Construção e Análise de Algoritmos

aula 02: O algoritmo de ordenação *Mergesort*

1 Introdução

Na aula passada, nós começamos a nossa discussão com um dos algoritmos de ordenação mais simples que existem: o *algoritmo da bolha*.

```
Procedimento ord-Bolha ( V[1..n] )
  {
      Para k = 1 Até n-1
          Para i = 1 Até n-k
              Se ( V[i] > V[i+1] )
               {
                  aux ← V[i];   V[i] ← V[i+1];   V[i+1] ← aux;
               }
  }
```

Nós vimos que esse algoritmo executa em tempo $O(n^2)$, o que não é lá essas coisas ...

Mas, utilizando a ideia de executar o algoritmo diversas vezes sobre porções diferentes da lista, nós conseguimos reduzir esse tempo para $O(n \log^2 n)$, o que já é muito bom.

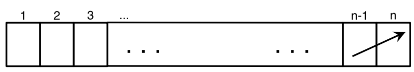
Hoje, nós vamos utilizar essa ideia novamente.

Antes disso, é útil ver que o algoritmo da bolha também pode ser colocado nesse formato.

A ideia é simples.

Suponha que nós executamos o algoritmo da bolha sobre a porção `V[n-1..n]` da lista.

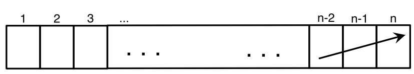
Então, essa porção fica ordenada



e, de fato, basta uma varredura para fazer isso.

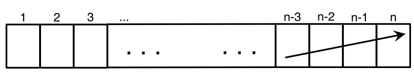
A seguir, suponha que nós executamos o algoritmo sobre a porção `V[n-2..n]`.

Então, a porção ordenada aumenta um pouquinho mais

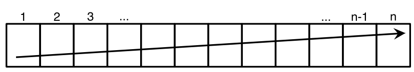


e, outra vez, apenas uma varredura basta.

Agora, é fácil ver que repetindo esse procedimento várias vezes, a lista inteira fica ordenada.



. . .



Essa observação nos permite escrever uma versão recursiva do algoritmo da bolha.

```
Procedimento ord-Bolha-Rec ( V[1..n], m )
  {
     Se (m = 0) Retorna;
     Para i = m Até n-1
         Se ( V[i] > V[i+1] )
          {
             aux ← V[i];   V[i] ← V[i+1];   V[i+1] ← aux;
          }
     ord-Bolha-Rec ( V[1..n], m-1 );
  }
```

onde a chamada é `ord-Bolha-Rec ( V[1..n], n-1 )`.

Mas, infelizmente, isso não reduz o tempo de execução do algoritmo ...

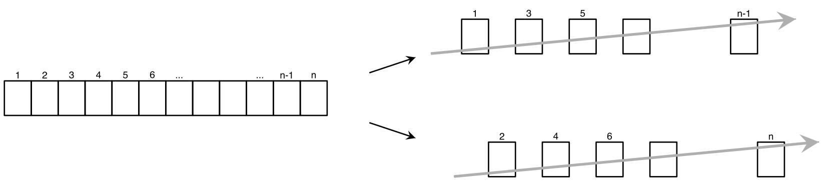
Como a primeira chamada recursiva executa em tempo 1, a segunda executa em tempo 2, e assim por diante, o tempo total do algoritmo é dado por:

$$1 \ + \ 2 \ + \ 3 \ + \ \ldots \ + \ n-1 \quad = \quad O(n^2)$$

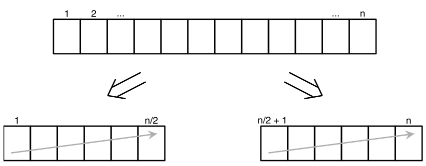
A seguir, nós vamos ver uma ideia melhor.

2 Ordenação por intercalação

Relembre que, na aula passada, nós dividimos os elementos da lista em duas metades, e ordenamos cada uma delas em separado.

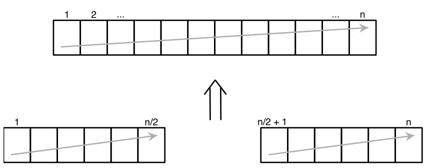


Mas, se pensarmos um pouquinho sobre o assunto, a maneira mais natural de fazer isso consiste em quebrar a lista exatamente no meio.



A ordenação das duas metades, é claro, não deixa a lista completamente ordenada.

E o procedimento de combinar as duas metades ordenadas em uma única lista ordenada é chamado de *intercalação*.



A itercalação de duas listas ordenadas é um procedimento relativamente simples.

```
Procedimento Intercalação ( V[1..m], V[m+1..n] )
 {
1.     i ← 1;   j ← m+1;   k ← 1

2.     Enquanto ( i ≤ m   e   j ≤ n )
       {
3.         Se ( V[i] > V[j] )  {
4.             A[k] ← V[i];   i++;   k++
5.         }

6.         Senão  {
7.             A[k] ← V[j];   j++;   k++
8.         }
       }

9.     Enquanto ( i ≤ m )
       {
10.        A[k] ← V[i];   i++;   k++
       }

11.    Enquanto ( j ≤ n )
       {
12.        A[k] ← V[j];   j++;   k++
       }

13.    V[1..n] ← A[1..n]   (essa instrução é, na realidade, um laço)
 }
```

Para determinar o tempo de execução desse procedimento, nós vamos assumir que

- os blocos formados pelas linhas `3-5` e `6-8` executam em tempo $O(1)$ (isto é, uma unidade de tempo)
- as linhas `10` e `12` também executam em tempo $O(1)$

Dessa maneira, o tempo de execução do algoritmo é dado pela soma dos números de iterações dos laços das linhas `2`, `9` e `11`, mais o laço da linha `13`.

Em princípio, não é possível determinar o número exato de iterações dos primeiros três laços, pois isso depende dos números que aparecem na lista. (Porque?)

Mas, como cada iteração desses laços coloca um elemento a mais na lista auxiliar `A`, e nós temos um total de n elementos, nós podemos deduzir que a soma das iterações é igual a n.

Finalmente, como o laço da linha `13` realiza n iterações, nós temos que o procedimento `Intercalação` executa em tempo

$$n \ + \ n \quad = \quad 2n \quad = \quad O(n)$$

Agora, é fácil determinar o tempo total de execução da nova estratégia de ordenação.

A ordenação de cada metade pelo algoritmo da bolha leva tempo $n^2/4$.

E, adicionando o tempo da intercalação, nós chegamos a

$$2 \cdot \frac{n^2}{4} \ + \ n \quad = \quad \frac{n^2}{2} \ + \ n$$

Ou seja, o tempo de execução do algoritmo da bolha foi reduzido aproximadamente à metade.

3 O algoritmo *Mergesort*

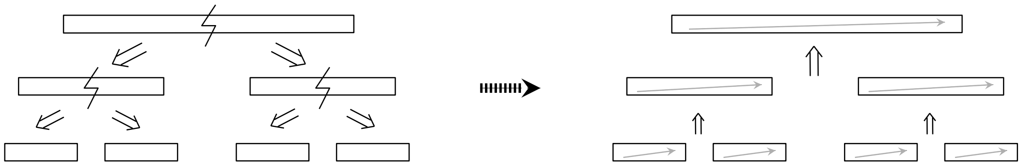
Vejamos o que nós acabamos de descobrir.

A estratégia de

A. quebrar a lista no meio
B. ordenar as duas metades em separado
C. intercalar as metades ordenadas resultantes

reduz o tempo de execução do algoritmo da bolha (aprox.) à metade.

Ora, mas se isso é o caso, então é natural aplicar a estratégia novamente sobre as duas metades.



Fazendo isso, o algoritmo agora realiza as seguintes operações:

- ordenar 4 sublistas de tamanho $n/4$ (pelo algoritmo da bolha)
- intercalar 2 pares de sublistas de tamanho $n/4$
- intercalar duas sublistas de tamanho $n/2$

e o seu tempo de execução é dado por

$$4 \cdot \frac{n^2}{16} \; + \; 2 \cdot \frac{n}{2} \; + \; n \quad = \quad \frac{n^2}{4} \; + \; 2n$$

Ou seja, o tempo de execução do algoritmo foi reduzido mais uma vez à metade!

Agora, não é difícil advinhar o que acontece quando aplicamos a estratégia de divisão também às sublistas de tamanho $n/4$:

- o algoritmo irá realizar 8 ordenações e 4 + 2 + 1 intercalações

e irá executar em tempo

$$8 \cdot \frac{n^2}{64} \; + \; 4 \cdot \frac{n}{4} \; + \; 2 \cdot \frac{n}{2} \; + \; n \quad = \quad \frac{n^2}{8} \; + \; 3n$$

Abaixo nós temos os tempos de execução das 4 versões do algoritmo que vimos até agora:

n^2	`(algor. da bolha)`
$\frac{n^2}{2} + n$	`(1 divisão)`
$\frac{n^2}{4} + n + n$	`(2 divisões)`
$\frac{n^2}{8} + n + n + n$	`(3 divisões)`

O que está acontecendo aqui é que a cada vez que nós aplicamos a estratégia de divisão, o termo quadrático se reduz à metade e um novo termo n é acrescentado à expressão.

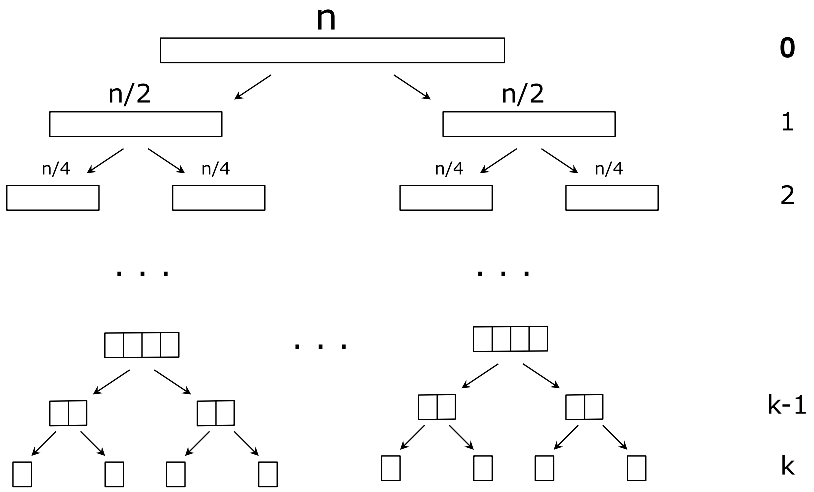
Logo, faz sentido continuar aplicando a estratégia de divisão até que o termo quadrático seja igual a n:

$$\frac{n^2}{2^k} \quad = \quad n$$

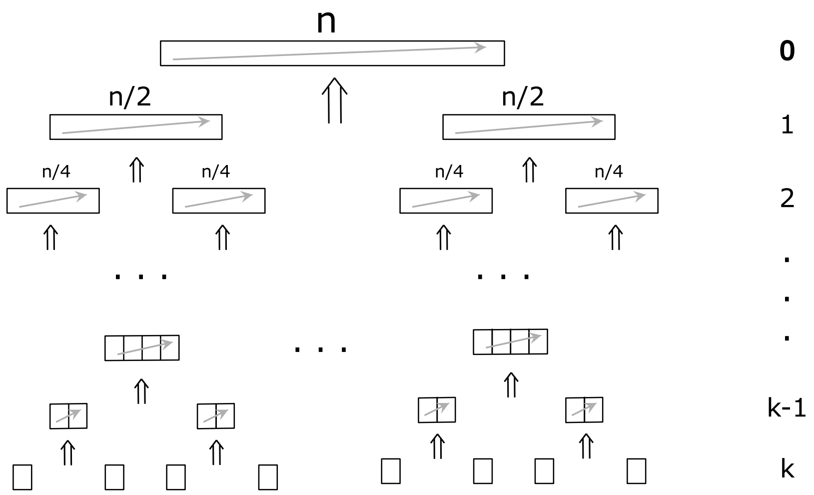
o que corresponde a $k \;=\; \log_2 n$.

Ou seja, para obter o maior benefício possível com a estratégia de divisão, nós devemos aplicá-la $\log_2 n$ vezes.

Mas, quando nós dividimos uma lista de tamanho n ao meio $\log_2 n$ vezes, nós essencialmente reduzimos todas as sublistas a apenas um elemento



E, quando nós fazemos isso, o algoritmo se resume a realizar intercalações



Isto é, não existem mais ordenações pelo método da bolha.

Esse algoritmo é conhecido como o algoritmo de ordenação *Mergesort*.

Seguindo o padrão que vimos acima, o seu tempo de execução é dado por

$$\frac{n^2}{2^{\log_2 n}} \;+\; \underbrace{n \;+\; \ldots \;+\; n}_{\log_2 n} \quad = \quad n\big(\log_2 n \;+\; 1\big) \quad = \quad O(n \log n)$$

o que corresponde ao melhor algoritmo de ordenação que nós vimos até agora.

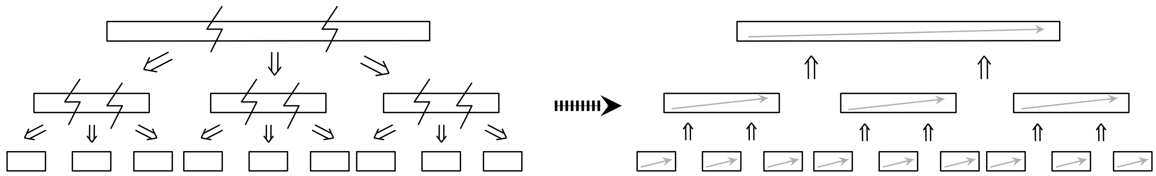
Devido às divisões sucessivas, a maneira mais fácil de implementar o algoritmo Mergesort é na forma de um procedimento recursivo.

```
Procedimento Mergesort-Rec ( V[i..j] )
  {
      Se (i = j)  Retorna;

      k ← (i+j) / 2

      Mergesort-Rec ( V[i..k] )

      Mergesort-Rec ( V[k+1..j] )

      Intercalacao ( V[1..k], V[k+1..j] );
  }
```

Exercícios

1. **3-Mergesort**

   Imagine que alguém decide modificar o algoritmo Mergesort para que ele divida a lista em 3 partes iguais a cada passo.



   a) Apresente o pseudocódigo do procedimento `3-Intercalação` e do procedimento principal do algoritmo `3-Mergesort`.

   b) Estime o tempo de execução do algoritmo `3-Mergesort`.

   c) Você acha que, na prática, essa variante executa mais rápido que o algoritmo original? Porque?

2. **Mergesort paralelo**

   Uma das vantagens da técnica de divisão é que, ao quebrar o problema original em múltiplos subproblemas, esses subproblemas podem ser resolvidos em paralelo por uma máquina com múltiplos processadores.

   Imagine que o algoritmo Mergesort é adaptado para executar em uma máquina com múltiplos processadores, para aproveitar ao máximo as oportunidades de paralelização.

   Mais especificamente, note que todo trabalho de ordenação da lista é realizado pelo procedimento `Intercalação`. Então, a ideia é que, sempre que possível, as chamadas a esse procedimento são executadas em paralelo.

   a) Estime o tempo de execução do algoritmo em uma máquina com 2 processadores.

   b) Estime o tempo de execução do algoritmo em uma máquina com n processadores.

3. **Algoritmo de desordenação** (OPCIONAL)

   Suponha que a linha `3` do procedimento `Intercalação` é trocada pelo seguinte fragmento de código:

```
3′.        lance uma moeda honesta
3″.        Se ( o resultado é cara )   {
```

   Agora, imagine que o algoritmo Mergesort é executado com essa variante do procedimento `Intercalação`.

   Esse algoritmo produz uma configuração perfeitamente aleatória da lista de entrada?

   (i.e., todas as configurações possíveis da lista têm a mesma probabilidade de serem produzidas pelo algoritmo?)

   Dica: Examine o comportamento do algoritmo com listas de tamanho 2,3,4.