ALGORITMOS E ESTRUTURA DE DADOS

Curso de Engenharia de Software
Lucas Sampaio Leite

Quick sort

INSTITUTO FEDERAL
Pernambuco

- O Quick Sort é um algoritmo de ordenação eficiente e amplamente utilizado, baseado na estratégia de divisão e conquista (divide and conquer).
- Seu funcionamento consiste em escolher um elemento da coleção, denominado pivô, e reorganizar os demais elementos de forma que os menores fiquem à sua esquerda e os maiores à sua direita.
- Em seguida, o mesmo processo é aplicado recursivamente às duas partições até que toda a coleção esteja ordenada.

Quick sort

INSTITUTO FEDERAL
Pernambuco

- Diferentemente do Merge Sort, o Quick Sort não necessita de memória auxiliar para realizar a intercalação dos elementos, tornando-o mais econômico em termos de espaço.
- Entretanto, as partições geradas pelo pivô nem sempre são equilibradas. Quando uma das partições se torna muito maior que a outra, o desempenho do algoritmo pode ser significativamente reduzido.

Quick sort

INSTITUTO FEDERAL
Pernambuco

- O Quick Sort baseia-se em uma operação chamada particionamento:
  - Um elemento é escolhido como pivô e os demais elementos são reorganizados de forma que os menores (ou iguais) fiquem à sua esquerda e os maiores à sua direita.
  - O processo é repetido recursivamente para as duas partições geradas.

Elementos menores que o pivô	pivô	Elementos maiores que o pivô

Quick sort

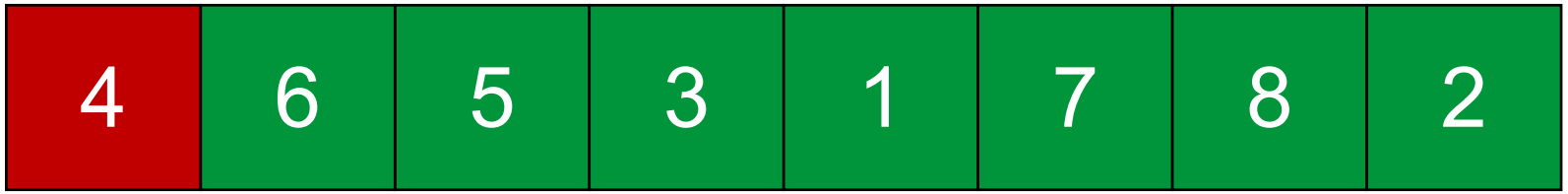
INSTITUTO FEDERAL
Pernambuco

- Passos:
  - Escolha do pivô: Selecione um elemento da coleção para atuar como pivô.
  - Particionamento: Reorganize os elementos de modo que todos os valores menores ou iguais ao pivô fiquem à sua esquerda e todos os valores maiores fiquem à sua direita. Ao final dessa etapa, o pivô estará em sua posição definitiva.
  - Divisão: A coleção é dividida em duas subcoleções: uma à esquerda e outra à direita do pivô.
  - Recursão: Aplique o Quick Sort recursivamente a cada subcoleção.
  - Término: O processo é encerrado quando as subcoleções possuem tamanho 0 ou 1, pois nesses casos elas já estão ordenadas.

Quick sort

INSTITUTO FEDERAL
Pernambuco

- Antes do particionamento:



- Após o particionamento:

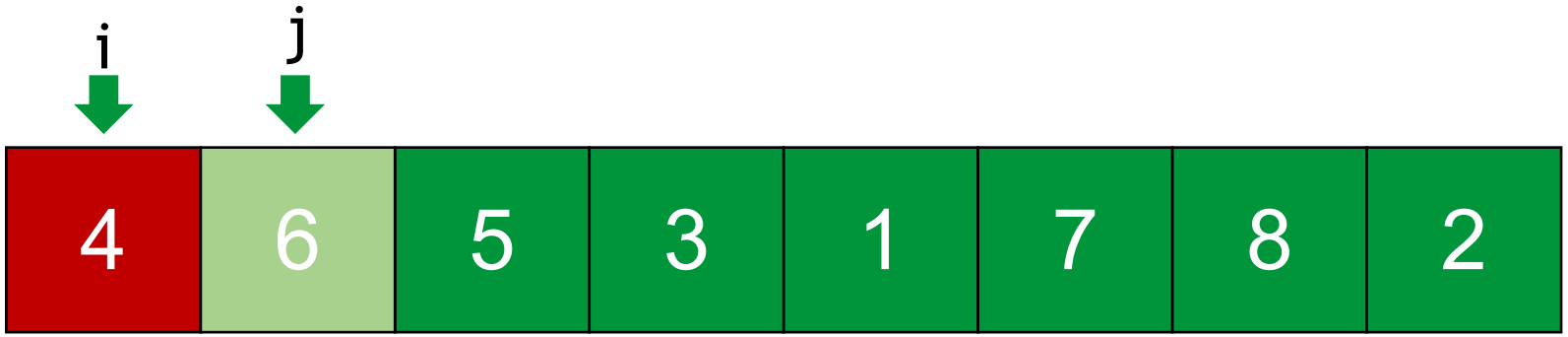
Quick sort

INSTITUTO FEDERAL
Pernambuco

- Particionamento de Lomuto: utiliza um índice para delimitar os elementos menores que o pivô. Durante o percurso da lista, cada elemento menor ou igual ao pivô é trocado para a região à esquerda. Ao final, o pivô é colocado entre os elementos menores e maiores, definindo sua posição definitiva.

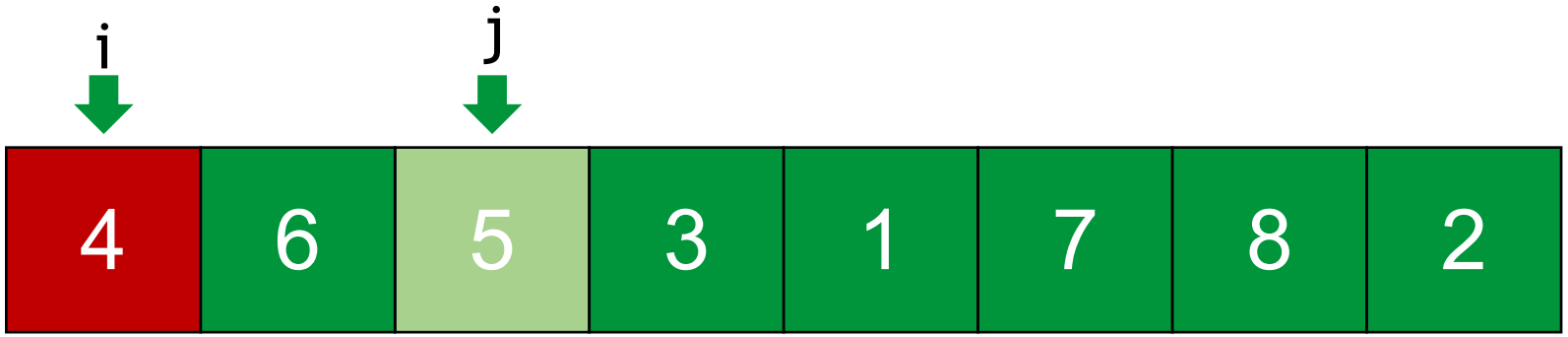
Quick sort

- Pivô = 4



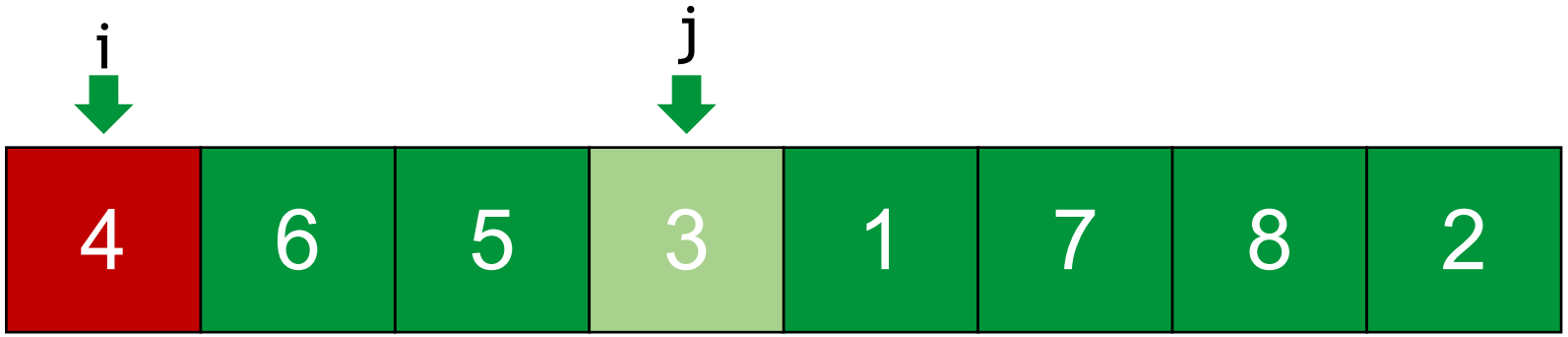
Quick sort

- Pivô = 4



Quick sort

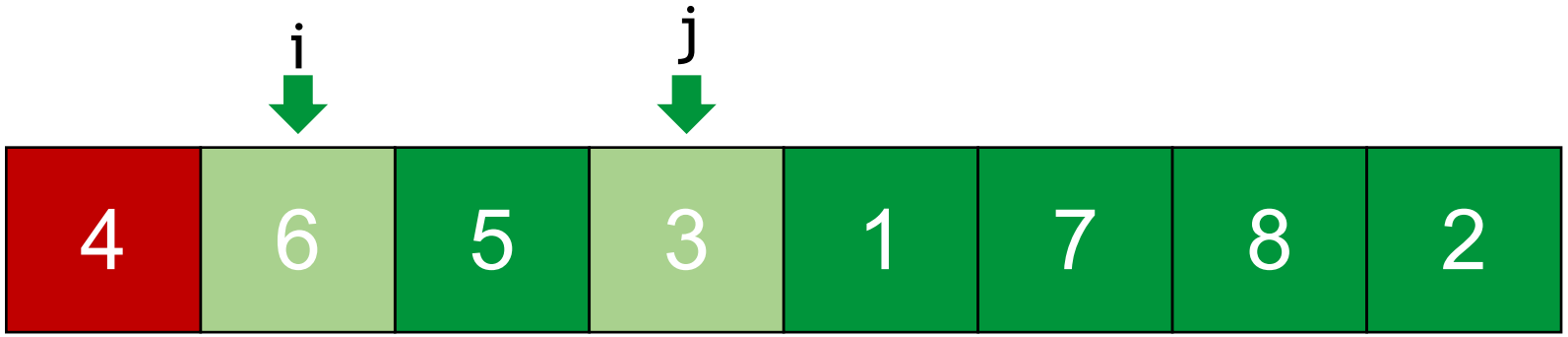
- Pivô = 4



Quick sort

INSTITUTO FEDERAL
Pernambuco

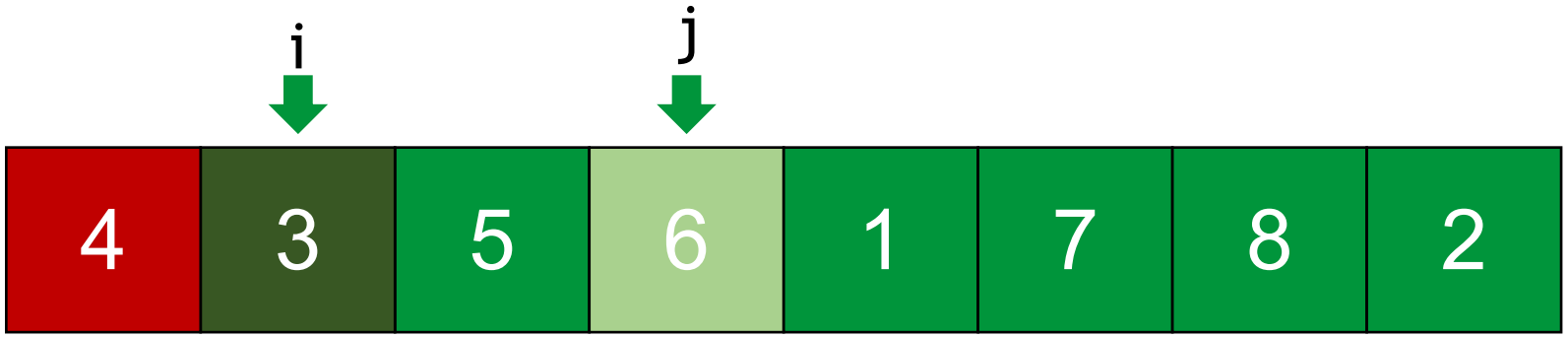
- Pivô = 4



Quick sort

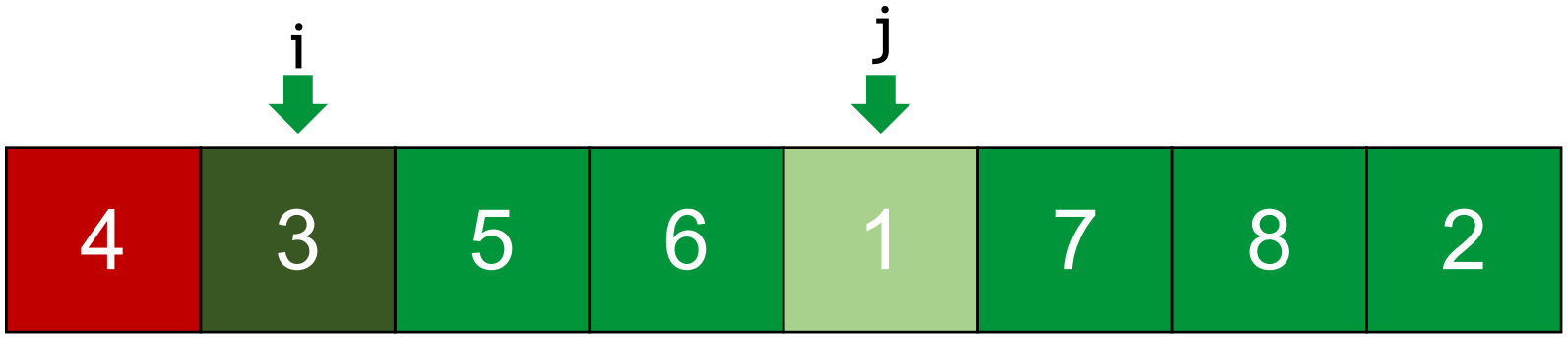
INSTITUTO FEDERAL
Pernambuco

- Pivô = 4



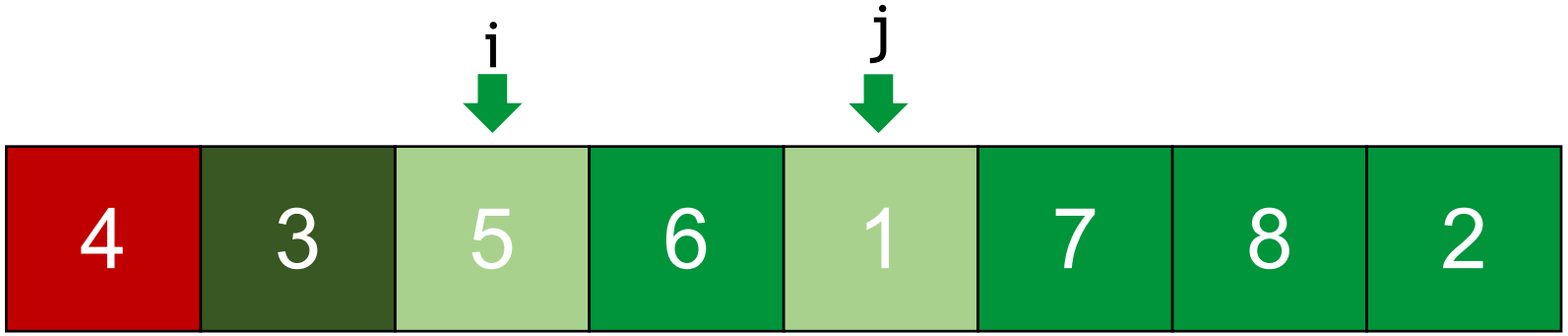
Quick sort

- Pivô = 4



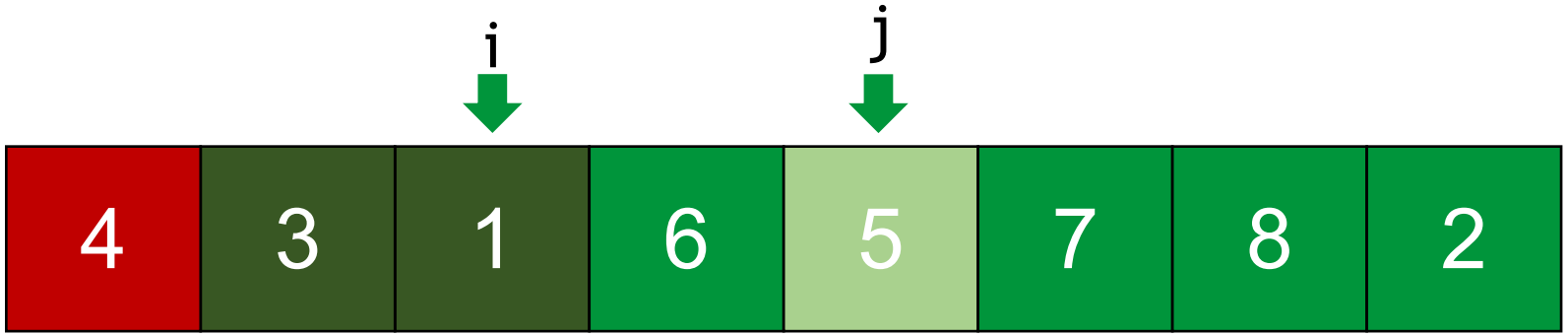
Quick sort

- Pivô = 4



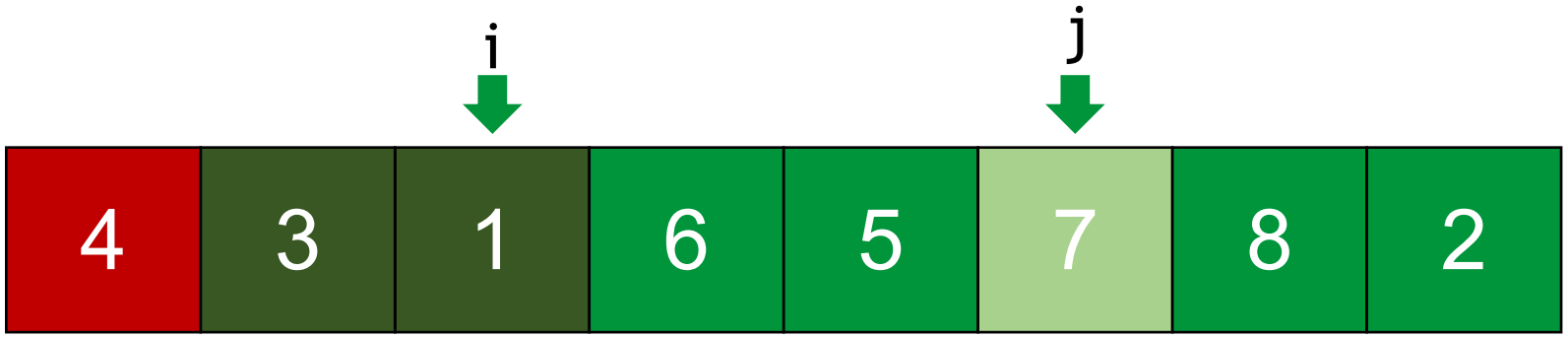
Quick sort

- Pivô = 4



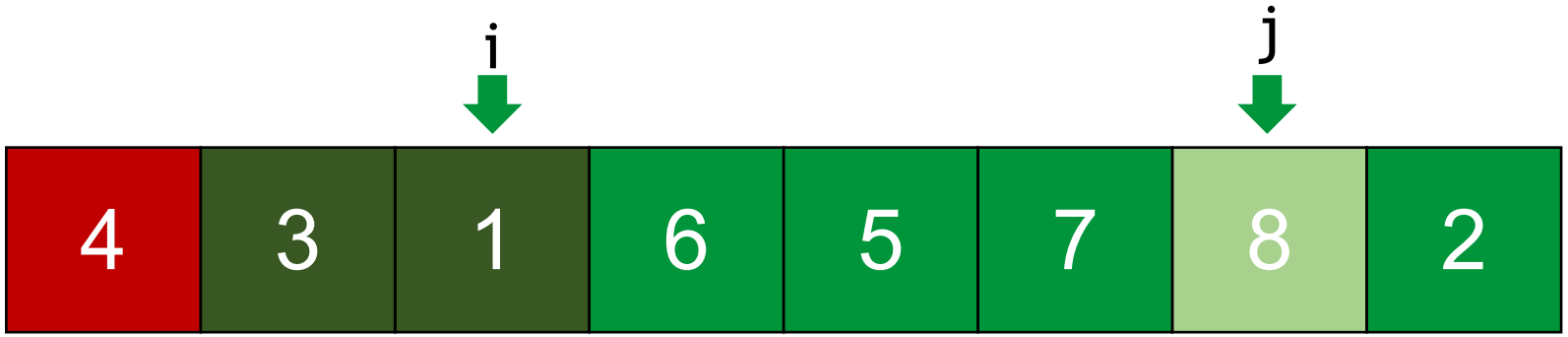
Quick sort

- Pivô = 4



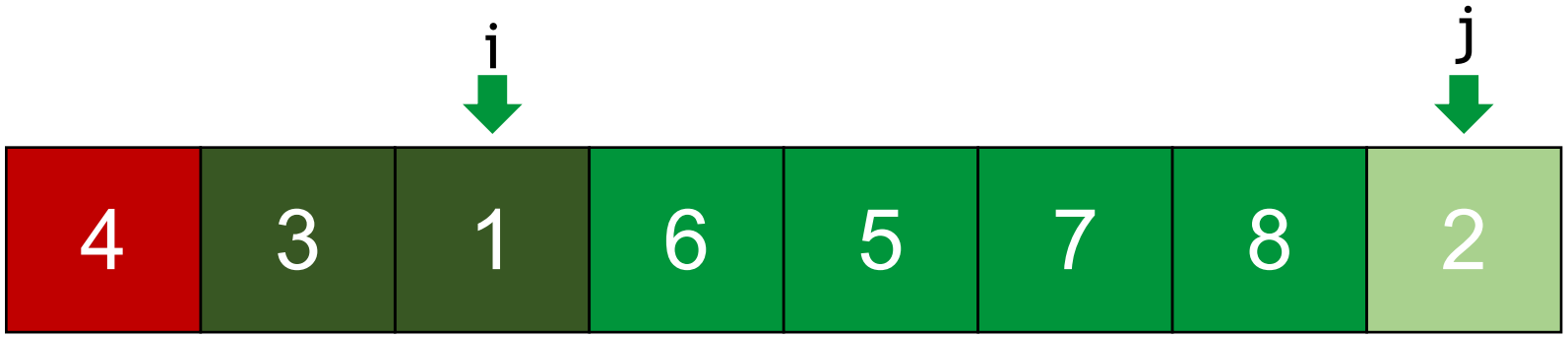
Quick sort

- Pivô = 4



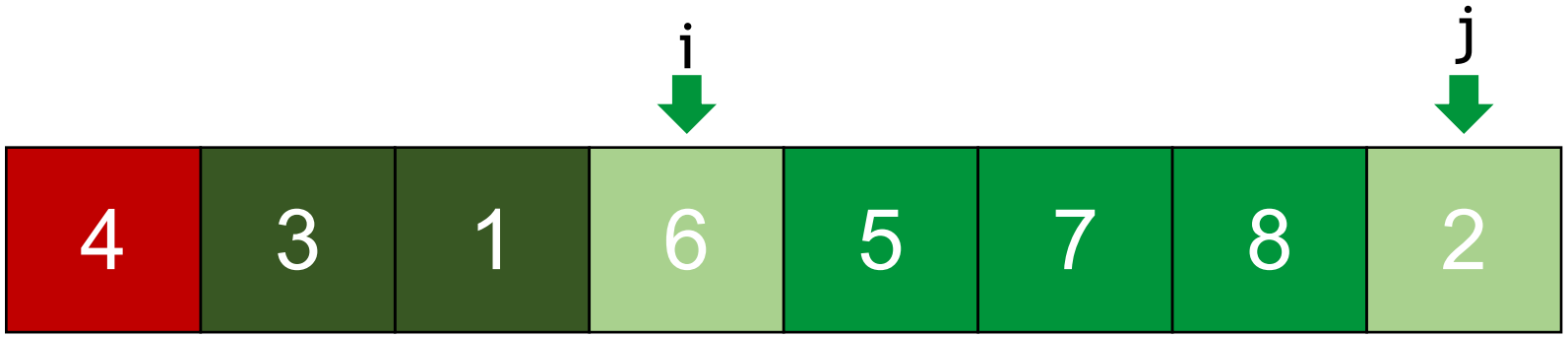
Quick sort

- Pivô = 4



Quick sort

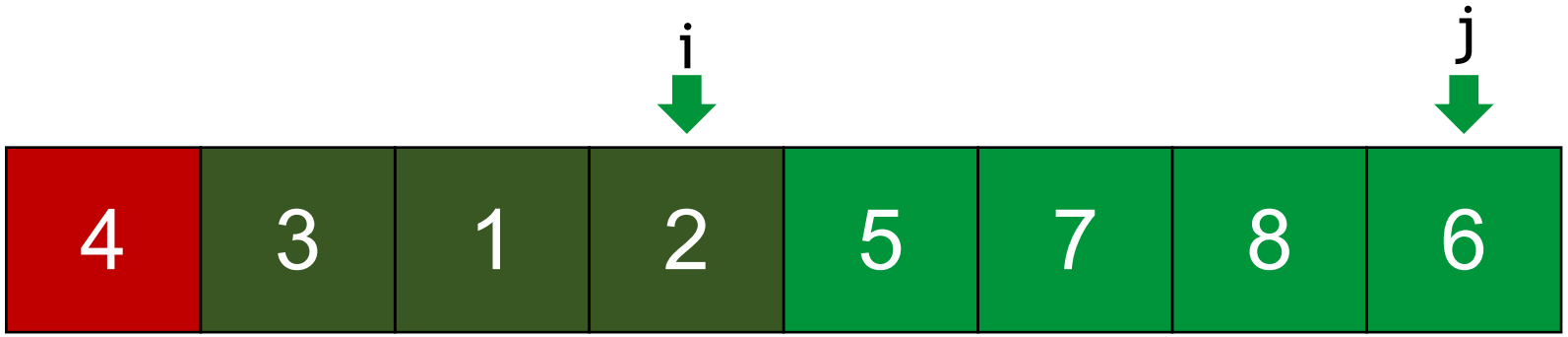
- Pivô = 4



Quick sort

INSTITUTO FEDERAL
Pernambuco

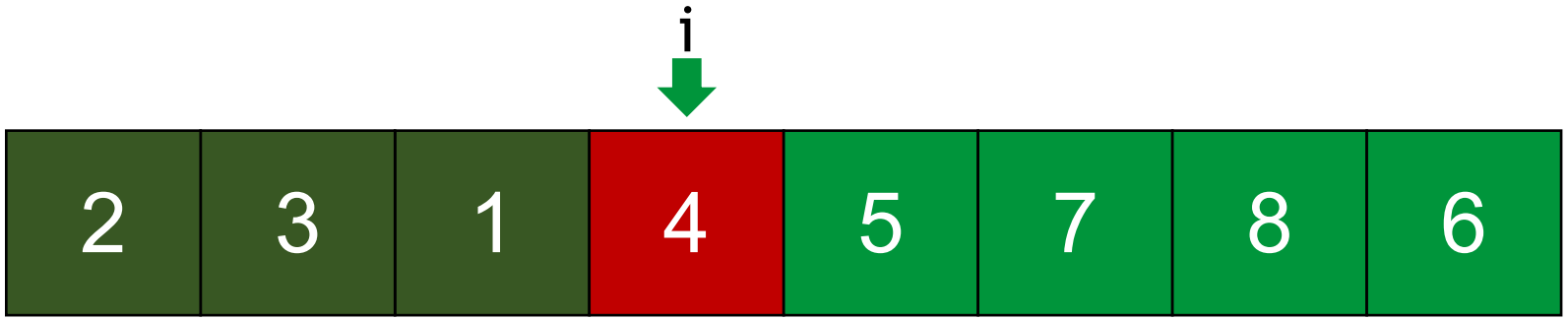
- Pivô = 4



Quick sort

INSTITUTO FEDERAL
Pernambuco

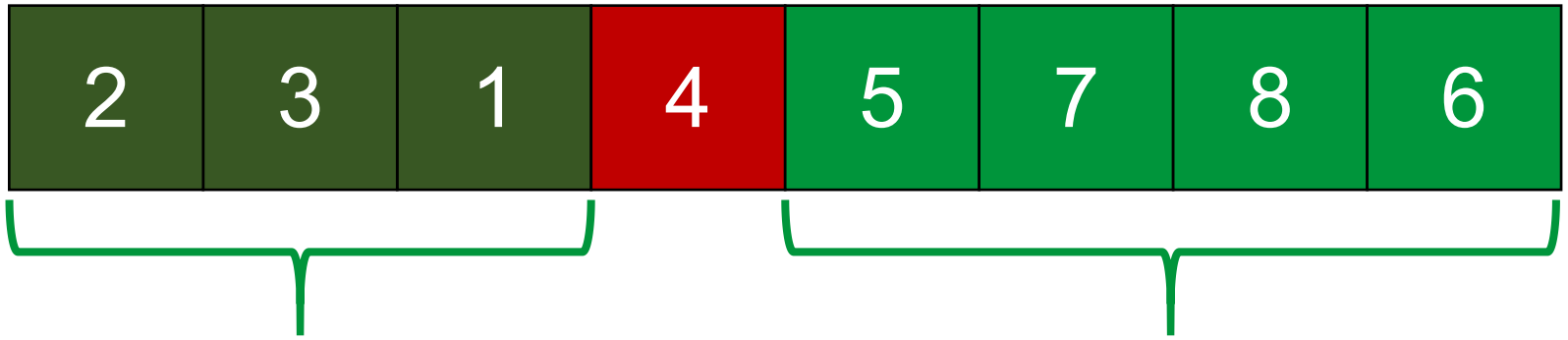
- Pivô = 4



Quick sort

INSTITUTO FEDERAL
Pernambuco

- Pivô = 4



- Aplica-se recursivamente o algoritmo às sublistas à esquerda e à direita do pivô até que tenham tamanho 0 ou 1, quando estarão ordenadas.

```
quickSort(vetor, inicio, indicePivo - 1);
quickSort(vetor, indicePivo + 1, fim);
```

Quick sort

INSTITUTO FEDERAL
Pernambuco

```
void quickSort(int vetor[], int inicio, int fim) {

    if (inicio < fim) {

        int indicePivo = particaoLomuto(vetor, inicio, fim);

        quickSort(vetor, inicio, indicePivo - 1);
        quickSort(vetor, indicePivo + 1, fim);
    }
}
```

Quick sort

```
int particaoLomuto(int vetor[], int inicio, int fim) {
    int pivo = vetor[inicio];
    int i = inicio;

    for (int j = inicio + 1; j <= fim; j++) {
        if (vetor[j] <= pivo) {
            i++;
            swap(vetor, i, j);
        }
    }

    swap(vetor, inicio, i);

    return i;
}
```

INSTITUTO FEDERAL
Pernambuco

Quick sort

INSTITUTO FEDERAL
Pernambuco

```
void swap(int vetor[], int i, int j) {
    int temp = vetor[i];
    vetor[i] = vetor[j];
    vetor[j] = temp;
}
```

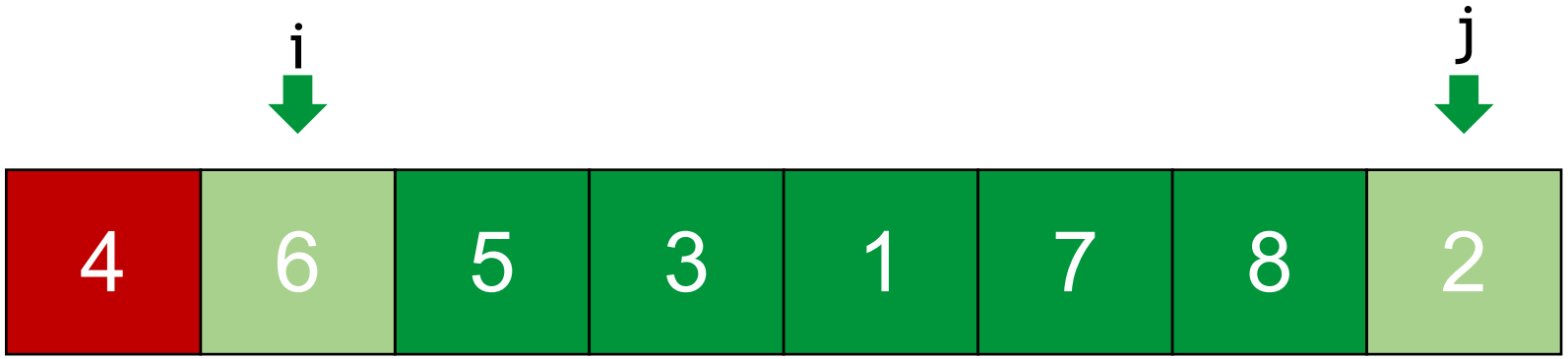
Quick sort

INSTITUTO FEDERAL
Pernambuco

- Particionamento de Hoare:
  - A ideia é utilizar dois índices para percorrer o array em sentidos opostos.
  - O primeiro índice, i, parte da posição imediatamente à direita do pivô e avança em direção ao final do array..
  - O segundo índice, j, parte da última posição e recua em direção ao início do array.
  - O índice i avança enquanto encontrar elementos menores ou iguais ao pivô. Já o índice j recua enquanto encontrar elementos maiores que o pivô.
  - Quando ambos param, os elementos apontados por i e j estão do lado errado da partição e são trocados. O processo é repetido até que os índices se cruzem.

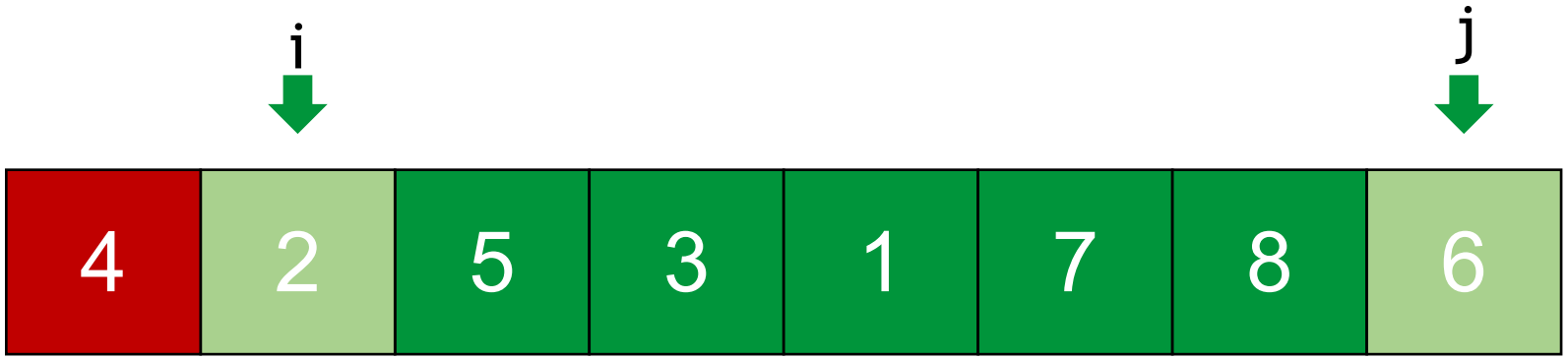
Quick sort

- Pivô = 4



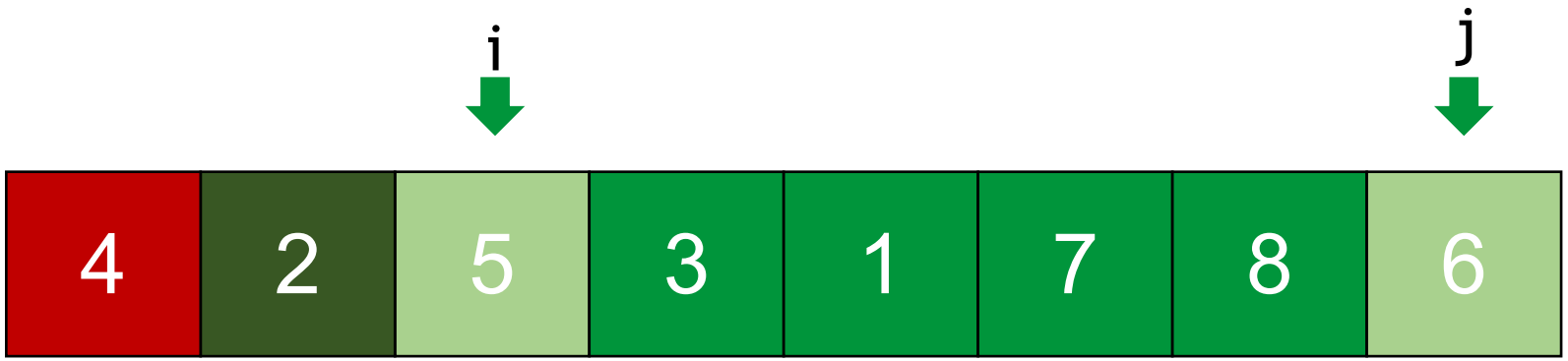
Quick sort

- Pivô = 4



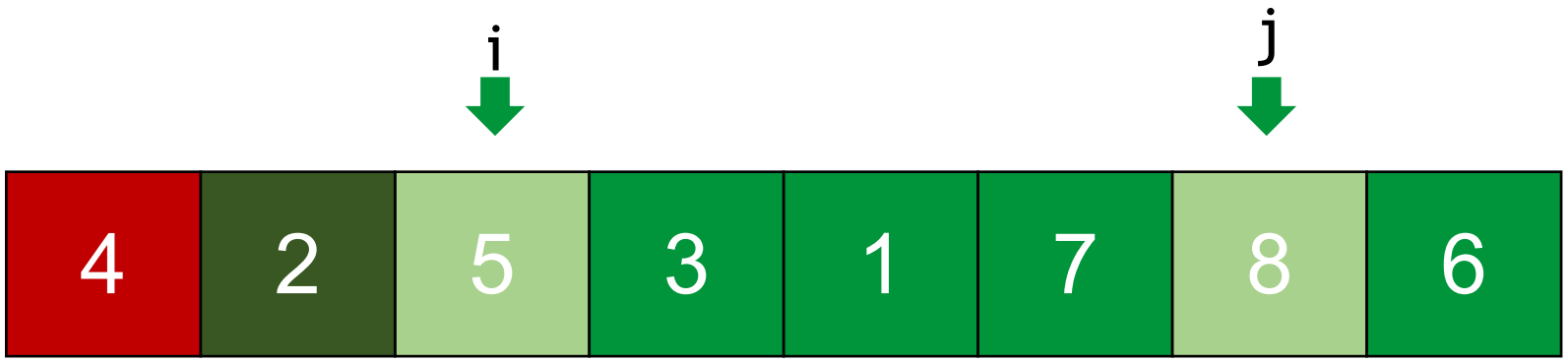
Quick sort

- Pivô = 4



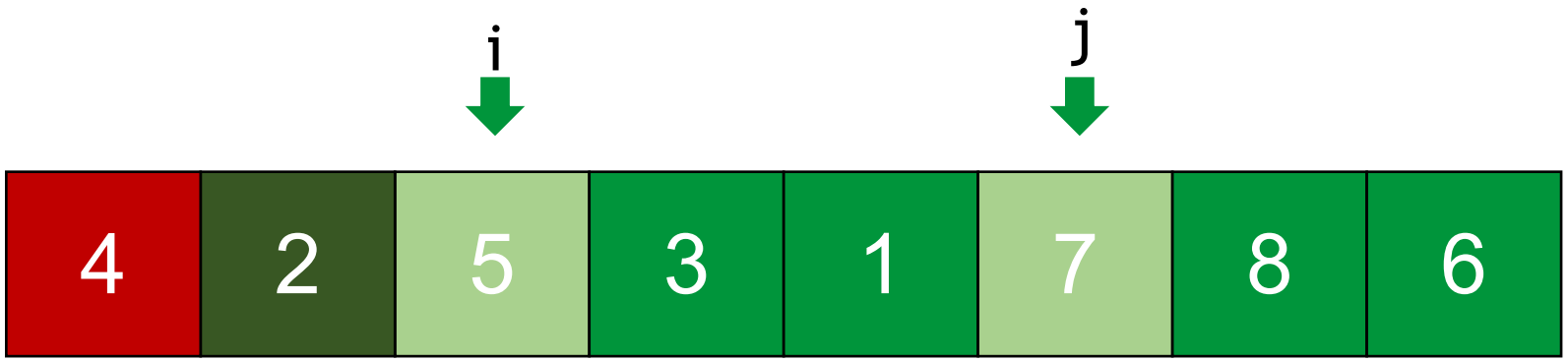
Quick sort

- Pivô = 4



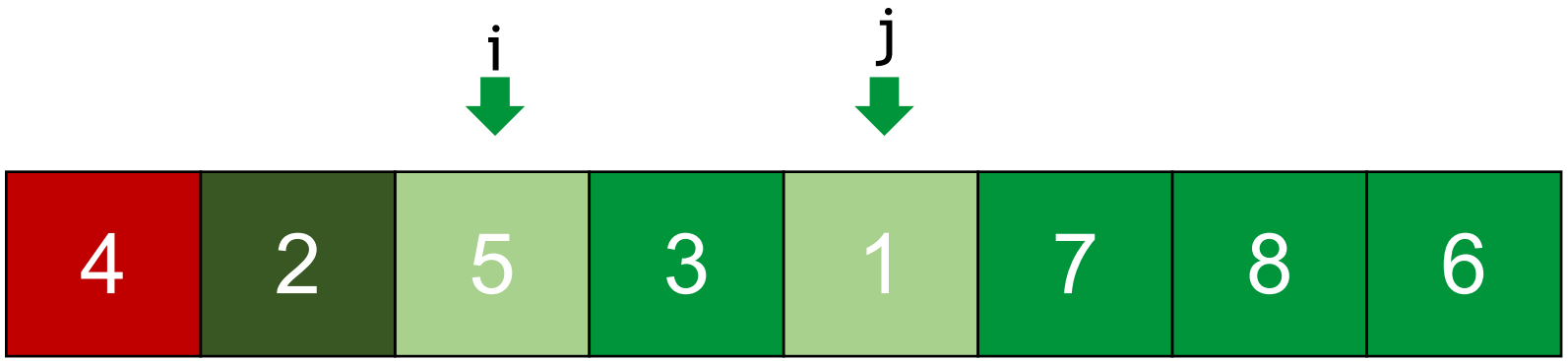
Quick sort

- Pivô = 4



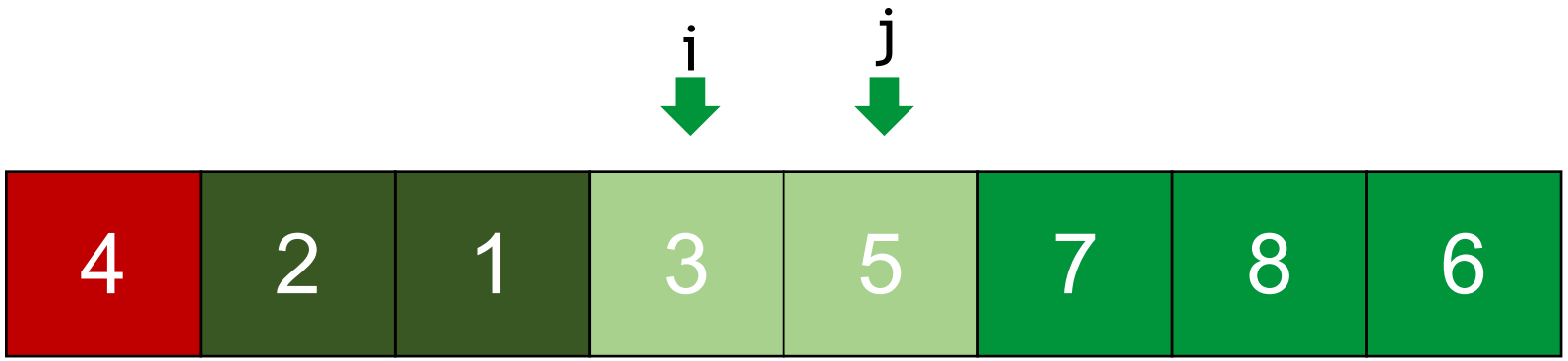
Quick sort

- Pivô = 4



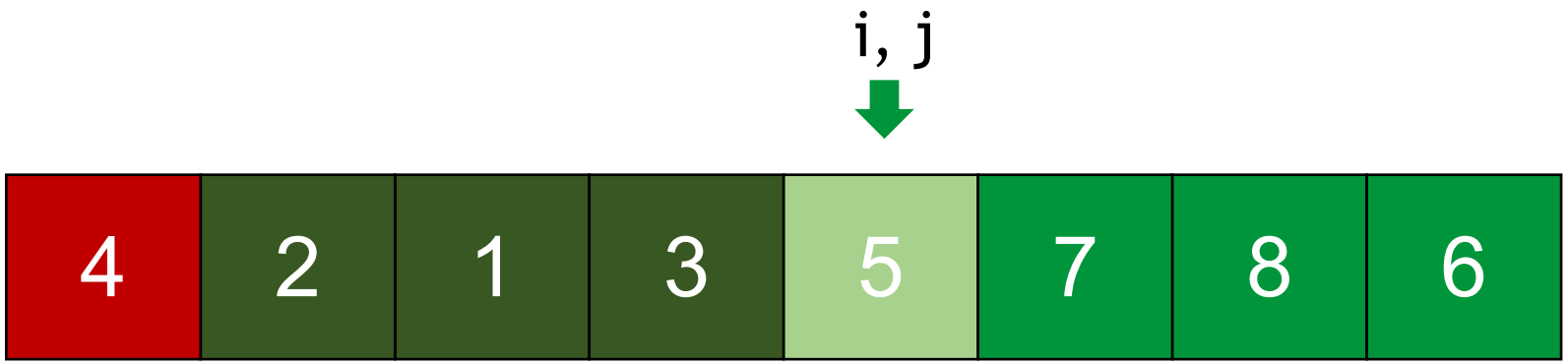
Quick sort

- Pivô = 4



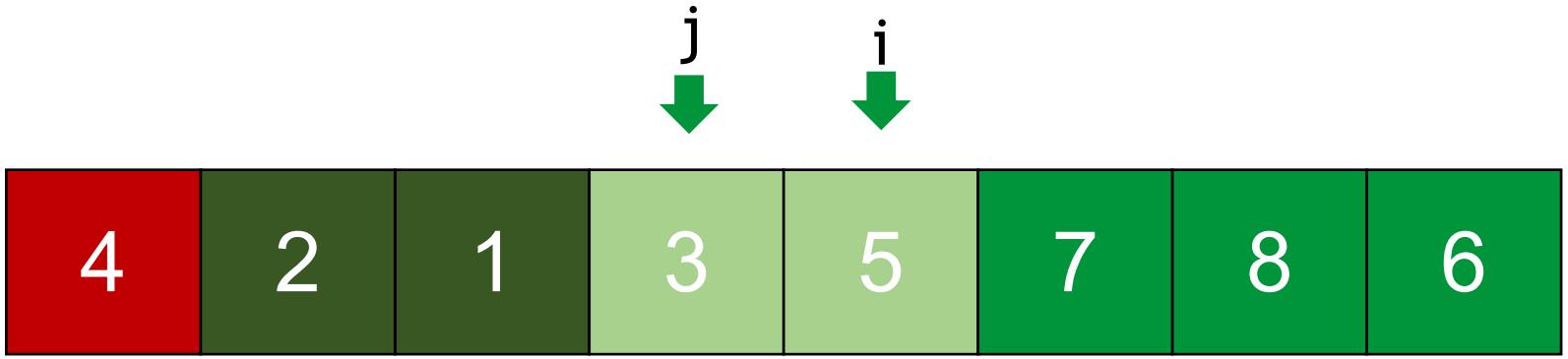
Quick sort

- Pivô = 4



Quick sort

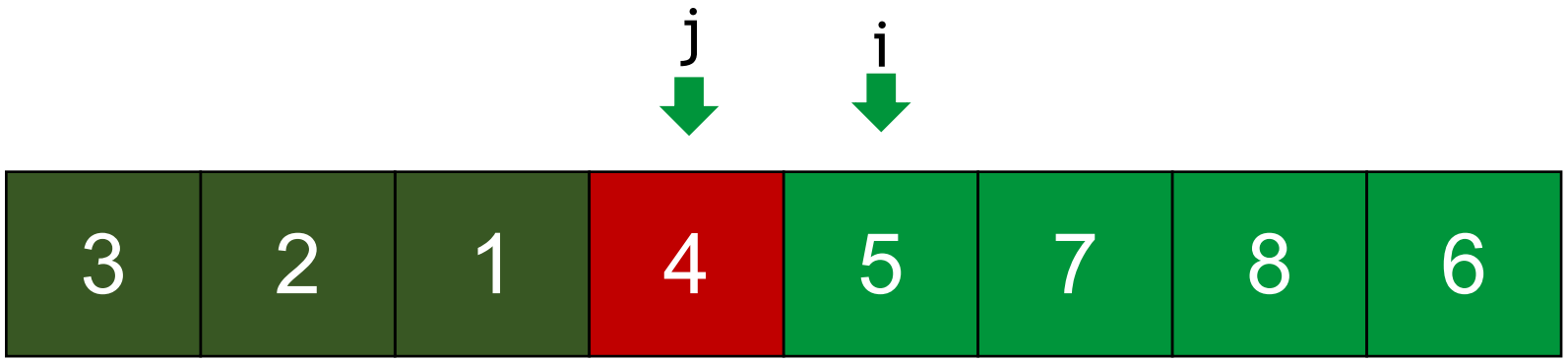
- Pivô = 4



Quick sort

INSTITUTO FEDERAL
Pernambuco

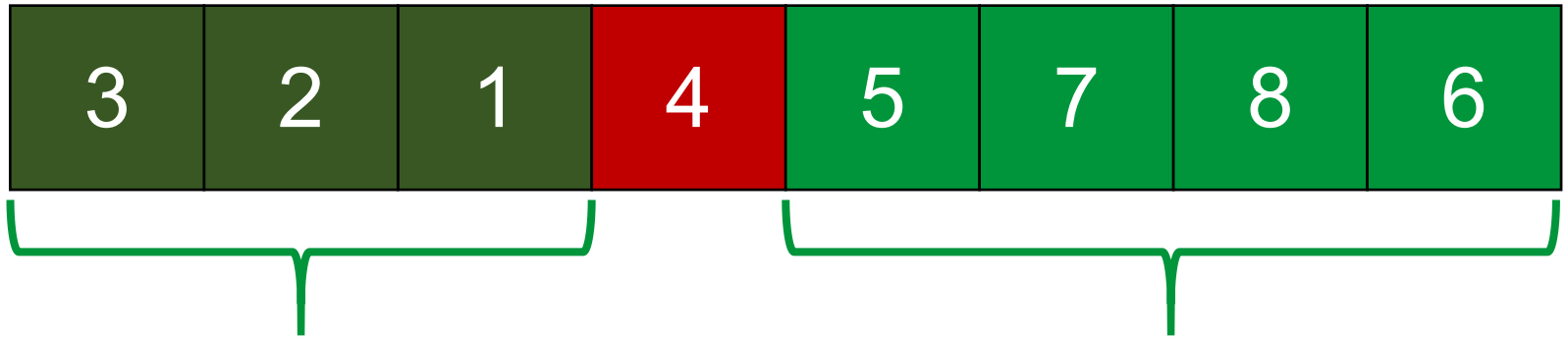
- Pivô = 4



Quick sort

INSTITUTO FEDERAL
Pernambuco

- Pivô = 4



- Aplica-se recursivamente o algoritmo às sublistas até que tenham tamanho 0 ou 1, quando estarão ordenadas.

```
quickSort(vetor, inicio, indicePivo - 1);
quickSort(vetor, indicePivo + 1, fim);
```

Quick sort

```
int particaoHoare(int vetor[], int inicio, int fim) {
    int i = inicio + 1;
    int j = fim;
    int pivo = vetor[inicio];

    while (i <= j) {
        while (i <= j && vetor[i] <= pivo)
            i++;
        while (i <= j && vetor[j] > pivo)
            j--;
        if (i < j)
            swap(vetor, i, j);
    }

    swap(vetor, inicio, j);
    return j;
}
```

INSTITUTO FEDERAL
Pernambuco

Quick sort

INSTITUTO FEDERAL
Pernambuco

```
void swap(int vetor[], int i, int j) {
    int temp = vetor[i];
    vetor[i] = vetor[j];
    vetor[j] = temp;
}
```

Quick sort

INSTITUTO FEDERAL
Pernambuco

- Particionamento Lomuto:
  - Vantagens
    - Mais simples de entender e implementar.
  - Desvantagens
    - Realiza mais trocas.
    - Pode apresentar desempenho inferior em vetores grandes.
    - Costuma ser menos eficiente quando há muitos elementos repetidos.

Quick sort

INSTITUTO FEDERAL
Pernambuco

- Particionamento Hoare:
  - Vantagens
    - Realiza menos trocas.
    - Costuma apresentar melhor desempenho para vetores grandes.
    - Hoare geralmente é mais rápido na prática, embora tenha a mesma complexidade teórica.
  - Desvantagens
    - Implementação mais complexa.
    - Mais sujeito a erros de implementação.
    - A lógica de movimentação dos índices é menos intuitiva.

Quick sort

INSTITUTO FEDERAL
Pernambuco

- Analisando a complexidade:
  - Para uma coleção com n elementos, se o pivô dividir a coleção em duas partes aproximadamente iguais a cada passo, serão realizadas cerca de $\log_2 n$ divisões.
  - Em cada nível da recursão, todos os elementos precisam ser comparados com o pivô durante o particionamento, totalizando aproximadamente n comparações.
  - Portanto, a complexidade do algoritmo é O(n log n).
  - Além disso, diferentemente do Merge Sort, o Quick Sort não necessita de memória auxiliar para intercalar os elementos.

Quick sort

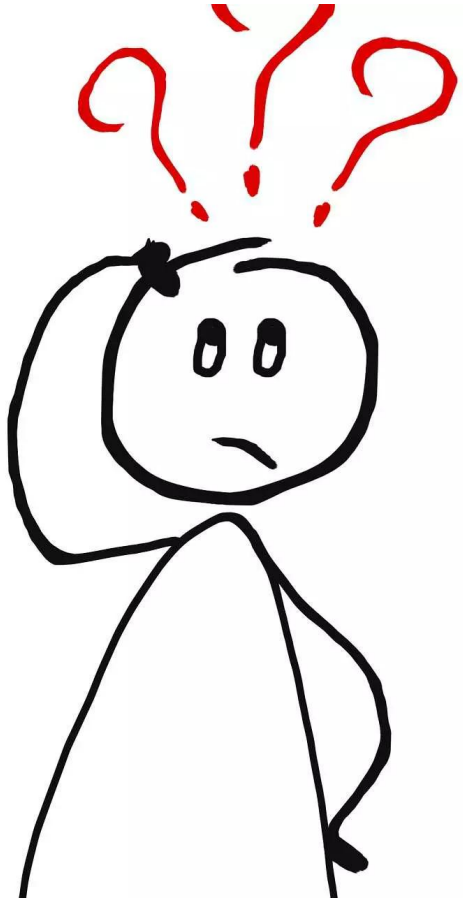
INSTITUTO FEDERAL
Pernambuco

- Analisando a complexidade:
  - No entanto, as partições nem sempre são equilibradas. Dependendo da escolha do pivô, a coleção pode ser dividida de forma muito desigual.
  - No pior caso, uma coleção com n elementos pode ser particionada em duas partes de tamanhos 0 e n – 1.
  - Em seguida, a subcoleção de tamanho n – 1 pode ser dividida em 0 e n – 2, e assim sucessivamente.
  - O resultado é uma complexidade de $O(n^2)$, além do custo associado às chamadas recursivas.

Quick sort

INSTITUTO FEDERAL
Pernambuco

- Como reduzir a chance do pior caso?



Quick sort

INSTITUTO FEDERAL
Pernambuco

- Como reduzir a chance do pior caso?
  - A escolha do pivô tem grande influência no desempenho do Quick Sort.
  - Uma estratégia simples é selecionar o pivô aleatoriamente, reduzindo a probabilidade de gerar partições muito desbalanceadas.
  - Se o pivô for escolhido aleatoriamente, a probabilidade de selecionar o pior pivô em uma coleção de tamanho n é 1/n.
  - Para que o pior caso ocorra em todas as chamadas recursivas, seria necessário escolher sucessivamente o pior pivô em cada partição:

$$\frac{1}{n} \times \frac{1}{n-1} \times \frac{1}{n-2} \times \cdots \times \frac{1}{2}$$

  - Como essa probabilidade é extremamente pequena, o Quick Sort apresenta desempenho próximo de O(n log n) na maioria das aplicações práticas.

Quick sort

INSTITUTO FEDERAL
Pernambuco

- Outras estratégias para escolha do pivô:
  - Escolher o elemento central da coleção.
  - Escolher um pivô aleatório.
  - Utilizar a mediana de três (primeiro, meio e último elementos).
  - Utilizar amostras maiores para estimar uma mediana mais representativa.
- Essas estratégias ajudam a produzir partições mais equilibradas e reduzem a chance de ocorrência do pior caso.

Quick sort vs Merge sort

INSTITUTO FEDERAL
Pernambuco

- Merge Sort
  - Sempre executa em O(n log n).
  - É um algoritmo estável.
  - Requer memória auxiliar para realizar a intercalação.
  - Não depende da ordem inicial dos elementos.
- Quick Sort
  - Geralmente apresenta melhor desempenho na prática.
  - Ordena o vetor in-place, utilizando pouca memória adicional (recursão).
  - O desempenho depende da qualidade das partições.
  - Pode atingir $O(n^2)$ no pior caso, embora isso seja raro quando o pivô é bem escolhido.

Exercício

INSTITUTO FEDERAL
Pernambuco

- Os exemplos apresentados em sala utilizam o primeiro elemento da subcoleção como pivô. Modifique as implementações do Quick Sort para que:

a) O pivô seja o elemento central do vetor.

b) O pivô seja escolhido aleatoriamente entre as posições inicio e fim.

Dúvidas

INSTITUTO FEDERAL
Pernambuco

ALGORITMOS E ESTRUTURA DE DADOS

Curso de Engenharia de Software
Lucas Sampaio Leite