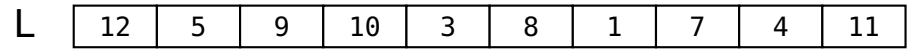
Construção e Análise de Algoritmos

aula 09: O problema da mediana

1 Introdução

Imagine que L é uma lista de tamanho n.

Por exemplo,



A tarefa consiste em

→ *Encontrar a mediana da lista* L

Bom, aqui nós podemos supor que n é par.

Daí que, a mediana é o elemento que é maior do que exatamente $n/2$ elementos da lista.

Certo.

Mas, como é que a gente encontra esse elemento ?

Bom, isso é fácil.

Quer dizer, basta ordenar a lista e examinar a posição $\frac{n}{2} + 1$.

Isso resolve o problema em tempo $O(n \log n)$.

Legal.

Mas, será que a gente consegue fazer melhor do que isso ?

Quer dizer, intuitivamente, a ordenação faz mais trabalho do que devia.

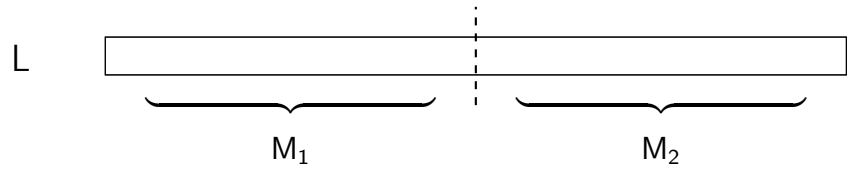
Daí que, deve ser possível fazer melhor do que isso.

É isso o que nós vamos ver na aula de hoje.

2 O algoritmo de divisão e conquista

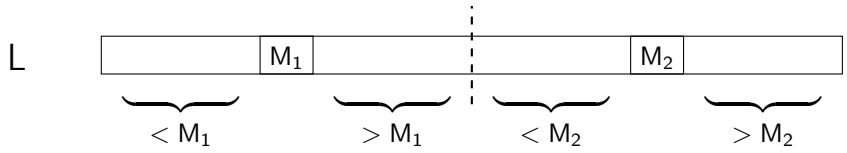
A nossa primeira tentativa é aplicar o método da divisão e conquista.

E para isso, a gente resolve o problema nas duas metades da lista



Mas, ninguém garante que M_1 ou M_2 é a mediana da lista inteira.

Então, a gente começa particionando as duas metades utilizando M_1 e M_2 como pivôs



E daí, a gente raciocina assim

- Suponha que $M_1 < M_2$.

  Então,

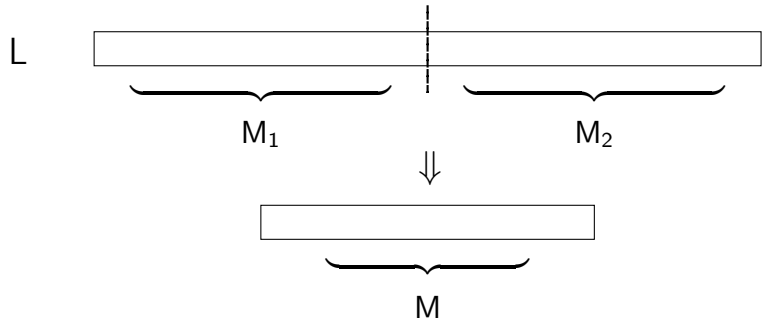
  - todo elemento à esquerda de M_1 é menor do que $n/2$ elementos (ou mais)
  - todo elemento à direita de M_2 é maior do que $n/2 + 2$ elementos (ou mais)

  Logo, nenhum desses elementos pode ser a mediana.

— (*e o caso em que* $M_2 < M_1$ *é análogo ...*)

Esse raciocínio nos permite identificar e excluir $n/2$ elementos — (*em tempo* $O(n)$)

E agora, basta encontrar a mediana dos elementos que restaram



Quer dizer, agora nós temos um algoritmo de divisão e conquista

```
Algoritmo  Mediana-DC ( L, inicio, fim )
 {
      Se ( fim = inicio + 1 )   Retorna ( L[fim] )
  O(1)
       meio  <--  ( inicio + fim ) / 2

T(n/2)  M1  <--  Mediana-DC  (L, inicio, meio)

T(n/2)  M2  <--  Mediana-DC  (L, meio+1, fim)

  O(n)  j,k  <--  Reorganiza (L, M1, M2, inicio, fim)

T(n/2)  M  <--  Mediana-DC  (L, j, k)

   O(1)  Retorna ( M )
 }
```

E agora, nós podemos analisar o tempo de execução do algoritmo pelo método da equação de recorrência.

Com base nas anotações do código, nós escrevemos a recorrência

$$T(n) \quad = \quad 3 \cdot T\left(\frac{n}{2}\right) \quad + \quad n$$

Para obter a solução, nós desenrolamos a recorrência

$$\begin{aligned}
T(n) \quad &= \quad 3 \cdot T\left(\tfrac{n}{2}\right) \quad + \quad n \\
&= \quad 3 \cdot \left[\, 3 \cdot T\left(\tfrac{n}{2^2}\right) \quad + \quad \tfrac{n}{2} \,\right] \quad + \quad n \\
&= \quad 3^2 \cdot T\left(\tfrac{n}{2^2}\right) \quad + \quad \tfrac{3n}{2} \quad + \quad n \\
&= \quad 3^3 \cdot T\left(\tfrac{n}{2^3}\right) \quad + \quad \tfrac{3^2 n}{2^2} \quad + \quad \tfrac{3n}{2} \quad + \quad n \\
&= \quad \ldots \\
&= \quad 3^{\log_2 n} \cdot T(1) \quad + \quad n \left(\tfrac{3}{2}\right)^{\log_2 n - 1} \quad + \quad \ldots \quad + \quad \tfrac{3n}{2} \quad + \quad n
\end{aligned}$$

Daí, a gente pode fazer a seguinte esperteza

$$3^{\log_2 n} \quad = \quad \left(2^{\log_2 3}\right)^{\log_2 n} \quad = \quad \left(2^{\log_2 n}\right)^{\log_2 3} \quad = \quad n^{\log_2 3}$$

Por outro lado, o restante dos termos podem ser reescritos assim

$$n \cdot \left[\, 1 \quad + \quad \tfrac{3}{2} \quad + \quad \left(\tfrac{3}{2}\right)^2 \quad + \quad \ldots \quad + \quad \left(\tfrac{3}{2}\right)^{\log_2 n - 1} \,\right]$$

Quer dizer, nós temos uma soma de PG dentro dos parênteses.

E a gente lembra que

$$\begin{aligned}
\mathsf{S} \quad &= \quad 1 \quad + \quad \tfrac{3}{2} \quad + \quad \left(\tfrac{3}{2}\right)^2 \quad + \quad \ldots \quad + \quad \left(\tfrac{3}{2}\right)^k \\
&= \quad 1 \quad + \quad \tfrac{3}{2} \cdot \Big[\, 1 \underbrace{\quad + \quad \left(\tfrac{3}{2}\right) \quad + \quad \ldots \quad + \quad \left(\tfrac{3}{2}\right)^{k-1}}_{\mathsf{S} \quad - \quad \left(\frac{3}{2}\right)^k} \,\Big]
\end{aligned}$$

donde

$$\mathsf{S} \quad = \quad 1 \quad + \quad \tfrac{3}{2} \cdot \mathsf{S} \quad - \quad \left(\tfrac{3}{2}\right)^{k+1}$$

donde

$$\mathsf{S} \quad = \quad 2 \cdot \left(\left(\tfrac{3}{2}\right)^{k+1} \quad - \quad 1 \right)$$

Substituindo esse resultado na expressão acima, nós obtemos

$$n \cdot \left(2 \cdot \left(\tfrac{3}{2}\right)^{\log_2 n} - 2 \right) = 2n \cdot \frac{n^{\log_2 3}}{n} - 2n$$

$$= 2 \cdot n^{\log_2 3} - 2n$$

Finalmente, juntando uma coisa com a outra, a gente obtém

$$T(n) = n^{\log_2 3} + 2 \cdot n^{\log_2 3} - 2n$$

$$= O(n^{\log_2 3}) = O(n^{1{,}58})$$

Quer dizer, isso é melhor do que o algoritmo porcaria.

Mas, não é tão bom quanto a solução via ordenação.

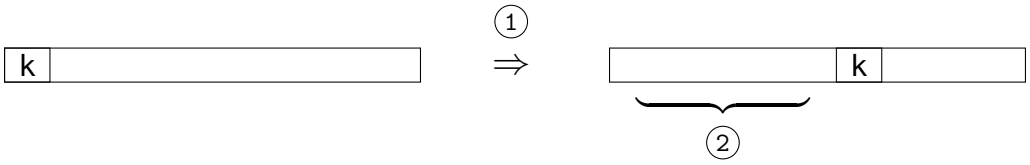
Então, ainda deve ser possível fazer melhor do que isso.

3 O algoritmo *quickMediana*

Existe uma ideia simples que funciona bem na prática.

A ideia consiste em utilizar a mesma estratégia do algoritmo quicksort

1. primeiro, a gente particiona a lista
2. e depois, a gente continua procurando no lado certo da partição




```
Algoritmo  quickMediana ( L, inicio, fim )
 {
     k  <--  Particao ( L, inicio, fim )

     Se ( k = n/2 + 1)    M  <--  L[k]

     Se ( k > n/2 + 1)    M  <--  quickMediana  (L, inicio, k-1)

     Se ( k < n/2 + 1)    M  <--  quickMediana  (L, k+1, fim)

     Retorna ( M )
 }
```

No pior caso, esse algoritmo é uma porcaria — (*porque?*)

Mas, ele executa em tempo médio $O(n)$.

Porque?

Bom, a ideia é utilizar a função `Partição-Boa()`.

Essa função executa em tempo médio $O(n)$.

E garante que o maior lado da partição tem tamanho no máximo $3n/4$.

Daí, a anotação do código fica assim

```
Algoritmo  quickMediana ( L, inicio, fim )
 {
  O(n)  k <--  Particao-Boa ( L, inicio, fim )

     O(1)  Se ( k = n/2 + 1)    M  <--  L[k]

T(3n/4)    Se ( k > n/2 + 1)    M  <--  quickMediana  (L, inicio, k-1)

T(3n/4)    Se ( k < n/2 + 1)    M  <--  quickMediana  (L, k+1, fim)

     O(1)  Retorna ( M )
 }
```

Daí, como apenas uma chamada recursiva é realizada, nós obtemos a seguinte recorrência

$$T(n) \quad = \quad T\left(\tfrac{3n}{4}\right) \quad + \quad n$$

Para obter a solução, nós desenrolamos a recorrência

$$\begin{aligned}
T(n) &= T\left(\tfrac{3n}{4}\right) + n \\
&= T\left(\tfrac{3^2 n}{4^2}\right) + \tfrac{3n}{4} + n \\
&= T\left(\tfrac{3^3 n}{4^3}\right) + \tfrac{3^2 n}{4^2} + \tfrac{3n}{4} + n \\
&= \ldots \\
&< n \cdot \left[1 + \tfrac{3}{4} + \tfrac{3^2}{4^2} + \tfrac{3^3}{4^3} + \ldots\right]
\end{aligned}$$

Quer dizer, a gente chegou em uma soma de PG outra vez.

E a gente resolve isso assim

$$\begin{aligned}
\mathsf{S} &= 1 + \tfrac{3}{4} + \left(\tfrac{3}{4}\right)^2 + \left(\tfrac{3}{4}\right)^3 + \ldots \\
&= 1 + \tfrac{3}{4} \cdot \Big[1 + \underbrace{\tfrac{3}{4} + \left(\tfrac{3}{4}\right)^2 + \ldots}_{\mathsf{S}}\Big]
\end{aligned}$$

donde

$$\mathsf{S} \quad = \quad 1 \quad + \quad \frac{3}{4} \cdot \mathsf{S}$$

donde

$$\mathsf{S} \quad = \quad 4$$

Daí que, o nosso algoritmo executa em tempo médio

$$T(n) \quad = \quad 4n \quad = \quad O(n)$$

Mas, no pior caso ele é uma porcaria.

Logo, ainda deve ser possível conseguir algo melhor do que isso.

4 Mediana em tempo linear

Mas, como é que a gente chega lá.

Bom, o único caminho é tentar entender melhor as coisas.

Quer dizer, nós já vimos que o uso de um bom pivô no procedimento de partição pode reduzir o tempo do algoritmo para $O(n)$.

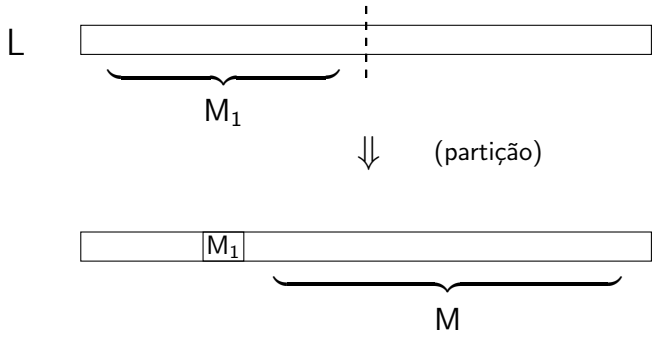
O problema é que a função `Partição-Boa( )` não consegue encontrar esse pivô em tempo $O(n)$ — (*no pior caso*)

Mas daí, a gente nota que o algoritmo de divisão e conquista encontrava dois bons pivôs — (*e reduzia o tamanho da lista para* $n/2$)

Só que, um bom pivô já basta.

Quer dizer, a mediana da primeira metade é um bom pivô.

E com esse pivô a gente consegue eliminar ao menos $n/4$ elementos — (*porque?*)



Daí, o algoritmo fica assim

```
Algoritmo  Mediana-DC2 ( L, inicio, fim )
 {
      Se ( fim = inicio + 1 )   Retorna ( L[fim] )
    O(1)
    meio  <--  ( inicio + fim ) / 2
T(n/2) M1  <--  Mediana-DC2  (L, inicio, meio)
  O(n)  k  <--  Particiona (M1, L, inicio, fim)
    O(1)  Se ( k = n/2 + 1)    M  <--  L[k]
T(3n/4)   Se ( k > n/2 + 1)    M  <--  Mediana-DC2  (L, inicio, k-1)
T(3n/4)   Se ( k < n/2 + 1)    M  <--  Mediana-DC2 (L, k+1, fim)
       Retorna ( M )
 }
```

onde

- as duas últimas chamadas recursivas tem estimativa $T(3n/4)$ porque o bom pivô (M_1) permite eliminar ao menos $n/4$ elementos do problema

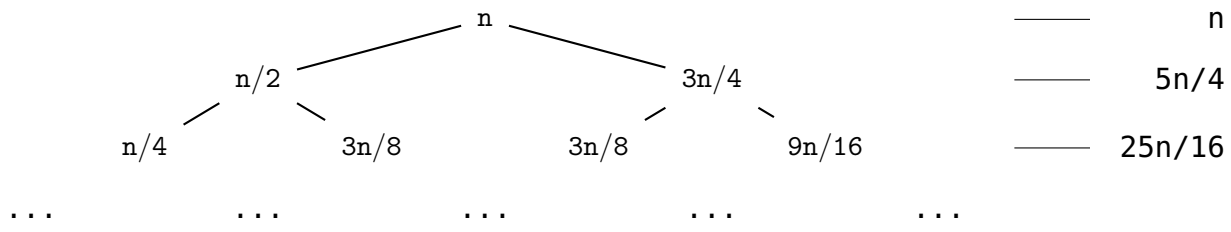
Daí que, a gente obtém a seguinte recorrência

$$T(n) \quad = \quad T\left(\frac{n}{2}\right) \quad + \quad T\left(\frac{3n}{4}\right) \quad + \quad n$$

E agora?

Como é que a gente resolve isso?

Bom, uma ideia é abrir a árvore de recursão, e fazer a soma por níveis



Quer dizer, a coisa cresce por um fator de 5/4.

E o ramo mais curto da árvore tem comprimento $\log_2 n$.

Daí que, a gente pode estimar o tempo de execução do algoritmo assim

$$\begin{aligned} T(n) \quad &> \quad n \quad + \quad \tfrac{5}{4}n \quad + \quad \left(\tfrac{5}{4}\right)^2 n \quad + \quad \ldots \quad + \quad \left(\tfrac{5}{4}\right)^{\log_2 n} n \\ &= \quad n \cdot \left[1 \quad + \quad \tfrac{5}{4} \quad + \quad \left(\tfrac{5}{4}\right)^2 \quad + \quad \ldots \quad + \quad \left(\tfrac{5}{4}\right)^{\log_2 n} \right] \end{aligned}$$

Quer dizer, a gente obteve uma soma de PG outra vez.

O que a gente resolve assim

$$\begin{aligned} \mathsf{S} \quad &= \quad 1 \quad + \quad \tfrac{5}{4} \quad + \quad \left(\tfrac{5}{4}\right)^2 \quad + \quad \ldots \quad + \quad \left(\tfrac{5}{4}\right)^k \\ &= \quad 1 \quad + \quad \tfrac{5}{4} \quad \cdot \quad \Big[1 \quad \underbrace{+ \quad \left(\tfrac{5}{4}\right) \quad + \quad \ldots \quad + \quad \left(\tfrac{5}{4}\right)^{k-1}}_{\mathsf{S} \quad - \quad \left(\frac{3}{2}\right)^k} \Big] \end{aligned}$$

donde

$$\mathsf{S} \quad = \quad 1 \quad + \quad \tfrac{5}{4} \cdot \mathsf{S} \quad - \quad \left(\tfrac{5}{4}\right)^{k+1}$$

donde

$$\mathsf{S} \quad = \quad 2 \cdot \left(\left(\tfrac{5}{4}\right)^{k+1} \quad - \quad 1 \right)$$

Daí que, o tempo de execução do algoritmo é

$$\begin{aligned} T(n) &> 2n \cdot \left(\left(\tfrac{5}{4}\right)^{\log_2 n} - 1 \right) \\ &= 2n \cdot n^{\log_2 5/4} - 2n \\ &= O(n^{1{,}32}) \end{aligned}$$

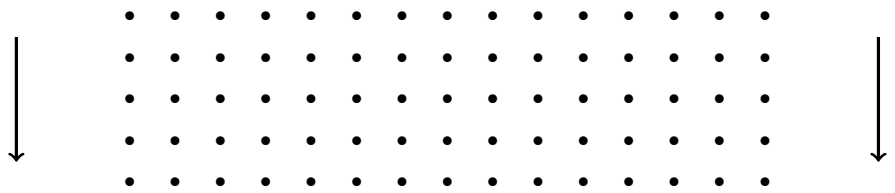
Bom, a gente ainda não chegou lá.

Mas, a ideia é boa ...

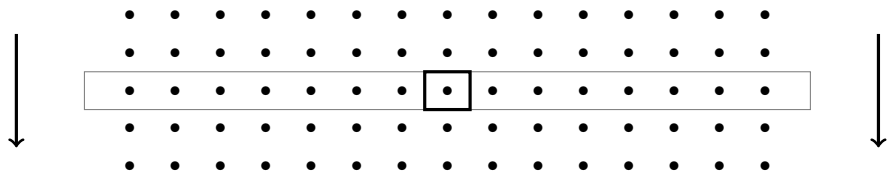
Quer dizer, tudo o que a gente precisa é de uma maneira eficiente de calcular um bom pivô.

E a solução é a seguinte

1. Divida os elementos da lista em grupinhos de 5, e ordene cada grupinho



2. Daí, considere a coleção dos elementos centrais de cada grupinho, e encontre a mediana dessa coleção



O que nós podemos dizer sobre esse elemento?

Bom, não é difícil de ver que ele é

- maior do que $\frac{3n}{10}$ elementos — (*aproximadamente*)
- menor do que $\frac{3n}{10}$ elementos — (*aproximadamente*)

Daí que, ele é um bom pivô.

Além disso, esse bom pivô é encontrado em tempo

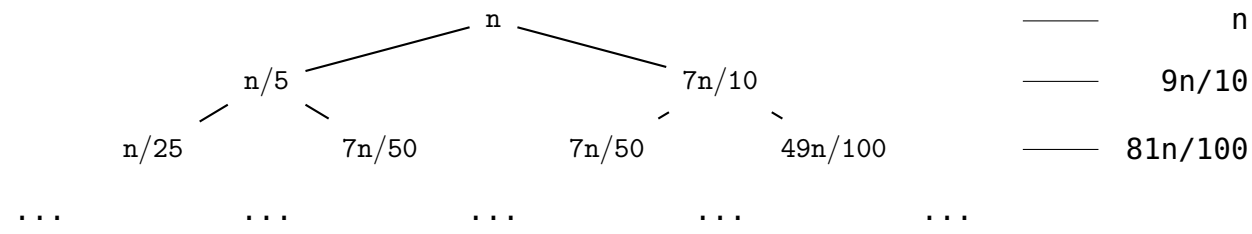
$$O(n) + T(n/5)$$

Utilizando esse esquema, a gente obtém um algoritmo para o problema que executa em tempo

$$T(n) = T(n/5) + T(7n/10) + n$$

Quanto é isso?

Bom, vamos expadir a árvore para ver



Quer dizer, a coisa cresce por um fator de 9/10.

Daí que, a gente pode estimar o tempo de execução do algoritmo assim

$$\begin{aligned} T(n) &< n + \tfrac{9}{10}n + \left(\tfrac{9}{10}\right)^2 n + \left(\tfrac{9}{10}\right)^3 n + \dots \\ &= n \cdot \left[1 + \tfrac{9}{10} + \left(\tfrac{9}{10}\right)^2 + \left(\tfrac{9}{10}\right)^3 + \dots\right] \end{aligned}$$

A gente obteve uma soma de PG outra vez.

E a gente resolve isso assim

$$\begin{aligned} \mathsf{S} &= 1 + \tfrac{9}{10} + \left(\tfrac{9}{10}\right)^2 + \left(\tfrac{9}{10}\right)^3 + \dots \\ &= 1 + \tfrac{9}{10} \cdot \Big[\underbrace{1 + \tfrac{9}{10} + \left(\tfrac{9}{10}\right)^2 + \dots}_{\mathsf{S}}\Big] \end{aligned}$$

donde

$$\mathsf{S} = 1 + \frac{9}{10} \cdot \mathsf{S}$$

donde

$$\mathsf{S} = 10$$

Daí que, o tempo de execução do algoritmo é

$$T(n) = 10 \cdot n = O(n)$$

Legal, não é?