ORDENAÇÃO

Aula 21 - 09 de junho de 2009

Quicksort

Quicksort

O algoritmo de ordenação *quicksort* foi desenvolvido por C. A. R. Hoare:

Quicksort

O algoritmo de ordenação *quicksort* foi desenvolvido por C. A. R. Hoare:

- C. A. R. Hoare. Quicksort. *Computer Journal*, 5(1) : 10-15, 1962.

Quicksort

O algoritmo de ordenação *quicksort* foi desenvolvido por C. A. R. Hoare:

- C. A. R. Hoare. Quicksort. *Computer Journal*, 5(1) : 10-15, 1962.

O quicksort não é um algoritmo ótimo para o problema da ordenação (ainda não vimos nenhum!), mas ele se comporta como tal para quase todas as entradas de um mesmo tamanho, n.

Quicksort

O algoritmo de ordenação *quicksort* foi desenvolvido por C. A. R. Hoare:

- C. A. R. Hoare. Quicksort. *Computer Journal*, 5(1) : 10-15, 1962.

O quicksort não é um algoritmo ótimo para o problema da ordenação (ainda não vimos nenhum!), mas ele se comporta como tal para quase todas as entradas de um mesmo tamanho, n.

Na prática, o quicksort é o algoritmo preferido. Nós veremos o porquê em uma outra aula. Primeiro, vamos tratar de entendê-lo.

Quicksort

Quicksort

A "alma" do quicksort é um procedimento auxiliar denominado *partição*. Este procedimento recebe uma seqüência, $A[l..r]$, de elementos de um vetor A com n elementos, rearranja os elementos no segmento $A[l..r]$ de A e retorna um inteiro q tal que

$$q \in \{l, \ldots, r\} \subseteq \{1, \ldots, n\}$$

e

$$A[i] \leq A[q] \quad \text{e} \quad A[q] \leq A[j]\,,$$

para todo

$$i \in \{1, \ldots, q-1\} \quad \text{e} \quad j \in \{q+1, \ldots, r\}\,.$$

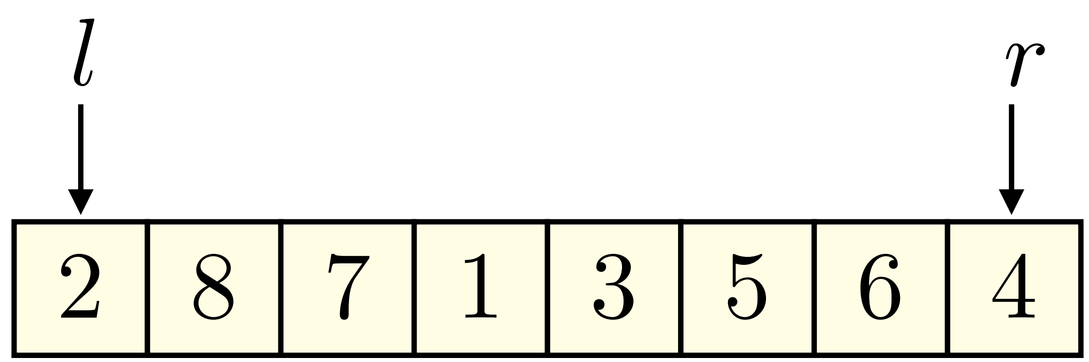
Quicksort

Quicksort

Por exemplo, suponha que a entrada de *partição* seja o segmento de um vetor A contendo os elementos mostrados abaixo:

Quicksort

Por exemplo, suponha que a entrada de *partição* seja o segmento de um vetor A contendo os elementos mostrados abaixo:



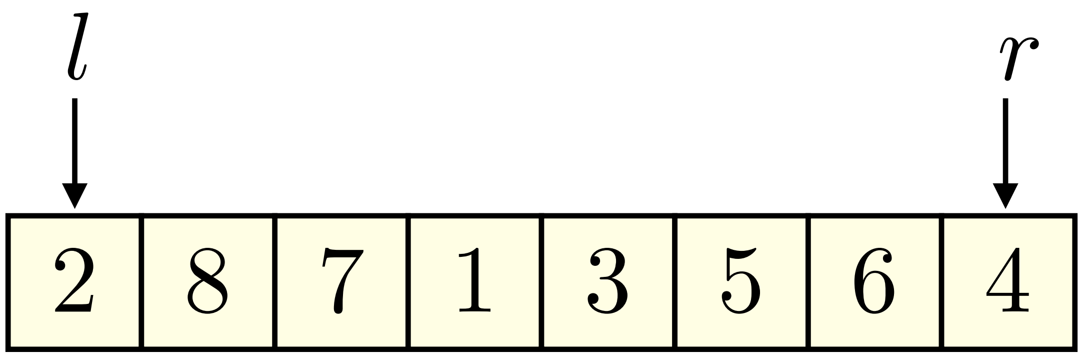
Quicksort

Por exemplo, suponha que a entrada de *partição* seja o segmento de um vetor A contendo os elementos mostrados abaixo:

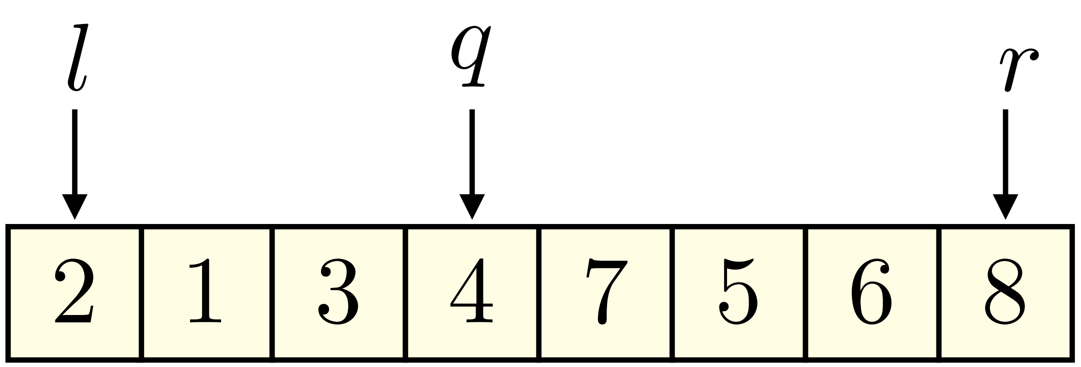
l							r
2	8	7	1	3	5	6	4

Uma possível saída satisfazendo as condições do slide anterior é

Por exemplo, suponha que a entrada de *partição* seja o segmento de um vetor A contendo os elementos mostrados abaixo:



Uma possível saída satisfazendo as condições do slide anterior é

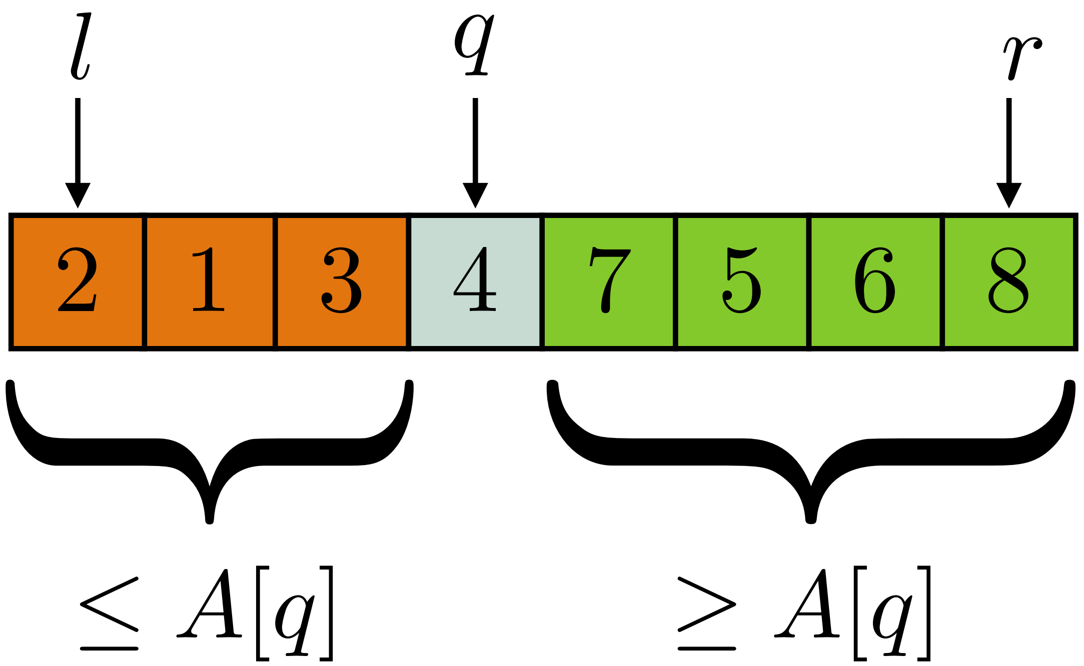


Quicksort

Quicksort

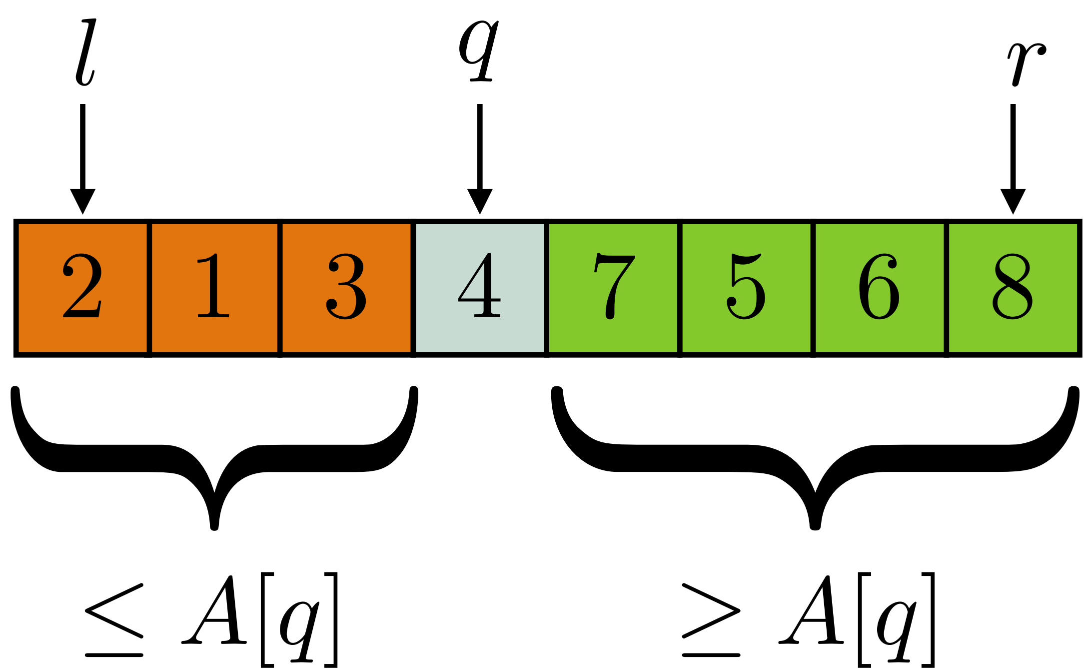
Note que

Note que



Quicksort

Note que



Note também que há muitas outras saídas que satisfazem as condições impostas. Mas, tudo que precisamos é gerar *uma* saída.

Quicksort

Quicksort

algoritmo *partição*(**ref** A, l, r)
    $x \leftarrow A[r]$
    $i \leftarrow l - 1$
    para $j \leftarrow l$ **até** $r - 1$ **faça**
        se *compara*$(A[j], x) \neq 1$ **então**
            $i \leftarrow i + 1$
            $A[i] \leftrightarrow A[j]$
        fimse
    fimpara
    $A[i+1] \leftrightarrow A[r]$
    retorne $i + 1$
fimalgoritmo

Quicksort

Quicksort

O algoritmo partição inicia selecionando um elemento $x = A[r]$, conhecido como **pivô**, em torno do qual a partição ocorrerá.

Quicksort

O algoritmo partição inicia selecionando um elemento $x = A[r]$, conhecido como **pivô**, em torno do qual a partição ocorrerá.

algoritmo *partição*(**ref** A, l, r)
    $x \leftarrow A[r]$
    $i \leftarrow l - 1$
    para $j \leftarrow l$ **até** $r - 1$ **faça**
        se *compara*$(A[j], x) \neq 1$ **então**
            $i \leftarrow i + 1$
            $A[i] \leftrightarrow A[j]$
        fimse
    fimpara
    $A[i + 1] \leftrightarrow A[r]$
    retorne $i + 1$
fimalgoritmo

Quicksort

O algoritmo partição inicia selecionando um elemento $x = A[r]$, conhecido como **pivô**, em torno do qual a partição ocorrerá.

algoritmo *partição*(**ref** A, l, r)
    $x \leftarrow A[r]$
    $i \leftarrow l - 1$
    para $j \leftarrow l$ **até** $r - 1$ **faça**
        se *compara*$(A[j], x) \neq 1$ **então**
            $i \leftarrow i + 1$
            $A[i] \leftrightarrow A[j]$
        fimse
    fimpara
    $A[i + 1] \leftrightarrow A[r]$
    retorne $i + 1$
fimalgoritmo

O pivô x estará na posição q do vetor quando partição terminar.

Quicksort

Quicksort

Quando o restante do procedimento é executado, o segmento $A[l..r]$ é particionado em **quatro** regiões (possivelmente vazias).

Quando o restante do procedimento é executado, o segmento $A[l..r]$ é particionado em **quatro** regiões (possivelmente vazias).

algoritmo *partição*(**ref** A, l, r)
 $x \leftarrow A[r]$
 $i \leftarrow l - 1$
 para $j \leftarrow l$ **até** $r - 1$ **faça**
  se *compara*$(A[j], x) \neq 1$ **então**
   $i \leftarrow i + 1$
   $A[i] \leftrightarrow A[j]$
  fimse
 fimpara
 $A[i+1] \leftrightarrow A[r]$
 retorne $i + 1$
fimalgoritmo

Quicksort

Quicksort

No início de cada iteração do laço **para**, cada região satisfaz certas propriedades. A saber, para cada índice k do vetor, temos que

Quicksort

No início de cada iteração do laço **para**, cada região satisfaz certas propriedades. A saber, para cada índice k do vetor, temos que

1) Se $l \leq k \leq i$ então $A[k] \leq x$.

Quicksort

No início de cada iteração do laço **para**, cada região satisfaz certas propriedades. A saber, para cada índice k do vetor, temos que

1) Se $l \leq k \leq i$ então $A[k] \leq x$.

2) Se $i + 1 \leq k \leq j - 1$ então $A[k] > x$.

No início de cada iteração do laço **para**, cada região satisfaz certas propriedades. A saber, para cada índice k do vetor, temos que

1) Se $l \leq k \leq i$ então $A[k] \leq x$.

2) Se $i+1 \leq k \leq j-1$ então $A[k] > x$.

3) Se $k = r$ então $A[k] = x$.

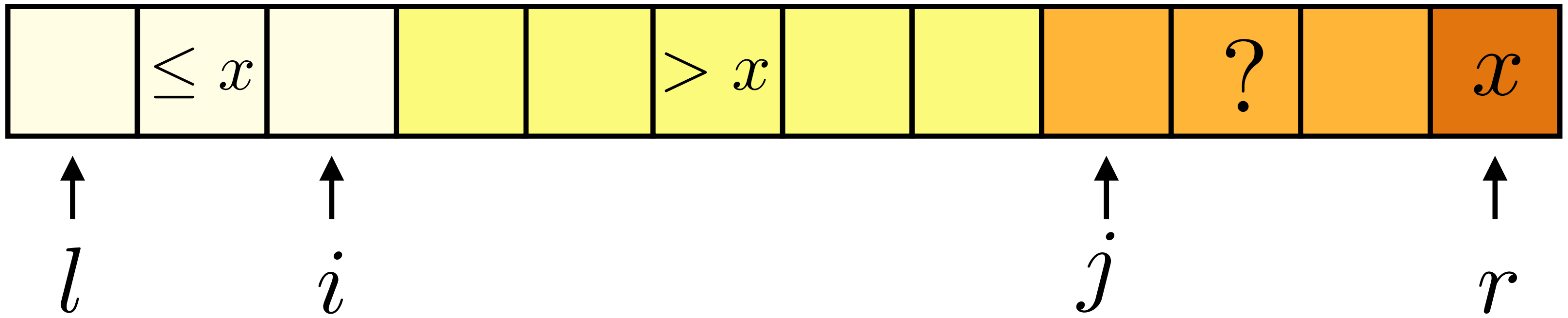
Quicksort

No início de cada iteração do laço **para**, cada região satisfaz certas propriedades. A saber, para cada índice k do vetor, temos que

1) Se $l \leq k \leq i$ então $A[k] \leq x$.

2) Se $i + 1 \leq k \leq j - 1$ então $A[k] > x$.

3) Se $k = r$ então $A[k] = x$.



Quicksort

Quicksort

O que fazem as linhas (04)-(09) do algoritmo?

Quicksort

O que fazem as linhas (04)-(09) do algoritmo?

(04) **para** $j \leftarrow l$ **até** $r - 1$ **faça**
(05)   **se** *compara*$(A[j], x) \neq 1$ **então**
(06) $i \leftarrow i + 1$
(07) $A[i] \leftrightarrow A[j]$
(08)   **fimse**
(09) **fimpara**

Quicksort

O que fazem as linhas (04)-(09) do algoritmo?

(04) **para** $j \leftarrow l$ **até** $r - 1$ **faça**
(05)   **se** *compara*$(A[j], x) \neq 1$ **então**
(06) $i \leftarrow i + 1$
(07) $A[i] \leftrightarrow A[j]$
(08)   **fimse**
(09) **fimpara**

O segmento $A[l..r]$ é visitado. A visita consiste em comparar cada elemento $A[j]$ do segmento com o pivô, x. Quando $A[j] \leq x$, o algoritmo incrementa i e troca os elementos $A[i]$ e $A[j]$.

Quicksort

Quicksort

Se $A[j] > x$, o marcador i não é incrementado. Logo, no início de cada iteração do laço, $A[l..i]$ contém apenas os elementos de $A[l..j-1]$ que são $\leq x$, enquanto $A[i+1..j-1]$ contém os maiores.

Quicksort

Se $A[j] > x$, o marcador i não é incrementado. Logo, no início de cada iteração do laço, $A[l..i]$ contém apenas os elementos de $A[l..j-1]$ que são $\leq x$, enquanto $A[i+1..j-1]$ contém os maiores.

```
para j ← l até r − 1 faça
    se compara(A[j], x) ≠ 1 então
        i ← i + 1
        A[i] ↔ A[j]
    fimse
fimpara
```

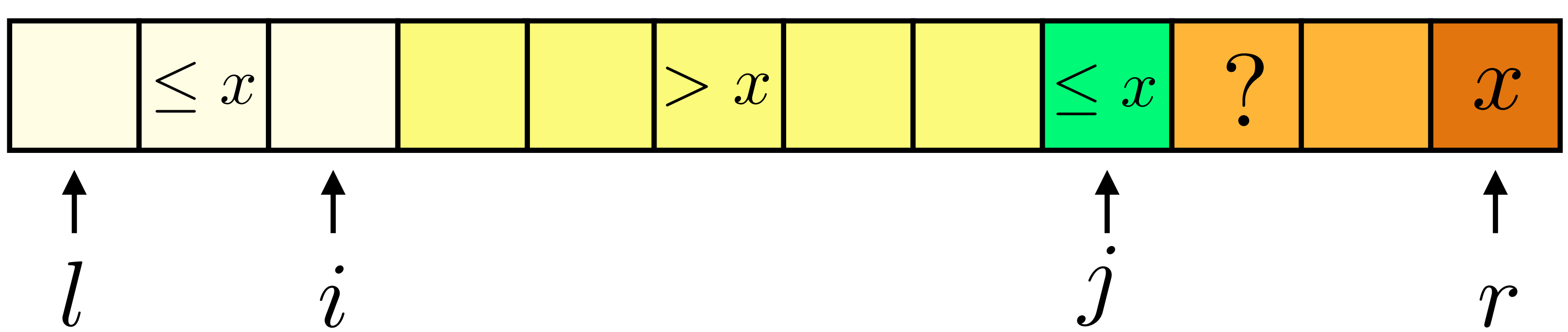
Quicksort

Quicksort

```
    se compara(A[j], x) ≠ 1 então
        i ← i + 1
        A[i] ↔ A[j]
    fimse
```

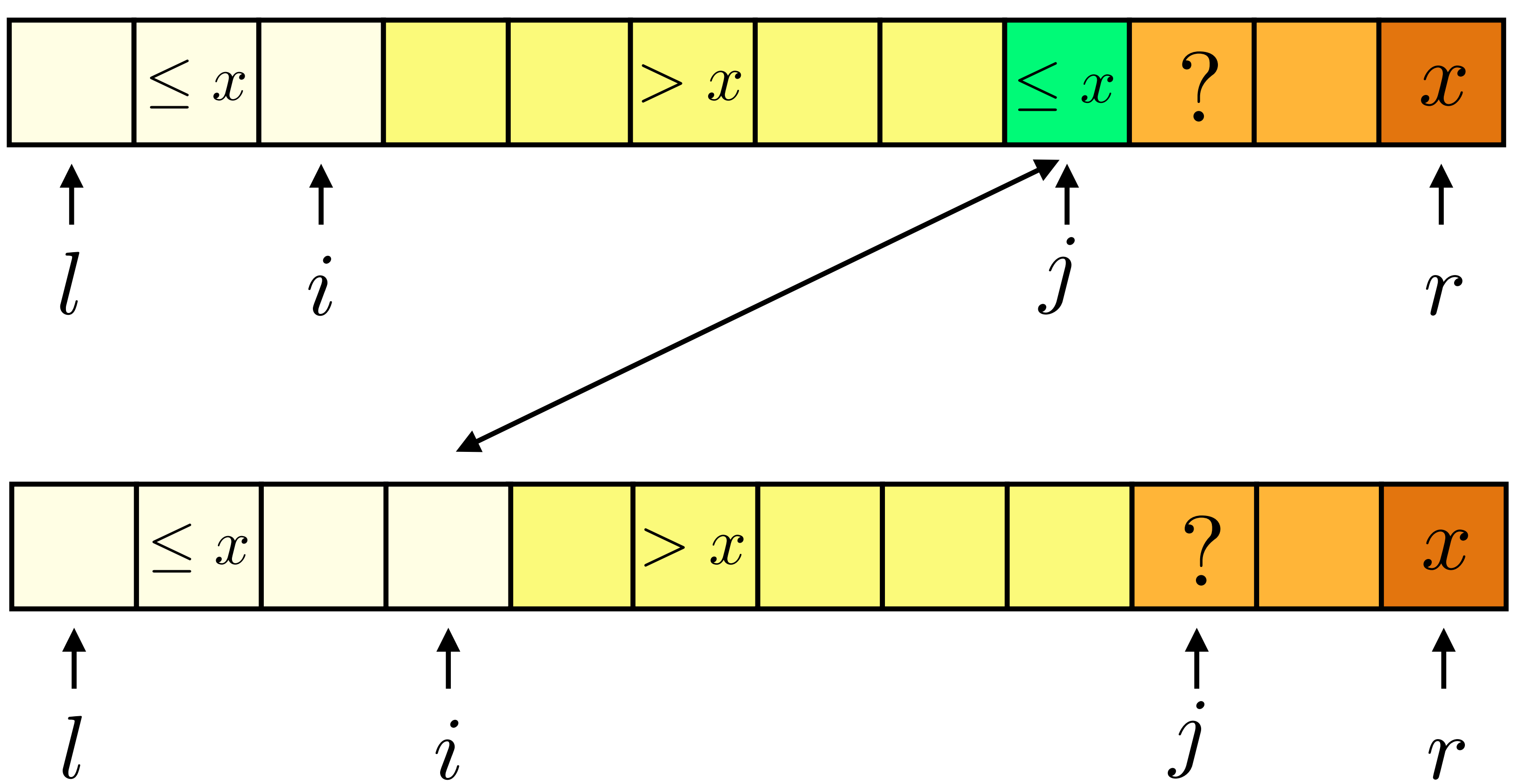
Quicksort

se *compara*$(A[j], x) \neq 1$ **então**
    $i \leftarrow i + 1$
    $A[i] \leftrightarrow A[j]$
fimse



Quicksort

```
se compara(A[j], x) ≠ 1 então
    i ← i + 1
    A[i] ↔ A[j]
fimse
```



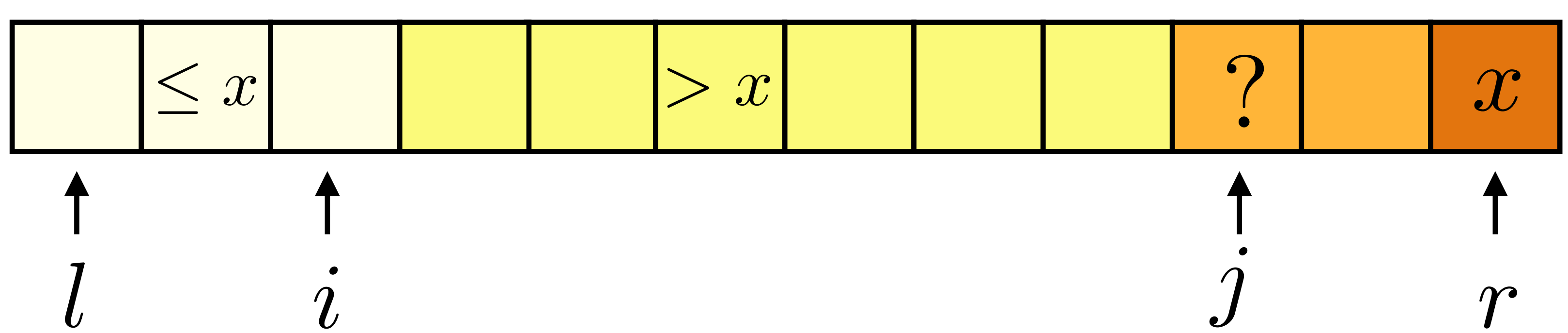
Quicksort

Quicksort

se *compara*$(A[j], x) \neq 1$ **então**
    $i \leftarrow i + 1$
    $A[i] \leftrightarrow A[j]$
fimse

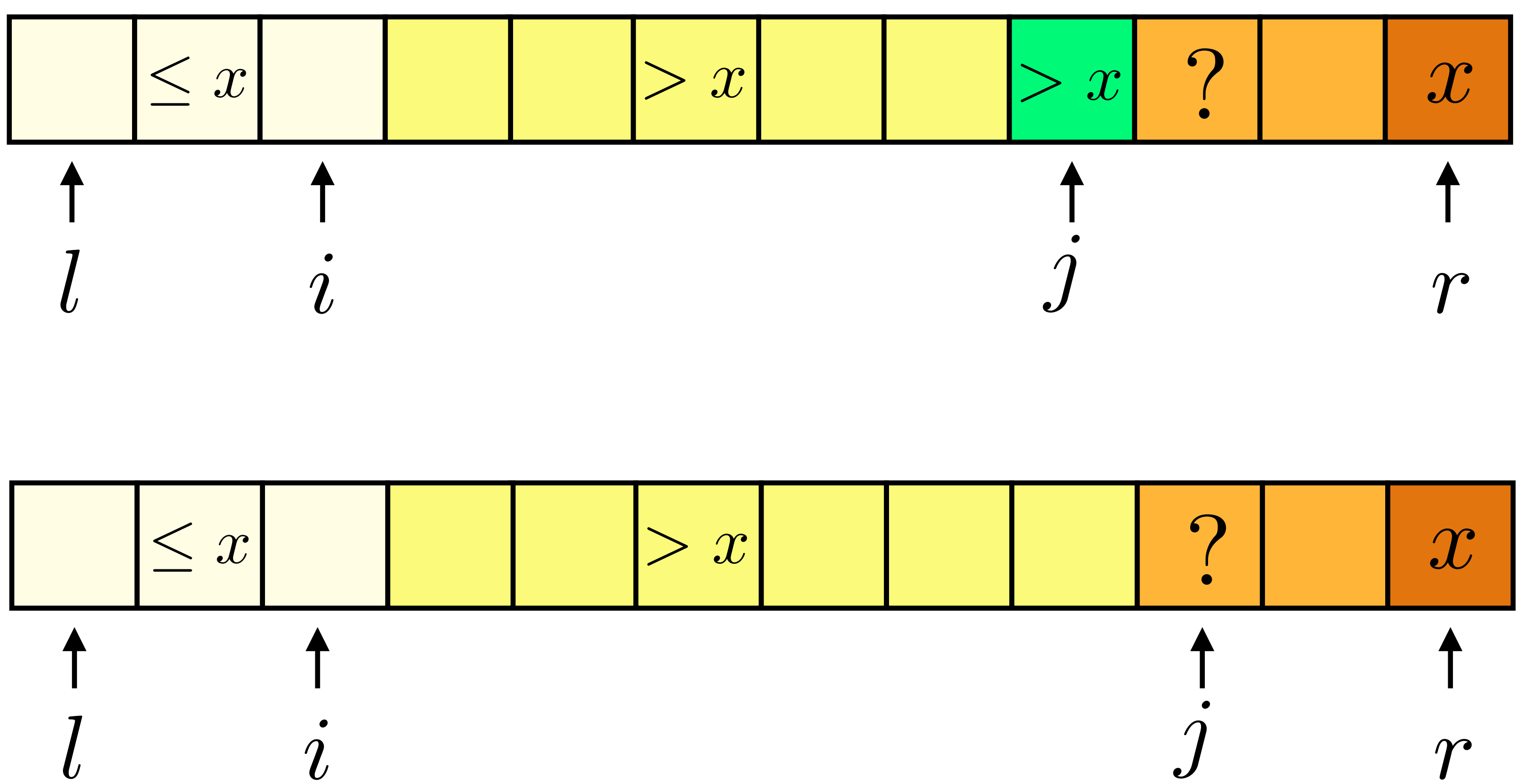
Quicksort

```
se compara(A[j], x) ≠ 1 então
    i ← i + 1
    A[i] ↔ A[j]
fimse
```



Quicksort

```
se compara(A[j], x) ≠ 1 então
    i ← i + 1
    A[i] ↔ A[j]
fimse
```



Quicksort

Quicksort

Quando o laço **para** termina, o pivô, x, troca de posição com o elemento $A[i+1]$.

Quicksort

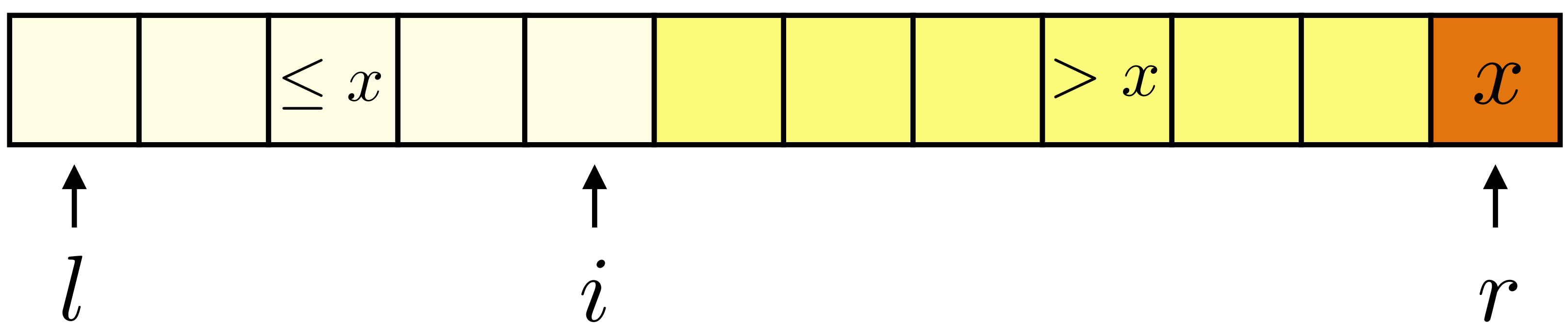
Quando o laço **para** termina, o pivô, x, troca de posição com o elemento $A[i+1]$.

$$(10) \qquad A[i+1] \leftrightarrow A[r]$$

Quicksort

Quando o laço **para** termina, o pivô, x, troca de posição com o elemento $A[i+1]$.

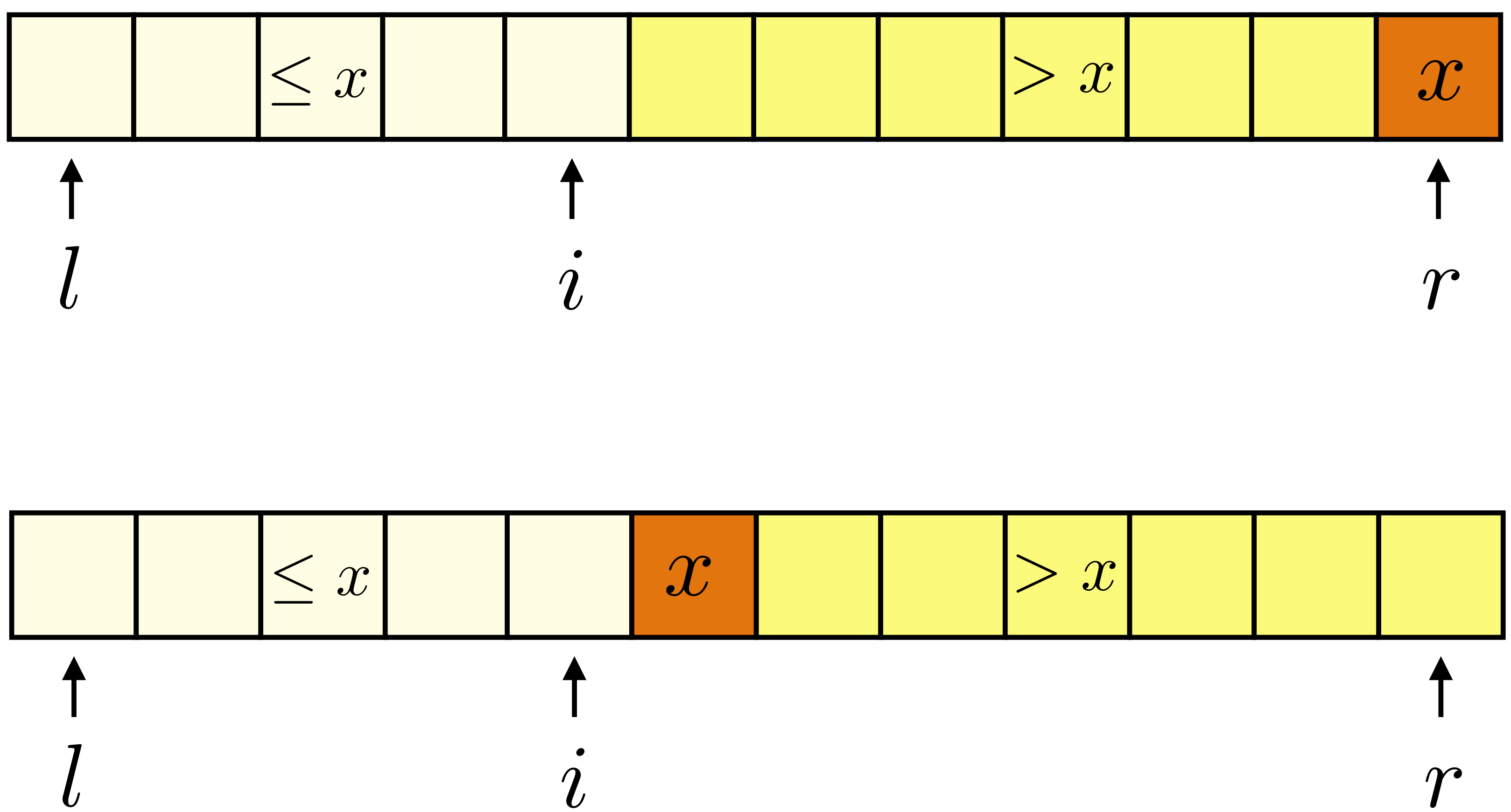
$$(10) \qquad A[i+1] \leftrightarrow A[r]$$



Quicksort

Quando o laço **para** termina, o pivô, x, troca de posição com o elemento $A[i+1]$.

$$(10) \qquad A[i+1] \leftrightarrow A[r]$$



Quicksort

Quicksort

A última linha de *partição* retorna a posição do pivô:

Quicksort

A última linha de *partição* retorna a posição do pivô:

$$(11) \qquad \textbf{retorne } i+1$$

Quicksort

A última linha de *partição* retorna a posição do pivô:

$$(11) \qquad \textbf{retorne } i+1$$

Logo, se fizermos

$$q \leftarrow \textit{partição}(A, l, r)$$

teremos

$$A[q] \geq A[k] \quad \text{para todo } k \in \{l, \ldots, q-1\}$$

e

$$A[q] < A[k] \quad \text{para todo } k \in \{q+1, \ldots, r\}$$

após o retorno de *partição*.

Quicksort

2	8	7	1	3	5	6	4

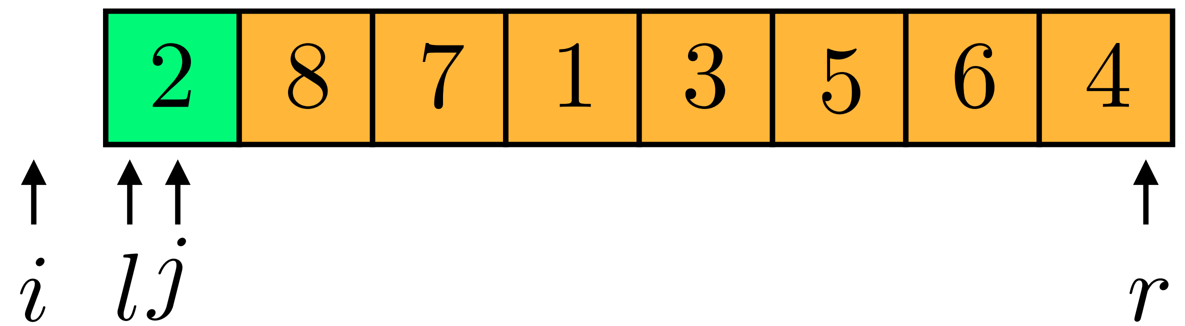
Quicksort

Exemplo:

2	8	7	1	3	5	6	4

Quicksort

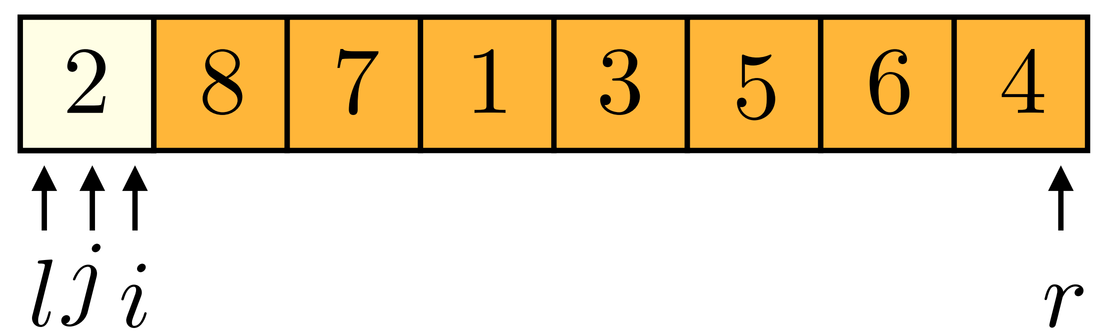
Exemplo:



$$A[j] \leq A[r]?$$

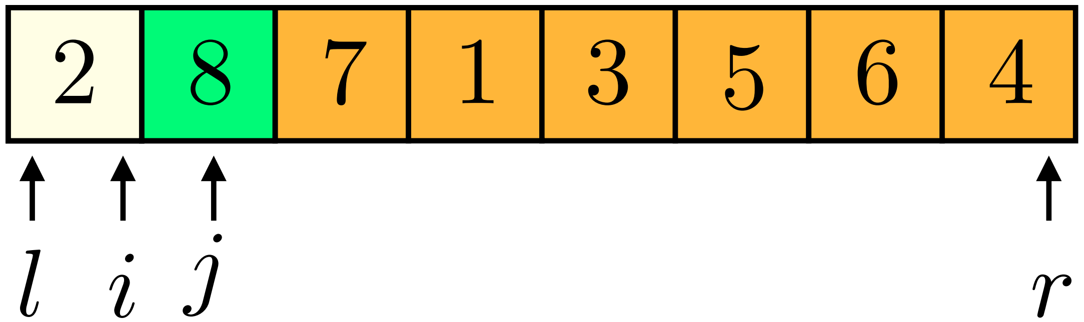
Quicksort

Exemplo:



$$A[j] \leq A[r]?$$

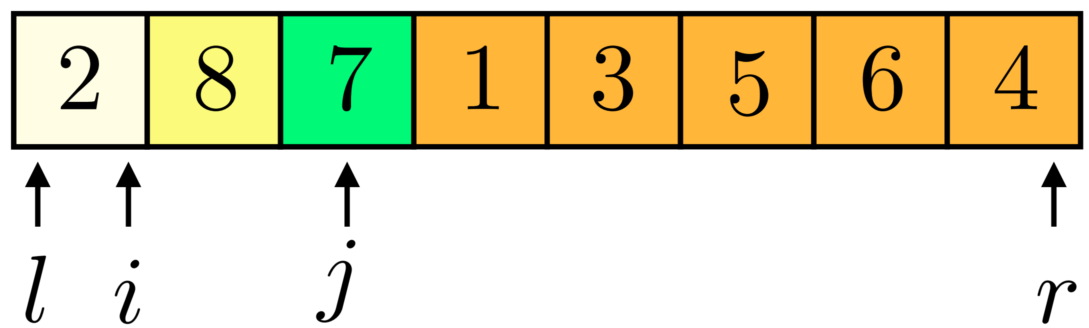
Exemplo:



$$A[j] \leq A[r]?$$

Quicksort

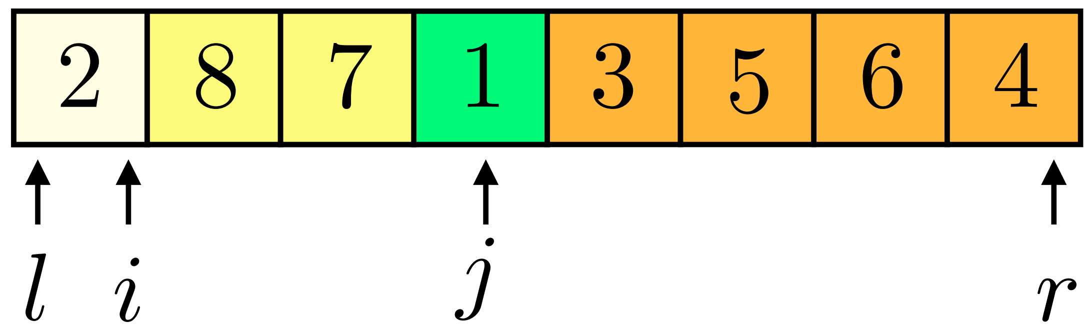
Exemplo:



$$A[j] \leq A[r]?$$

Quicksort

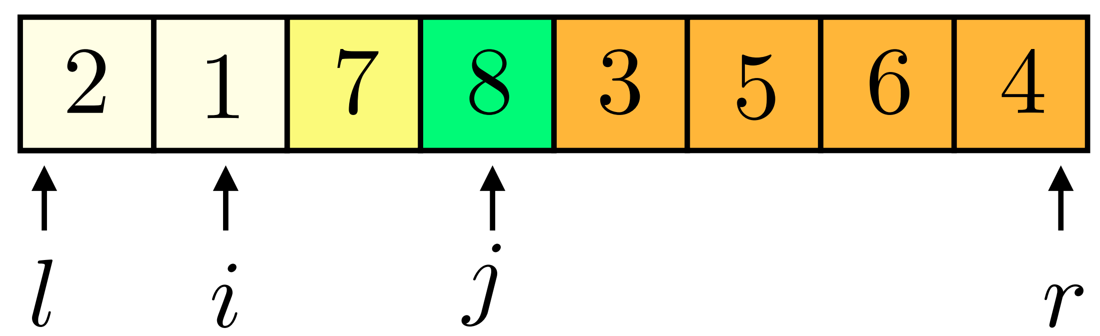
Exemplo:



$$A[j] \leq A[r]?$$

Quicksort

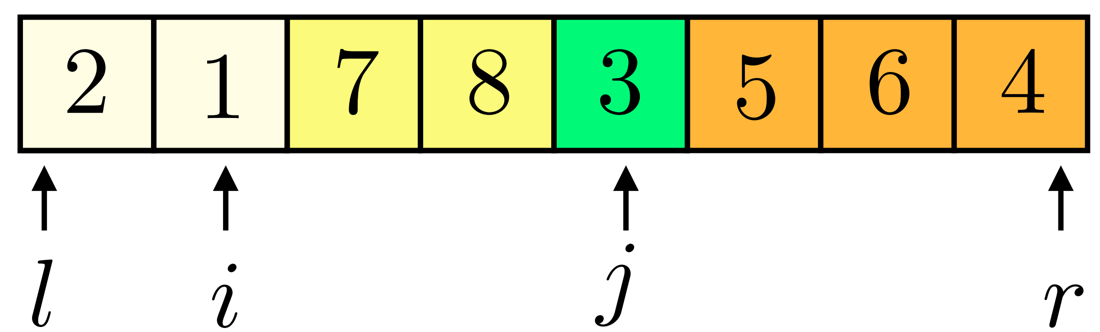
Exemplo:



$$A[j] \leq A[r]?$$

Quicksort

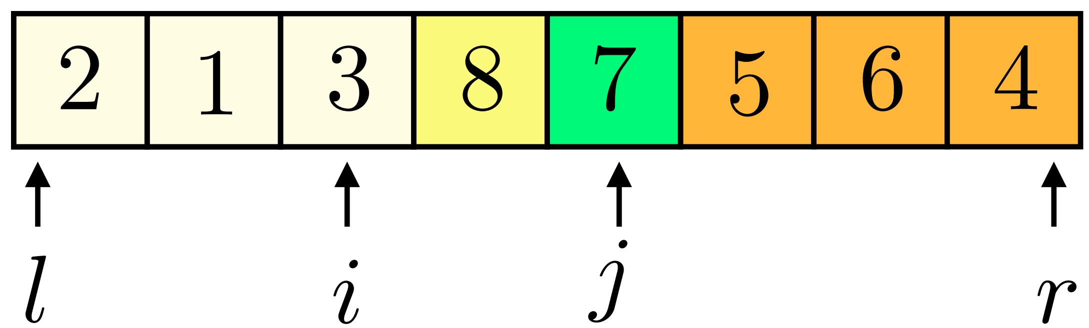
Exemplo:



$$A[j] \leq A[r]?$$

Quicksort

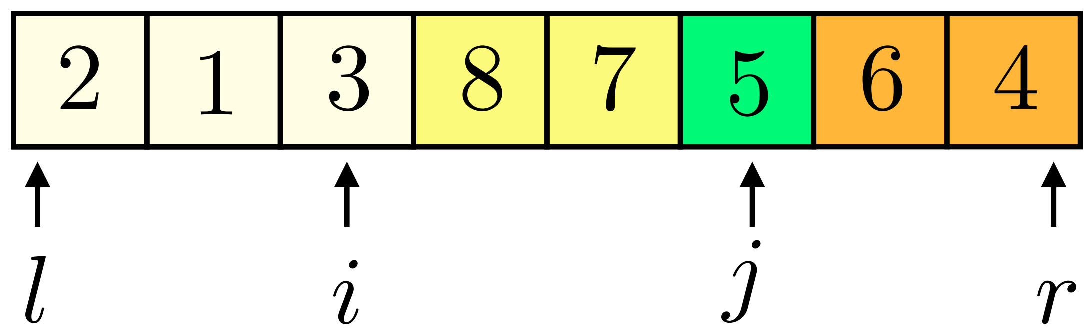
Exemplo:



$$A[j] \leq A[r]?$$

Quicksort

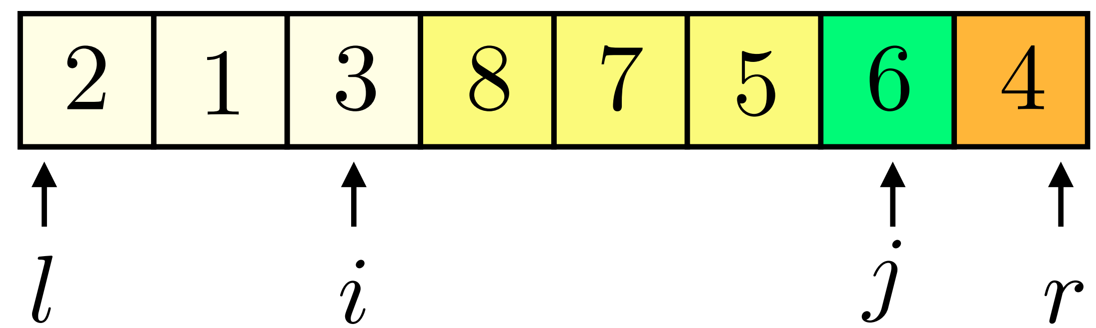
Exemplo:



$$A[j] \leq A[r]?$$

Quicksort

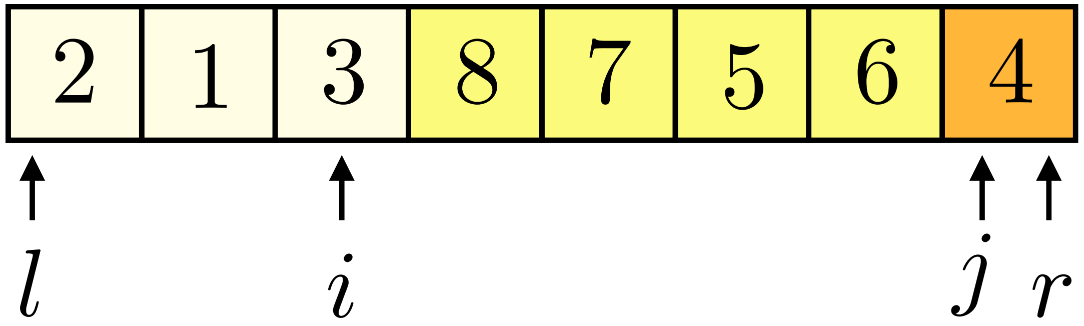
Exemplo:



$$A[j] \leq A[r]?$$

Quicksort

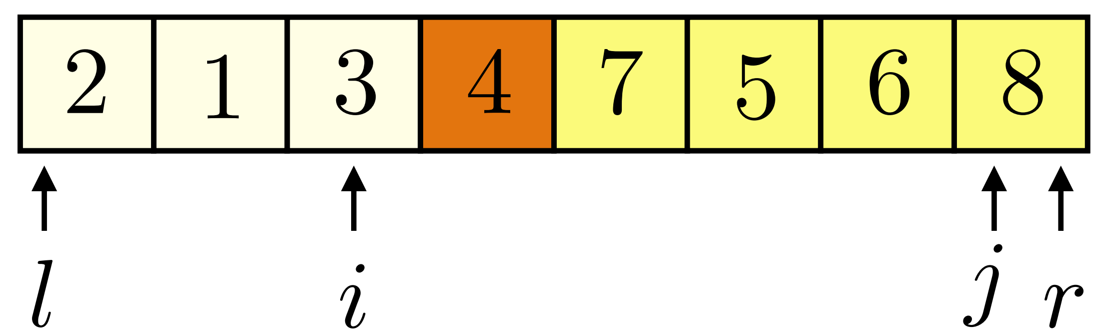
Exemplo:



$$A[i+1] \leftrightarrow A[r]$$

Quicksort

Exemplo:



$$A[i+1] \leftrightarrow A[r]$$

Quicksort

Quicksort

A pergunta que podemos nos fazer agora é: como nos utilizarmos de *partição* para **ordenar** um dado vetor A com n elementos?

Quicksort

A pergunta que podemos nos fazer agora é: como nos utilizarmos de *partição* para **ordenar** um dado vetor A com n elementos?

Para começar, o que podemos dizer sobre o elemento $A[q]$?

Quicksort

A pergunta que podemos nos fazer agora é: como nos utilizarmos de *partição* para **ordenar** um dado vetor A com n elementos?

Para começar, o que podemos dizer sobre o elemento $A[q]$?

Ele está na posição em que estaria se o vetor estivesse ordenado!

A pergunta que podemos nos fazer agora é: como nos utilizarmos de *partição* para **ordenar** um dado vetor A com n elementos?

Para começar, o que podemos dizer sobre o elemento $A[q]$?

Ele está na posição em que estaria se o vetor estivesse ordenado!

Não podemos dizer o mesmo sobre os demais elementos, mas eles devem estar "próximos" de onde gostaríamos que eles estivessem.

Quicksort

Quicksort

Como os elementos à esquerda de $A[q]$ são menores ou iguais a $A[q]$ e os elementos à direita são maiores, podemos chamar *partição* para cada um desses segmentos de forma independente.

Quicksort

Como os elementos à esquerda de $A[q]$ são menores ou iguais a $A[q]$ e os elementos à direita são maiores, podemos chamar *partição* para cada um desses segmentos de forma independente.

O que acontece se repetirmos o procedimento acima, de forma recursiva, até que todas as partições tenham tamanho 0 ou 1?

Quicksort

Como os elementos à esquerda de $A[q]$ são menores ou iguais a $A[q]$ e os elementos à direita são maiores, podemos chamar *partição* para cada um desses segmentos de forma independente.

O que acontece se repetirmos o procedimento acima, de forma recursiva, até que todas as partições tenham tamanho 0 ou 1?

Isto é exatamente o que faz o algoritmo *quicksort*.

Como os elementos à esquerda de $A[q]$ são menores ou iguais a $A[q]$ e os elementos à direita são maiores, podemos chamar *partição* para cada um desses segmentos de forma independente.

O que acontece se repetirmos o procedimento acima, de forma recursiva, até que todas as partições tenham tamanho 0 ou 1?

Isto é exatamente o que faz o algoritmo *quicksort*.

(01) **algoritmo** *quicksort*(**ref** A, n)
(02)   *quicksort_aux*$(A, 1, n)$
(03) **fimalgoritmo**

Quicksort

Quicksort

algoritmo *quicksort_aux*(**ref** A, l, r)
    se $l < r$ **então**
        $q \leftarrow$ *partição*(A, l, r)
        quicksort_aux$(A, l, q-1)$
        quicksort_aux$(A, q+1, r)$
    fimse
fimalgoritmo

Quicksort

Quicksort

Qual é a complexidade de pior caso de *partição*?

Quicksort

Qual é a complexidade de pior caso de *partição*?

```
algoritmo partição(ref A, l, r)
    x ← A[r]
    i ← l − 1
    para j ← l até r − 1 faça
        se compara(A[j], x) ≠ 1 então
            i ← i + 1
            A[i] ↔ A[j]
        fimse
    fimpara
    A[i + 1] ↔ A[r]
    retorne i + 1
fimalgoritmo
```

Quicksort

Qual é a complexidade de pior caso de *partição*?

```
algoritmo partição(ref A, l, r)
    x ← A[r]
    i ← l − 1
    para j ← l até r − 1 faça
        se compara(A[j], x) ≠ 1 então
            i ← i + 1
            A[i] ↔ A[j]
        fimse
    fimpara
    A[i + 1] ↔ A[r]
    retorne i + 1
fimalgoritmo
```

Pelo que vemos acima, podemos deduzir que é $\Theta(r - l)$.

Quicksort

Quicksort

Na verdade, o tempo de execução de *partição* (no melhor ou pior caso) é proporcional ao tamanho $r - l + 1$ do segmento $A[l..r]$ de A.

Quicksort

Na verdade, o tempo de execução de *partição* (no melhor ou pior caso) é proporcional ao tamanho $r - l + 1$ do segmento $A[l..r]$ de A.

Qual é a complexidade de pior caso de *quicksort*?

Quicksort

Na verdade, o tempo de execução de *partição* (no melhor ou pior caso) é proporcional ao tamanho $r - l + 1$ do segmento $A[l..r]$ de A.

Qual é a complexidade de pior caso de *quicksort*?

Obviamente, a complexidade de pior caso de *quicksort* é a mesma da chamada inicial ao procedimento *quicksort_aux*; isto é,

$$\textit{quicksort\_aux}(A, 1, n)$$

Quicksort

Quicksort

Como *quicksort_aux* foi descrito de forma recursiva, podemos expressar o número de OPs de *quicksort_aux* por uma recorrência.

Quicksort

Como *quicksort_aux* foi descrito de forma recursiva, podemos expressar o número de OPs de *quicksort_aux* por uma recorrência.

```
algoritmo quicksort_aux(ref A, l, r)
    se l < r então
        q ← partição(A, l, r)
        quicksort_aux(A, l, q − 1)
        quicksort_aux(A, q + 1, r)
    fimse
fimalgoritmo
```

Quicksort

Quicksort

Seja $t(n)$ o número de OPs de *quicksort* em um vetor A de tamanho n. Então,

$$t(n) = t(q-1) + t(n-q) + \Theta(n)\,,$$

onde $\Theta(n)$ é o número de OPs de *partição* na entrada $A[1..n]$, $t(q-1)$ é o número de OPs da chamada recursiva de *quicksort* para o segmento $A[1..q-1]$ de A e $t(n-q)$ é o número de OPs da chamada recursiva de *quicksort* para o segmento $A[q+1..n]$ de A.

Quicksort

Quicksort

Como estamos interessados na complexidade de pior caso, queremos na verdade determinar um valor de q tal que $t(n)$ seja máximo:

$$t(n) = \max_{1 \leq q \leq n} \{t(q-1) + t(n-q)\} + \Theta(n)\,.$$

Quicksort

Como estamos interessados na complexidade de pior caso, queremos na verdade determinar um valor de q tal que $t(n)$ seja máximo:

$$t(n) = \max_{1 \leq q \leq n} \{t(q-1) + t(n-q)\} + \Theta(n)\,.$$

Embora não seja nada intuitivo, podemos mostrar que os valores de q que maximizam $t(n)$ em $\{1, \ldots, n\}$ são $q = 1$ e $q = n$.

Quicksort

Como estamos interessados na complexidade de pior caso, queremos na verdade determinar um valor de q tal que $t(n)$ seja máximo:

$$t(n) = \max_{1 \leq q \leq n} \{t(q-1) + t(n-q)\} + \Theta(n) \, .$$

Embora não seja nada intuitivo, podemos mostrar que os valores de q que maximizam $t(n)$ em $\{1, \ldots, n\}$ são $q = 1$ e $q = n$.

Se isto acontecer em cada chamada ao procedimento *partição*, temos que $t(n) = \Theta(n^2)$; ou seja, *quicksort* é quadrático no **pior caso**.

Quicksort

Quicksort

Note que $q = 1$ ou $q = n$ exatamente quando o procedimento *partição* cria uma partição vazia e outra com $n-1$ elementos.

Note que $q = 1$ ou $q = n$ exatamente quando o procedimento *partição* cria uma partição vazia e outra com $n - 1$ elementos.

E quando isso ocorre?

Note que $q = 1$ ou $q = n$ exatamente quando o procedimento *partição* cria uma partição vazia e outra com $n-1$ elementos.

E quando isso ocorre?

Quando $A[1..n]$ está ordenado em ordem crescente ou decrescente.

Note que $q = 1$ ou $q = n$ exatamente quando o procedimento *partição* cria uma partição vazia e outra com $n - 1$ elementos.

E quando isso ocorre?

Quando $A[1..n]$ está ordenado em ordem crescente ou decrescente.

Neste caso, toda vez que *partição* é chamado para um segmento $A[l..r]$, este segmento está ordenado! Logo, haverá n chamadas a *quicksort_aux*. A primeira com um vetor de n elementos, a segunda com um vetor de $n - 1$ elementos, a terceira com um vetor de $n - 2$ elementos, e assim por diante.

Quicksort

Quicksort

Em cada chamada a *quicksort_aux*, o número de OPs de *partição* é proporcional ao tamanho do vetor. Então, o número total de OPs de *quicksort_aux* é proporcional à soma dos n termos

$$n + (n-1) + (n-2) + \cdots + 1 = \Theta(n^2)\,.$$

Quicksort

Em cada chamada a *quicksort_aux*, o número de OPs de *partição* é proporcional ao tamanho do vetor. Então, o número total de OPs de *quicksort_aux* é proporcional à soma dos n termos

$$n + (n-1) + (n-2) + \cdots + 1 = \Theta(n^2)\,.$$

Isto apenas confirma o que dissemos antes: $t(n) = \Theta(n^2)$ no pior caso. Logo, o pior caso de *quicksort* ocorre justamente quando o vetor A está ordenado em ordem crescente ou decrescente.

Quicksort

Quicksort

Qual é a complexidade de melhor caso de *quicksort*?

Quicksort

Qual é a complexidade de melhor caso de *quicksort*?

Usando o mesmo raciocínio de antes, o número mínimo de OPs de *quicksort_aux* é dado pela seguinte relação de recorrência:

Quicksort

Qual é a complexidade de melhor caso de *quicksort*?

Usando o mesmo raciocínio de antes, o número mínimo de OPs de *quicksort_aux* é dado pela seguinte relação de recorrência:

$$t(n) = \min_{1 \leq q \leq n} \{t(q-1) + t(n-q-1)\} + \Theta(n)\,.$$

Quicksort

Qual é a complexidade de melhor caso de *quicksort*?

Usando o mesmo raciocínio de antes, o número mínimo de OPs de *quicksort_aux* é dado pela seguinte relação de recorrência:

$$t(n) = \min_{1 \leq q \leq n} \{t(q-1) + t(n-q-1)\} + \Theta(n)\,.$$

O valor de q que minimiza $t(n)$ em cada chamada a *quicksort_aux* é

$$q = \frac{n+1}{2}\,.$$

Em outras palavras, o melhor caso ocorre quando *partição* consegue dividir o segmento do vetor em duas partes de tamanhos "iguais".

Quicksort

Quicksort

Logo,

$$t(n) \leq 2 \cdot t(n/2) + \Theta(n) = \Theta(n \lg n)\,.$$

Quicksort

Logo,

$$t(n) \leq 2 \cdot t(n/2) + \Theta(n) = \Theta(n \lg n) \,.$$

Então, podemos dizer que *quicksort* possui complexidade de tempo $\Theta(n^2)$ no pior caso (assim como os algoritmos de ordenação por inserção, seleção e bolha) e $\Theta(n \lg n)$ no melhor caso (que é melhor do que os algoritmos de ordenação por seleção e bolha e pior do que o algoritmo de ordenação por inserção).

Quicksort

Quicksort

Uma diferença fundamental entre *quicksort* e os algoritmos que vimos anteriormente é que a complexidade de caso médio de *quicksort* é $\Theta(n \lg n)$, enquanto a daqueles outros algoritmos é $\Theta(n^2)$.

Além disso, *quicksort* possui complexidade de tempo $\Theta(n \lg n)$ para quase todas as entradas (aquelas que correspondem ao pior caso são uma pequena parte de todas as possíveis permutações de A).

A observação acima, juntamente com os fatos que veremos em outras aulas, fazem do *quicksort* o algoritmo de ordenação preferido na prática.

Referências

Referências

- Paulo A. Azeredo
  Métodos de Classificação de Dados, Editora Campus, 1996.

Existem inúmeros applets na Internet que ilustram o funcionamento dos algoritmos de ordenação mais conhecidos. Eu recomendo o seguinte:

http://www.cs.oswego.edu/~mohammad/classes/csc241/samples/sort/Sort2-E.html