Classificação e Pesquisa de Dados

Cristiano Santos

cristiano.santos@amf.edu.br

QuickSort

Algoritmo Quick Sort

- Proposto por Hoare em 1960 e publicado em 1962.
- É um dos algoritmos mais rápidos de ordenação considerando uma ampla variedade de situações.
- Provavelmente é o mais utilizado.

Algoritmo Quick Sort

- A idéia básica é **dividir o problema de ordenar um conjunto com n itens em dois sub-problemas** menores.
- Os **problemas menores são ordenados independentemente**.
- Os **resultados são combinados para produzir a solução final**.

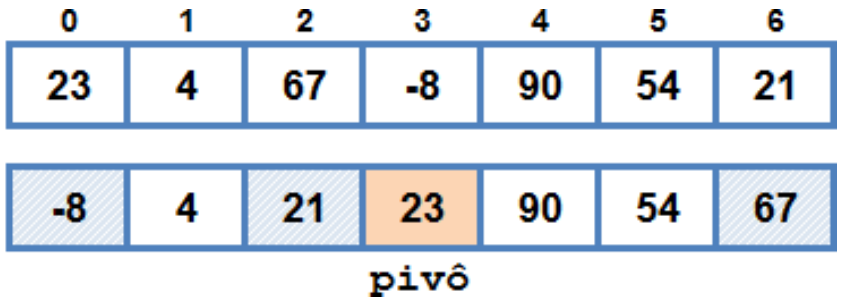
Algoritmo Quick Sort

- **Também conhecido como ordenação por partição**
  - É outro algoritmo recursivo que usa a **idéia de *dividir para conquistar*** para ordenar os dados
  - Se baseia no problema da separação
    - Em inglês, ***partition subproblem***

Algoritmo Quick Sort

Problema da separação

- Consiste em rearranjar o array usando um valor como **pivô**
  - **Valores menores** do que o **pivô** ficam a **esquerda**
  - **Valores maiores** do que o **pivô** ficam a **direita**



Algoritmo Quick Sort

- Algoritmo para o particionamento:

Algoritmo Quick Sort

- Algoritmo para o particionamento:
  1. **Escolha** arbitrariamente um **pivô x**.

Algoritmo Quick Sort

- Algoritmo para o particionamento:
  1. **Escolha** arbitrariamente um **pivô x**.
  2. **Percorra** o vetor a **partir da esquerda** até **que v[i] ≥ x**.

Algoritmo Quick Sort

- Algoritmo para o particionamento:
  1. **Escolha** arbitrariamente um **pivô x**.
  2. **Percorra** o vetor a **partir da esquerda** até **que v[i] ≥ x**.
  3. **Percorra** o vetor a **partir da direita** até que **v[j] ≤ x.**

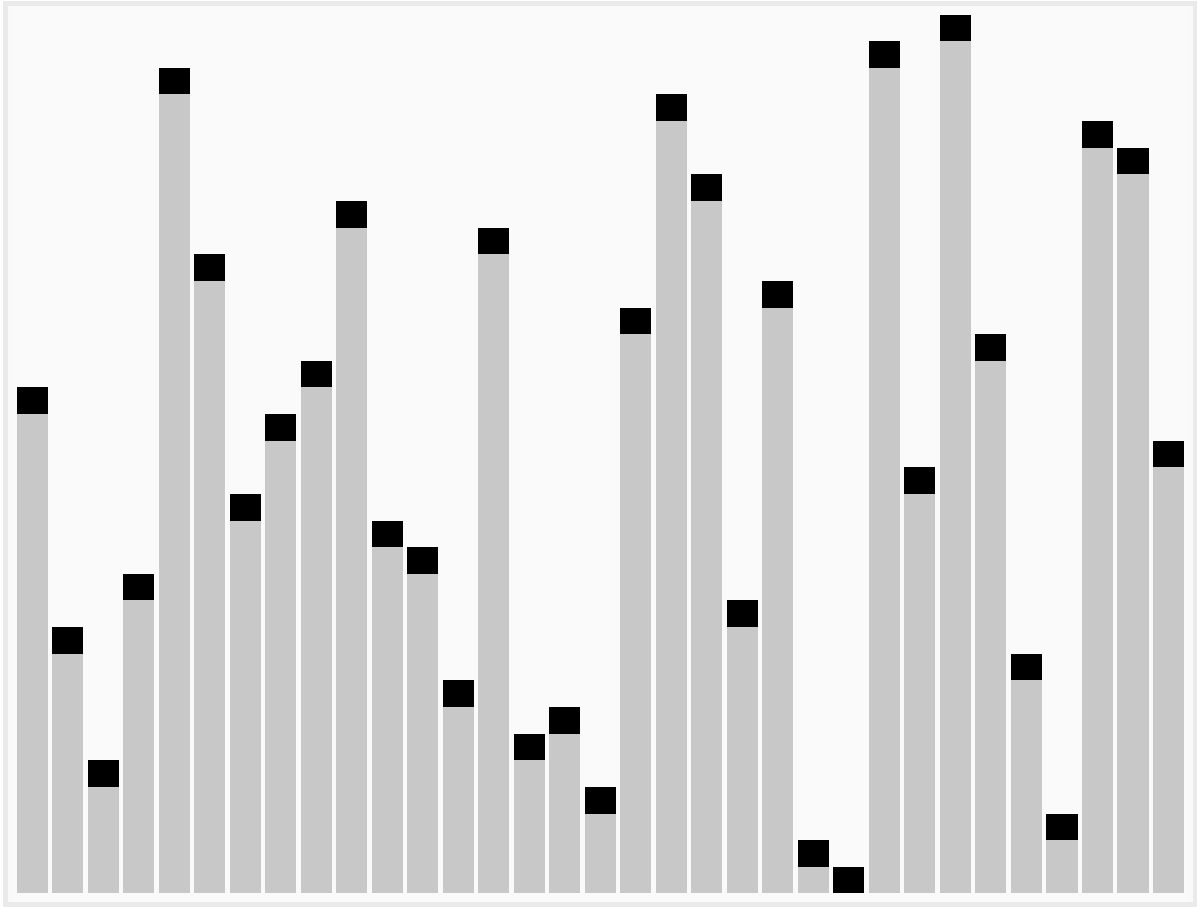
Algoritmo Quick Sort

- Algoritmo para o particionamento:
  1. **Escolha** arbitrariamente um **pivô x**.
  2. **Percorra** o vetor a **partir da esquerda** até **que v[i] ≥ x**.
  3. **Percorra** o vetor a **partir da direita** até que **v[j] ≤ x.**
  4. **Troque v[i] com v[j].**

Algoritmo Quick Sort

- Algoritmo para o particionamento:
  1. **Escolha** arbitrariamente um **pivô x**.
  2. **Percorra** o vetor a **partir da esquerda** até **que v[i] ≥ x**.
  3. **Percorra** o vetor a **partir da direita** até que **v[j] ≤ x.**
  4. **Troque v[i] com v[j].**
  5. **Continue** este processo **até os** apontadores i e j se cruzarem.

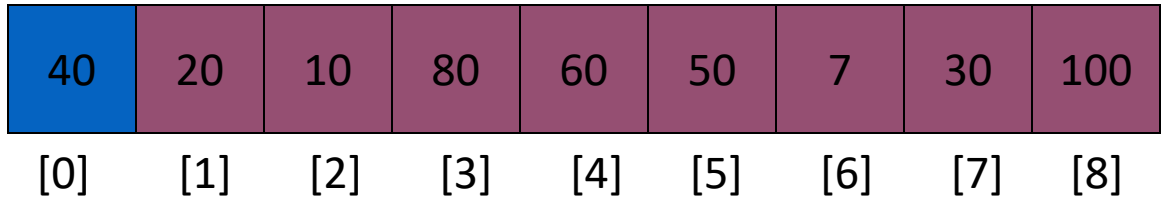
Algoritmo Quick Sort



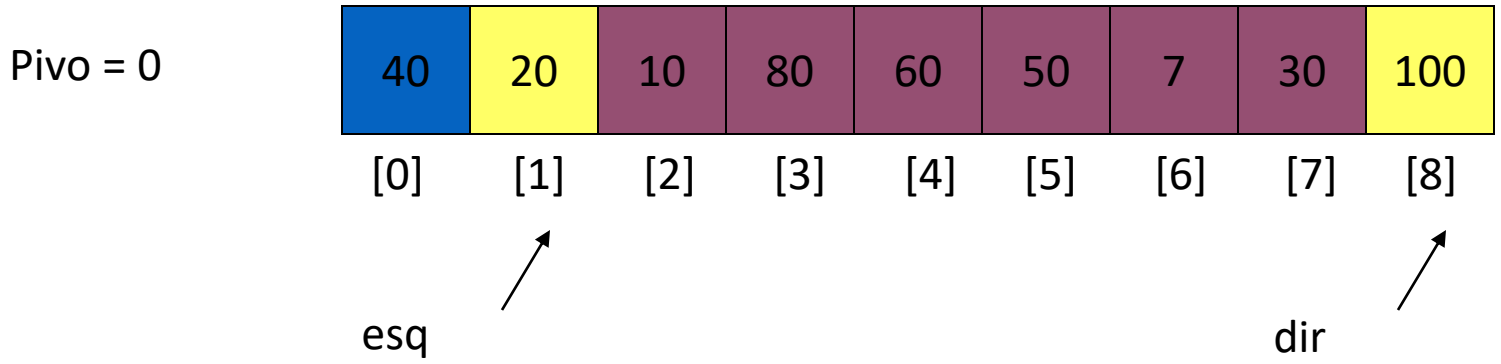
Escolha do elemento pivô

Existem várias maneiras de escolher o elemento pivô.

Neste exemplo, usaremos o primeiro elemento:

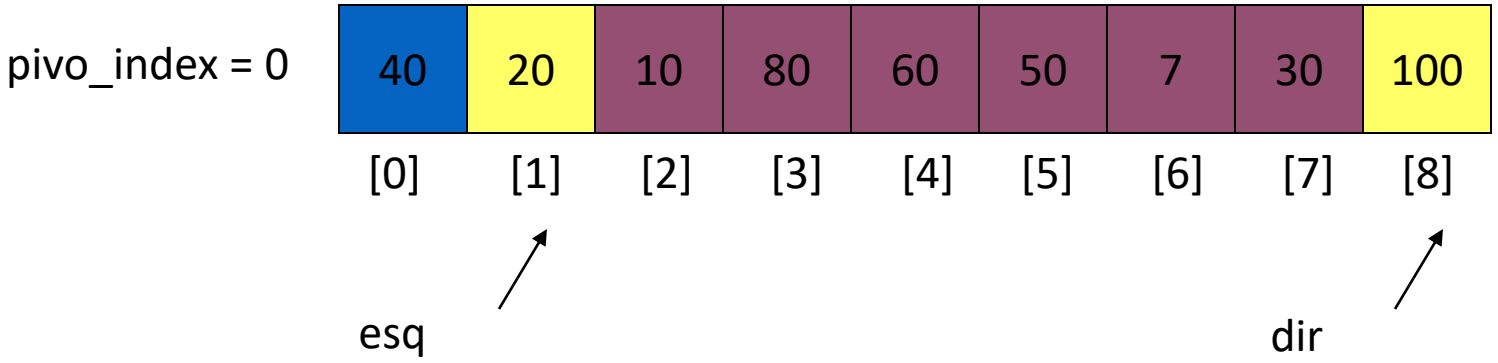


Partição: Funcionamento



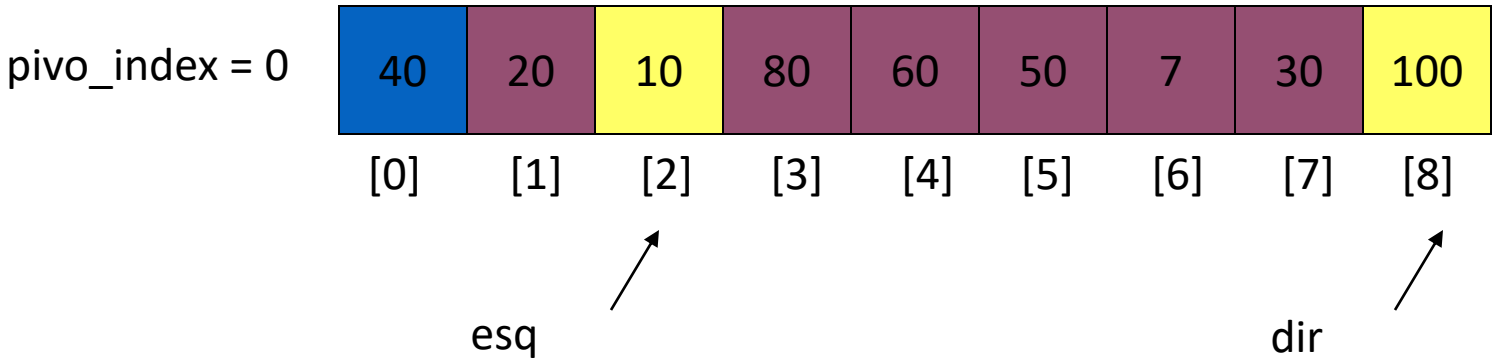
Partição: Funcionamento

1. While **lista[esq]** <= **lista[pivo]**
   ++esq



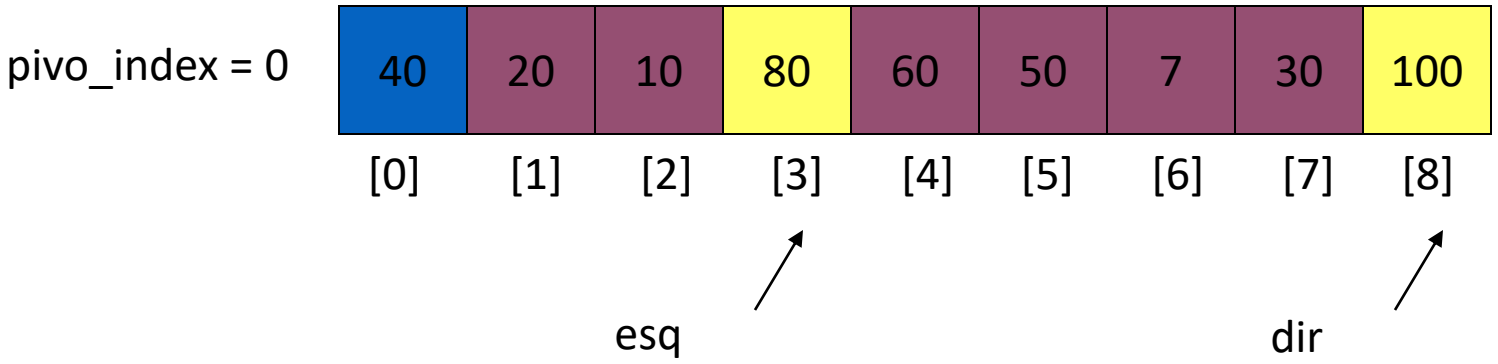
Partição: Funcionamento

1. While lista[esq] <= lista[pivo]
   ++esq



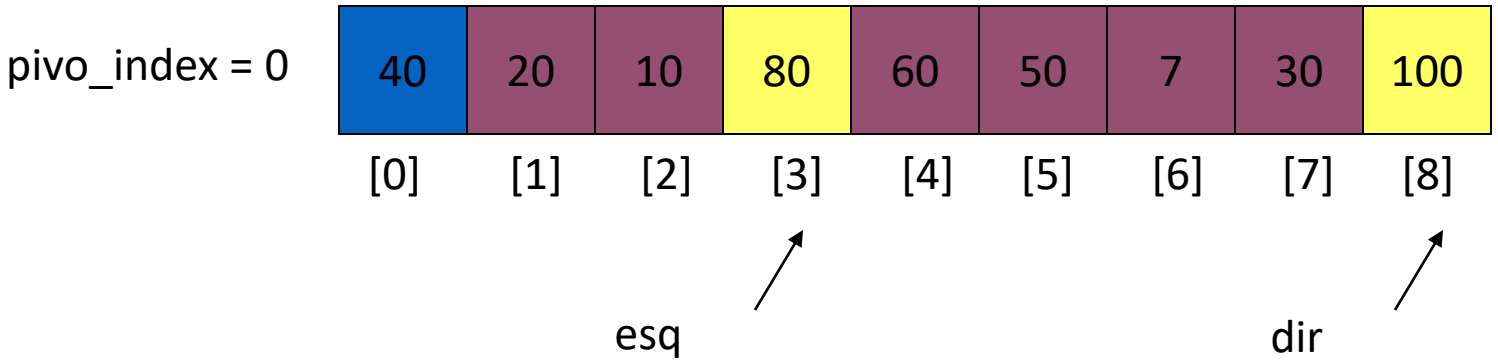
Partição: Funcionamento

1. While **lista[esq] <= lista[pivo]**
   ++esq



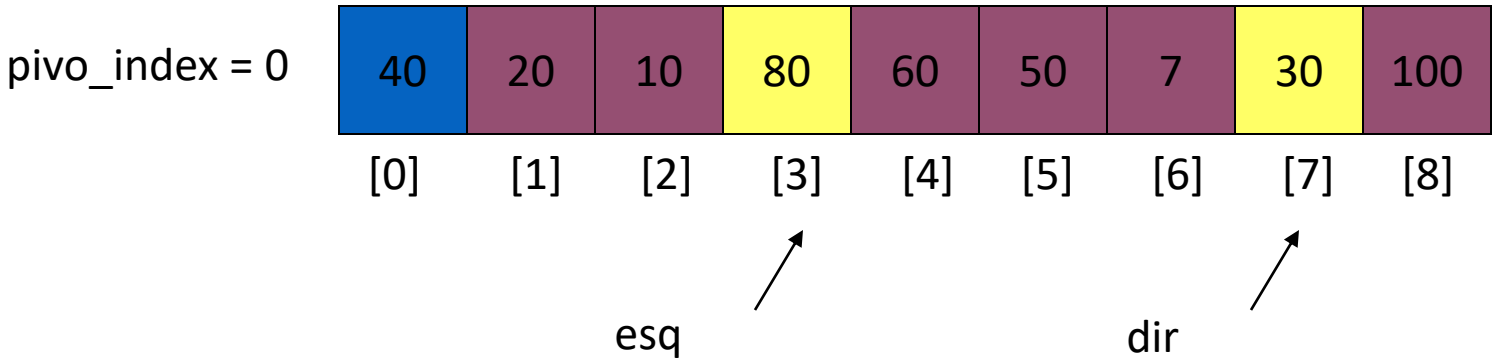
Partição: Funcionamento

1. While lista[esq] <= lista[pivo]
   ++esq
2. While lista[dir] > lista[pivo]
   --dir



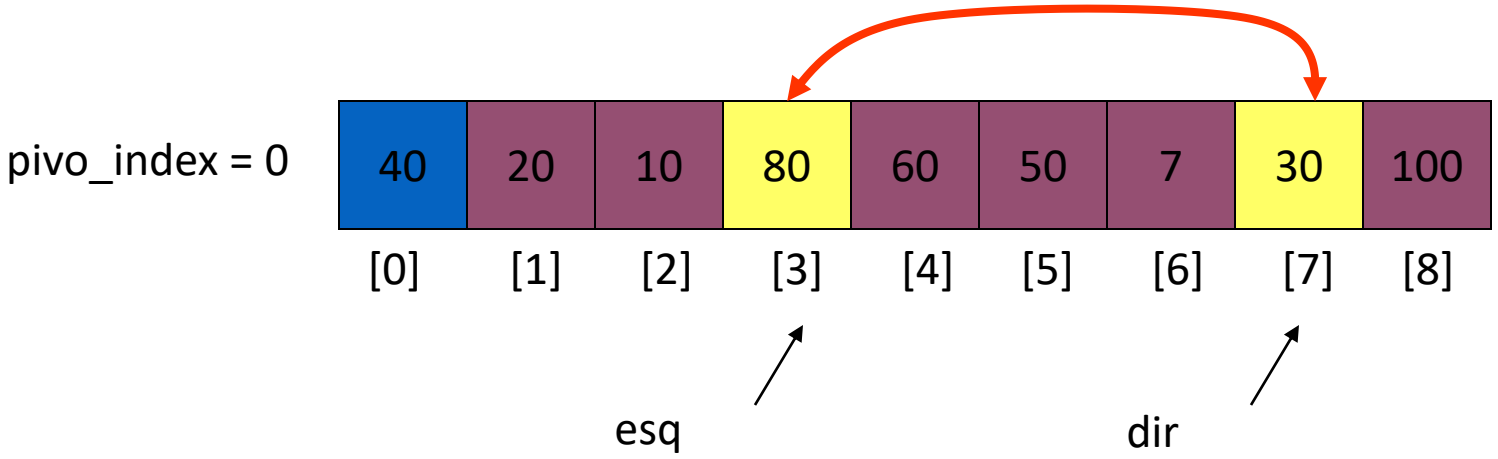
Partição: Funcionamento

1. While lista[esq] <= lista[pivo]
   ++esq
2. While **lista[dir] > lista[pivo]**
   --dir



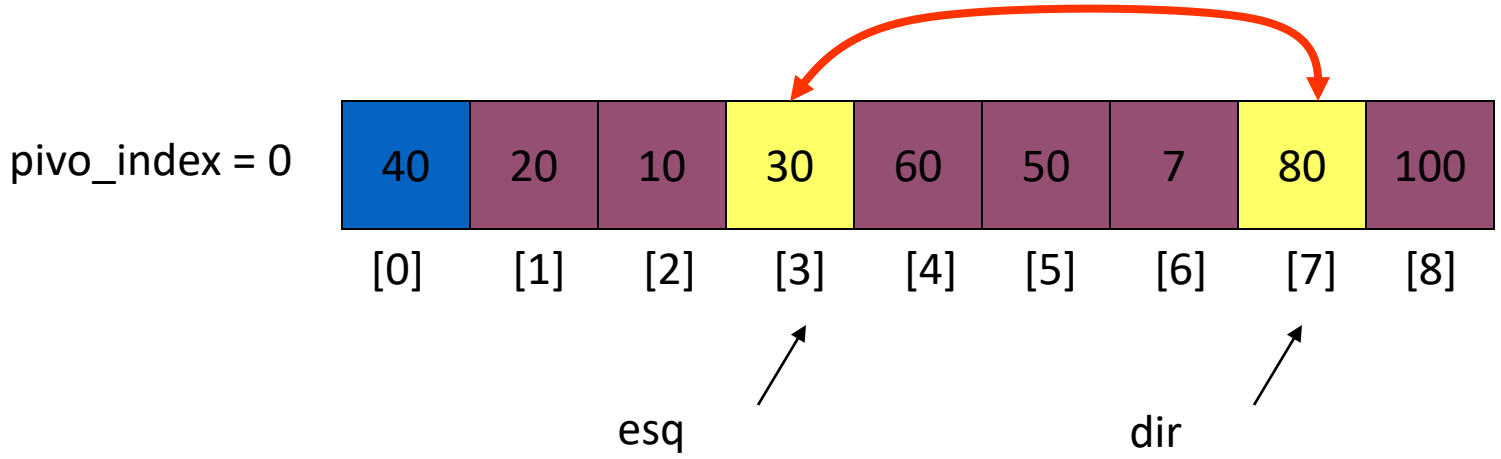
Partição: Funcionamento

1. While lista[esq] <= lista[pivo]
   ++esq
2. While lista[dir] > lista[pivo]
   --dir
3. If esq < dir
   troca lista[esq] and lista[dir]



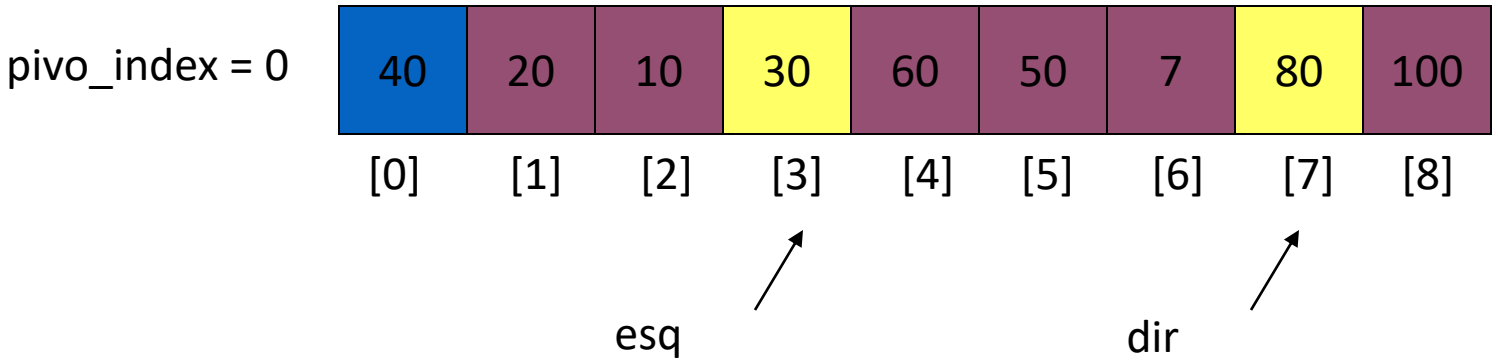
Partição: Funcionamento

1. While lista[esq] <= lista[pivo]
   ++esq
2. While lista[dir] > lista[pivo]
   --dir
3. If esq < dir
   troca lista[esq] and lista[dir]



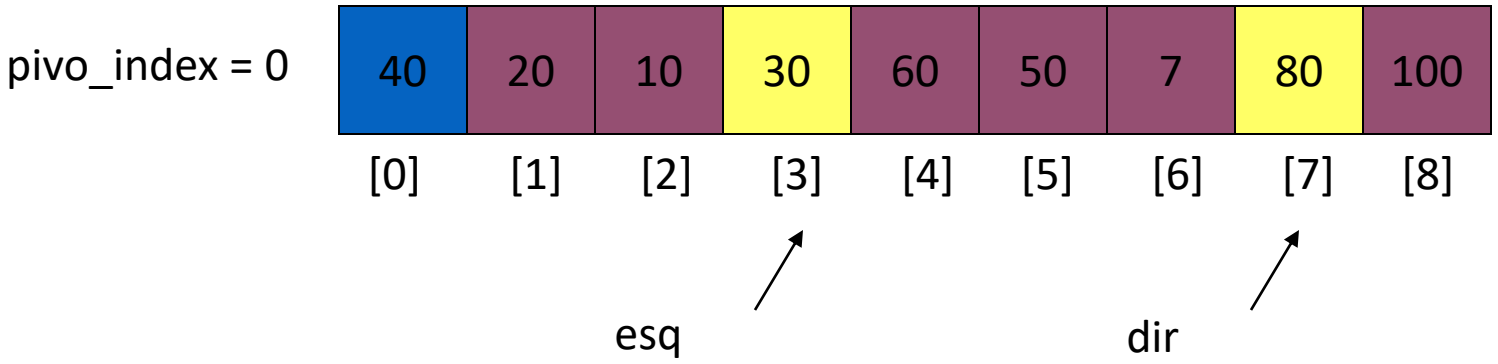
Partição: Funcionamento

1. While lista[esq] <= lista[pivo]
   ++esq
2. While lista[dir] > lista[pivo]
   --dir
3. If esq < dir
   troca lista[esq] and lista[dir]
4. While dir > esq, **go to** 1.



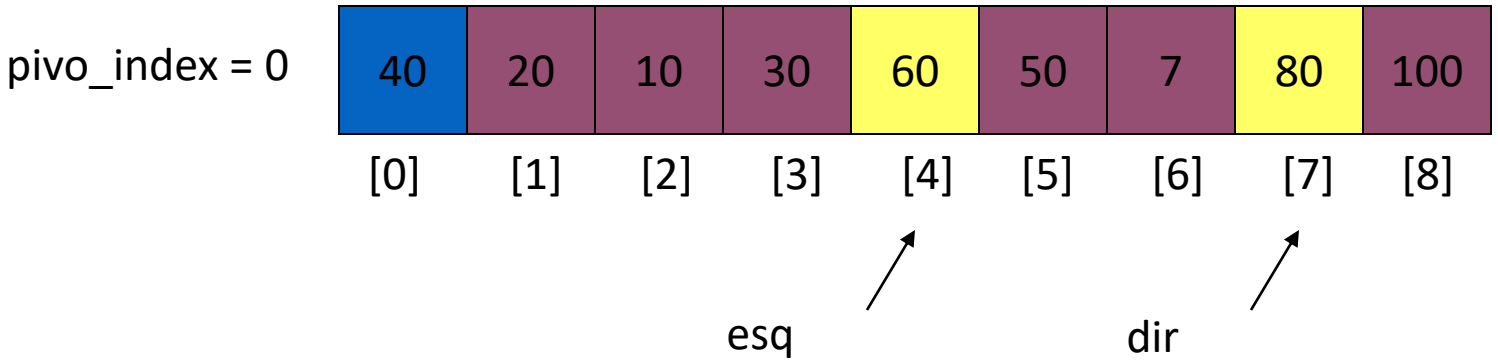
Partição: Funcionamento

1. While lista[esq] <= lista[pivo]
   ++esq
2. While lista[dir] > lista[pivo]
   --dir
3. If esq < dir
   troca lista[esq] and lista[dir]
4. While dir > esq, **go to** 1.



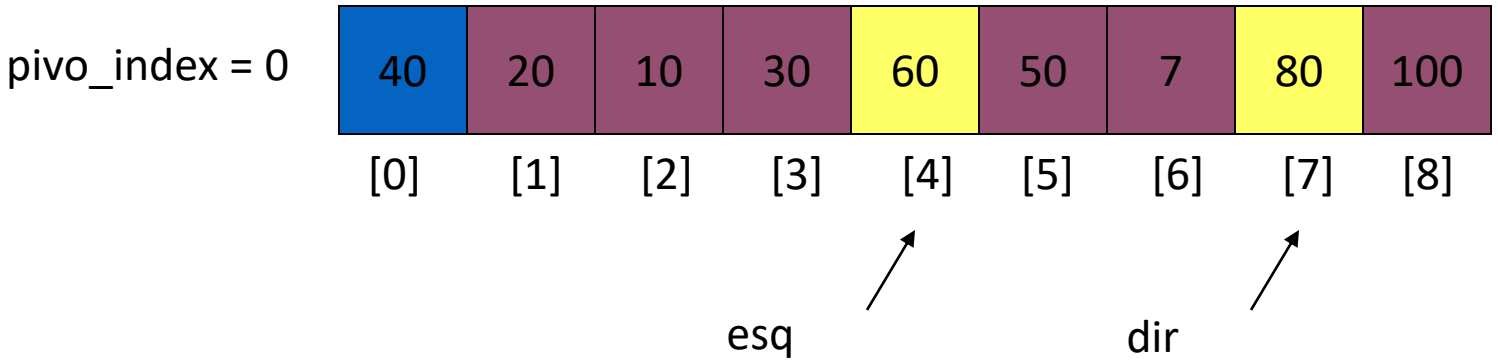
Partição: Funcionamento

1. While **lista[esq] <= lista[pivo]**
   ++esq
2. While lista[dir] > lista[pivo]
   --dir
3. If esq < dir
   troca lista[esq] and lista[dir]
4. While dir > esq, **go to** 1.



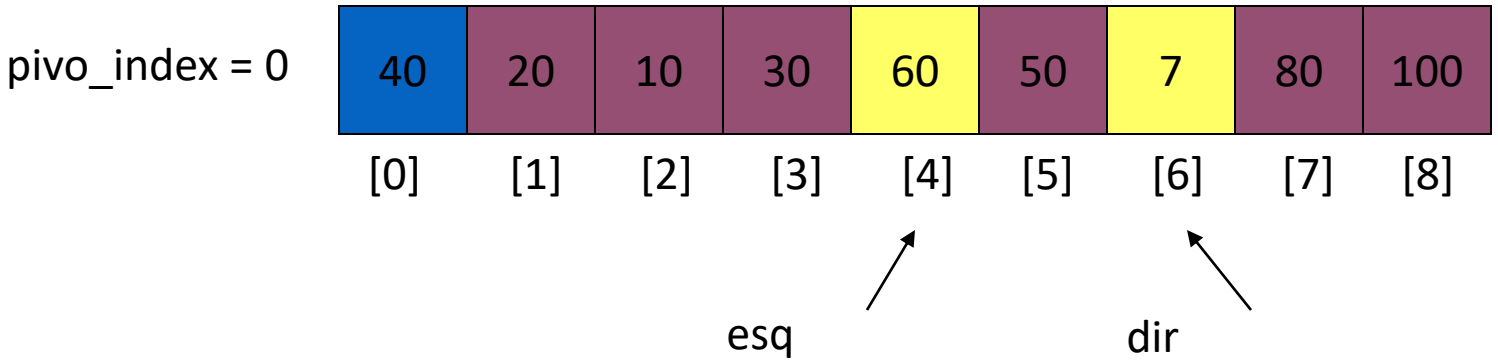
Partição: Funcionamento

1. While lista[esq] <= lista[pivo]
   ++esq
2. While lista[dir] > lista[pivo]
   --dir
3. If esq < dir
   troca lista[esq] and lista[dir]
4. While dir > esq, **go to** 1.



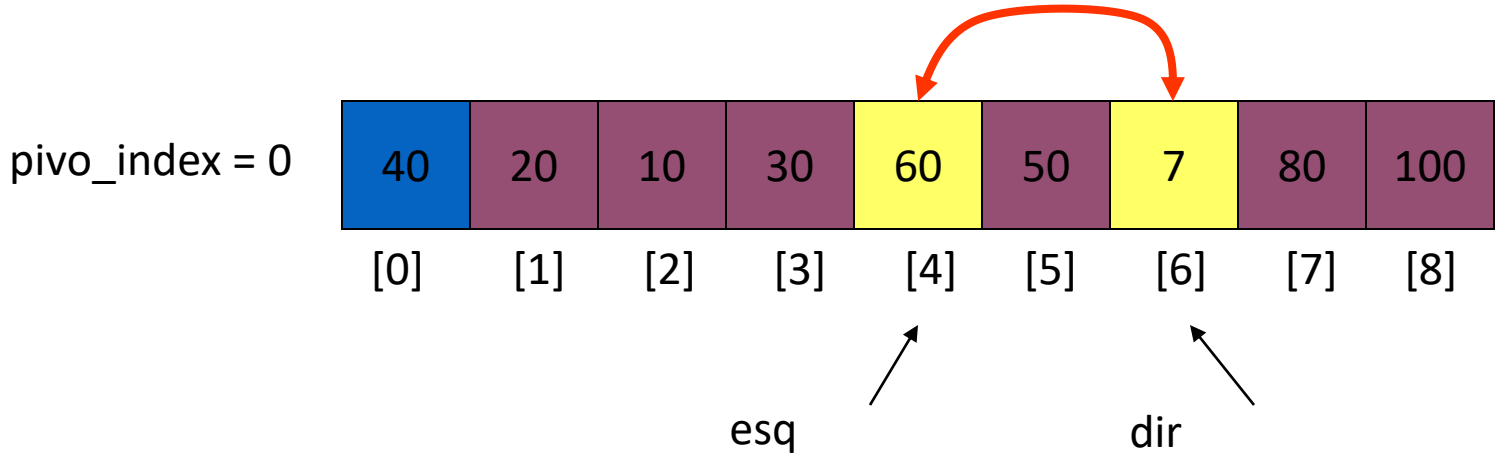
Partição: Funcionamento

1. While lista[esq] <= lista[pivo]
   ++esq
2. While **lista[dir] > lista[pivo]**
   --dir
3. If esq < dir
   troca lista[esq] and lista[dir]
4. While dir > esq, **go to** 1.



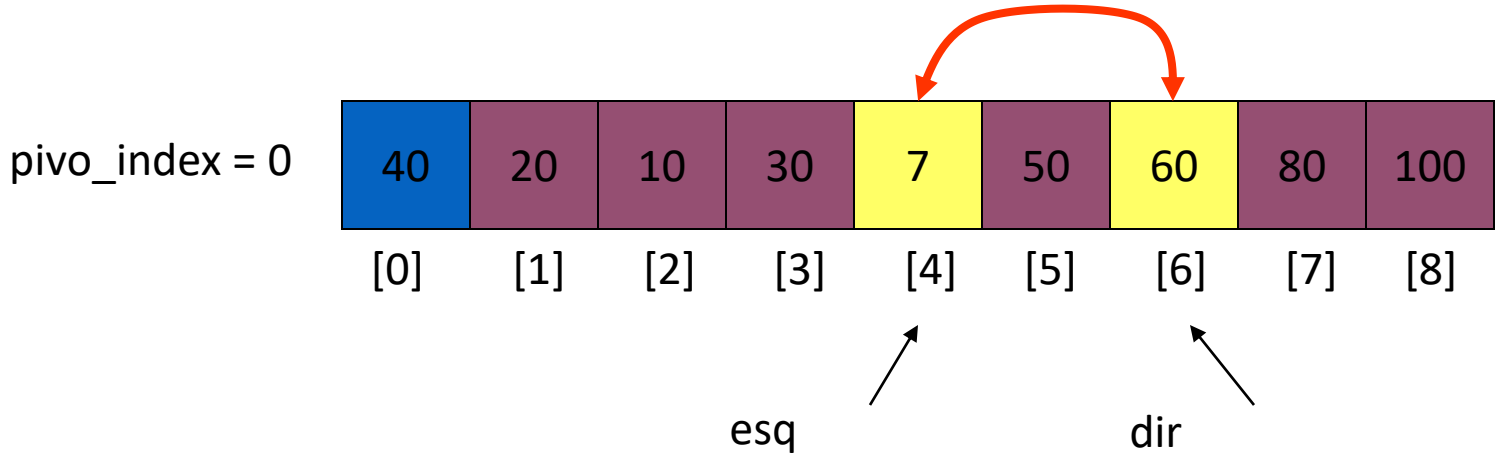
Partição: Funcionamento

1. While lista[esq] <= lista[pivo]
   ++esq
2. While lista[dir] > lista[pivo]
   --dir
3. If esq < dir
   troca lista[esq] and lista[dir]
4. While dir > esq, **go to** 1.



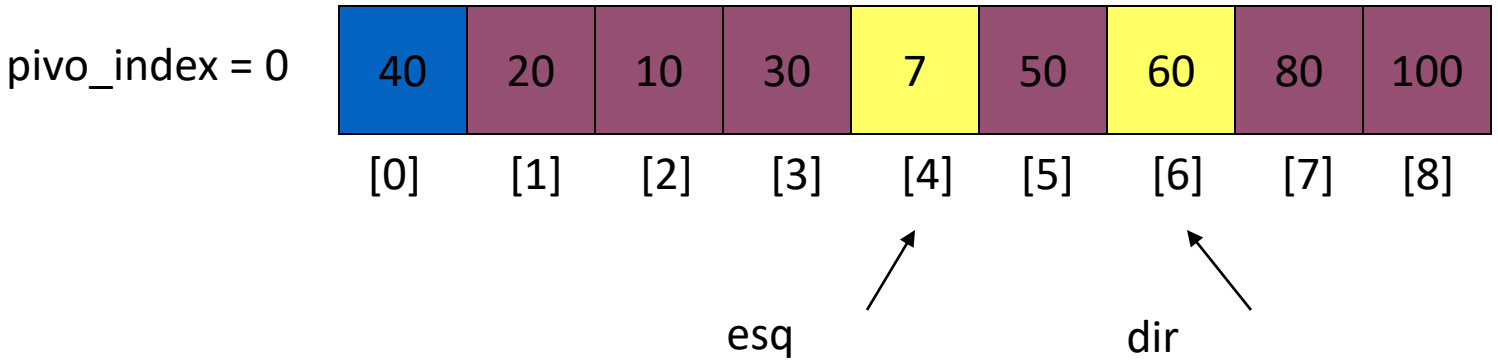
Partição: Funcionamento

1. While lista[esq] <= lista[pivo]
   ++esq
2. While lista[dir] > lista[pivo]
   --dir
3. If esq < dir
   troca lista[esq] and lista[dir]
4. While dir > esq, **go to** 1.



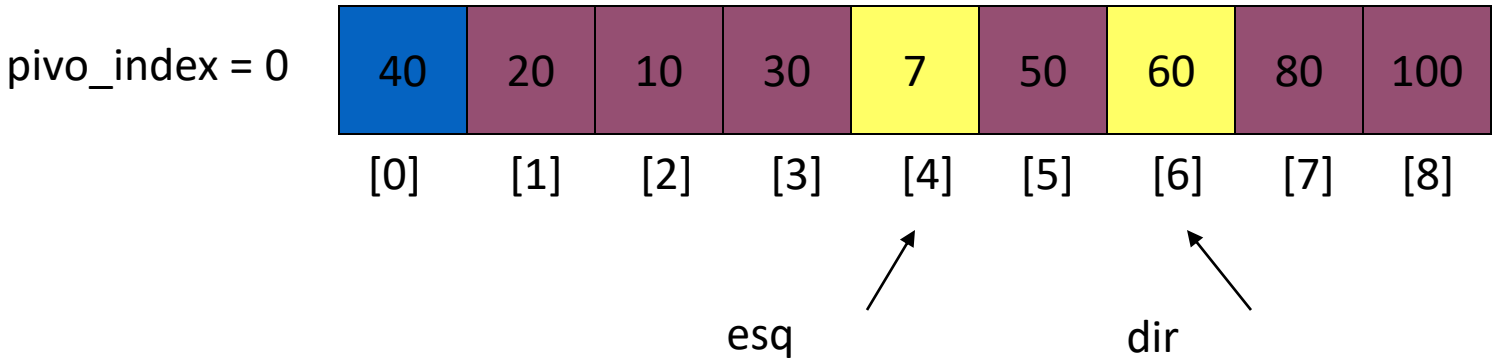
Partição: Funcionamento

1. While lista[esq] <= lista[pivo]
   ++esq
2. While lista[dir] > lista[pivo]
   --dir
3. If esq < dir
   troca lista[esq] and lista[dir]
4. While dir > esq, **go to** 1.



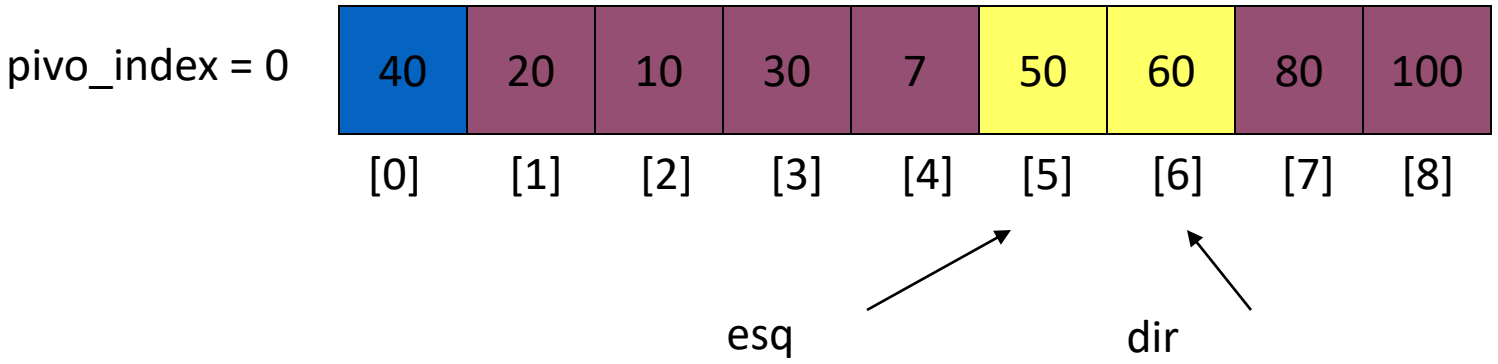
Partição: Funcionamento

1. While lista[esq] <= lista[pivo]
   ++esq
2. While lista[dir] > lista[pivo]
   --dir
3. If esq < dir
   troca lista[esq] and lista[dir]
4. While dir > esq, **go to** 1.



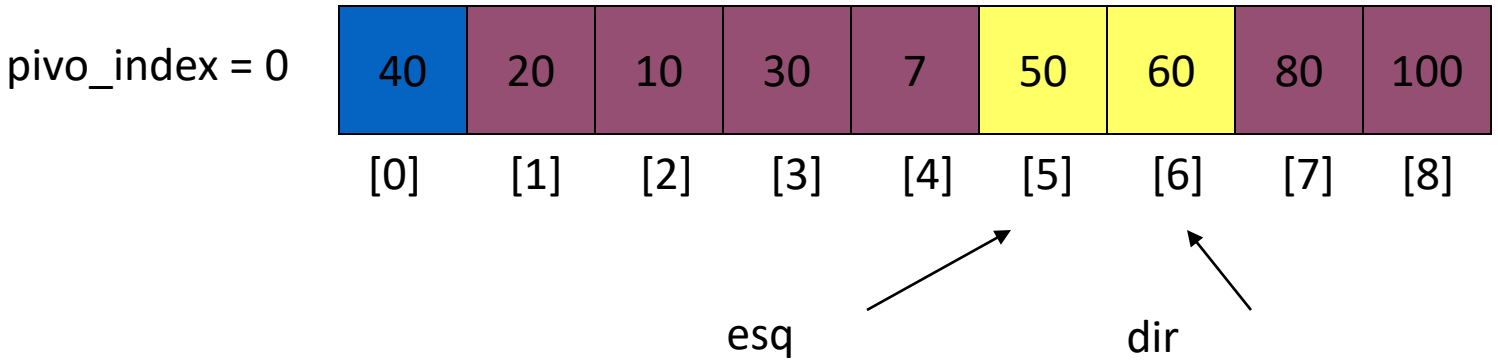
Partição: Funcionamento

1. While **lista[esq] <= lista[pivo]**
   ++esq
2. While lista[dir] > lista[pivo]
   --dir
3. If esq < dir
   troca lista[esq] and lista[dir]
4. While dir > esq, **go to** 1.



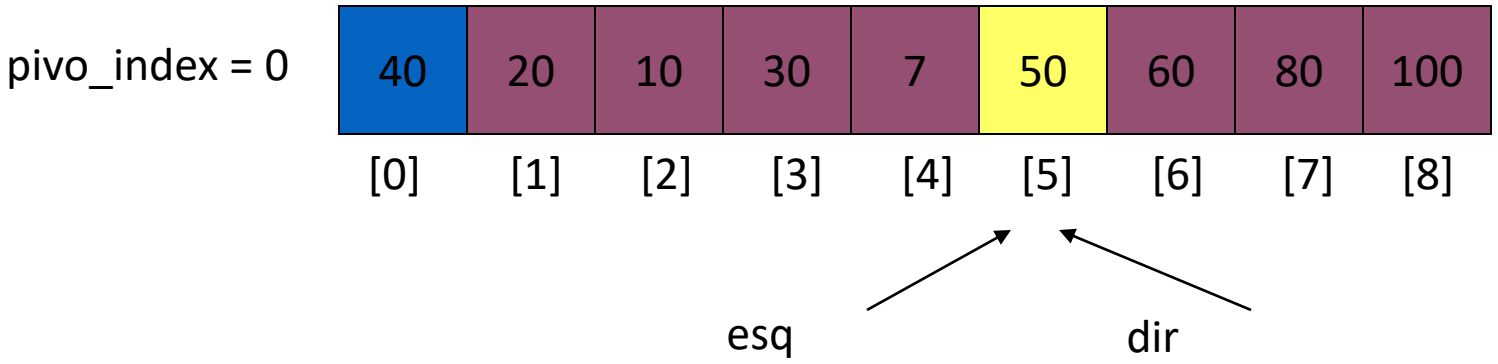
Partição: Funcionamento

1. While lista[esq] <= lista[pivo]
   ++esq
2. While lista[dir] > lista[pivo]
   --dir
3. If esq < dir
   troca lista[esq] and lista[dir]
4. While dir > esq, **go to** 1.



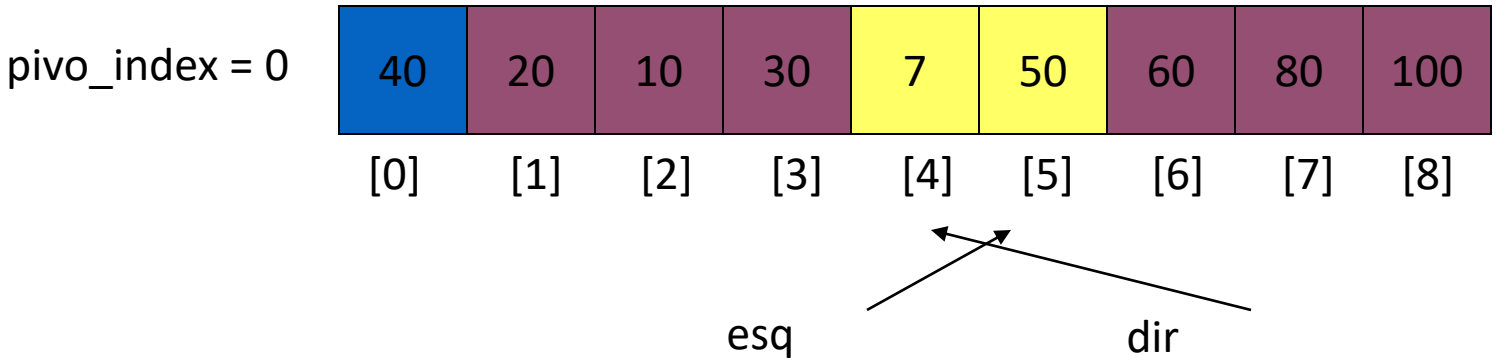
Partição: Funcionamento

1. While lista[esq] <= lista[pivo]
   ++esq
2. While lista[dir] > lista[pivo]
   --dir
3. If esq < dir
   troca lista[esq] and lista[dir]
4. While dir > esq, **go to** 1.



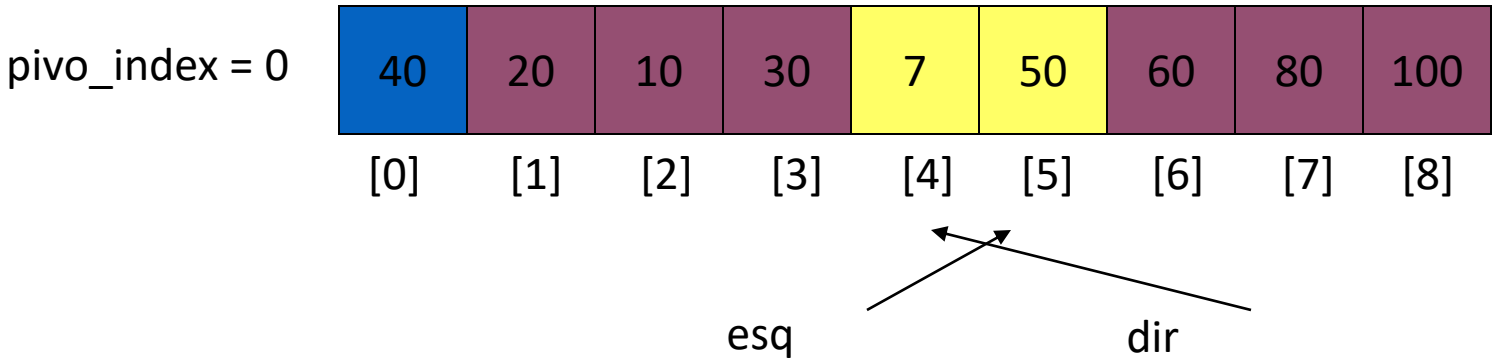
Partição: Funcionamento

1. While lista[esq] <= lista[pivo]
   ++esq
2. While lista[dir] > lista[pivo]
   --dir
3. If esq < dir
   troca lista[esq] and lista[dir]
4. While dir > esq, **go to** 1.



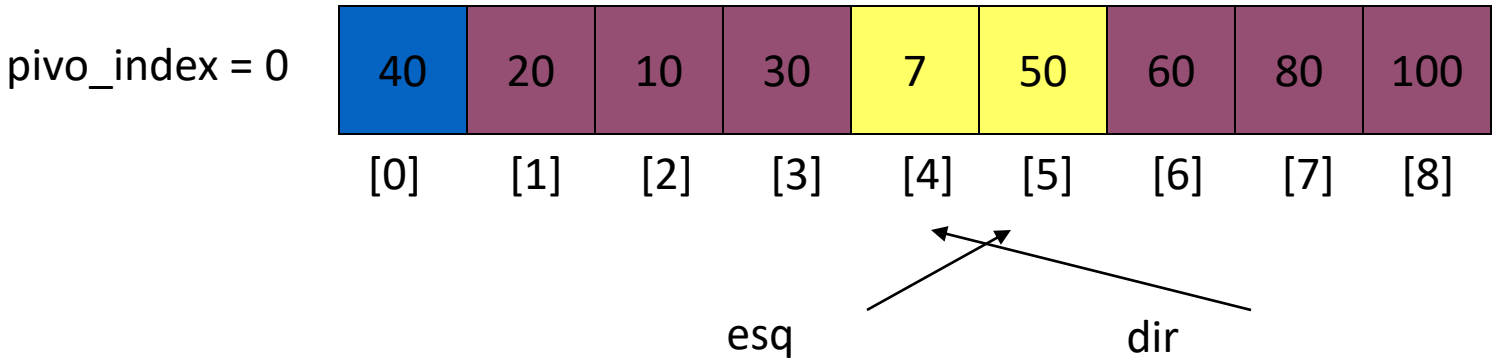
Partição: Funcionamento

1. While lista[esq] <= lista[pivo]
   ++esq
2. While lista[dir] > lista[pivo]
   --dir
3. If esq < dir
   troca lista[esq] and lista[dir]
4. While dir > esq, **go to** 1.



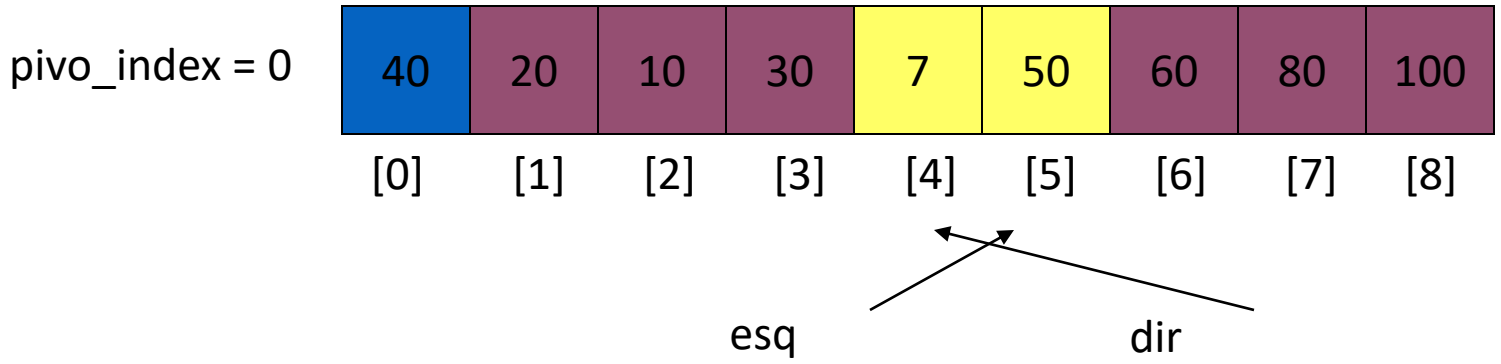
Partição: Funcionamento

1. While lista[esq] <= lista[pivo]
   ++esq
2. While lista[dir] > lista[pivo]
   --dir
3. If esq < dir
   troca lista[esq] and lista[dir]
4. While dir > esq, **go to** 1.



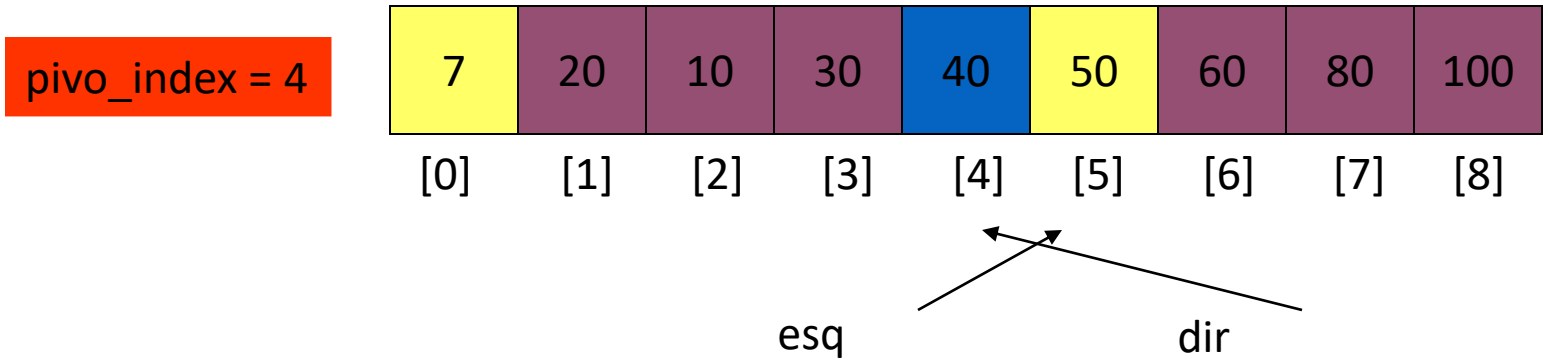
Partição: Funcionamento

1. While lista[esq] <= lista[pivo]
   ++esq
2. While lista[dir] > lista[pivo]
   --dir
3. If esq < dir
   troca lista[esq] and lista[dir]
4. While dir > esq, **go to** 1.
5. Swap data[dir] and lista[pivo]

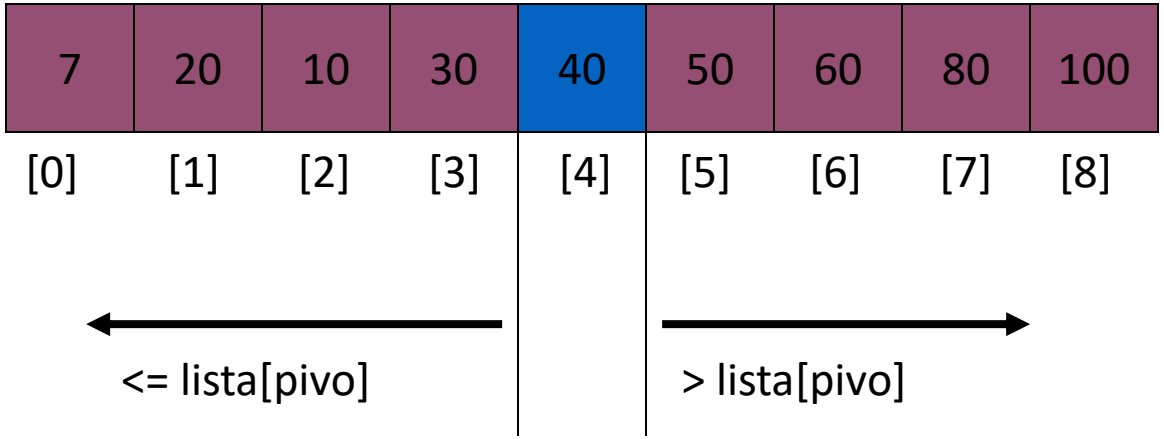


Partição: Funcionamento

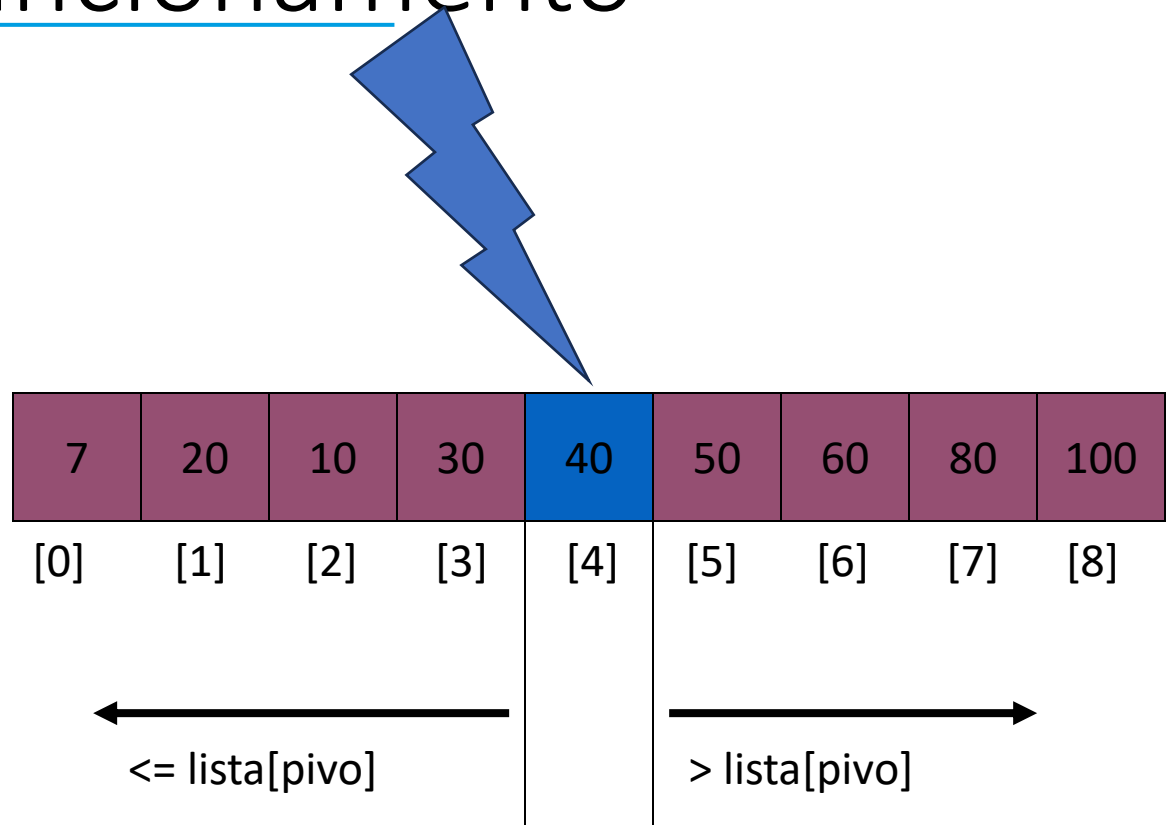
1. While lista[esq] <= lista[pivo]
   ++esq
2. While lista[dir] > lista[pivo]
   --dir
3. If esq < dir
   troca lista[esq] and lista[dir]
4. While dir > esq, **go to** 1.
5. Swap data[dir] and lista[pivo]



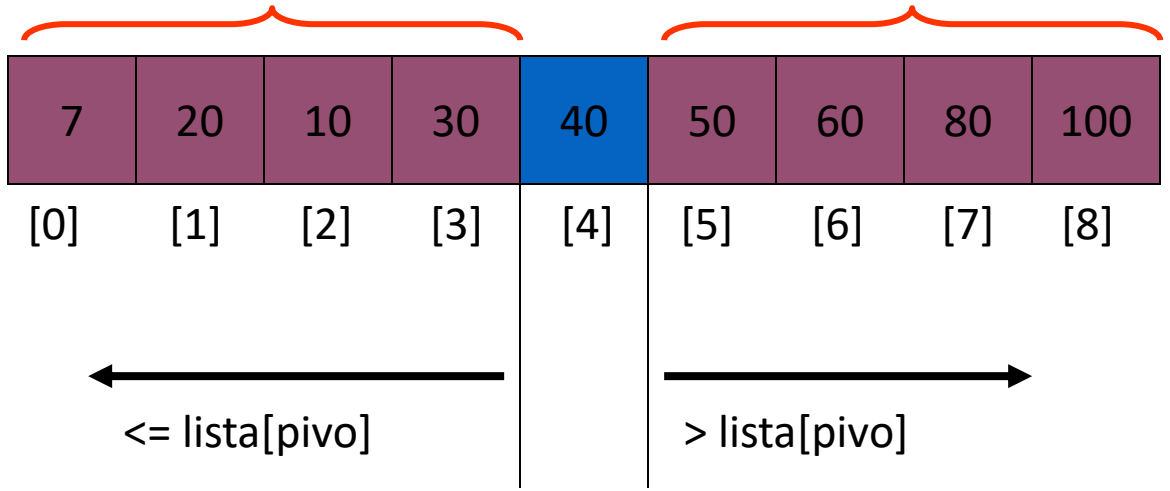
Partição: Funcionamento



Partição: Funcionamento



Partição: Recursão



Partição – Resumo da tal partição

- **Dado um pivô**, **particione** os elementos de uma lista de modo que a lista resultante consista em:
  - **Uma sub-lista contém elementos > = pivô**
  - **Outra sub-lista** contém **elementos < pivot**

As **sub-listas** são **armazenados** na **lista de dados originais**.

Particionando loops, os elementos devem ser trocados esquerda/direita do pivô.

Algoritmo: Quick Sort

- **Como implementar???**

Algoritmo: Quick Sort

Algoritmo usa 2 funções

- **quickSort** : divide os dados em arrays cada vez menores

Algoritmo: Quick Sort

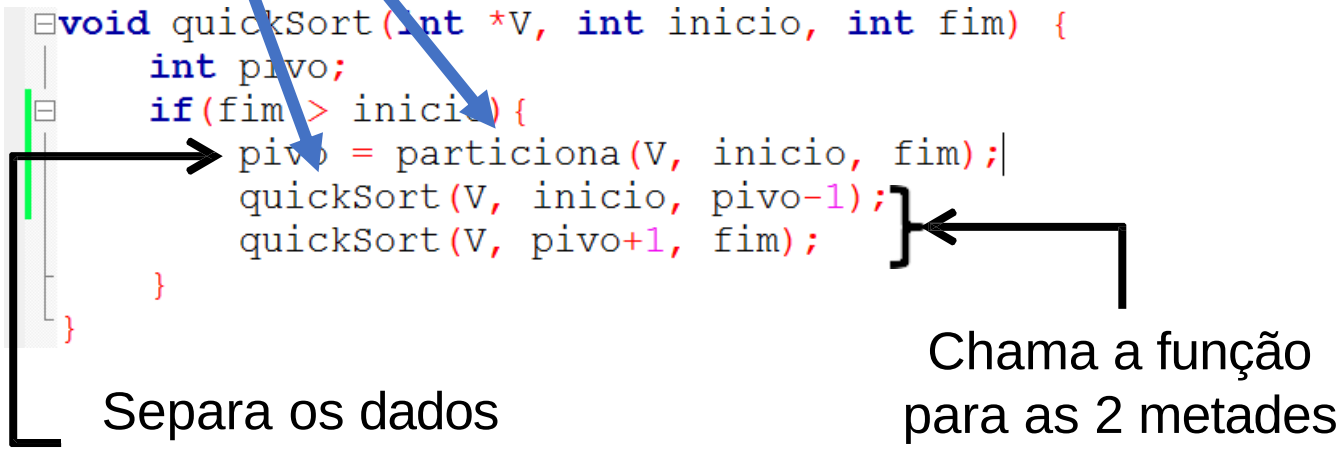
Algoritmo usa 2 funções

- **quickSort** : divide os dados em arrays cada vez menores
- **particiona**: calcula o pivô e rearranja os dados

Algoritmo: Quick Sort

Algoritmo usa 2 funções

- **quickSort** : divide os dados em arrays cada vez menores
- **particiona**: calcula o pivô e rearranja os dados



Algoritmo Quick Sort

- Algoritmo

```
19  int particiona(int *V, int inicio, int final ){
20      int esq, dir, pivo, aux;
21      esq = inicio;
22      dir = final;
23      pivo = V[inicio];
24      while(esq < dir){
25          while(esq <= final && V[esq] <= pivo)
26              esq++;
27
28          while(dir >= 0 && V[dir] > pivo)
29              dir--;
30
31          if(esq < dir){
32              aux = V[esq];
33              V[esq] = V[dir];
34              V[dir] = aux;
35          }
36      }
37      V[inicio] = V[dir];
38      V[dir] = pivo;
39      return dir;
40  }
```

Algoritmo Quick Sort

- Algoritmo

```
19  int particiona(int *V, int inicio, int final ){
20      int esq, dir, pivo, aux;
21      esq = inicio;
22      dir = final;
23      pivo = V[inicio];
24      while(esq < dir){
25          while(esq <= final && V[esq] <= pivo)
26              esq++;
27
28          while(dir >= 0 && V[dir] > pivo)
29              dir--;
30
31          if(esq < dir){
32              aux = V[esq];
33              V[esq] = V[dir];
34              V[dir] = aux;
35          }
36      }
37      V[inicio] = V[dir];
38      V[dir] = pivo;
39      return dir;
40  }
```

Avança posição da esquerda

Algoritmo Quick Sort

- Algoritmo

```
19  int particiona(int *V, int inicio, int final ){
20      int esq, dir, pivo, aux;
21      esq = inicio;
22      dir = final;
23      pivo = V[inicio];
24      while(esq < dir){
25          while(esq <= final && V[esq] <= pivo)
26              esq++;
27
28          while(dir >= 0 && V[dir] > pivo)
29              dir--;
30
31          if(esq < dir){
32              aux = V[esq];
33              V[esq] = V[dir];
34              V[dir] = aux;
35          }
36      }
37      V[inicio] = V[dir];
38      V[dir] = pivo;
39      return dir;
40  }
```

Avança posição da esquerda

Recua posição da direita

Algoritmo Quick Sort

- Algoritmo

```
19  int particiona(int *V, int inicio, int final ){
20      int esq, dir, pivo, aux;
21      esq = inicio;
22      dir = final;
23      pivo = V[inicio];
24      while(esq < dir){
25          while(esq <= final && V[esq] <= pivo)
26              esq++;
27
28          while(dir >= 0 && V[dir] > pivo)
29              dir--;
30
31          if(esq < dir){
32              aux = V[esq];
33              V[esq] = V[dir];
34              V[dir] = aux;
35          }
36      }
37      V[inicio] = V[dir];
38      V[dir] = pivo;
39      return dir;
40  }
```

Avança posição da esquerda

Recua posição da direita

Trocar esq e dir

Algoritmo Quick Sort

- Algoritmo

```
19  int particiona(int *V, int inicio, int final ){
20      int esq, dir, pivo, aux;
21      esq = inicio;
22      dir = final;
23      pivo = V[inicio];
24      while(esq < dir){
25          while(esq <= final && V[esq] <= pivo)
26              esq++;
27
28          while(dir >= 0 && V[dir] > pivo)
29              dir--;
30
31          if(esq < dir){
32              aux = V[esq];
33              V[esq] = V[dir];
34              V[dir] = aux;
35          }
36      }
37      V[inicio] = V[dir];
38      V[dir] = pivo;
39      return dir;
40  }
```

Avança posição da esquerda

Recua posição da direita

Trocar esq e dir

Ajusta o pivô no local correto

Algoritmo Quick Sort

- Complexidade
  - Em geral, é algoritmo muito rápido. Porém, é um algoritmo lento em alguns casos especiais
    - Por exemplo, quando o particionamento não é balanceado
  - Considerando um array com **N** elementos, o tempo de execução é:
    - ***O(N log N)***, melhor caso e caso médio;
    - ***O(N²)***, pior caso.

Algoritmo Quick Sort

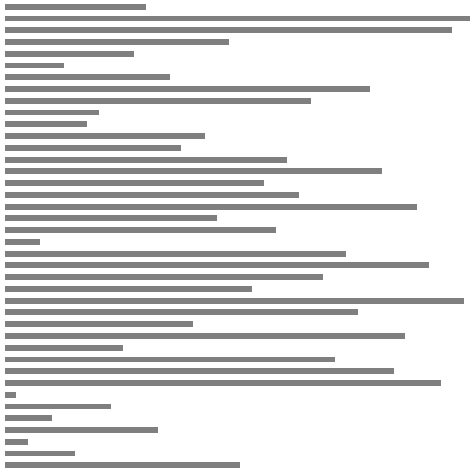
- Desvantagens
  - Não é um algoritmo estável
  - **Como escolher o pivô?**
    - Existem várias abordagens diferentes
    - No pior caso o pivô divide o array de **N** em dois: uma partição com **N-1** elementos e outra com **0** elementos
    - **Particionamento não é balanceado**
    - Quando isso acontece a cada nível da recursão, temos o tempo de execução de ***O(N²)***

Algoritmo Quick Sort

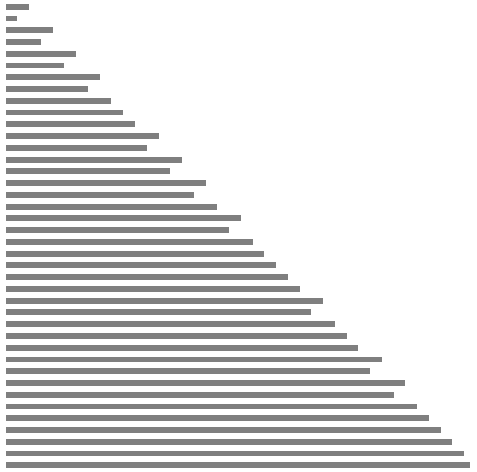
- Desvantagens
  - No caso de um particionamento não balanceado, **o insertion sort acaba sendo mais eficiente que o quick sort**
    - O pior caso do quick sort ocorre quando o array já está ordenado, uma situação onde a complexidade é ***O(N)*** no insertion sort
- **Vantagem**
  - **Apesar de seu pior caso ser quadrático, costuma ser a melhor opção prática para ordenação de grandes conjuntos de dados**

Estudo de Casos

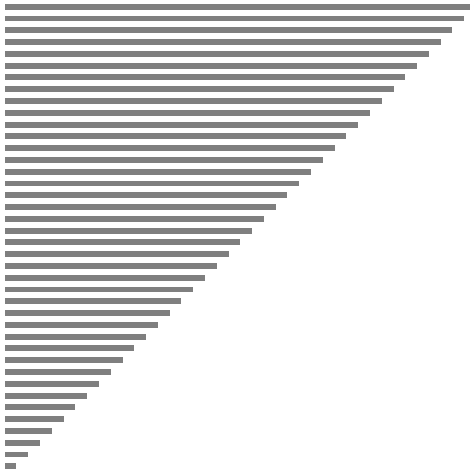
Aleatório



Quase ordenado



Inverso



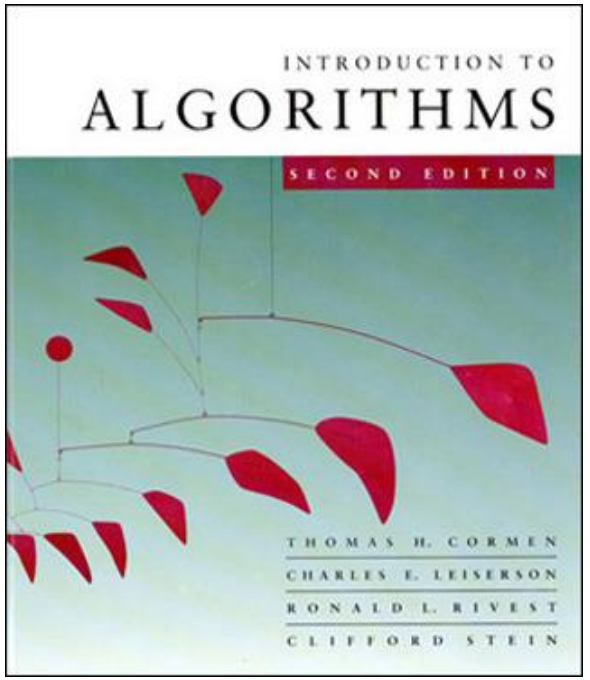
Valores repetidos

Quick Sort

- Pior caso
  - Quando o pivô é o maior ou menor do conjunto, dessa forma não existe divisão dos dados
- Melhor caso
  - O pivô é sempre o elemento central

Leitura importante

livro “Algorithms” de Cormen et al.



Shellsort

Classificação e Pesquisa de Dados

Cristiano Santos

cristiano.santos@amf.edu.br