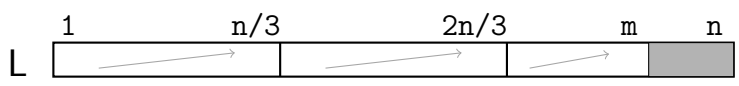
Repetindo a esperteza

Certo.

Mas agora que a gente entendeu, a gente pode levar a coisa mais longe.

Quer dizer, a gente pode pensar em dividir a lista em 3 partes



Daí,

- a busca vai levar tempo $3 \log n$ (aprox.)
- e a inserção e a remoção vão levar tempo $n/3$ (aprox.)

Quer dizer, o tempo da busca aumentou outra vez.

Mas agora, os tempos da inserção e da remoção foram reduzidos a 1/3 do tempo original.

Já dá para advinhar que isso vai continuar acontecendo quando a gente dividir a lista em mais partes.

E daí, a gente pode perguntar

- *Qual é o número de partes ótimo ?*

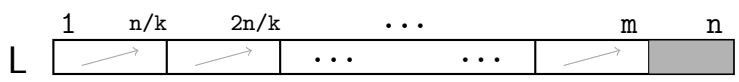
Só que dessa vez a pergunta não tem resposta ...

Quer dizer, tudo depende do contexto.

Se a gente tem uma situação onde acontecem muitas buscas e quase nenhuma inserção e remoção, então a gente deve manter o tempo da busca o menor possível.

Por outro lado, se ocorrem muitas inserções e remoções, então a gente vai querer manter os tempos da inserção e remoção pequenos.

Para entender melhor o que está acontecendo, imagine que a lista é dividida em k partes



Daí, no pior caso, a busca precisa examinar todas as partes.

E isso leva tempo

$$k \cdot \log (n/k) \qquad \text{(aprox.)}$$

Por outro lado, a inserção sempre ocorre na última parte.

E como essa parte tem tamanho no máximo n/k, a coisa leva tempo

$$n/k \qquad \text{(aprox.)}$$

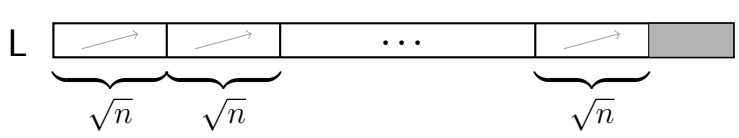
Finalmente, a remoção envolve uma busca seguida de um deslocamento.

E isso leva tempo

$$k \cdot \log (n/k) \quad + \quad n/k \qquad \text{(aprox.)}$$

Legal.

A estrutura de dados *multilista* corresponde à situação em que a gente divide a lista em blocos de tamanho $\sqrt{n}$



E nesse caso, os tempos de execução das operações de busca, inserção e remoção ficam assim

	busca	inserção	remoção
multilista	$O(\sqrt{n} \cdot \log n)$	$O(\sqrt{n})$	$O(\sqrt{n} \cdot \log n)$