Agora nós vamos utilizar outra ideia.

Quer dizer, a gente começa perguntando

- *Qual é a ineficiência básica do algoritmo da bolha?*

Porque, se a gente consegue remover essa ineficiência, então o algoritmo pode ficar mais rápido.

Bom, a ineficiência básica do algoritmo da bolha é a seguinte

→ *os elementos andam apenas uma posição de cada vez*

Porque o que faz os elementos andarem é a troca.

E a troca sempre troca dois elementos adjacentes.

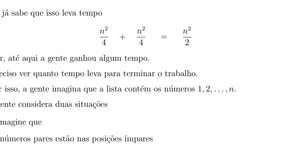
Certo.

Então a solução é simples.

Quer dizer, a ideia é trocar elementos que estejam mais distantes um do outro.

A versão mais simples dessa ideia consiste em comparar e trocar elementos à distância 2

Na prática, isso corresponde a executar o algoritmo da bolha sobre elementos ímpares e elementos pares, em separado



Em geral, isso ainda não resolve o problema completamente.

Mas daí, a gente executa varreduras até que a lista fique completamente ordenada

Análise

Mas será que isso é uma boa ideia?

Bom, a primeira observação é que a gente está aplicando a estratégia da divisão outra vez

→ *nós executamos o algoritmo da bolha uma vez em cada metade da lista*

E a gente já sabe que isso leva tempo

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

Quer dizer, até aqui a gente ganhou algum tempo.

Mas, é preciso ver quanto tempo leva para terminar o trabalho.

Para fazer isso, a gente imagina que a lista contém os números $1, 2, \ldots, n$.

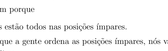
E daí, a gente considera duas situações

caso 1: Imagine que

- os números pares estão nas posições ímpares
- os números ímpares estão nas posições pares

— (*em uma ordem qualquer*)

Então, após a ordenação das posições ímpares e das posições pares, a lista fica assim



E aqui, basta uma varredura para colocar a lista em ordem.

O que leva tempo $O(n)$.

Logo, nesse caso, o tempo de execução do algoritmo é

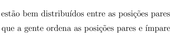
$$\frac{n^2}{2} \quad + \quad n$$

caso 2: Imagine que

- os números $1, 2, \ldots, \frac{n}{2}$ estão nas posições ímpares
- os números $\frac{n}{2}+1, \ldots, n$ estão nas posições pares

— (*em uma ordem qualquer*)

Então, após a ordenação das posições ímpares e das posições pares, a lista fica assim



E agora, é preciso fazer varreduras para colocar a lista em ordem.

Quantas varreduras?

Bom, vamos concentrar a nossa atenção no elemento $n/2$.

Esse elemento está a $n/2$ posições de distância do seu lugar correto — (*aprox.*)

E a cada varredura ele anda uma posição nessa direção — (*porque?..*)

Logo, vão ser necessárias $n/2$ varreduras para colocar a lista em ordem — (*aprox.*)

E o tempo de execução do algoritmo nesse caso é

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

Quer dizer, a gente não ganhou nada.

O algoritmo bolha-32

Bom, a gente ainda não ganhou nada.

Mas, a gente ainda pode ganhar alguma coisa.

Quer dizer, a gente pode ganhar entendimento.

E o entendimento vem quando a gente faz as seguintes perguntas

- *O que é que torna o melhor caso bom?*
  O que é que torna o pior caso ruim?

Bom, é mais fácil começar com a segunda pergunta.

Quer dizer, o pior caso é ruim porque

1. Os números pequenos estão todos nas posições ímpares.
   Daí que, na hora em que a gente ordena as posições ímpares, nós vamos ter números pequenos no final da lista.
2. Os números grandes estão todos nas posições pares.
   Daí que, na hora em que a gente ordena as posições pares, nós vamos ter números grandes no início da lista.

E daí, é preciso fazer um monte de varreduras para colocar a lista em ordem.

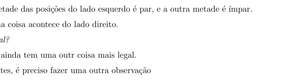
Por outro lado, o melhor caso é bom porque

1. Os números pequenos estão bem distribuídos entre as posições pares e ímpares.
   Daí que, na hora em que a gente ordena as posições pares e ímpares, eles vão parar todos no início da lista.
2. Os números grandes estão bem distribuídos entre as posições pares e ímpares.
   Daí que, na hora em que a gente ordena as posições pares e ímpares, eles vão parar todos no fim da lista.

Legal.

Agora que a gente entendeu, nós podemos tentar forçar que o melhor caso aconteça.

Quer dizer, a gente pode adicionar uma etapa ao algoritmo, que prepara o melhor caso para a fase de ordenação



E aqui é a hora para uma grande esperteza.

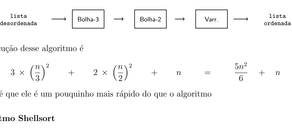
Quer dizer, a ideia é colocar uma ordenação da bolha com salto 3 na etapa preliminar

Porque?

Bom, aqui há mais coisa pra gente entender outra vez.

Quer dizer, na prática, o algoritmo da bolha com salto 3

- ordena as posições múltiplas de 3 — (*i.e., da forma* $3k$)
- ordena as posições da forma $3k+1$
- ordena as posições da forma $3k+2$



E isso faz com que, em cada lista

- os pequenos fiquem do lado esquerdo
- e os grandes fiquem do lado direito

Mas, agora veja quem está do lado esquerdo.

Sim! a metade das posições do lado esquerdo é par, e a outra metade é ímpar.

E a mesma coisa acontece do lado direito.

Não é legal?

Sim, mas ainda tem uma outr coisa mais legal.

Só que antes, é preciso fazer uma outra observação

→ *As ordenações feitas pela cairinha* `Bolha-2`
não desfazem as ordenações feitas pela cairinha `Bolha-3`

Quer dizer, as posições da forma $3k$, $3k+1$, $3k+2$ continuam ordenadas após a execução do `Bolha-2`.

Porque?

Bom, isso é um excelente exercício para o pensamento analítico — (*e será deixado pra você ...*)

O que é importante é que nesse ponto, nós temos um monte de relações de ordem

E agora, a gente concentra a atenção sobre o elemento `x` que está na posição 8

Quer dizer,

- o `x` é menor do que o elemento da posição 11, e todos que estão à direita dele
  — (*porque a linha* $3k+2$ *está ordenada*)
- o `x` é menor do que o elemento da posição 10, e todos que estão à direita dele
  — (*porque as posições pares e a linha* $3k+1$ *estão ordenadas*)
- o `x` é menor do que o elemento da posição 12, e todos que estão à direita dele
  — (*porque as posições pares e a linha* $3k$ *estão ordenadas*)

A gente também pode fazer um raciocínio semelhante para a esquerda.

E a partir daí, a gente conclui que

- ou o elemento `x` está na posição correta
- ou ele está ao lado da posição correta

A mesma coisa, claro, vale para todos os elementos da lista.

Logo, apenas uma varredura basta para deixar a lista completamente ordenada.

Não é legal?

Sim, e abaixo nós temos o esquema completo do algoritmo Bolha-32

O tempo de execução desse algoritmo é

$$3 \times \left(\frac{n}{3}\right)^2 \quad + \quad 2 \times \left(\frac{n}{2}\right)^2 \quad + \quad n \quad = \quad \frac{5n^2}{6} \quad + \quad n$$

E aqui a gente vê que ele é um pouquinho mais rápido do que o algoritmo

O algoritmo Shellsort

(. . .)