Construção e Análise de Algoritmos

Discussão: *Mergesort* in-place

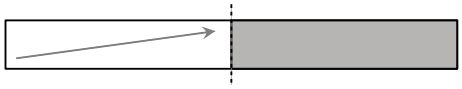
Nós vimos na aula 02 que o algoritmo Mergesort requer o uso de uma memória auxiliar do mesmo tamanho da lista que está sendo ordenada.

Mas, isso só vale para a implementação padrão do procedimento de intercalação.

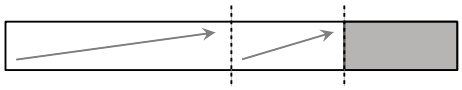
Fazendo as coisas de maneira inteligente, é possível eliminar o uso da memória auxiliar.

Vejamos.

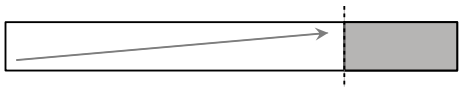
A primeira observação é que a primeira metade da lista pode ser ordenada (via mergesort) utilizando a segunda metade (ainda desordenada) como memória auxiliar.



De maneira análoga, o terceiro quarto da lista pode ser ordenado (via mergesort) utilizando o último quarto como memória auxiliar.



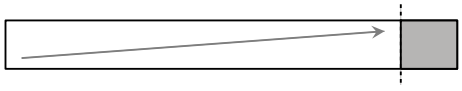
E a ideia agora é fazer a intercalação das duas partes ordenadas A e B, utilizando a parte desordenada C como memória auxiliar.



Se você conseguir fazer isso, então o problema está basicamente resolvido.

Isto é, em seguida nós ordenamos o sétimos oitavo da lista (via mergesort) utilizando o último oitavo como memória auxiliar

e depois repetimos o novo procedimento de intercalação



para aumentar a porção ordenada da lista.

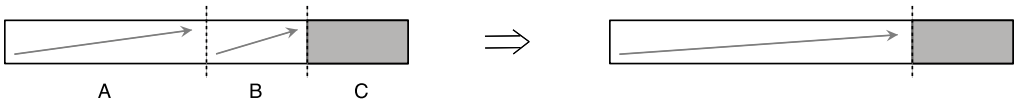
Continuando dessa maneira, a lista inteira ficará completamente ordenada.

A operação chave para essa implementação in-place do algoritmo Mergesort é uma variante do procedimento `Intercalação` que utiliza uma porção da própria lista como memória auxiliar.

A seguir, nós vamos apresentar duas soluções para a implementação desse procedimento — com o pseudo-código para a segunda delas.

Solução 1: (movimentação de blocos)

Há alguns dias, nós vimos uma solução para o seguinte problema de intercalação

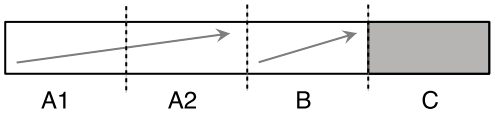


Isto é, intercalar os elementos das porções ordenadas A e B da lista, utilizando a porção C como memória auxiliar.

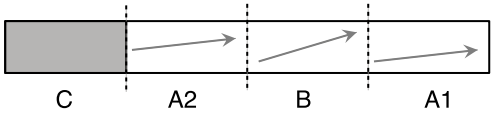
A solução que nós apresentamos era baseada na movimentação individual de elementos.

A seguir, nós vamos esboçar uma solução alternativa baseada na movimentação de blocos inteiros.

O procedimento começa com a divisão da lista em 4 blocos

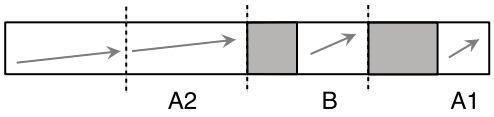


O primeiro passo consiste em mover o bloco A1 para a última posição

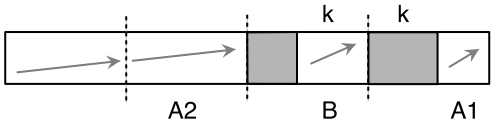


Daí, nós podemos iniciar a intercalação de A1 e B, movendo os elementos para o início da lisaa.

Quando todas as posições do primeiro bloco tiverem sido utilizadas, nós teremos uma situação mais ou menos assim:

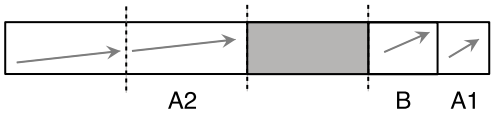


A observação chave, a seguir, é que a porção restante do bloco B possui exatamente o mesmo tamanho da parte vazia do último bloco

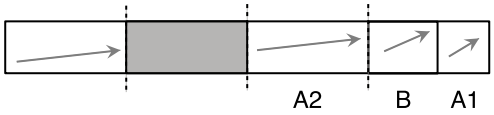


(porque?)

Então, nós movemos a porção restante de B para o último bloco



E, em seguida, nós movemos a porção A2 para o terceiro bloco

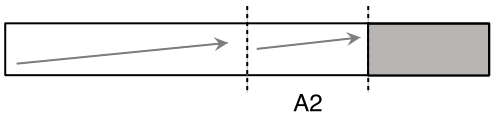


Pronto, agora o procedimento de intercalação pode continuar.

Caso os elementos da porção A1 se esgotem, a intercalação continua com os elementos de A2.

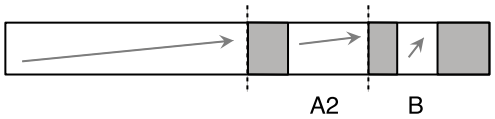
Quando todas as posições do segundo bloco tiverem sido ocupadas, existem basicamente dois casos possíveis.

caso 1: os elementos de B foram esgotados

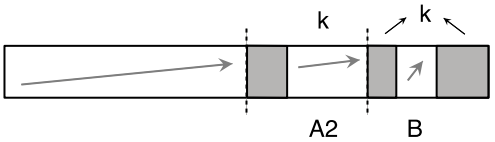


Nesse caso, não há nada a fazer: o procedimento de intercalação acabou.

caso 2: ainda existem elementos em B

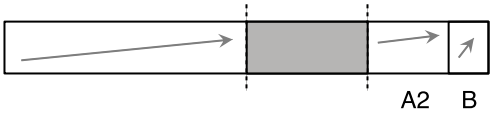


Nesse caso, a porção que resta de A2 tem o mesmo tamanho que a soma das partes vazias no último bloco

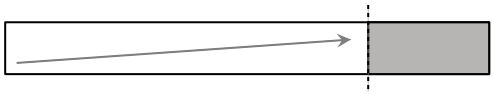


(porque?)

A ideia, então, é deslocar a porção restante de B para o fim do último bloco, e deslocar a porção restante de A2 também para o último bloco.



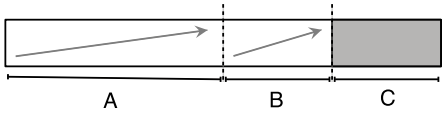
Pronto, agora o procedimento de intercalação pode continuar e completar a tarefa.



Você consegue implementar essa solução?

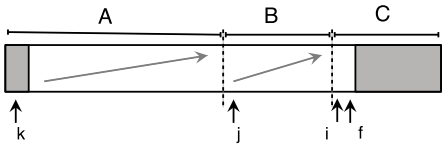
Solução 2: (movimentação de elementos)

A ideia dessa solução é:



- mover os elementos de A um a um para a parte auxiliar C, sempre que necessário, para dar espaço aos elementos que estão sendo intercalados
- o menor elemento de A (que sempre estará na parte C) é comparado com o menor elemento de B, e o menor deles é movido para frente

No início do procedimento, o primeiro elemento de A é movido para a parte C, e os índices i, j, k, f são incializados da seguinte maneira:



onde

- `k` é a posição onde o próximo elemento da intercalação será colocado
- `j` é a posição do menor elemento de B
- `i` é a posição do menor elemento de A
- `f` é a posição do último elemento de A em C

(Note que a porção de A na parte auxiliar C será uma lista circular.)

A seguir começam as comparações e movimentações dos elementos.

Nós também vamos utilizar um contador `c` para registrar quantos elementos de A já foram intercalados.

```
Procedimento Intercalacao-inplace ( V[1..n] )
 {
    Troca (1, 3n/4 + 1)
    k <-- 1;   j <-- n/2 + 1;   i,f <-- 3n/4 + 1

    c <-- 0     // conta quantos elementos de A já foram intercalados

    Enquanto ( c < n/2  e  j <= 3n/4 )
     {
        Se ( V[j] < V[i] )    // o elemento de B é menor
         {
            Troca (k,j);  k++; j++;

            Se ( j < 3n/4)
             {
               f++;  Se ( f > n )  f <-- 3n/4 + 1
               Troca (f,k)
             {
         }
        Senão                 // o elemento de A é menor
         {
            c++
            Troca (k,i);  k++;
            i++;  Se ( i > n )  i <-- 3n/4 + 1

            Se ( k <= n/2 )   // se existem elementos de A na parte da frente
             {
               f++;  Se ( f > n )  f <-- 3n/4 + 1
               Troca (f,k)
             }
         }
     }

    Se ( c = n/2 )
     {
       // não há nada a fazer, os elementos de A já foram todos intercalados
       // e os elementos restantes de B já estão no lugar correto
     }

    Se ( j > 3n/4 )  // os elementos de B foram todos intercalados
                     // basta mover os elementos restantes de A para frente
     {
        Enquanto ( c < n/2 )
         {
            c++
            Troca (k,i);  k++;
            i++;  Se ( i > n )  i <-- 3n/4 + 1
         }
     }
 }
```