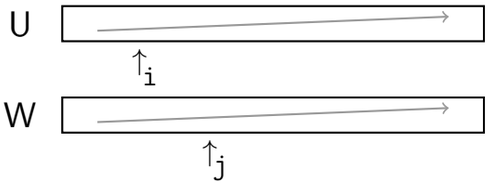
Então agora, a gente começa a usar aquilo que a gente aprendeu.

Quer dizer, a gente ja aprendeu a ordenar uma lista de maneira eficiente.

E a boa notícia é que é bem rápido contar os elementos comuns quando as duas listas estão ordenadas.

Basta percorrer as listas simultaneamente com dois dedos



A coisa fica assim

```
func contaComunsOrd (U[a..b], W[c..d])

    cont ← 0
    i ← a    j ← c
    Enquanto ( i ≤ b  E  j ≤ d )
        Se ( U[i] = W[j] )
            cont++;   i++;   j++
        Se ( U[i] < W[j] )    i++
        Senão                   j++

    Retorna (cont)
```

O(n)

Daí, antes de fazer a contagem dos elementos comuns, nós precisamos ordenar as faixas de U e W — (*antes e depois do pivô*)

Mas, nós não podemos mudar a ordem dos elementos de U e W para não atrapalhar a contagem.

Então, nós vamos utilizar duas listas auxiliares G, H para isso.

A coisa fica assim

```
func contaTE-3.0 (U, W)

    cont ← 0

    Para i ← 2 Até N-1
        pivo ← U[i]
        j ← Localiza (pivo, W[1..N])
        G[1..i-1] ← Ordena (U[1..i-1])
        G[i+1..N] ← Ordena (U[i+1..N])
        H[1..j-1] ← Ordena (W[1..j-1])
        H[j+1..N] ← Ordena (W[j+1..N])
        C1 ← contaComunsOrd (U[1..i-1], W[1..j-1])
        C2 ← contaComunsOrd (U[i+1..N], W[j+1..N])
        cont ← C1 * C2

    Retorna (cont)
```

O(n)

O(nlogn)

O(n)

E a complexidade do nosso terceiro algoritmo é

$$O(n) \quad \times \quad \Big[O(n \log n) \quad + \quad O(n) \Big] \qquad = \qquad O(n^2 \log n)$$

Bom, as coisas estão melhorando.

Mas isso ainda é uma porcaria.