# 17 Aula 17: 8/OUT/2019

## 17.1 Aulas passadas

```
* recursão (várias)
* busca sequencial e binária (aula 15)
* aulas passadas: Notação assintótica, ordenação por inserção (aula 16)
```

## 17.2 Hoje

- intercalação
- mergesort: versão recursiva, versão iterativa
- análise do consumo de tempo: experimental e analítica
- algoritmo por **divisão-e-conquista** (este modelo será o EP11 e EPX2)

## 17.3 Intercalação

### 17.3.1 Problema

Dados `v[e:m]` e `v[m:d]` crescentes, rearranjar `v[e:d]` de modo que ele fique em ordem crescente.

### 17.3.2 Solução

```
#-----------------------------------------------
def intercale(e, m, d, v):
    ''' (int,int,int,list) -> None

    Recebe uma lista v tal que

        - v[e:m] é crescente e
        - v[m:e] é crescente

    e rearranja os elementos de v de tal forma
    que v[e:d] seja crescente.

    Consumo de tempo é O(n) onde n = p-r
    '''
    # crie lista auxiliar
    w = []
    i = p
    j = q
    while i < m and j < d:
        if v[i] < v[j]:
            w.append(v[i])
            i += 1
        else:
            w.append(v[j])
            j += 1

    # copie os elementos que sobraram em v[e:d]
    while i < m:
        w.append(v[i])
        i += 1

    # copie os elementos que sobraram
    while j < r:
        w.append(v[j])
        j += 1

    # copie w[0:r-p] para v[e:d]
    # o mesmo que: v[e:d] = w
    for i in range(r-p):
        v[e+i] = w[i]
```

### 17.3.3 Outra solução

```
#-----------------------------------------------
def intercale(v, e, m, r):
    ve = v[e:m] # clone
    vd = v[m:d] # clone
    vd.reverse() # podemos fazer uma função recursiva...
    w = ve + vd # concatenação
    i = 0
    j = -1 # j = r - p - 1
    for k in range(e,r):
        if w[i] < w[j]:
            v[k] = w[i]
            i += 1
        else:
            v[k] = w[j]
            j -= 1
```

## 17.4 Mergesort

Ordenação por intercalação.

```
def merge_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 = (p+r)//2
        merge_sort(e, m, v)
        merge_sort(m, d, v)
        intercale(e, m, d, v)
```

### 17.4.1 Correção

A função está correta?

A correção da função, que es apoia na correção da função `intercale()` pode ser demonstrada por indução em `n := r-p`.

### 17.4.2 Consumo de tempo

Olhando a "árvore da recursão" vemos que o consumo de tempo em cada nível é proporcional a `n` e que há aproximadamente `lg n` níveis.

## 17.5 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

## 17.6 Mergesort iterativo

Cada iteração intercala blocos de tamanho `b`.

Simular com

```
+----+----+----+----+----+----+----+----+----+
| 55 | 33 | 66 | 44 | 99 | 11 | 77 | 22 | 88 |  b = 1
+----+----+----+----+----+----+----+----+----+
  e    m    d
            e    m    d
                      e    m    d
                                e    m    d
                                          e    m    d # intercala faz nada

+----+----+----+----+----+----+----+----+----+
| 33 | 55 | 44 | 66 | 11 | 99 | 22 | 77 | 88 |  b = 2
+----+----+----+----+----+----+----+----+----+
  e         m         d
                      e         m         d
                                          e         m         d  # intercala faz nada

+----+----+----+----+----+----+----+----+----+
| 33 | 44 | 55 | 66 | 11 | 22 | 77 | 99 | 88 |  b = 4
+----+----+----+----+----+----+----+----+----+
  e                   m                   d


+----+----+----+----+----+----+----+----+----+
| 11 | 22 | 33 | 44 | 55 | 66 | 77 | 99 | 88 |  b = 4
+----+----+----+----+----+----+----+----+----+
  e                   m                   e
                                          e                   m                   d


+----+----+----+----+----+----+----+----+----+
| 11 | 22 | 33 | 44 | 55 | 66 | 77 | 99 | 88 |  b = 4
+----+----+----+----+----+----+----+----+----+
  e                                     m    d
```

```
def merge_sortI(v):
    '''(list) -> None

    Recebe uma lista v e rearranja seu elementos de
    maneira que fiquem crescente.

    É uma implementação iterativa do Mergesort.
    '''
    n = len(v)
    b = 1
    while b < n:
        e = 0
        while e + b < n:
            m = e +   b
            d = e + 2*b
            if d > n: d = n # remendo para o último
            intercale(e, m, d, v)
            e = r
        b = 2*b
```