DIVISÃO-E-CONQUISTA E MERGESORT (Cormen, §2.3)

Como dito em [Cormen, 2ª ed., §2.3]:

> "Existem muitas maneiras de projetar algoritmos. A ordenação por inserção usa uma abordagem INCREMENTAL: tendo ordenado o subvetor A[i..j-1], nós inserimos o elemento A[j] na sua posição correta, obtendo o subvetor ordenado A[i..j]."

Outra técnica de projeto de algoritmos, amplamente utilizada, é a DIVISÃO E CONQUISTA: dada uma instância para um problema,

a) Se a instância for trivial, resolva-a diretamente.

b) Se a instância não for trivial:

```
1) DIVISÃO:    divida-a em instâncias menores.
2) CONQUISTA:  resolva essas instâncias menores RECURSIVAMENTE
               (isto é, pela mesma estratégia de divisão e conquista).
3) COMBINAÇÃO: combine as soluções das instâncias menores,
               de forma a obter uma solução para a instância original.
```

ORDENAÇÃO POR ENTRELAÇAMENTO (MERGESORT)

O seguinte algoritmo ordena um vetor por meio da técnica de divisão e conquista:

```
==============================================================================
Algoritmo:    mergesort
Entrada:      vetores V[1..n] e B[1..n], índices "i" e "f".
Pré-condição: 1 <= i <= f <= n ou n = 0.
Saída:        nenhuma.
Pós-condição: se n != 0, então V[i..f] estará ordenado.
------------------------------------------------------------------------------
1. SE i < f
2. | m := i + ⌊(f-i)/2⌋   // = ⌊i + (f-i)/2⌋ = ⌊(i+f)/2⌋.
3. | mergesort(V,B,i,m)
4. | mergesort(V,B,m+1,f)
5. | entrelaçar(V,B,i,m,f)
==============================================================================
```

O cerne desse algoritmo é a função "entrelaçar":

```
================================================================================
Algoritmo:    entrelaçar
Entrada:      vetores V[1..n] e B[1..n], índices "i", "m" e "f".
Pré-condição: 1 <= i <= m < f <= n e trechos V[i..m] e V[m+1..f] ordenados.
Saída:        nenhuma.
Pós-condição: V[i..f] estará ordenado.
--------------------------------------------------------------------------------
a := i, b := m+1, k := i
    /*
     * INVARIANTES: * i   <= a <= m+1
     *              * m+1 <= b <= f+1
     *              * k-i = a-i + b-(m+1)
     *              * B[i..k-1] contém V[i..a-1] U V[m+1..b-1] ordenado.
     *
     * VARIANTES: * m+1-a + f+1-b
     *            * f-k+1
     */
REPITA SEMPRE
| SE V[a] <= V[b]
| | B[k] := V[a]
| | ++k, ++a
| | SE m < a
| | | REPITA
| | | | B[k] := V[b]
| | | | ++k, ++b
| | | ENQUANTO b <= f.
| | | SAIA_DO_LAÇO     // "break"
| SENÃO
| | B[k] := V[b]
| | ++k, ++b
| | SE f < b
| | | REPITA
| | | | B[k] := V[a]
| | | | ++k, ++a
| | | ENQUANTO a <= m.
| | | SAIA_DO_LAÇO     // "break"
k := i
REPITA
| V[k] := B[k]
| ++k
ENQUANTO k <= f.
================================================================================
```

TEMPO DE EXECUÇÃO DO MERGESORT

Como o algoritmo "entrelaçar" executa em tempo $\theta(t)$ sobre um intervalo de tamanho $t = f-i+1 >= 1$, então o tempo de execução do Mergesort é limitado superiormente por

$f(t) = f(\lceil t/2\rceil) + f(\lfloor t/2\rfloor) + a*t$, ---> termo constante englobado em "a*n".

para "t" e "a" grandes o suficiente. O método da árvore de recursão aplicado à função acima nos dá aproximadamente lg(t) níveis ao custo de "a*t" cada, o que nos leva à estimativa de que $f(t) = O(t*\lg t)$, que pode ser provada por indução:

```
1. TESE: f(n) = O(n*lg n).
   ----
================================================================================
ESBOÇO DA PROVA (passo indutivo):

  * TESE: ∀ n >= ..., f(n) <= c*n*lg n.

  * BASE (... <= n <= ...): ...

  * H.I.: ∀ ... <= l < n, f(l) <= c*l*lg l.

  * PASSO INDUTIVO (n >= ... >= 100):

      f(n)  = f(⌈n/2⌉) + f(⌊n/2⌋) + a*n
           <= c*⌈n/2⌉*lg ⌈n/2⌉ + c*⌊n/2⌋*lg ⌊n/2⌋ + a*n   ---> H.I.
           <= c*⌈n/2⌉*lg ⌈n/2⌉ + c*⌊n/2⌋*lg ⌈n/2⌉ + a*n
            = c*(⌈n/2⌉ + ⌊n/2⌋)*lg ⌈n/2⌉ + a*n
            = c*n*lg ⌈n/2⌉ + a*n
           <= c*n*lg(n/2 + 1) + a*n
           <= c*n*lg(n/2 + n/100) + a*n                   ---> n >= 100
            = c*n*lg(50n/100 + n/100) + a*n
            = c*n*lg(n/(100/51)) + a*n
            = c*n*[lg(n) - lg(100/51)] + a*n
            = c*n*lg(n) - c*n*lg(100/51) + a*n
           <= c*n*lg(n), CQD,                             ---> veja abaixo

    onde a última inequação vale porque

      a*n - c*n*lg(100/51) <= 0  <->  a*n <= c*n*lg(100/51)
                                 <->  a <= c*lg(100/51)      ---> n >= 100 > 0
                                 <->  a/lg(100/51) <= c ,

    o que é possível, basta escolher "c" grande o suficiente, já que

      1 = 51/51 < 100/51 < 102/51 = 2

    e portanto 0 = lg(1) < lg(100/51) < lg(2) = 1.

================================================================================
```

EXERCÍCIOS E LEITURA ADICIONAL

Veja os interessantes exercícios do Rudini que podem ser resolvidos "a la" Mergesort:

  http://lia.ufc.br/~rudini/ufc/2014ii/cana.list1.pdf .

Há algumas claras melhorias possíveis no Mergesort:

  a) Muitas cópias são feitas desnecessariamente de um vetor para o outro. Escreva uma versão do algoritmo em que os passos de entrelaçamento são feitos alternando-se entre um vetor e outro.

  b) A versão original do Mergesort não atenta para o fato de que partes consideráveis do vetor podem já estar ordenadas. Faça uma versão do algoritmo que começa dividindo o vetor em trechos já ordenados, e que então entrelaça pares de trechos sucessivos até restar apenas um trecho.

Aqui está uma boa leitura adicional sobre o Mergesort:

  http://dx.doi.org/10.1007/3-540-62592-5_74 .