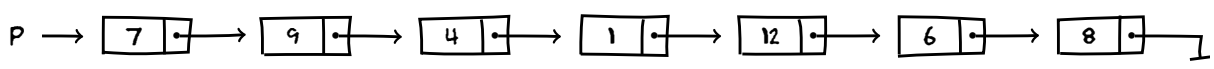
## A operação de partição

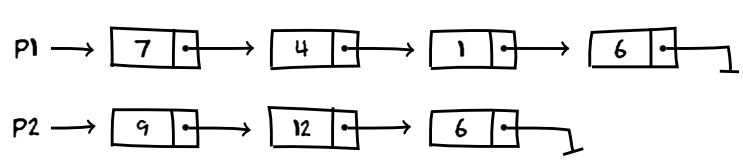
Imagine que o ponteiro p aponta para uma lista encadeada




A tarefa é

→ *particionar a lista* p *em duas listas* p1 *e* p2
*onde os elementos da lista* p1 *são todos* $\geq$ k
*e os elementos da lista* p2 *são todos* $<$ k

No exemplo acima, isso nos daria




O raciocínio recursivo para essa tarefa fica assim

1. primeiro a gente particiona o restante da lista
   e obtém os ponteiros p1 e p2 — (*fazendo uma chamada recursiva*)
2. depois, a gente compara o primeiro elemento com o k
   e coloca ele em p1 ou p2

E a coisa fica assim

```
func partição-Rec (q,k)

      Se (q == Nulo)   Retorna (Nulo,Nulo)

      p1,p2 = partição-Rec (q->prox)                // particiona o restante da lista

      Se ( q->num ≥ k)                              // o primeiro elemento
            q->prox = p1;   Retorna (q,p2)          //
      Senão                                         //
            q->prox = p2;   Retorna (p1,q)          //
```

### O algoritmo Quicksort

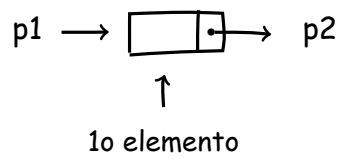
Agora que nós temos a função de partição, é bem fácil escrever o algoritmo quicksort.

Quer dizer, a ideia é utilizar o primeiro elemento como pivô da partição — (*i.e., o valor do* k)

Daí, a gente particiona o restante da lista.

E ordena recursivamente as listas p1 e p2 — (*chamando a função* quicksort-Rec())

Finalmente, basta fazer a concatenação




A coisa fica assim

```
func quicksort-Rec (q)

      Se (q == Nulo)   Retorna (q)                 // já ordenei ...

      p1,p2 = partição-Rec (q->prox,q->num)         // particiona o restante da lista

      p1 = quicksort-Rec (p1)                       // ordena a 1a parte

      p2 = quicksort-Rec (p2)                       // ordena a 2a parte

      q = concatena (p1,q,p2)                       // e concatena

      Retorna (q)
```

◇