam as do melhor caso e as divisões ruins sejam as do pior caso

Intuição para o caso médio



Intuição para o caso médio



$$T(n) = O(n \log n)$$

Ordenação por comparação

- Os algoritmos vistos compartilham uma propriedade
  - A seqüência ordenada é obtida apenas usando comparações entre os elementos de entrada
  - Esta classe de algoritmos chama-se **ordenação por comparação**

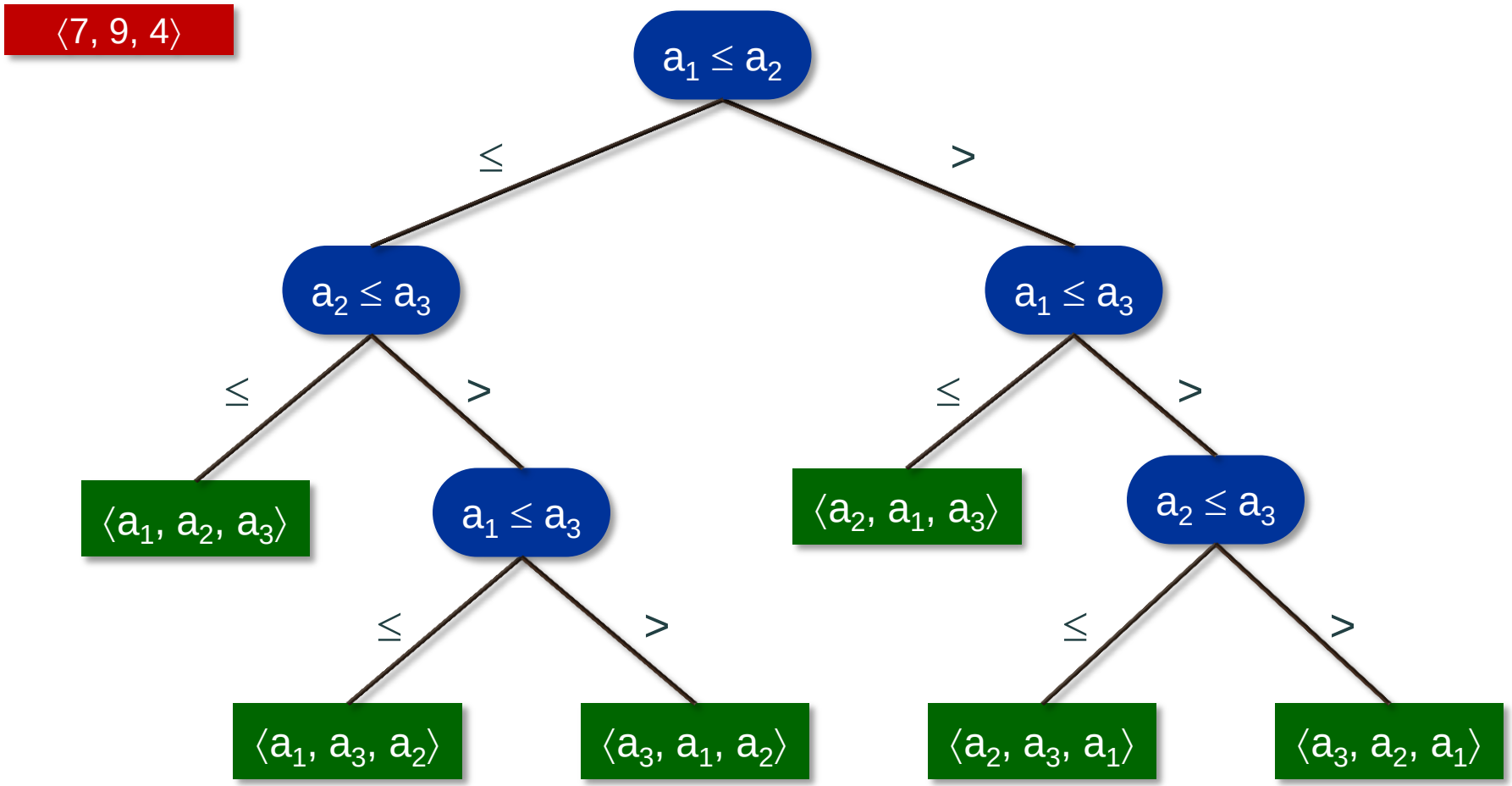
- Pertencem a esta classe
  - Bubblesort
  - Insertionsort
  - Mergesort
  - Heapsort
  - Quicksort
  - ...

Limite inferior

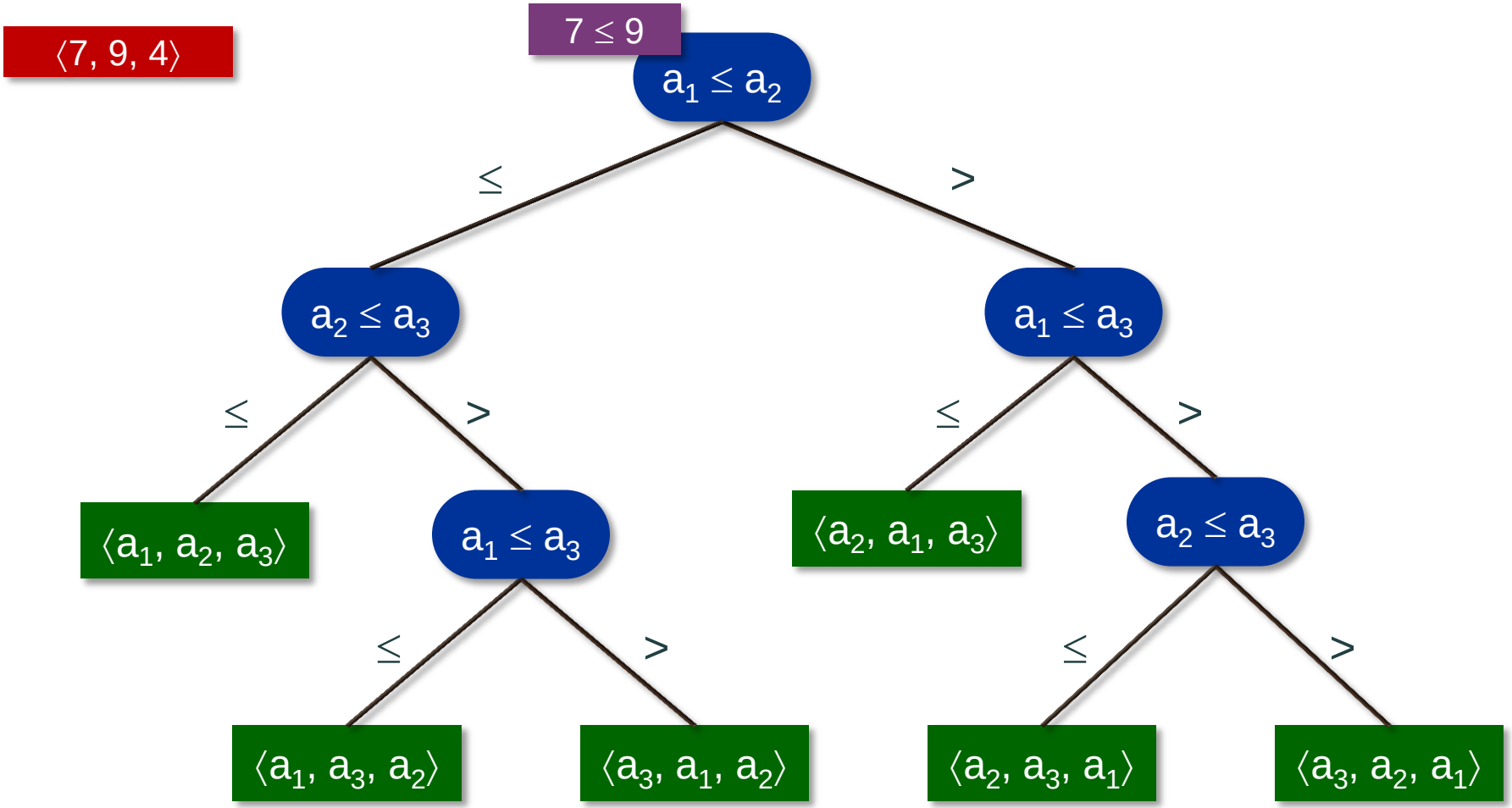
- Nenhum algoritmo de ordenação por comparação pode ser melhor do que $\Omega(n \log n)$
  - Para provar isso é necessário montar uma **árvore de decisão**

- A cada dois elementos a_i, a_j ,
  - Precisamos definir quem é o menor
  - Fazemos a comparação $a_i \leq a_j$

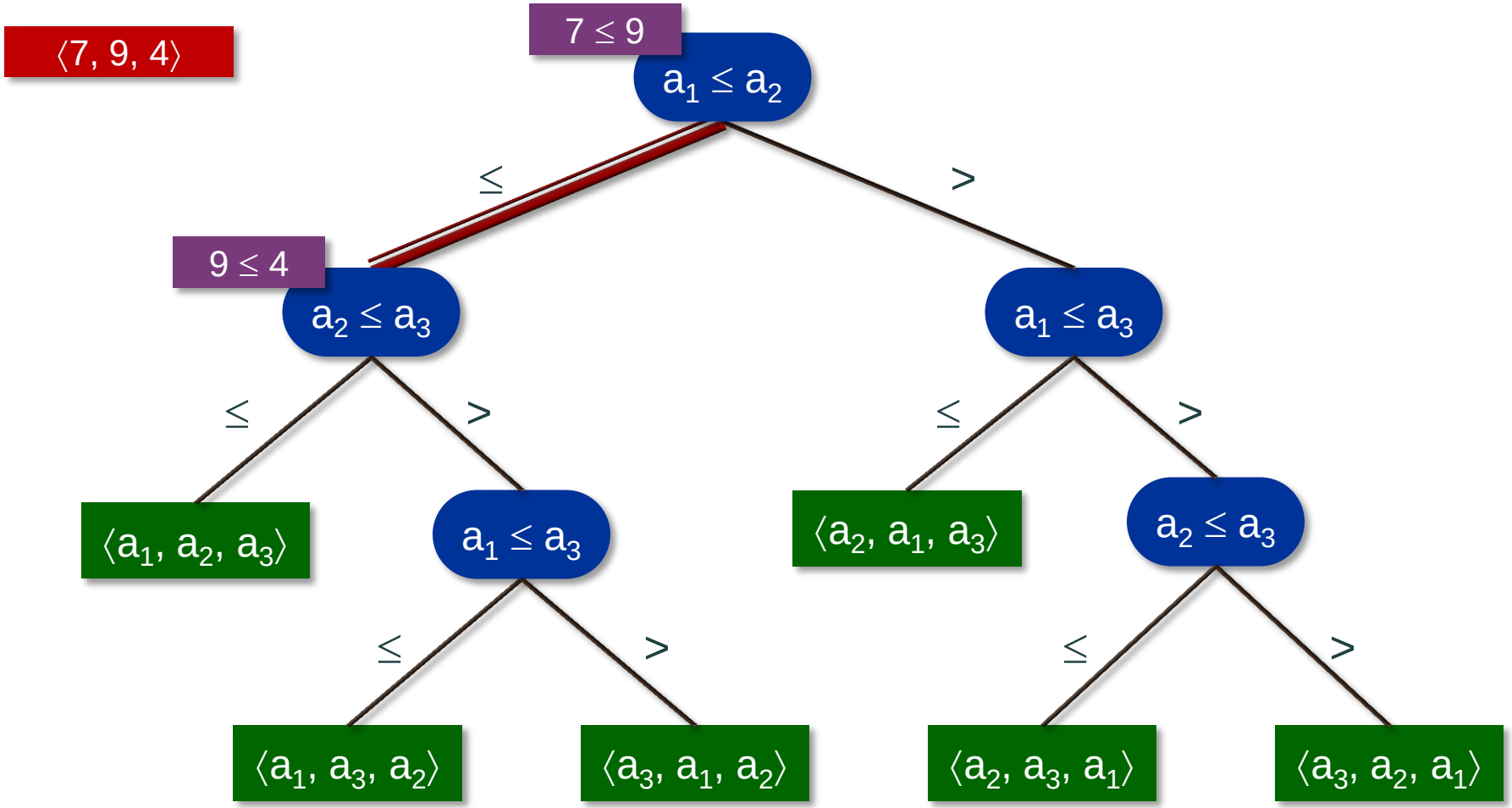
Árvore de decisão para o Insertionsort



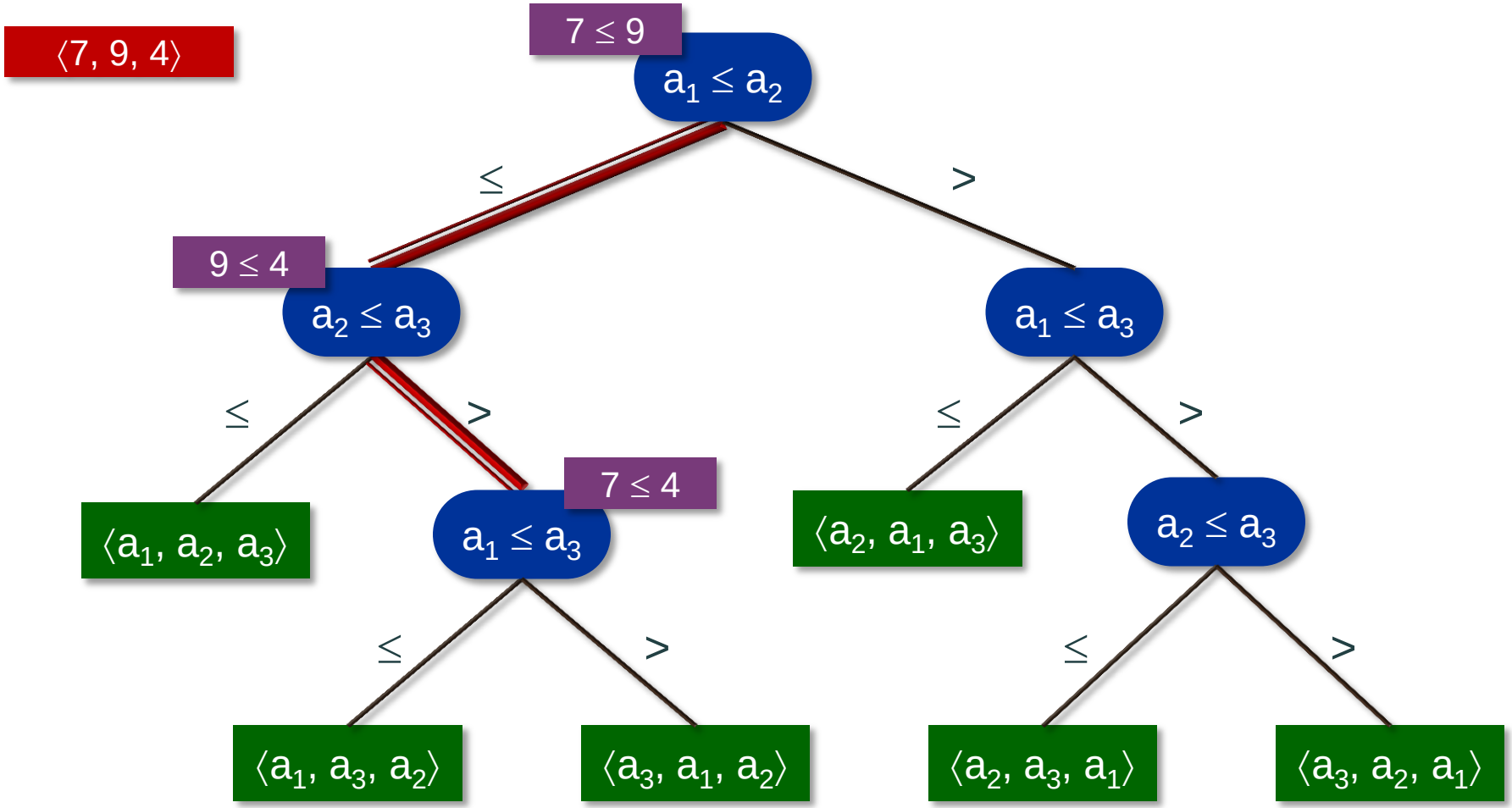
Árvore de decisão para o Insertionsort



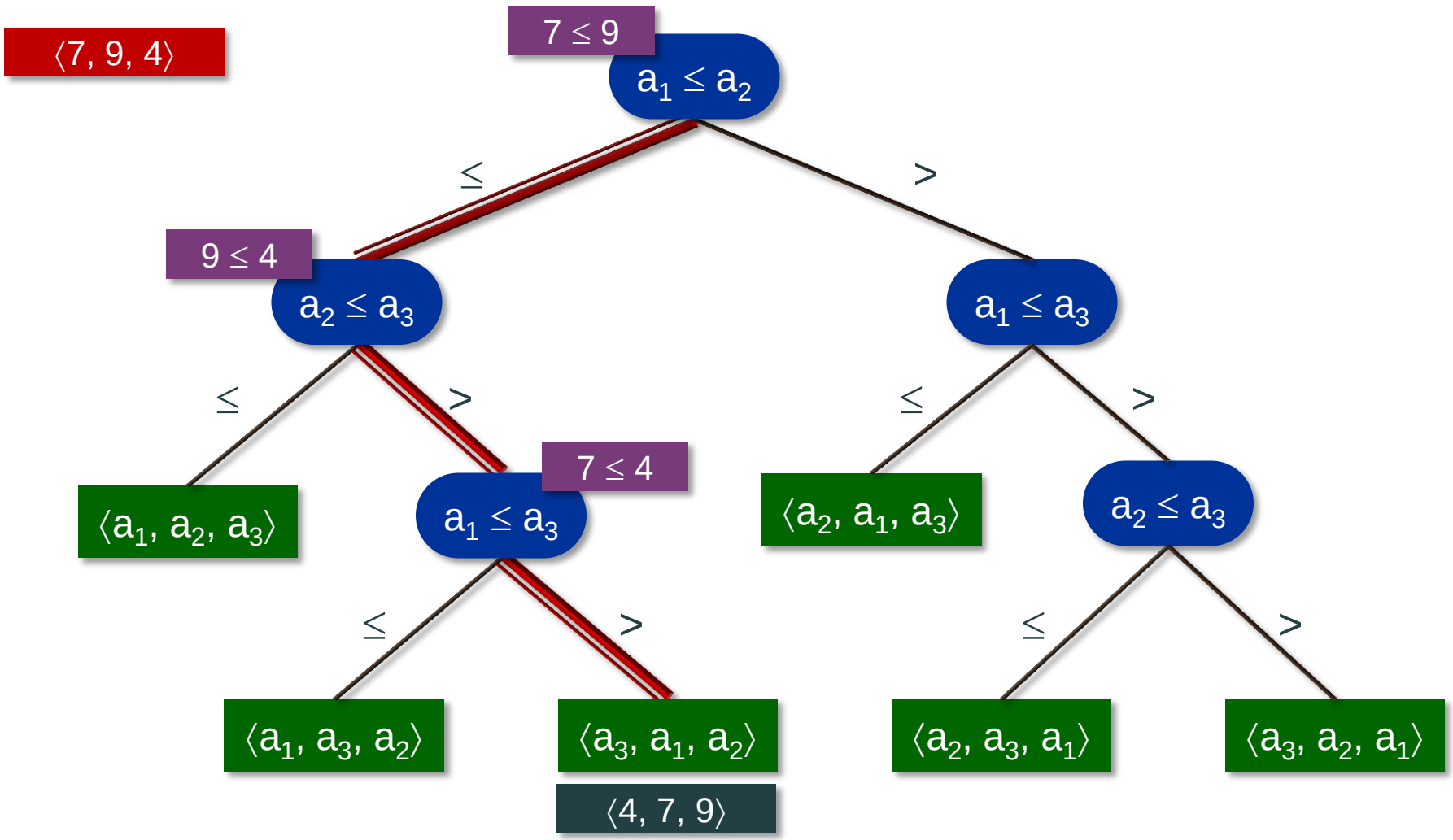
Árvore de decisão para o Insertionsort



Árvore de decisão para o Insertionsort



Árvore de decisão para o Insertionsort



Limite inferior

- O número de folhas (f) da árvore corresponde a todas as possíveis permutações do arranjo
  - $f = n!$
- O número de folhas (f) de uma árvore binária de altura h é no máximo 2^h
  - $f \leq 2^h$
- Logo
  - $n! \leq 2^h \Rightarrow h \geq \log(n!) \Rightarrow h \geq \Theta(n \log n) \Rightarrow h = \Omega(n \log n)$
- No pior caso o custo será altura da árvore!

Exercício

- Qual valor de q o PARTITION retorna quando todos os elementos do arranjo $A[p..r]$ são iguais?
- Modifique o PARTITION para que $q = (p+r) / 2$ quando todos os elementos do arranjo $A[p..r]$ são iguais