Algoritmos e Programação de Computadores

Ordenação: Quick Sort

Marcelo Jara P.
Dr. Eng. Eletrica, FEEC, Unicamp
mjara.perez@ic.unicamp.br
MC102-Z, 06 Novembro, 2018
Instituto de Computação (IC/Unicamp)

Introdução

Será utilizada a técnica de **recursão** para resolver um problema clássico de **ordenação**.

- **Problema**:
  - Temos uma lista `v` de inteiros de tamanho `n`.
  - Devemos deixar `v` ordenada em ordem crescente de valores.
- **QuickSort**
  - Algoritmo de Ordenação proposto por T. Hoare in 1959, pub. in 1961
  - Também conhecido como **partition-exchange sort**.

Dividir e Conquistar

- Temos que resolver um problema `P` de tamanho `n`.
- **Dividir**: Quebra-se `P` em sub-problemas menores.
- Resolvemos os sub-problemas de forma recursiva.
- **Conquistar**: Unimos as soluções dos sub-problemas para obter solução do problema maior `P`.

Quick Sort

Quick Sort

- Vamos supor que devemos ordenar uma lista de uma posição `inicial` até `fim`.
- **Dividir**:
  - Escolha em elemento particular da lista chamado `pivô`. O procedimento de escolha de `pivô` pode ser aleatório ou não
  - Particione a lista em uma posição `pos` tal que todos elementos de `inicial` até `pos - 1` são menores ou iguais do que o `pivô`, e todos elementos de `pos` até `fim` são maiores ou iguais ao `pivô`.

Quick Sort

- **Resumo**
  - O problema de ordenação é resolvido de forma **recursiva** para as duas sub-listas (uma de `inicial` até `pos-1` e a outra de `pos` até `fim`).
- **Algoritmo**:
  - **Particionar**: reordenar parcialmente a lista original, que será dividida no fim deste processo em 2 sub-listas, com um valor “pos” definindo o elemento inicial da partição direita
  - **QuickSort**. Em este procedimento a função será invocada recursivamente e a funçao “Particionar” será executada uma vez em cada chamada de **QuickSort**

Quick Sort: Particionamento

Define-se um valor `p` como `pivô`, este valor será a referência para todas as comparações com os elementos da lsta

1. A lista (ou vetor) é “varrida” do `início` ao `fim` até encontrar um elemento maior que o `pivô` (direção `esquerda` a `direita`)
2. “Varre-se” de novo a lista (ou vetor) do `fim` ao `início` até encontrar um elemento menor ou igual ao `pivô`. (direção `direita` a `esquerda`)
3. Trocam-se então as posições destes elementos e continua-se com o procedimento até verificar todas as posições da lista (ou vetor).

Quick Sort: 1o passo - Particionament

1. Dada uma lista, determinar um elemento `pivô`. Ex 5.
2. Definir 2 marcadores (*pointers*), um do lado **esquerdo** e outro do lado **direito** dos extremos da lista (na fig. ao lado, > *start pointer-left* `(L)`, *end pointer-right* `(R)`
3. Compare o valor do elemento apontado pelo marcador **esquerdo** (*left-pointer*), **`L`**, com o `pivô`

a) Caso (`lista[L] < pivô`), incremente (desloque) o marcador `L`, um lugar na direção direita ... **>**

b) Caso contrario, **não** incrementar o marcador `L` .

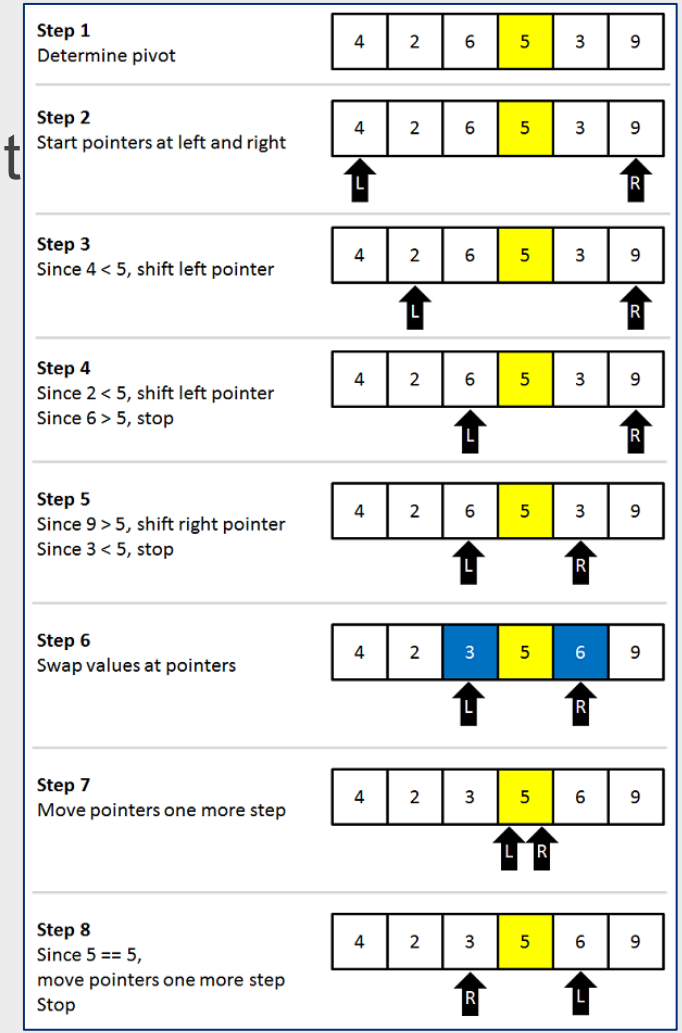
4. Compare o valor do marcador **direito** (*right pointer*) , **`R`**, com o `pivô`.

a) Caso (`lista[R] > pivô`), decremente (desloque) o marcador `R`, um lugar na direção esquerda ... <

b) Caso contrario, pare de decrementar o marcador `R`.

5. Troque de posições os 2 ultimos elementos indicados pelos marcadores (*left/right pointers,* `lista[L]` *e* `lista[R]`), ao redor do `pivô` (*Step* 6 na fig. ao lado)

... Continue até que 2 marcadores (*pointers*) se encontrem



Quick Sort: Particionamento

A função `particiona` retorna a posição de partição. Em esta versão, considera-se o último elemento da lista como o `pivô`.

```
def particiona (v, inicio, fim):
    pivo = v[fim]
    while (inicio < fim):
    # o laço para quando inicio == fim => checar o vetor inteiro
        while (inicio < fim) and (v[inicio] <= pivo):
        # acha posição de elemento maior que pivo
            inicio = inicio + 1
        while (inicio < fim) and (v[fim] > pivo) :
        # acha posição de elemento menor ou igual que pivo
            fim = fim - 1
        v[inicio], v[fim] = v[fim], v[inicio] # troca elementos de posição
    return inicio
```

Quick Sort: Particionamento

Exemplo: `(1,9,3,7,6,2,3,8,5)` e `pivô=5`.

Para este exemplo, foi determinado que o `pivô`, será o **valor** indicado pelo último elemento da lista, e os valores (marcadores), serão os **índices** dos elementos de início e final da lista, i.e.

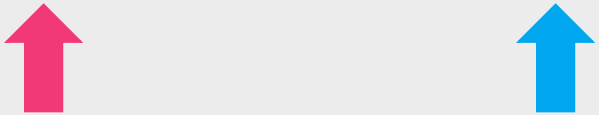
`inicio = 0 (índice do 1° elemento),`
`fim = len(lista)- 1 = 8 (índice do ultimo elemento)`

Se `N = len(lista)`, `pivô = lista[N-1]`
Logo, para
`N= 9`
`pivô = lista[8] = 5`

Quick Sort: Particionamento

Exemplo: `(1,9,3,7,6,2,3,8,5)` e `pivô=5`.

- `(1,9,3,7,6,2,3,8,5)`



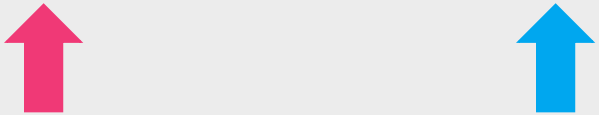
inicio = 0
fim = 8
pivo = 5

```
def particiona (v, inicio, fim):
    pivo = v[fim]
    while (inicio < fim):
        while (inicio < fim) and (v[inicio] <=
        pivo):
            inicio = inicio + 1
        while (inicio < fim) and (v[fim] > pivo) :
            fim = fim - 1
        v[inicio], v[fim] = v[fim], v[inicio]
    return inicio
```

Quick Sort: Particionamento

Exemplo: `(1,9,3,7,6,2,3,8,5)` e `pivô=5`.

- `(1,9,3,7,6,2,3,8,5)`



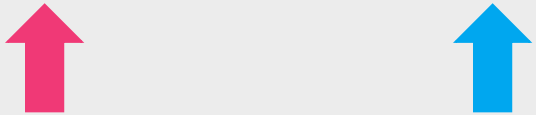
inicio = 0
fim = 8
pivo = 5

```
def particiona (v, inicio, fim):
    pivo = v[fim]
    while (inicio < fim):
        while (inicio < fim) and (v[inicio] <=
        pivo):
            inicio = inicio + 1
        while (inicio < fim) and (v[fim] > pivo) :
            fim = fim - 1
        v[inicio], v[fim] = v[fim], v[inicio]
    return inicio
```

Quick Sort: Particionamento

Exemplo: `(1,9,3,7,6,2,3,8,5)` e `pivô=5`.

- `(1,9,3,7,6,2,3,8,5)`



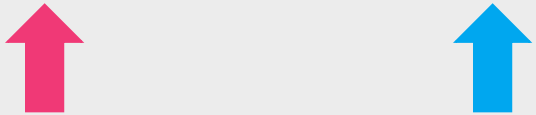
inicio = 1
fim = 8
pivo = 5

```
def particiona (v, inicio, fim):
    pivo = v[fim]
    while (inicio < fim):
        while (inicio < fim) and (v[inicio] <=
        pivo):
            inicio = inicio + 1
        while (inicio < fim) and (v[fim] > pivo) :
            fim = fim - 1
        v[inicio], v[fim] = v[fim], v[inicio]
    return inicio
```

Quick Sort: Particionamento

Exemplo: `(1,9,3,7,6,2,3,8,5)` e `pivô=5`.

- `(1,9,3,7,6,2,3,8,5)`



inicio = 1
fim = 8
pivo = 5

```
def particiona (v, inicio, fim):
    pivo = v[fim]
    while (inicio < fim):
        while (inicio < fim) and (v[inicio] <=
        pivo):
            inicio = inicio + 1
        while (inicio < fim) and (v[fim] > pivo) :
            fim = fim - 1
        v[inicio], v[fim] = v[fim], v[inicio]
    return inicio
```

Quick Sort: Particionamento

Exemplo: `(1,9,3,7,6,2,3,8,5)` e `pivô=5`.

- `(1,9,3,7,6,2,3,8,5)` → `(1,5,3,7,6,2,3,8,9)`

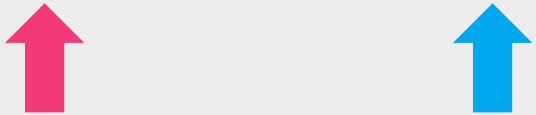
inicio = 1
fim = 8
pivo = 5

```
def particiona (v, inicio, fim):
    pivo = v[fim]
    while (inicio < fim):
        while (inicio < fim) and (v[inicio] <=
        pivo):
            inicio = inicio + 1
        while (inicio < fim) and (v[fim] > pivo) :
            fim = fim - 1
        v[inicio], v[fim] = v[fim], v[inicio]
    return inicio
```

Quick Sort: Particionamento

Exemplo: `(1,9,3,7,6,2,3,8,5)` e `pivô=5`.

- `(1,5,3,7,6,2,3,8,9)`



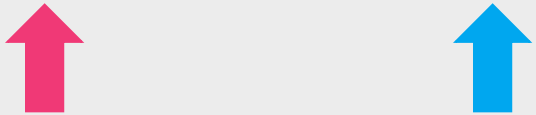
inicio = 1
fim = 8
pivo = 5

```
def particiona (v, inicio, fim):
    pivo = v[fim]
    while (inicio < fim):
        while (inicio < fim) and (v[inicio] <=
        pivo):
            inicio = inicio + 1
        while (inicio < fim) and (v[fim] > pivo) :
            fim = fim - 1
        v[inicio], v[fim] = v[fim], v[inicio]
    return inicio
```

Quick Sort: Particionamento

Exemplo: `(1,9,3,7,6,2,3,8,5)` e `pivô=5`.

- `(1,5,3,7,6,2,3,8,9)`



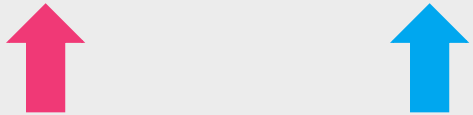
inicio = 1
fim = 8
pivo = 5

```
def particiona (v, inicio, fim):
    pivo = v[fim]
    while (inicio < fim):
        while (inicio < fim) and (v[inicio] <=
        pivo):
            inicio = inicio + 1
        while (inicio < fim) and (v[fim] > pivo) :
            fim = fim - 1
        v[inicio], v[fim] = v[fim], v[inicio]
    return inicio
```

Quick Sort: Particionamento

Exemplo: `(1,9,3,7,6,2,3,8,5)` e `pivô=5`.

- `(1,5,3,7,6,2,3,8,9)`



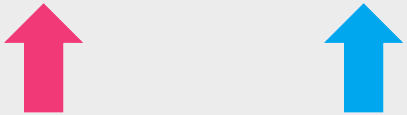
inicio = 2
fim = 8
pivo = 5

```
def particiona (v, inicio, fim):
    pivo = v[fim]
    while (inicio < fim):
        while (inicio < fim) and (v[inicio] <=
        pivo):
            inicio = inicio + 1
        while (inicio < fim) and (v[fim] > pivo) :
            fim = fim - 1
        v[inicio], v[fim] = v[fim], v[inicio]
    return inicio
```

Quick Sort: Particionamento

Exemplo: `(1,9,3,7,6,2,3,8,5)` e `pivô=5`.

- `(1,5,3,7,6,2,3,8,9)`



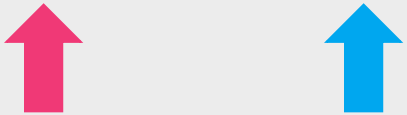
inicio = 3
fim = 8
pivo = 5

```
def particiona (v, inicio, fim):
    pivo = v[fim]
    while (inicio < fim):
        while (inicio < fim) and (v[inicio] <=
        pivo):
            inicio = inicio + 1
        while (inicio < fim) and (v[fim] > pivo) :
            fim = fim - 1
        v[inicio], v[fim] = v[fim], v[inicio]
    return inicio
```

Quick Sort: Particionamento

Exemplo: `(1,9,3,7,6,2,3,8,5)` e `pivô=5`.

- `(1,5,3,7,6,2,3,8,9)`



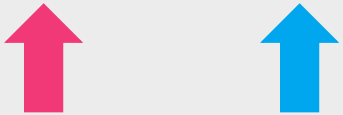
inicio = 3
fim = 8
pivo = 5

```
def particiona (v, inicio, fim):
    pivo = v[fim]
    while (inicio < fim):
        while (inicio < fim) and (v[inicio] <=
        pivo):
            inicio = inicio + 1
        while (inicio < fim) and (v[fim] > pivo) :
            fim = fim - 1
        v[inicio], v[fim] = v[fim], v[inicio]
    return inicio
```

Quick Sort: Particionamento

Exemplo: `(1,9,3,7,6,2,3,8,5)` e `pivô=5`.

- `(1,5,3,7,6,2,3,8,9)`



inicio = 3
fim = 7
pivo = 5

```
def particiona (v, inicio, fim):
    pivo = v[fim]
    while (inicio < fim):
        while (inicio < fim) and (v[inicio] <=
        pivo):
            inicio = inicio + 1
        while (inicio < fim) and (v[fim] > pivo) :
            fim = fim - 1
        v[inicio], v[fim] = v[fim], v[inicio]
    return inicio
```

Quick Sort: Particionamento

Exemplo: `(1,9,3,7,6,2,3,8,5)` e `pivô=5`.

- `(1,5,3,7,6,2,3,8,9)`

inicio = 3
fim = 6
pivo = 5

```
def particiona (v, inicio, fim):
    pivo = v[fim]
    while (inicio < fim):
        while (inicio < fim) and (v[inicio] <=
        pivo):
            inicio = inicio + 1
        while (inicio < fim) and (v[fim] > pivo) :
            fim = fim - 1
        v[inicio], v[fim] = v[fim], v[inicio]
    return inicio
```

Quick Sort: Particionamento

Exemplo: `(1,9,3,7,6,2,3,8,5)` e `pivô=5`.

- `(1,5,3,7,6,2,3,8,9)` → `(1,5,3,3,6,2,7,8,9)`

inicio = 3
fim = 6
pivo = 5

```
def particiona (v, inicio, fim):
    pivo = v[fim]
    while (inicio < fim):
        while (inicio < fim) and (v[inicio] <=
        pivo):
            inicio = inicio + 1
        while (inicio < fim) and (v[fim] > pivo) :
            fim = fim - 1
        v[inicio], v[fim] = v[fim], v[inicio]
    return inicio
```

Quick Sort: Particionamento

Exemplo: `(1,9,3,7,6,2,3,8,5)` e `pivô=5`.

- `(1,5,3,3,6,2,7,8,9)`

inicio = 3
fim = 6
pivo = 5

```
def particiona (v, inicio, fim):
    pivo = v[fim]
    while (inicio < fim):
        while (inicio < fim) and (v[inicio] <=
        pivo):
            inicio = inicio + 1
        while (inicio < fim) and (v[fim] > pivo) :
            fim = fim - 1
        v[inicio], v[fim] = v[fim], v[inicio]
    return inicio
```

Quick Sort: Particionamento

Exemplo: `(1,9,3,7,6,2,3,8,5)` e `pivô=5`.

- `(1,5,3,3,6,2,7,8,9)`

inicio = 3
fim = 6
pivo = 5

```
def particiona (v, inicio, fim):
    pivo = v[fim]
    while (inicio < fim):
        while (inicio < fim) and (v[inicio] <=
        pivo):
            inicio = inicio + 1
        while (inicio < fim) and (v[fim] > pivo) :
            fim = fim - 1
        v[inicio], v[fim] = v[fim], v[inicio]
    return inicio
```

Quick Sort: Particionamento

Exemplo: `(1,9,3,7,6,2,3,8,5)` e `pivô=5`.

- `(1,5,3,3,6,2,7,8,9)`

inicio = 4
fim = 6
pivo = 5

```
def particiona (v, inicio, fim):
    pivo = v[fim]
    while (inicio < fim):
        while (inicio < fim) and (v[inicio] <=
        pivo):
            inicio = inicio + 1
        while (inicio < fim) and (v[fim] > pivo) :
            fim = fim - 1
        v[inicio], v[fim] = v[fim], v[inicio]
    return inicio
```

Quick Sort: Particionamento

Exemplo: `(1,9,3,7,6,2,3,8,5)` e `pivô=5`.

- `(1,5,3,3,6,2,7,8,9)`

inicio = 4
fim = 6
pivo = 5

```
def particiona (v, inicio, fim):
    pivo = v[fim]
    while (inicio < fim):
        while (inicio < fim) and (v[inicio] <=
        pivo):
            inicio = inicio + 1
        while (inicio < fim) and (v[fim] > pivo) :
            fim = fim - 1
        v[inicio], v[fim] = v[fim], v[inicio]
    return inicio
```

Quick Sort: Particionamento

Exemplo: `(1,9,3,7,6,2,3,8,5)` e `pivô=5`.

- `(1,5,3,3,6,2,7,8,9)`

inicio = 4
fim = 5
pivo = 5

```
def particiona (v, inicio, fim):
    pivo = v[fim]
    while (inicio < fim):
        while (inicio < fim) and (v[inicio] <=
        pivo):
            inicio = inicio + 1
        while (inicio < fim) and (v[fim] > pivo) :
            fim = fim - 1
        v[inicio], v[fim] = v[fim], v[inicio]
    return inicio
```

Quick Sort: Particionamento

Exemplo: `(1,9,3,7,6,2,3,8,5)` e `pivô=5`.

- `(1,5,3,3,6,2,7,8,9)` → `(1,5,3,3,2,6,7,8,9)`

inicio = 4
fim = 5
pivo = 5

```
def particiona (v, inicio, fim):
    pivo = v[fim]
    while (inicio < fim):
        while (inicio < fim) and (v[inicio] <=
        pivo):
            inicio = inicio + 1
        while (inicio < fim) and (v[fim] > pivo) :
            fim = fim - 1
        v[inicio], v[fim] = v[fim], v[inicio]
    return inicio
```

Quick Sort: Particionamento

Exemplo: `(1,9,3,7,6,2,3,8,5)` e `pivô=5`.

- `(1,5,3,3,2,6,7,8,9)`

inicio = 4
fim = 5
pivo = 5

```
def particiona (v, inicio, fim):
    pivo = v[fim]
    while (inicio < fim):
        while (inicio < fim) and (v[inicio] <=
        pivo):
            inicio = inicio + 1
        while (inicio < fim) and (v[fim] > pivo) :
            fim = fim - 1
        v[inicio], v[fim] = v[fim], v[inicio]
    return inicio
```

Quick Sort: Particionamento

Exemplo: `(1,9,3,7,6,2,3,8,5)` e `pivô=5`.

- `(1,5,3,3,2,6,7,8,9)`

inicio = 4
fim = 5
pivo = 5

```
def particiona (v, inicio, fim):
    pivo = v[fim]
    while (inicio < fim):
        while (inicio < fim) and (v[inicio] <=
        pivo):
            inicio = inicio + 1
        while (inicio < fim) and (v[fim] > pivo) :
            fim = fim - 1
        v[inicio], v[fim] = v[fim], v[inicio]
    return inicio
```

Quick Sort: Particionamento

Exemplo: `(1,9,3,7,6,2,3,8,5)` e `pivô=5`.

- `(1,5,3,3,2,6,7,8,9)`

inicio = 5
fim = 5
pivo = 5

```
def particiona (v, inicio, fim):
    pivo = v[fim]
    while (inicio < fim):
        while (inicio < fim) and (v[inicio] <=
        pivo):
            inicio = inicio + 1
        while (inicio < fim) and (v[fim] > pivo) :
            fim = fim - 1
        v[inicio], v[fim] = v[fim], v[inicio]
    return inicio
```

Quick Sort: Particionamento

Exemplo: `(1,9,3,7,6,2,3,8,5)` e `pivô=5`.

- `(1,5,3,3,2,6,7,8,9)` → Retorna posição `5 (inicio=5)`.

inicio = 5
fim = 5
pivo = 5

```
def particiona (v, inicio, fim):
    pivo = v[fim]
    while (inicio < fim):
        while (inicio < fim) and (v[inicio] <=
        pivo):
            inicio = inicio + 1
        while (inicio < fim) and (v[fim] > pivo) :
            fim = fim - 1
        v[inicio], v[fim] = v[fim], v[inicio]
    return inicio
```

Quick Sort: Particionamento

Exemplo: `(1,9,3,7,6,2,3,8,5)` e `pivô=5`.

- `(1,9,3,7,6,2,3,8,5)` → `(1,5,3,7,6,2,3,8,9)`
- `(1,5,3,7,6,2,3,8,9)` → `(1,5,3,3,6,2,7,8,9)`
- `(1,5,3,3,6,2,7,8,9)` → `(1,5,3,3,2,6,7,8,9)`
- `(1,5,3,3,2,6,7,8,9)` → Retorna posição `5`.

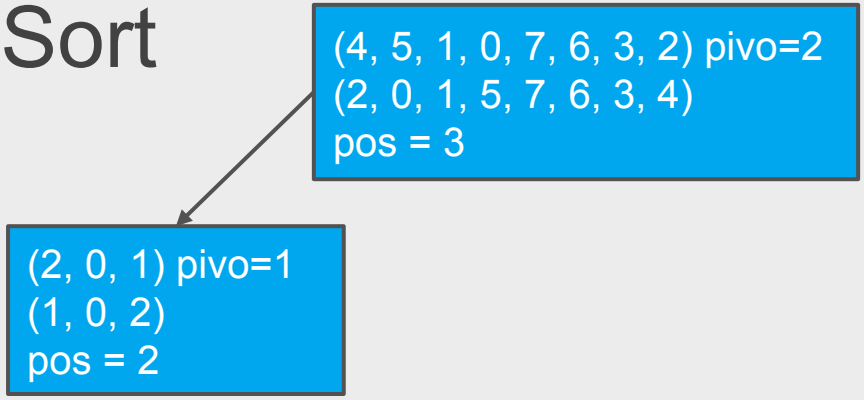
Quick Sort

```
def quickSort(v, inicio, fim):
    if (inicio < fim):
    # tem pelo menos 2 elementos a serem ordenados
        pos = particiona(v, inicio, fim)
        quickSort(v, inicio, pos-1)
        quickSort(v, pos, fim)
```

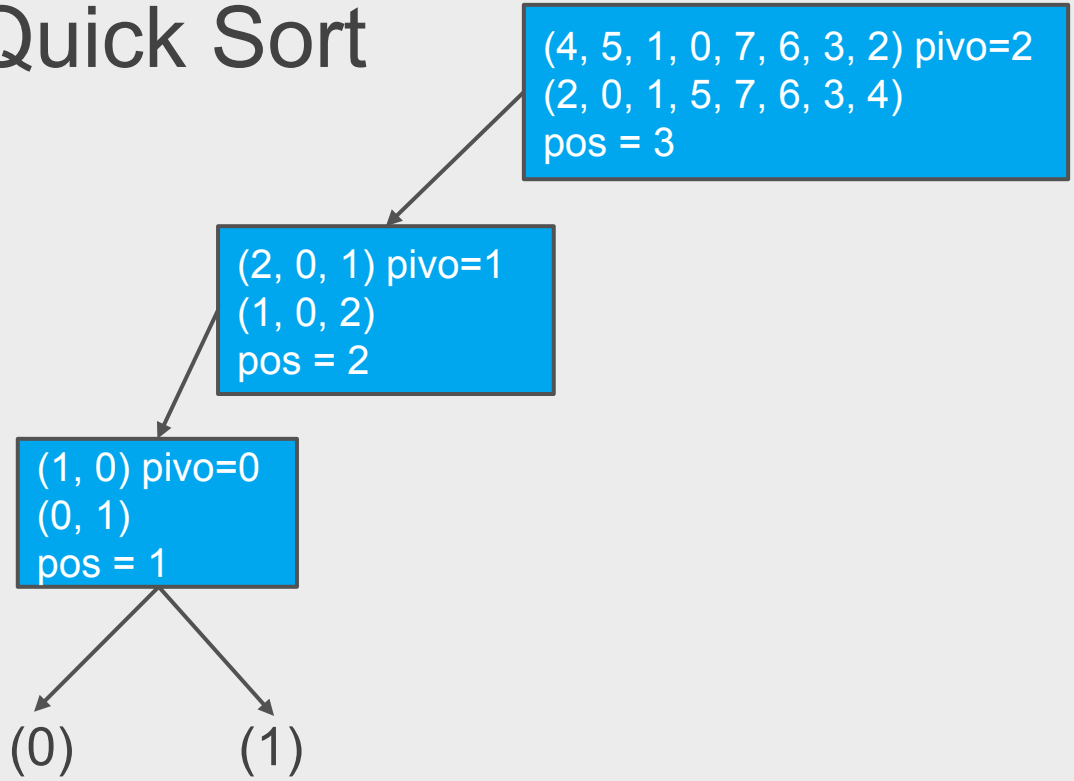
Quick Sort

(4, 5, 1, 0, 7, 6, 3, 2) pivo=2
(2, 0, 1, 5, 7, 6, 3, 4)
pos = 3

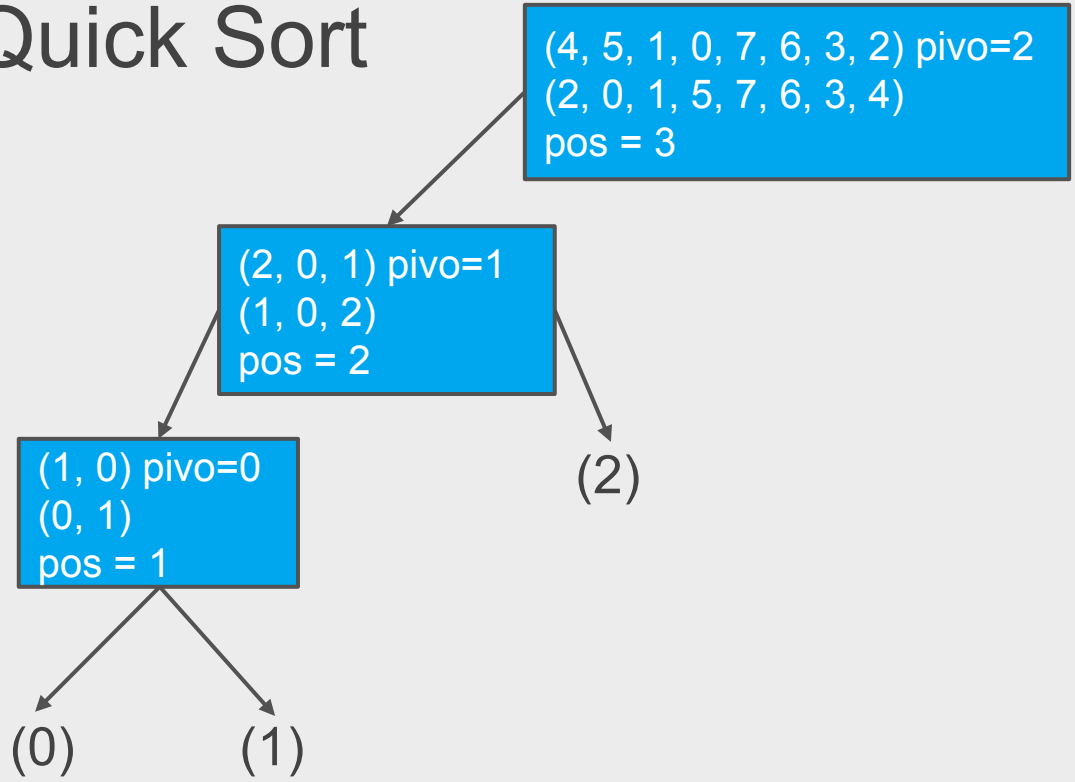
Quick Sort



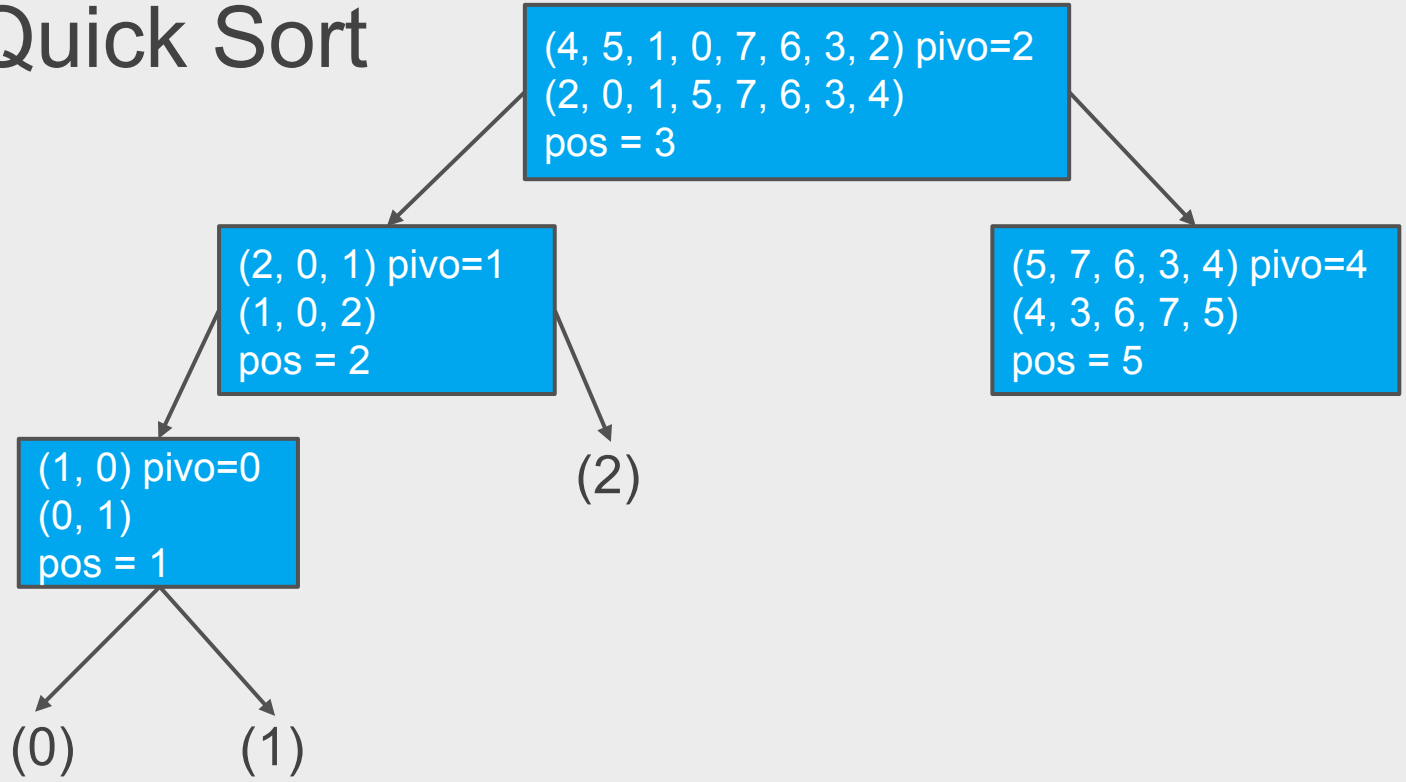
Quick Sort


(4, 5, 1, 0, 7, 6, 3, 2) pivo=2
(2, 0, 1, 5, 7, 6, 3, 4)
pos = 3
(2, 0, 1) pivo=1
(1, 0, 2)
pos = 2
(1, 0) pivo=0
(0, 1)
pos = 1
(0)
(1)

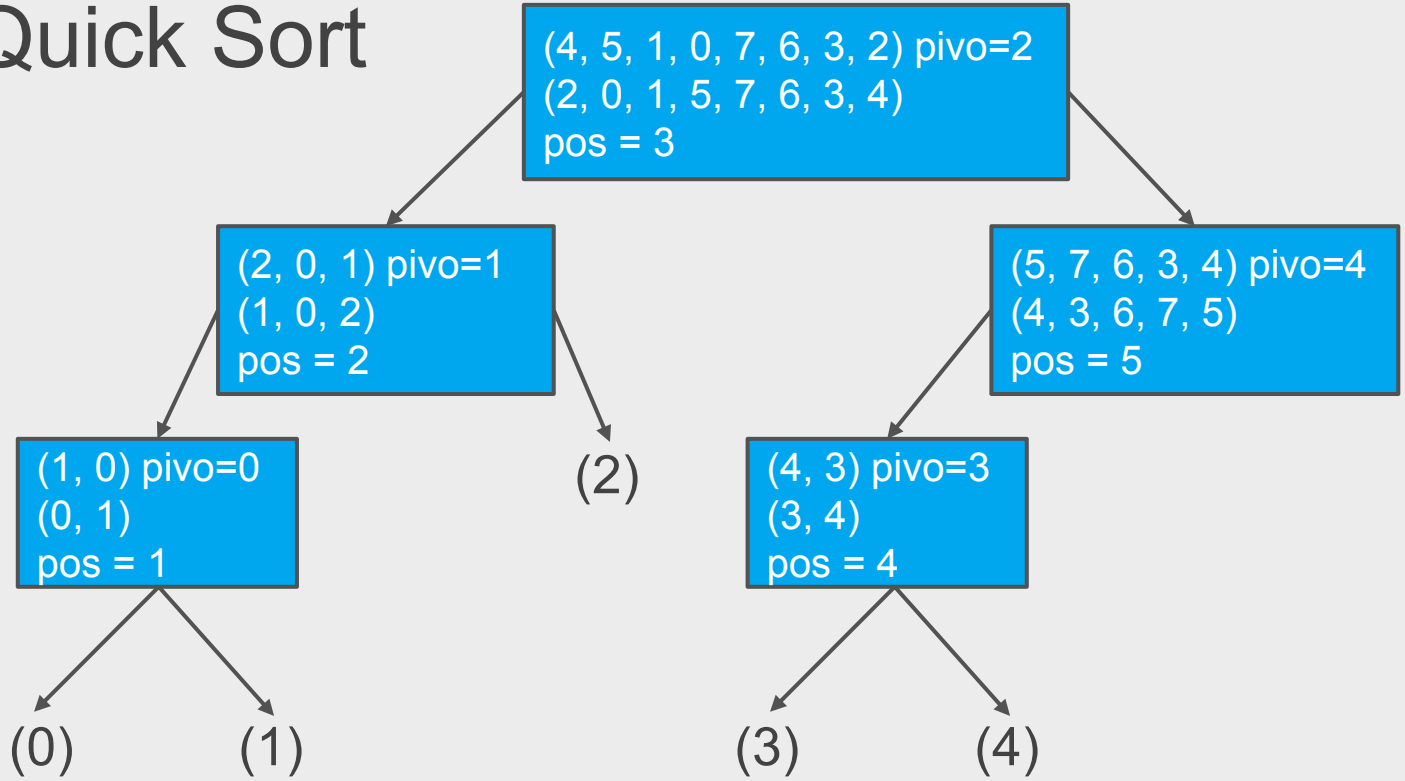
Quick Sort


(4, 5, 1, 0, 7, 6, 3, 2) pivo=2
(2, 0, 1, 5, 7, 6, 3, 4)
pos = 3
(2, 0, 1) pivo=1
(1, 0, 2)
pos = 2
(1, 0) pivo=0
(0, 1)
pos = 1
(2)
(0)
(1)

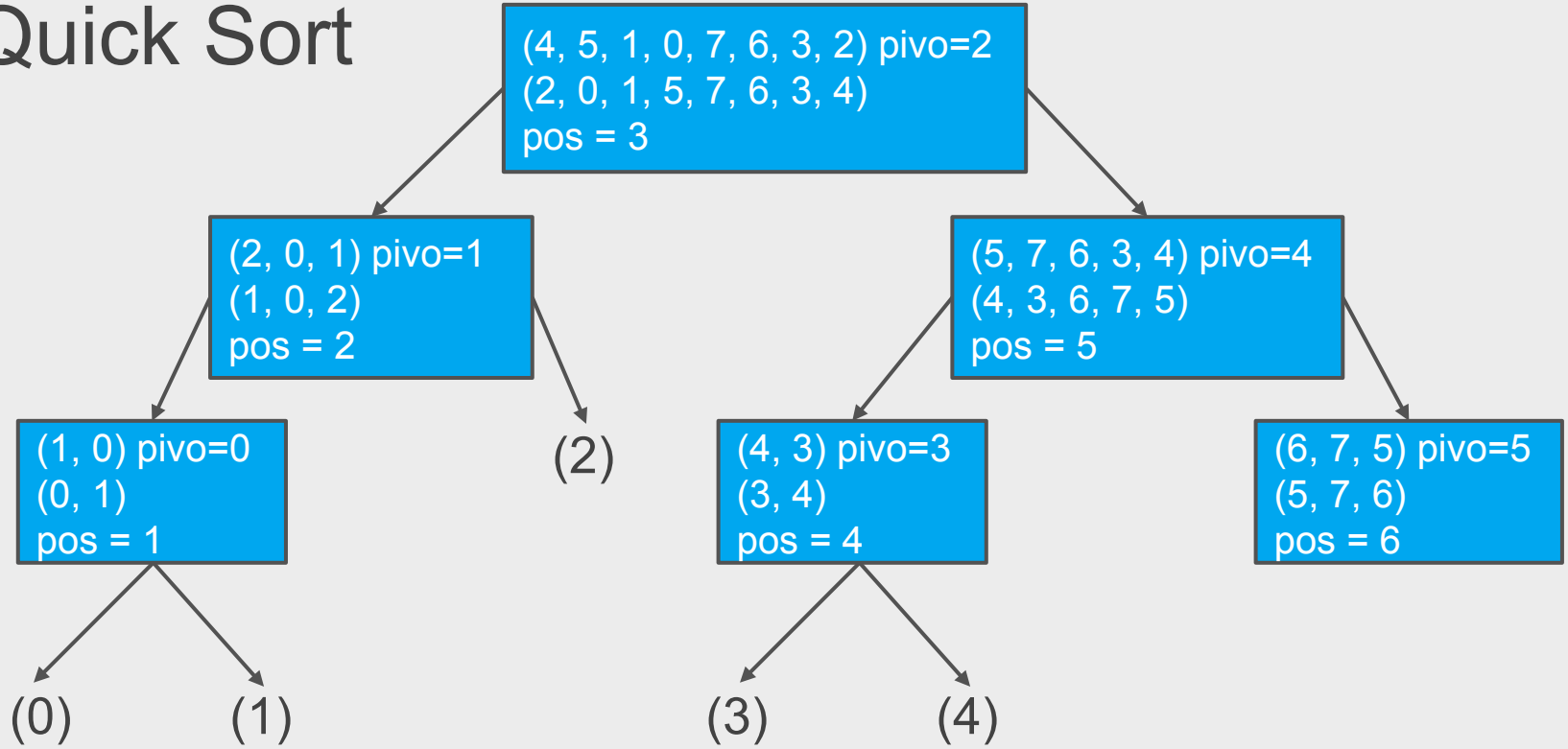
Quick Sort


(4, 5, 1, 0, 7, 6, 3, 2) pivo=2
(2, 0, 1, 5, 7, 6, 3, 4)
pos = 3
(2, 0, 1) pivo=1
(1, 0, 2)
pos = 2
(5, 7, 6, 3, 4) pivo=4
(4, 3, 6, 7, 5)
pos = 5
(1, 0) pivo=0
(0, 1)
pos = 1
(2)
(0)
(1)

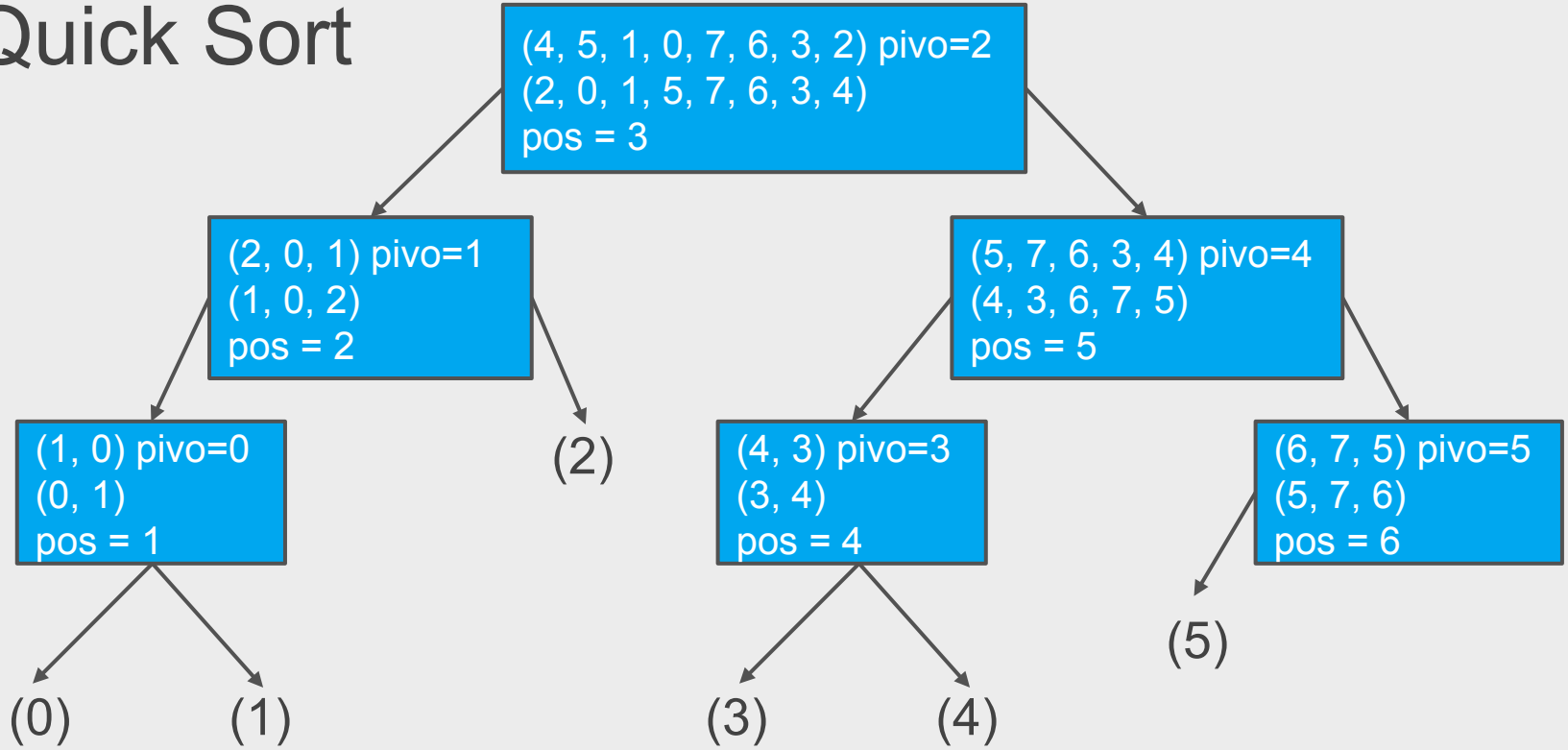
Quick Sort


(4, 5, 1, 0, 7, 6, 3, 2) pivo=2
(2, 0, 1, 5, 7, 6, 3, 4)
pos = 3
(2, 0, 1) pivo=1
(1, 0, 2)
pos = 2
(5, 7, 6, 3, 4) pivo=4
(4, 3, 6, 7, 5)
pos = 5
(1, 0) pivo=0
(0, 1)
pos = 1
(2)
(4, 3) pivo=3
(3, 4)
pos = 4
(0)
(1)
(3)
(4)

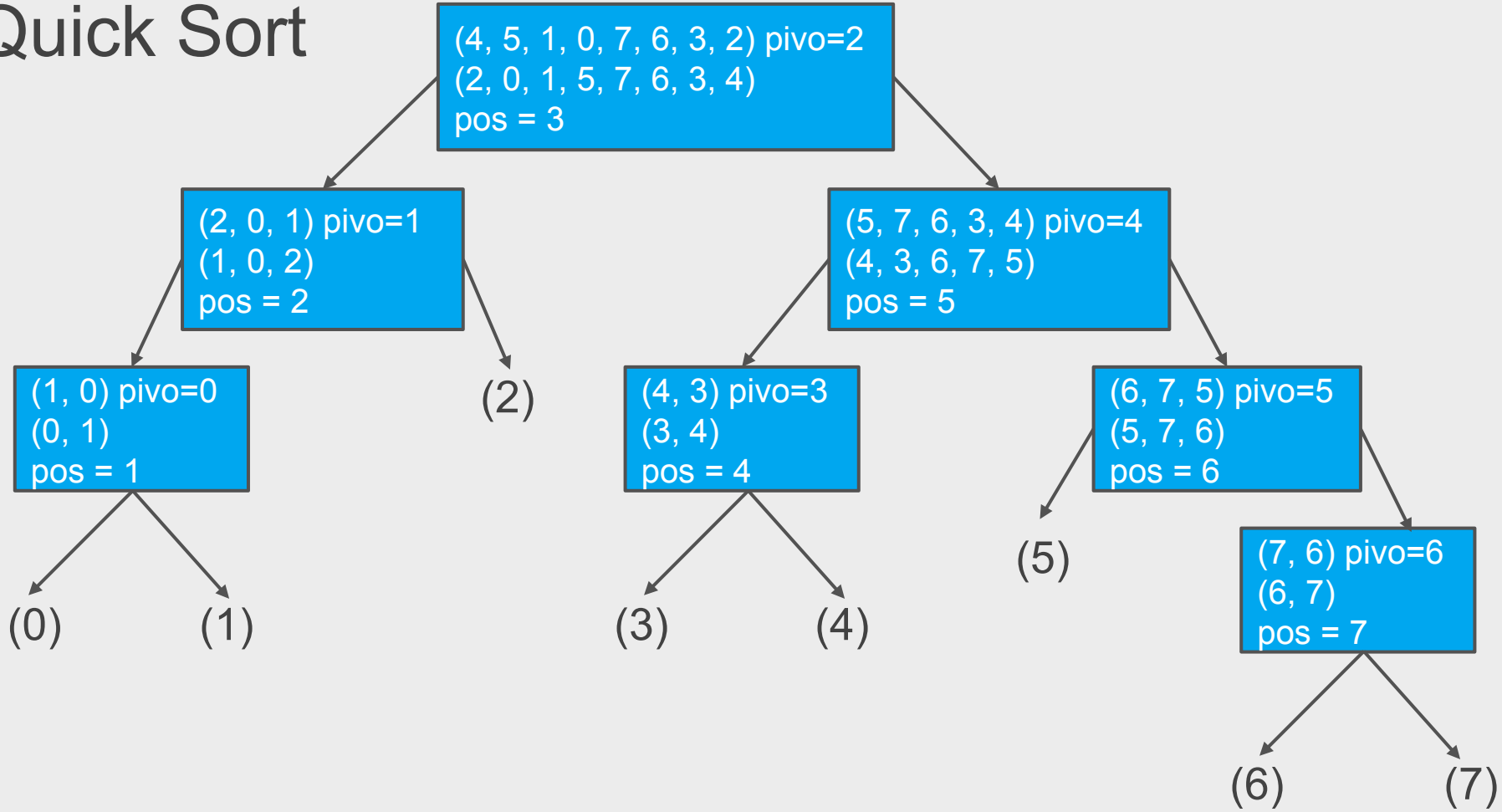
Quick Sort


(4, 5, 1, 0, 7, 6, 3, 2) pivo=2
(2, 0, 1, 5, 7, 6, 3, 4)
pos = 3
(2, 0, 1) pivo=1
(1, 0, 2)
pos = 2
(5, 7, 6, 3, 4) pivo=4
(4, 3, 6, 7, 5)
pos = 5
(1, 0) pivo=0
(0, 1)
pos = 1
(2)
(4, 3) pivo=3
(3, 4)
pos = 4
(6, 7, 5) pivo=5
(5, 7, 6)
pos = 6
(0)
(1)
(3)
(4)

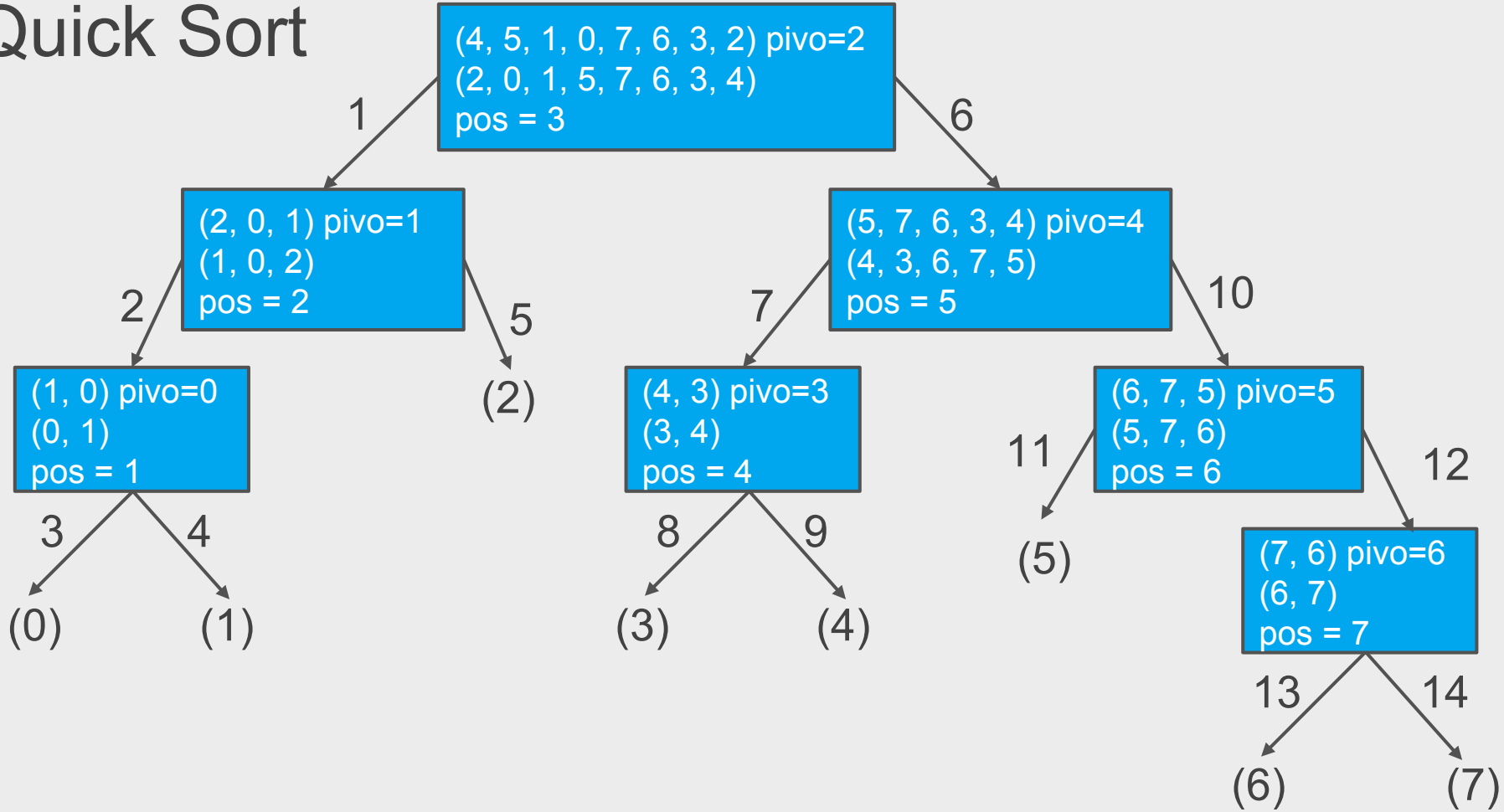
Quick Sort


(4, 5, 1, 0, 7, 6, 3, 2) pivo=2
(2, 0, 1, 5, 7, 6, 3, 4)
pos = 3
(2, 0, 1) pivo=1
(1, 0, 2)
pos = 2
(5, 7, 6, 3, 4) pivo=4
(4, 3, 6, 7, 5)
pos = 5
(1, 0) pivo=0
(0, 1)
pos = 1
(2)
(4, 3) pivo=3
(3, 4)
pos = 4
(6, 7, 5) pivo=5
(5, 7, 6)
pos = 6
(0)
(1)
(3)
(4)
(5)

Quick Sort


(4, 5, 1, 0, 7, 6, 3, 2) pivo=2
(2, 0, 1, 5, 7, 6, 3, 4)
pos = 3
(2, 0, 1) pivo=1
(1, 0, 2)
pos = 2
(5, 7, 6, 3, 4) pivo=4
(4, 3, 6, 7, 5)
pos = 5
(1, 0) pivo=0
(0, 1)
pos = 1
(2)
(4, 3) pivo=3
(3, 4)
pos = 4
(6, 7, 5) pivo=5
(5, 7, 6)
pos = 6
(0)
(1)
(3)
(4)
(5)
(7, 6) pivo=6
(6, 7)
pos = 7
(6)
(7)

Quick Sort


(4, 5, 1, 0, 7, 6, 3, 2) pivo=2
(2, 0, 1, 5, 7, 6, 3, 4)
pos = 3
1
6
(2, 0, 1) pivo=1
(1, 0, 2)
pos = 2
(5, 7, 6, 3, 4) pivo=4
(4, 3, 6, 7, 5)
pos = 5
2
5
7
10
(1, 0) pivo=0
(0, 1)
pos = 1
(2)
(4, 3) pivo=3
(3, 4)
pos = 4
(6, 7, 5) pivo=5
(5, 7, 6)
pos = 6
11
12
3
4
8
9
(5)
(7, 6) pivo=6
(6, 7)
pos = 7
(0)
(1)
(3)
(4)
13
14
(6)
(7)

Quick Sort

- Se o QuickSort particionar o vetor de tal forma que cada partição tenha mais ou menos o mesmo tamanho ele é mais eficiente.
- Porém se a partição for muito desigual (`n - 1` de um lado e `1` de outro) o algoritmo é menos ineficiente.
- Quando um vetor já está ordenado ou quase ordenado, ocorre este caso ruim. Por que?

Quick Sort: Tratando o pior caso

- Podemos implementar o QuickSort de tal forma a diminuirmos a chance de ocorrência do pior caso.
- Ao invés de escolhermos o pivô como um elemento de uma posição fixa, podemos escolher como pivô o elemento de uma posição aleatória.
- Podemos usar a função `random.randint(a,b)` da biblioteca `random` que retorna um número de forma aleatória entre `a` e `b`.

Random Quick Sort

- A única diferença é que escolhemos um elemento aleatório.
- Tal elemento é trocado com o que está no fim (será o pivô).

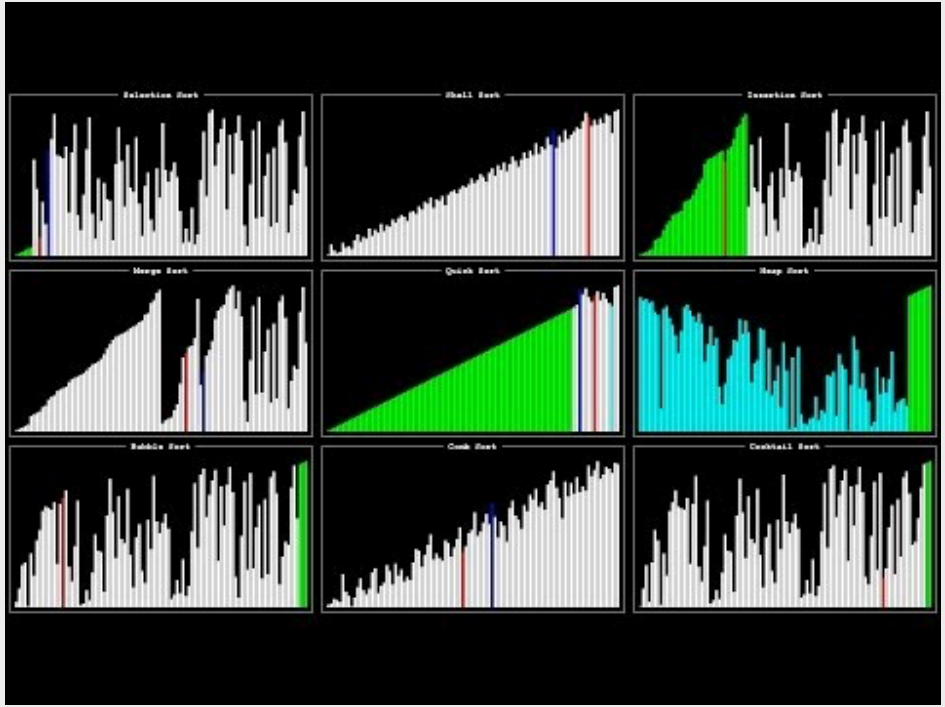
```
import random
def randomQuickSort(v, inicio, fim):
    if (inicio < fim):
        j = random.randint(inicio, fim)
        v[j], v[fim] = v[fim], v[j]
        pos = particiona(v, inicio, fim)
        randomQuickSort(v, inicio, pos-1)
        randomQuickSort(v, pos, fim)
```

Exercícios

1. Aplique o algoritmo de particionamento sobre o vetor `(13,19,9,5,12,21,7,4,11,2,6,6)` com pivô igual a `6`.
2. Qual o valor retornado pelo algoritmo de particionamento se todos os elementos do vetor tiverem valores iguais?
3. Faça uma execução passo-a-passo do `quickSort` com o vetor `(4,3,6,7,9,10,5,8)`.
4. Modifique o algoritmo `quickSort` para ordenar vetores em ordem decrescente.

Visualization and Comparison of Sorting Algorithms

https://www.youtube.com/watch?v=ZZuD6iUe3Pc



Referências

- Os slides dessa aula foram baseados no material de referência MC102 dos Profs. Sandra Ávila e Eduardo Xavier (IC/Unicamp), com modificações e exercícios implementados pelo prof. Marcelo Jara (IC, Unicamp, PNPD).
- Livros e Outras referencias:
  - Python Programming Fundamentals, Undergraduate Topics in Computer Science (UTiCS), por Kent D. Lee, Springer London (2011). Chapter 5. Defining Functions, section 5.9 Recursive Functions.