# 18 Aula 18: 10/OUT/2019

# 18.1 Aula passada

- intercale()
- mergesort: recursão, iteração, consumo de tempo  ${\tt n}$  lg  ${\tt n}$
- algoritmos por divisão-e-conquista

# 18.2 Hoje

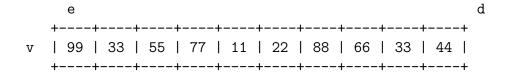
**Motivação** Um problema do mergesort é o consumo de espaço extra de O(n). Será que podemos ter um algoritmo de ordenação com consumo de tempo  $O(n \lg n)$  e memória O(1)?

- separe()
- quicksort()
- algoritmo por divisão-e-conquista

# 18.3 Separação

#### 18.3.1 Problema

Rearranjar uma dada lista v[e:d] e retorna um índice m,  $e \le m < d$ , tais que  $v[e:m] \le v[m] < v[m+1:d]$ . Entra:



Sai:

#### 18.3.2 Solução CLRS

#### 18.3.3 Correção

Em #A# vale que  $v[e:i] \le x < v[i+1:j]$ 

## 18.3.4 Solução PSADS

```
def separeR(e,d,v):
x = v[e] \# pivo
e = p+1
d = r-1
while e <= d:
    while e \le d and v[e] \le x:
         e += 1
    while d \ge e and v[d] \ge x:
         d = 1
    \# v[e] > x \text{ and } v[e : e-1] \le x
    \# v[d] \le x \text{ and } v[d+1: d] > x
    if e < d:
        v[e], v[d] = v[d], v[e]
         e += 1
         d = 1
v[e],v[d] = v[d],v[e]
return d
```

### 18.3.5 Consumo de tempo

O consumo de tempo da função separe() é proporcional a n := r-p.

O consumo de tempo da função separe() é O(n).

## 18.4 Quicksort

```
def quick_sort(e,d,v):
 '''(int,int,list) -> None

 Recebe uma lista v[e:d] e rearranja seu elementos
 de maneira que fique crescente.
 '''
 if p < r-1:
     q = separe(e,d,v)
     quick_sort(e,m,v)
     quick_sort(m,d,v)</pre>
```

## 18.4.1 Correção

A função está correta?

A correção da função, que se apoia na correção da função **separe()** pode ser demonstrada por indução em n := r-p.

### 18.4.2 Consumo de tempo

Desenhar "árvore da recursão" e verificar que no **pior caso** há cerca de **n** níveis e o consumo de tempo desses níveis é **n**, **n**-1,...,1.

Desenhar "árvore da recursão" e verificar que no **melhor caso** há cerca de  $\log_2 n$  níveis e o consumo de tempo de cada um desses níveis é aproximadamente n.

#### 18.4.3 Conclusões

O consumo da função quick sort() no pior caso é proporcional a n².

O consumo da função quick\_sort() no pior caso é  $O(n^2)$ .

O consumo da função quick\_sort() no melhor caso é proporcional a n $\log n.$ 

O consumo da função quick\_sort() no melhor caso é  $O(n \log n)$ .

# 18.5 Divisão e conquista

Algoritmos por divisão-e-conquista, tipicamente, têm três passos em cada nível da recursão:

- dividir: o problema é dividido em subproblemas de tamanho menor
- conquistar: os subproblema são resolvidos recursivamente e subproblemas "pequenos" são resolvido diretamente.
- combinar: as soluções dos subproblemas são combinadas para obter uma solução do problema original

## 18.6 Versão iterativa

```
def quick_sort(v):
 n = len(v)
 p = 0
 r = n
 pilha = Stack()
 pilha.push((e,d))
 while pilha.isEmpty():
     e, r = pilha.pop()
     if p < r-1: # r-p > 1 ou n > 1
         q = separe(e, d, v)
         # segmento inicial
         pilha.push((e, q))
         # segmente finac
         pilha.push((q+1, d))
```

#### 18.7 k-ésimo menor elemento

**Problema.** encontrar o k-ésimo menor elemento de uma lista v[0:n] supondo  $k \le n$ .

Solução inspirada em separe(). Ao final o k-ésimo menor elemento está na posição v[].

```
def k_esimo(k, e, d, v):
 q = separe(e, d, v)
 if q == k-1: return
 if q >= k:
     return k_esimo(k, e, m, v)
 return k_esimo(k, q+1, d, v)
```