CENTRO DE CIÊNCIA E TECNOLOGIA
LABORATÓRIO DE CIÊNCIAS MATEMÁTICAS
UNIVERSIDADE ESTADUAL DO NORTE FLUMINENSE

Algoritmos de Ordenação
Quicksort

Disciplina: Estrutura de Dados I

Prof. Fermín Alfredo Tang Montané

Curso: Ciência da Computação

Algoritmos de Ordenação
Quicksort

- Algoritmo proposto por C.A.R Hoare em 1962.
- O *quicksort*, ordenação rápida, é um algoritmo cujo tempo de execução de pior caso é $O(n^2)$ sobre um arranjo de n números.
- Apesar desse tempo de **execução lento no pior caso**, o *quicksort* com frequência é a melhor opção prática para ordenação, devido a sua notável eficiência na média.
- Seu tempo de execução no **caso médio** é O(n log n) e os fatores constantes ocultos na notação O(n log n) são bastante pequenos.
- Ele também apresenta vantagem na ordenação local e funciona bem até mesmo em ambientes de memória virtual.

Algoritmos de Ordenação
Quicksort

- O *quicksort*, assim como o *mergesort*, se baseia no princípio de **dividir e conquistar**. O processo para ordenar um subarranjo A[p, ..., r] pode ser descrito em dois passos:

- **Dividir:** O arranjo A[p .. r] é particionado (reorganizado) em dois subarranjos, A[p, ... ,q-l] e A[q+1, ... ,r]:
  - no primeiro subarranjo (A[p , ..., q-1]) cada elemento é menor que ou igual a A [q];
  - no segundo subarranjo (A[q+1, ..., r]) cada elemento é maior que ou igual a A [q];
  - O índice q é calculado como parte desse procedimento.
- **Conquistar:** Os dois subarranjos A[p .. q-1] e A[q + 1..r] são ordenados por chamadas recursivas a *quicksort*.

Algoritmos de Ordenação
Quicksort - Algoritmo

- O Algoritmo *Quicksort* precisa da uma chamada inicial. Nesta chamada, pede-se para ordenar um arranjo A[1.. N], onde N = Tamanho (A). Assim temos:

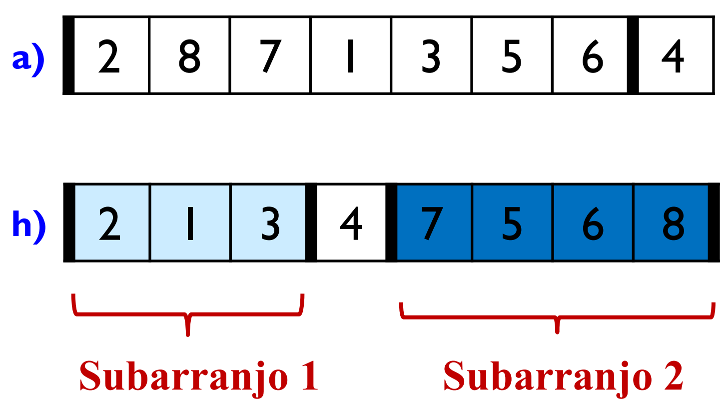
```
QUICKSORT (A, 1, N );          { ORDENA O ARRANJO A [1 .. N] , delimitado entre 1 e N }
```

- A primeira chamada desencadeia uma série de chamadas recursivas, que executam o procedimento de ordenar um subarranjo A [p..r] mostrado a seguir:

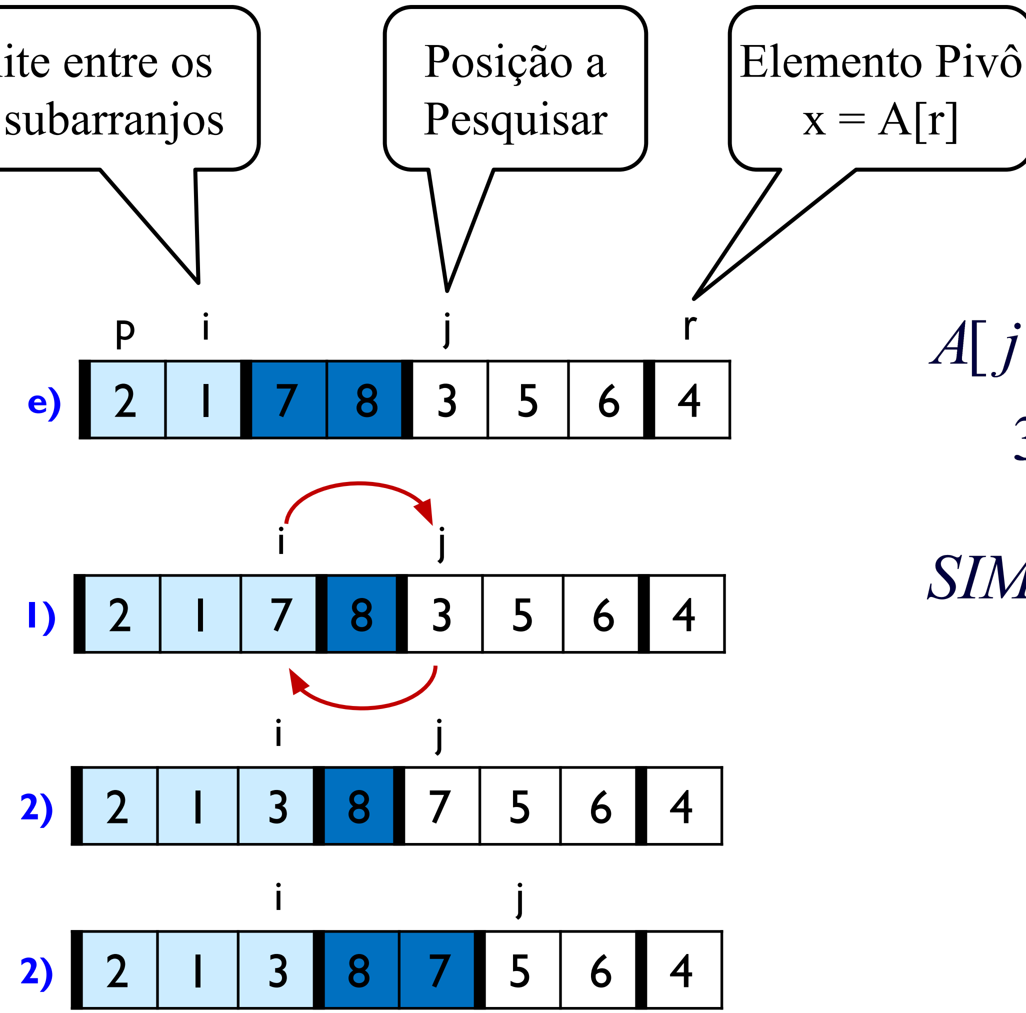
```
QUICKSORT (A, p, r)            { ORDENA O ARRANJO A [p .. r] , delimitado entre p e r }
inicio
    se p < r então
        { escolhe um elemento A[k] do arranjo A[p..r]; reorganiza o arranjo em dois subarranjos
          com elementos menores e maiores do que A[k] respectivamente }
        q ← PARTICIONA (A, p, r);   { retorna a posição q certa, para o elemento A[k] }
        QUICKSORT (A, p, q-1);      { ordena o subarranjo A [p..q-1] }
        QUICKSORT (A, q+1, r) ;     { ordena o subarranjo A [q+1..r] }
    fim-se
fim
```

Algoritmos de Ordenação
Quicksort - Algoritmo

- O elemento fundamental do algoritmo *Quicksort* é o procedimento PARTICIONA (A, p, r) que escolhe um elemento **pivô** A[k] e reorganiza o subarranjo em torno dele.
- O procedimento PARTICIONA produz dois subarranjos:
  - O primeiro contém elementos menores ou iguais do que A[k];
  - O segundo contém elementos maiores do que A[k];
- Embora, os subarranjos produzidos **não fiquem ordenados,** o elemento A[k] é colocado na **posição certa** do arranjo, **posição q**.
- O procedimento **retorna q**, a posição certa de A[k] no arranjo.



Algoritmos de Ordenação
Quicksort - Algoritmo



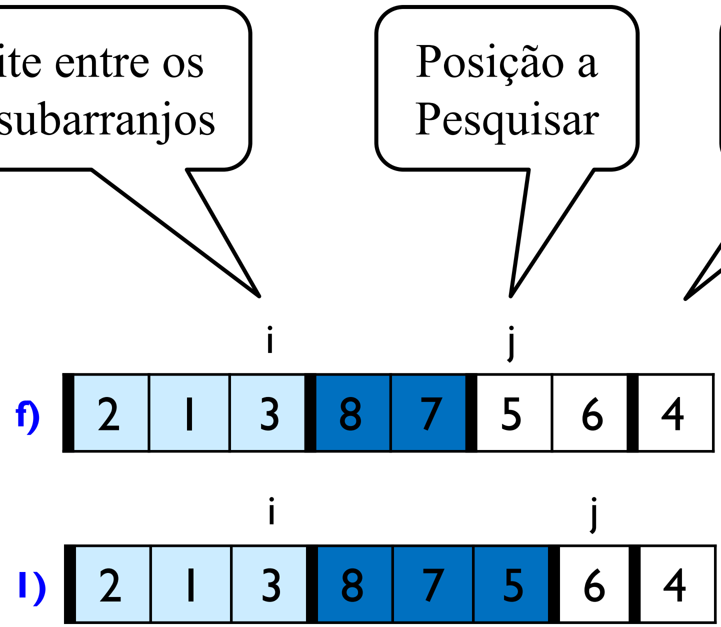
$A[j] \leq x \quad ?$

$3 \leq 4 \quad ?$

SIM

Algoritmos de Ordenação

Quicksort - Algoritmo

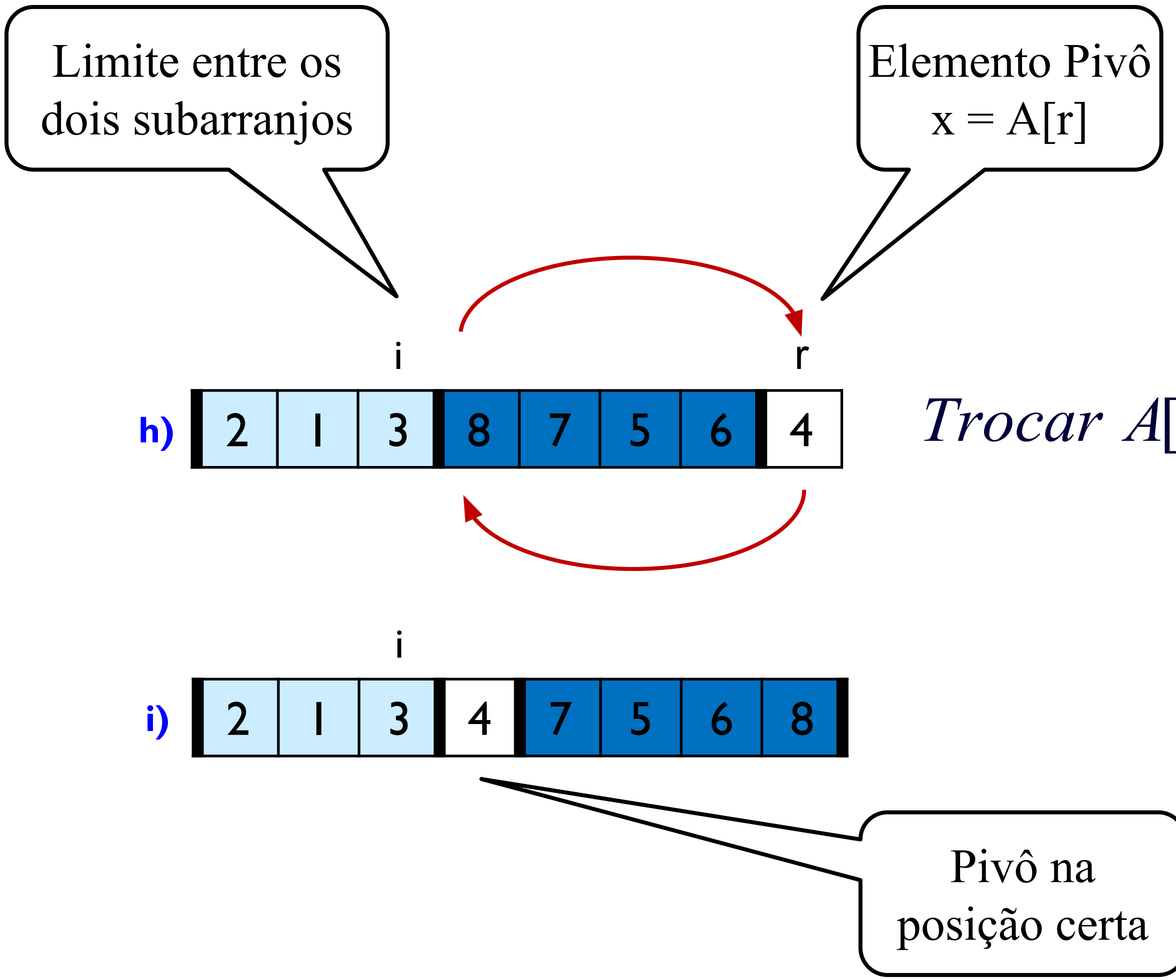


$A[j] \leq x$?

$5 \leq 4$?

NÃO

Algoritmos de Ordenação
Quicksort - Algoritmo



$$Trocar\ A[i+1] \leq A[r]$$

Algoritmos de Ordenação

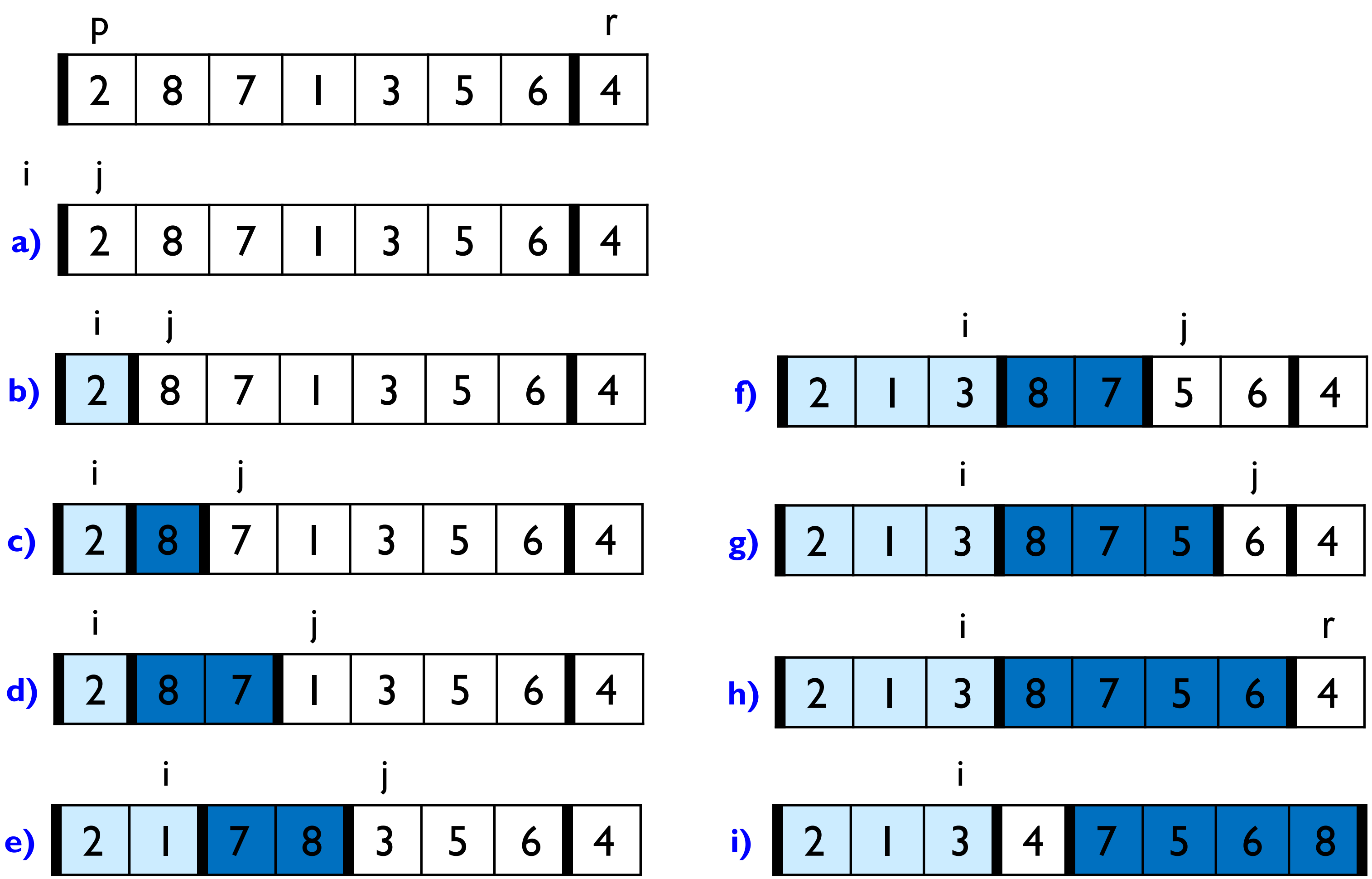
Quicksort - Algoritmo

- Apresentamos uma versão do algoritmo PARTICIONA (A, p, r), onde sempre se escolhe como elemento A[k] o último elemento do subarranjo. Esta versão é descrita a continuação:

```
PARTICIONA (A, p, r)              { ORDENA O ARRANJO A [p .. r] , delimitado entre p e r }
inicio
   { Inicialização }
   x ← A [r];                     { Escolhe o último elemento do subarranjo A [p .. r] }
   i ← p – 1;                     { O indice i indica o limite entre os subarranjos }
   { Separa os elementos do arranjo A [p..r] em dois subarranjos }
   para j ← p até r - 1 fazer     { O indice j indica o elemento a pesquisar }
       se A [j] ≤ x então
           i ← i + 1;             { O Primeiro subarranjo cresce }
           trocar A [i] e A[j];   { Troca elementos: o primeiro do subarranjo 2 e o pesquisado }
       fim-se
   fim-fazer
   { Coloca o pivô na posição certa }
   trocar A [i+1] e A[r];
   Retorna i+1;
fim
```

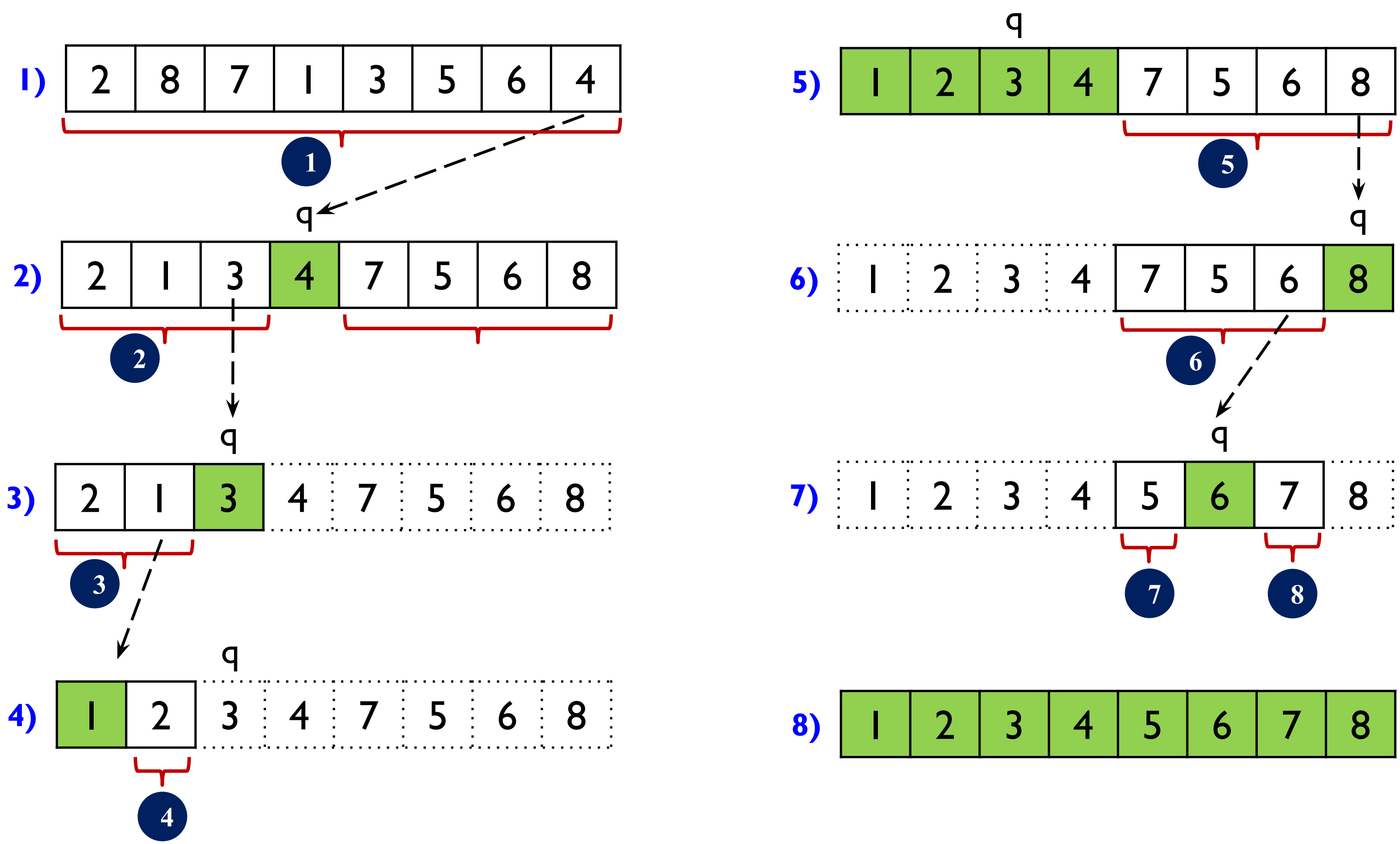
Quicksort – Exemplo do PARTICIONA

- O algoritmo PARTICIONA (A, p, r), escolhe como pivô o último elemento.



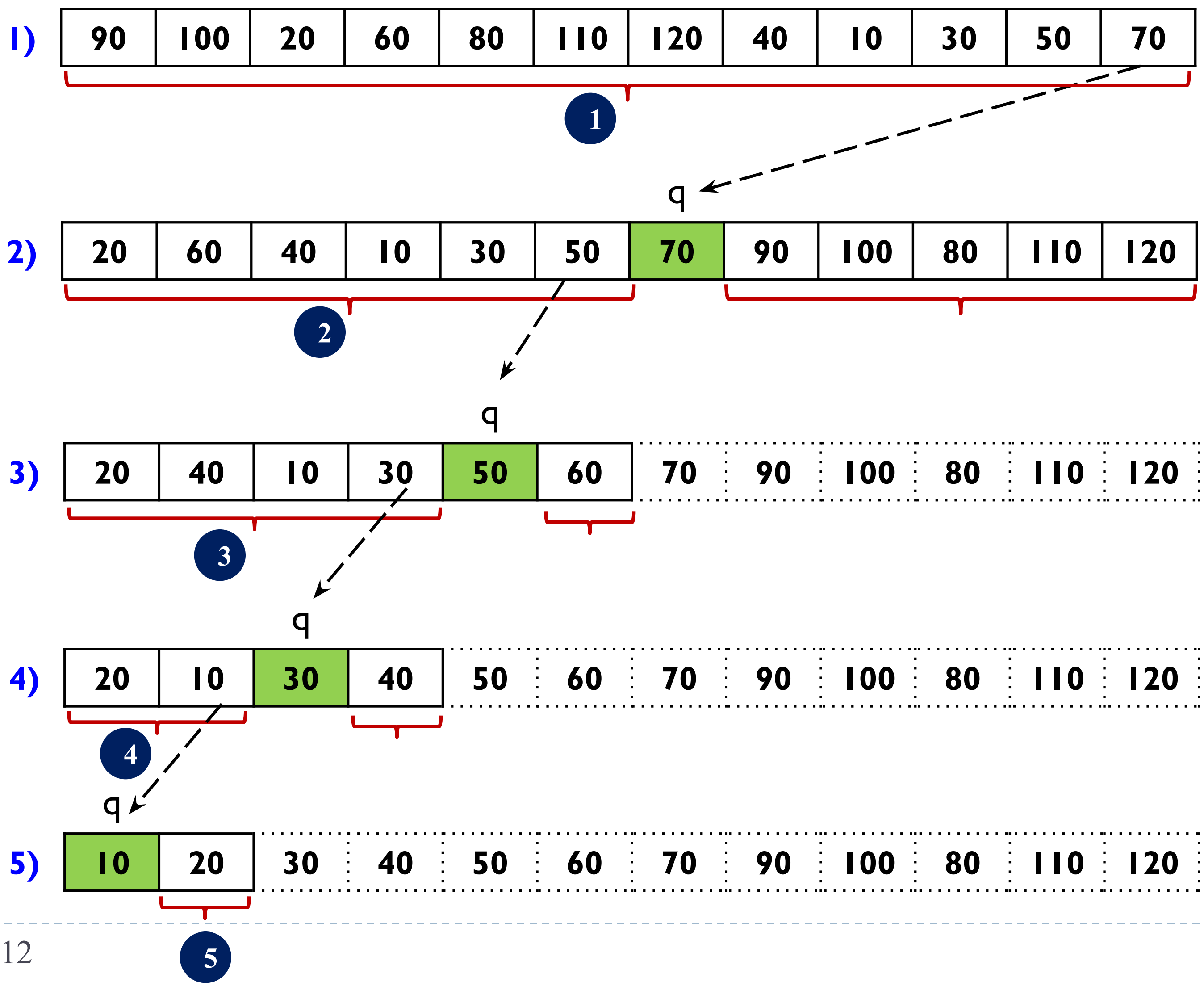
Algoritmos de Ordenação

Quicksort – Exemplo Completo

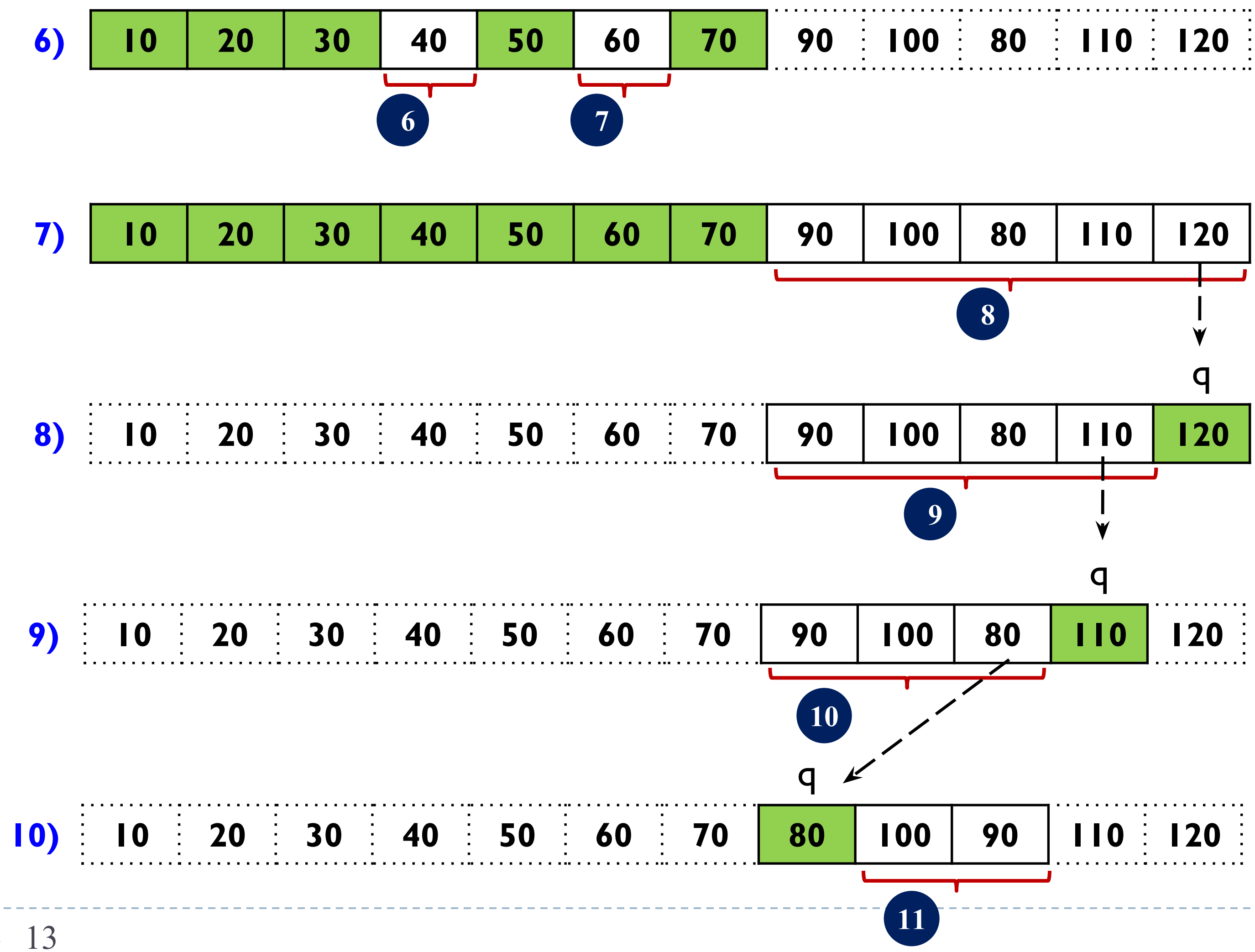


Algoritmos de Ordenação

Quicksort – Exemplo2 Completo



Algoritmos de Ordenação
Quicksort – Exemplo2 Completo



Algoritmos de Ordenação

Quicksort – Exemplo2 Completo

11)

								q			
10	20	30	40	50	60	70	80	90	100	110	120

12

12)

10	20	30	40	50	60	70	80	90	100	110	120

Ordenação Quicksort

Complexidade do Algoritmo – Comparações e trocas

- O número de comparações dependerá dos valores contidos no vetor, existem três casos a considerar:
  - **Pior caso**.- Acontece quando a árvore de subproblemas fica totalmente desbalanceada, com altura n. Por exemplo, o vetor se encontra em ordem descendente (ou ascendente). **Executa em tempo $O(n^2)$.**
  - **Caso Médio**.- Acontece quando a árvore de subproblemas fica parcialmente desbalanceada. Tempo de execução difícil de estimar.
  - **Melhor Caso**.- Acontece com dados aleatórios, quando o **pivô** corresponde ao valor médio dos itens ordenados. Isso gera uma árvore balanceada de subproblemas com altura $\log n + 1$. Cada particionamento de n elementos executa em $\boldsymbol{O(n)}$. **Executa em $O(n\log n)$**.

Algoritmo QuickSort

Complexidade do Algoritmo – Comparações e Trocas

- O número de comparações depende exclusivamente do processo de particionamento. Considerando um conjunto de n elementos. Para particionar esse conjunto, **no melhor caso,** são necessárias $n-1$ comparações e $n/2$ trocas.

n	Elementos *n log n*	Comp.	Trocas
2	2	2	1
4	8	8	4
8	24	24	12
16	64	64	32
32	160	160	80
64	384	384	192
128	896	896	448

Algoritmos Avançados
Comparação de Complexidade

- O algoritmo Quicksort é ligeiramente superior realizando um número menor de trocas do que o equivalente de copias do Mergesort.

		Mergesort		**Quicksort**	
n	Elementos ***n log n***	Comp. ***n log n***	Copias ***2(n log n)***	Comp. ***n log n***	Trocas ***(n log n)/2***
2	2	2	4	2	1
4	8	8	16	8	4
8	24	24	48	24	12
16	64	64	128	64	32
32	160	160	320	160	80
64	384	384	768	384	192
128	896	896	1792	896	448

Comparação de Complexidade

- A comparação entre os três algoritmos em termos de complexidade:

	Shellsort	Mergesort	Quicksort
Caso	Complex. tempo	Complex. Tempo	Complex. Tempo
Pior	$O(n^{3/2})$	$O(n \log n)$	$O(n^2)$
Médio	$O(n^{3/2})$	$O(n \log n)$	$O(n \log n)$
Melhor	$O(n^{3/2})$	$O(n \log n)$	$O(n \log n)$

Algoritmos de Ordenação

Quicksort - Aleatorizado

- Tornamos nosso algoritmo aleatório usando uma técnica de aleatoriedade, chamada **amostragem aleatória**.
- Em vez de sempre usar A[r] como pivô, usaremos um elemento escolhido aleatoriamente a partir do subarranjo A[p .. r].
- Faremos isso permutando o elemento A [r] com um elemento escolhido aleatorio de A[p .. r].
- Essa modificação, em que fazemos a amostragem aleatória do intervalo (p, ..., r,) garante que o elemento pivô x = A[r] tem a mesma probabilidade de ser escolhido dentre os (r - p + 1) elementos do subarranjo.
- Como o elemento pivô é escolhido de forma aleatória, espera-se que a divisão do arranjo de entrada seja razoavelmente bem equilibrada na média.
- Muitas pessoas consideram a versão aleatória resultante de quicksort o algoritmo de ordenação preferido para entrada grandes o suficiente.

Algoritmos de Ordenação
Quicksort Aleatorizado - Algoritmo

- O Algoritmo *Quicksort-Aleatorizado* realiza uma chamada inicial, que desencadeia uma serie de chamadas recursivas.

```
QUICKSORT - ALEATORIZADO  (A, 1, N );    { ORDENA O ARRANJO A [1 .. N] }
```

- Cada chamada recursiva procura ordenar um subarranjo A[p..r]. Esta variante é praticamente idêntica ao *quicksort* simples, com exceção do particionamento.

```
QUICKSORT - ALEATORIZADO (A, p, r)       { ORDENA O ARRANJO A [p .. r] }
inicio
    se p < r  então
        { escolhe o elemento A[k] do arranjo A[p..r] aleatoriamente e reorganiza o arranjo
          em dois subarranjos  com  elementos menores e maiores do que A[k] respectivamente }
        q ← PARTICIONA - ALEATORIZADO (A, p, r);  { posição q certa para o pivô }
        QUICKSORT - ALEATORIZADO  (A,  p, q-1);      { ordena o subarranjo A [p..q-1]  }
        QUICKSORT - ALEATORIZADO (A, q+1, r) ;      { ordena o subarranjo A [q+1..r]  }
    fim-se
fim
```

Algoritmos de Ordenação
Quicksort Aleatorizado - Algoritmo

- O procedimento *Particiona – Aleatorizado* introduz modificações mínimas no procedimento *Particiona* anterior.

```
PARTICIONA - ALEATORIZADO (A, p, r)      { ORDENA O ARRANJO A [p .. r] }
inicio
    i ← ALEATORIO ( p, r);               { escolha aleatória entre p e r }
    trocar A [i+1] e A[r];               { troca do elemento aleatório com o último }
    retorna  PARTICIONA (A, p, r)        { particionamento tradicional }
fim
```

Referências

- Thomas **Cormen**, Charles **Leiserson,** et al.**.** Algoritmos. Teoria e Prática. 2ª Edição. 2002.
- Robert **Lafore**. Estruturas de Dados e Algoritmos em Java. Editora Ciencia Moderna. 2ª Edição. 2004.