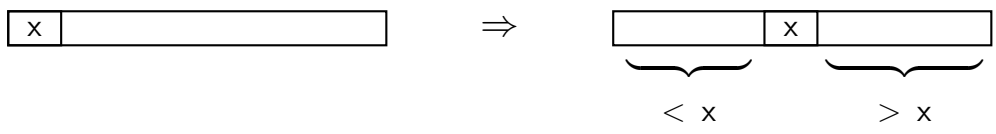
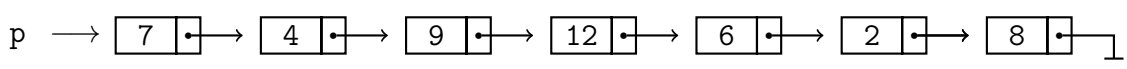
Lembre que a operação de partição reorganiza os elementos de um vetor assim



Quer dizer, os elementos menores que x vão para o início, e os elementos maiores que x vão para o fim do vetor — (*e o elemento* x *fica entre eles*)

Agora, nós queremos implementar essa operação na lista encadeada.

Para isso, nós imaginamos que o ponteiro p aponta para uma lista encadeada



E daí, a gente nota que esse exemplo é quase igual o exemplo (a).

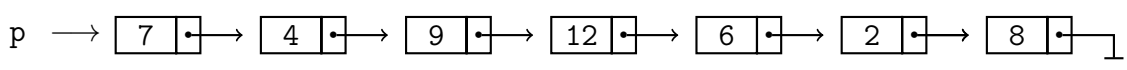
Quer dizer, lá a gente separou pares e ímpares, e agora a gente vai separar menores e maiores que o primeiro elemento.

Então, basta fazer a mesma coisa

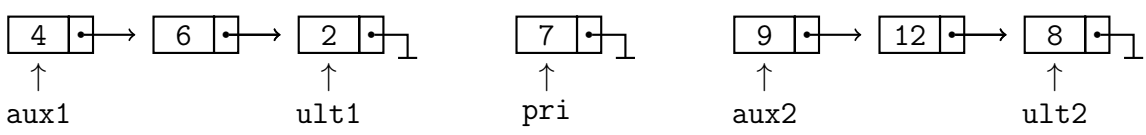
```
func Particao ( p )

      Se ( p == Nulo  OU  p->prox == Nulo)   Retorna (p)

      pri = p;    p = p->prox                          // separa o primeiro elemento

      aux1 = Nulo;    ult1 = Nulo
      aux2 = Nulo;    ult2 = Nulo

      Enquanto (p != Nulo )
            q = p;    p = p->prox                      // tira o 1o elemento de p
            q->prox = Nulo

            Se (q->num < pri->num)                     // Você é menor que o pivô?
                  Se (aux1 == Nulo)
                        aux1 = q;    ult1 = q
                  Senão
                        ult1->prox = q;    ult1 = q
            Senão                                      // Não?  então você é impar ...
                  Se (aux2 == Nulo)
                        aux2 = q;    ult2 = q
                  Senão
                        ult2->prox = q;    ult2 = q


      pri->prox = aux2                           // o primeiro entre as duas partes

      Se (aux1 == Nulo)    Retorna (pri)
      Senão
            ult1->prox = pri;    Retorna (aux1)
```

O algoritmo Quicksort

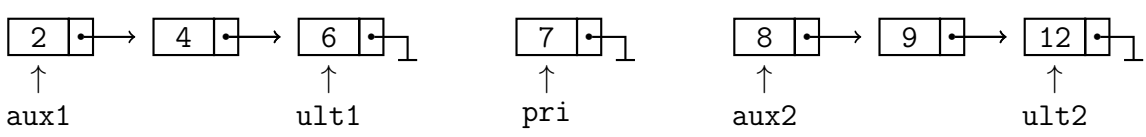
Vamos olhar mais uma vez para a lista p



Imagine que a operação de partição deixa as 3 partes desconectadas



E agora, veja o que acontece quando a gente aplica a partição na primeira e na terceira partes



Sim! a lista ficou completamente ordenada — (*basta conectar as 3 partes*)

Bom, aqui a gente deu um pouco de sorte — (*porque bastou uma partição em cada parte*)

Em geral, a gente vai ter que aplicar a partição nas partes das partes — (*e nas partes das partes das partes ...*)

Mas a coisa sempre funciona.

Quer dizer, no final a lista sempre fica ordenada.

Esse é o algoritmo de ordenação *Quicksort*.

E agora, nós já podemos executá-lo sobre uma lista encadeada.

A coisa fica assim

```
func Quicksort ( p, u )                          // ponteiros p/ 1o e ult elem

      Se ( p == Nulo  OU  p->prox == Nulo)
            Retorna (p,u)                        // nada a fazer

      aux1,utl1,pri,aux2,ult2 = Particao (p,u)

      aux1,utl1 = Quicksort (aux1,ult1)

      aux2,utl2 = Quicksort (aux2,ult2)

      p,u  Conecta (aux1,ult1,pri,aux2,ult2)     // junta tudo

      Retorna (p,u)
```

◇