Contando faixas bacanas

Imagine que L é uma lista de tamanho n que contém números positivos e negativos

Imagine que nós temos dois números MIN, MAX — (*onde o primeiro é menor do que o segundo*)

E imagine que uma faixa de elementos consecutivos é *bacana* se a soma dos seus elementos está entre MIN e MAX.

O problema é

→ *Contar quantas faixas bacanas existem na lista* L

Certo.

É bem fácil resolver esse problema em tempo $O(n^3)$.

Basta calcular a soma de cada faixa `L[i..j]` em separado — (*e ir contando quantas delas são bacanas*)

Mas, $O(n^3)$ é uma grande porcaria ...

Daí, pensando um pouquinho, a gente vê que dá pra calcular as somas de todas as faixas que começam em `L[i]` em tempo $O(n)$.

Basta manter um dedo parado em `L[i]`, e fazer um dedo `j` andar



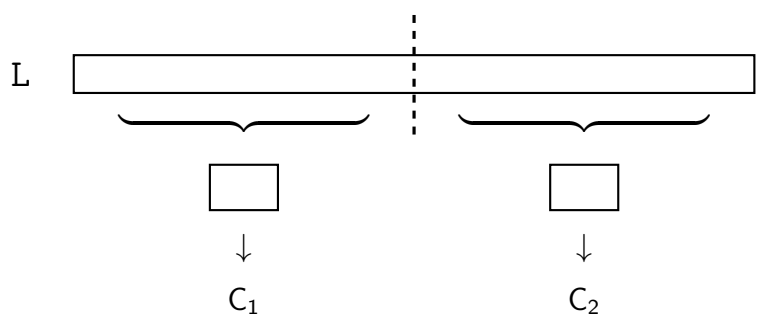
Isso vai dar um algoritmo de tempo $O(n^2)$.

Mas isso ainda é uma porcaria ...

Divisão e conquista

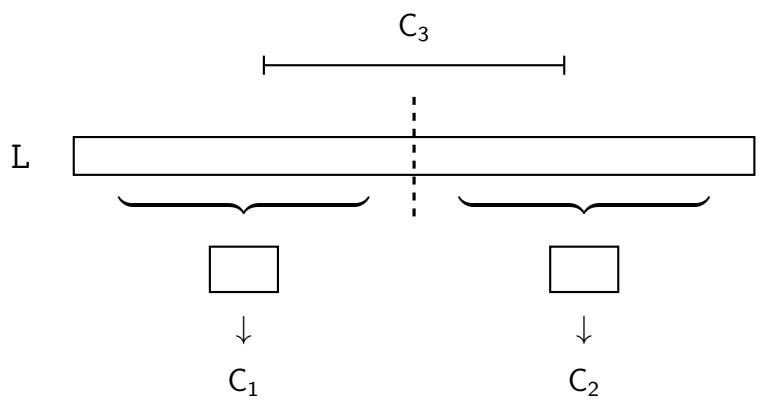
Então agora, está na hora de usar o que a gente aprendeu.

Quer dizer, imagine que a gente quebra a lista no meio, e resolve os problemas de cada lado — (*recursivamente*)



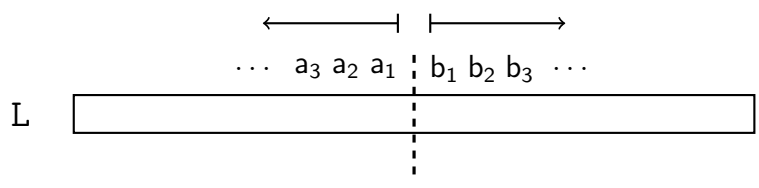
Isso, é claro, ainda não resolve o nosso problema.

Porque ainda é preciso contar as faixas bacanas que possuem elementos nas duas metades

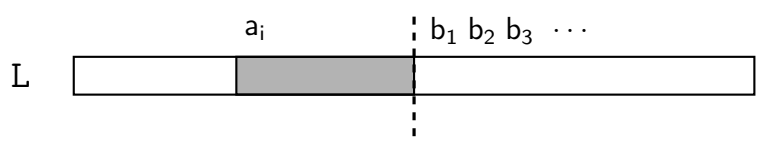


Então agora, a gente se concentra nesse problema.

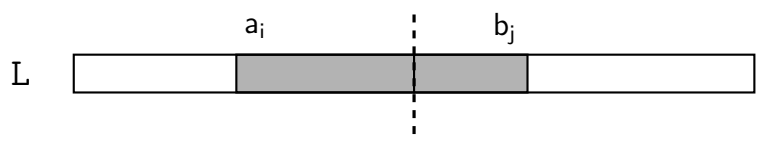
E a ideia natural é pecorrer a lista do meio para as duas pontas —: (*calculando a soma até cada ponto*)



Daí, a gente foca a atenção em um dos pedaços do lado esquerdo



E vai contar quantos pedaços do lado direito completam uma faixa bacana



$$\text{MIN} \;\leq\; a_i + b_j \;\leq\; \text{MAX}$$

Só que, examinar cada pedaço do lado direito leva tempo $O(n)$.

E fazer isso para cada pedaço do lado esquerdo leva tempo $O(n^2)$ — (*uma porcaria ...*)

Então, a gente precisa ser mais esperto do que isso.

E a esperteza consiste em ordenar os números do lado direito

Agora, a gente consegue determinar quantos b_j's satisfazem

$$\text{MIN} - a_i \;\leq\; b_j \;\leq\; \text{MAX} - a_i$$

em tempo $O(\log n)$ — (*usando a busca binária*)

E fazer isso para cada pedaço do lado esquerdo leva tempo $O(n \log n)$.

Agora a gente pode escrever a função que faz isso

```
func contaFaixaMeio ( L, inicio, meio, fim )

    k ← 0;     soma ← 0

    Para i ← meio Até inicio

        soma ← soma + L[i]

        k++;     A[K] ← soma

    ℓ ← 0;     soma ← 0

    Para j ← meio+1 Até fim

        soma ← soma + L[j]

        ℓ++;     B[ℓ] ← soma

    ordena ( B )

    C ← 0

    Para i ← 1 Até K                    // p/ cada elemento de A

        b1 ← buscaBin ( Min - A[i], B )

        b2 ← buscaBin ( Max - A[i], B )

        C ← b2 − b1

    Retorna ( C )
```

É bem fácil ver que esse algoritmo executa em tempo $O(n \log n)$.

E agora, nós podemos escrever o nosso algoritmo de divisão e conquista

```
T(n) =  func CFB-DC ( L, inicio, fim )

          Se ( inicio = fim )

              Se ( MIN ≤ L[inicio] ≤ MAX )     Retorna ( 1 )
O(1)
              Senão                                Retorna ( 0 )

          meio ← ( inicio + fim ) / 2

T(n/2) ←  C1 ← maiorRetangulo-DC ( L, inicio, meio )

T(n/2) ←  C2 ← maiorRetangulo-DC ( L, meio+1, fim )

O(n log n) ←  C3 ← retanguloMeio ( L, inicio, meio, fim )

          Retorna ( C1 + C2 + C3 )
```

Análise

Seguindo as anotações, nós obtemos a seguinte equação de recorrência para o tempo de execução do algoritmo

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

E agora, é só desenrolar o rocambole

(. . .)

O que nos dá

$$T(n) \quad = \quad O(n \log^2 n)$$