Operação `max_heapify`

Escreva uma função, chamada `max_heapify`, que recebe como entrada um vetor contendo uma "quase-*heap*" máxima e transforma esse vetor em uma *heap* máxima.

Uma "quase-*heap*" máxima é um vetor $A=(a_1,a_2,a_3,...,a_{n-1})$ no qual todos os elementos respeitam a propriedade de *heap* máxima, com exceção do primeiro, que pode ou não respeitar a propriedade de *heap* máxima.
Para transformar essa "quase-*heap*" em uma *heap*, compare o elemento na posição a ser testada do vetor com os seus dois filhos (se houver). Caso ele viole a propriedade de *heap* com algum dos filhos, troque-o de posição com o filho mais promissor.
Aplique `max_heapify` recursivamente até que a propriedade de *heap* não seja mais violada.

Para recordar, dado um elemento $A[i]$ em uma *heap*...

- O pai de $A[i]$ pode ser encontrado no índice $pai(i)=\lfloor \frac{i-1}{2} \rfloor$;
- O elemento $A[0]$ não possui pai;
- O filho esquerdo de $A[i]$, se existir, pode ser encontrado no índice $2i+1$ e o filho direito, se existir, em $2i+2$;
- A propriedade de *heap* máxima é $A[A[pai(i)] \geqslant A[i]$.

Em seguida, faça um programa principal que lê várias listas em notação Python, representando "quase-*heaps*". Para cada uma imprima a lista resultante da operação `max_heapify`. A entrada termina com a lista vazia (`[]`).

Exemplos de Entrada e Saída

Entrada	`[7, 3, 2]`
	`[5, 3, 4]`
	`[3, 2, 7]`
	`[4, 6, 8, 3, 5, 5, 1, 2, 2, 5]`
	`[]`

Saída	`[7, 3, 2]`
	`[5, 3, 4]`
	`[7, 2, 3]`
	`[8, 6, 5, 3, 5, 4, 1, 2, 2, 5]`