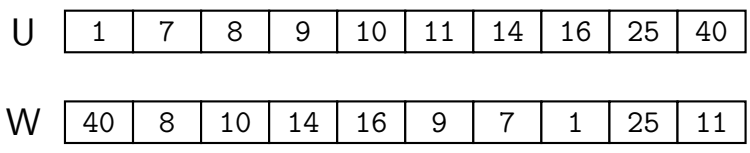
Bom, a gente pode tentar.

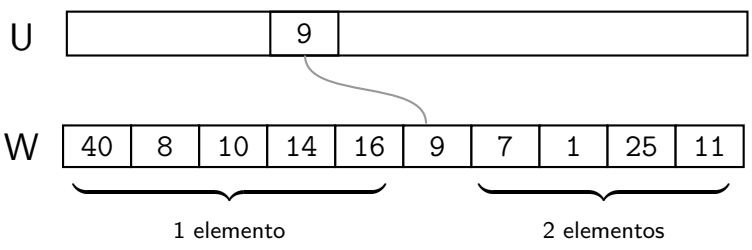
Só que para isso, é preciso modificar o nosso ponto de vista — (*porque nós já chegamos ao limite do ponto de vista anterior*)

Quer dizer, imagine agora que os elementos da lista U estão em ordem crescente



Bom, isso facilita bastante as coisas.

Porque agora, para contar os elementos comuns basta procurar os elementos menores que o pivô do lado esquerdo, e os elementos maiores que o pivô do lado direito.

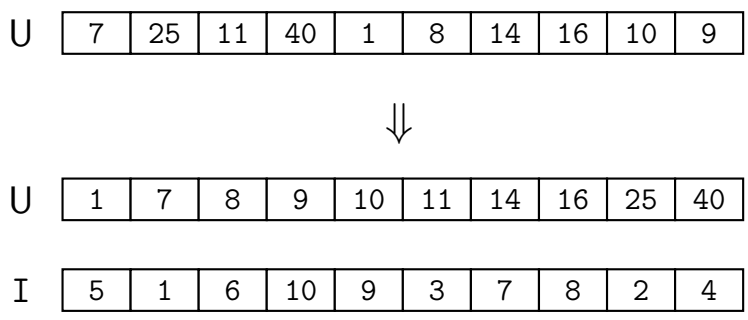


E isso nos dá um algoritmo bem mais simples de tempo $O(n^2)$.

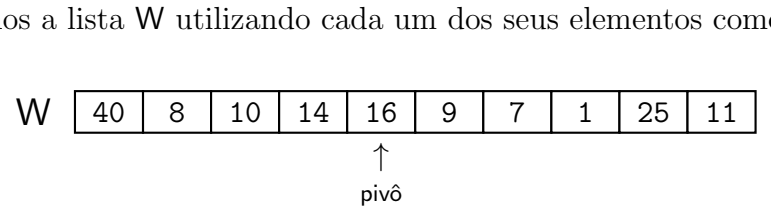
O problema é que no nosso problema a lista U não está ordenada.

Ora, mas se esse é o problema, então a gente pode ordenar a lista U.

E para não perder informação, a gente mantém os índices da configuração original em uma lista auxiliar I



Agora, nós percorremos a lista W utilizando cada um dos seus elementos como pivô



E para cada um deles nós contamos

- número de elementos na primeira parte de W que estão à esquerda do pivô em U
- número de elementos na segunda parte de W que estão à direita do pivô em U

Para fazer isso, nós

- fazemos uma busca binária na versão ordenada de U
- obtemos a posição do elemento na versão original de U, consultando a lista I
- e depois comparamos com a posição do pivô na versão original de U

Isso leva tempo $O(\log n)$ para cada elemento.

O que dá tempo $O(n \log n)$ para processar cada pivô.

O que dá um algoritmo de tempo $O(n^2 \log n)$.

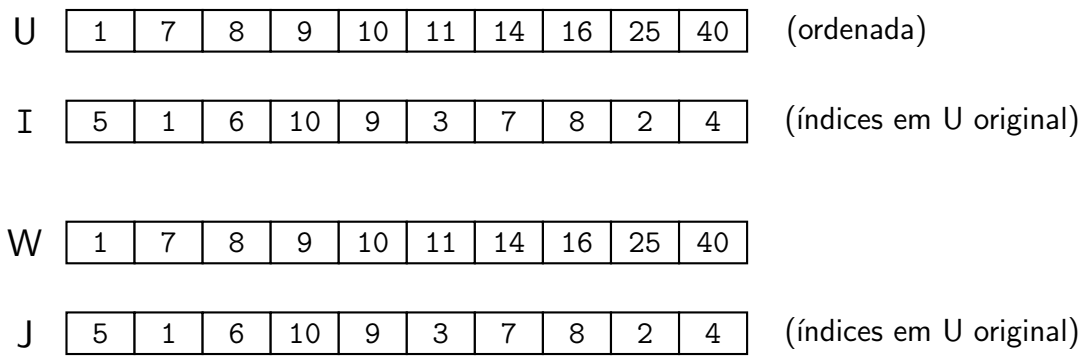
Isso não é muito legal.

Mas aqui a gente pode ser um pouquinho mais esperto.

Quer dizer, nós estamos fazendo várias vezes a busca binária com cada elemento de W.

Então, a ideia é fazer uma vez só.

E guardar os elementos em uma lista auxiliar J



Essas informações podem ser computadas em tempo $O(n \log n)$.

E agora, olhando para a lista J, nós estamos na situação em que U é uma lista ordenada.

E nessa situação, nós sabemos resolver o problema em tempo $O(n^2)$.

Não é legal?