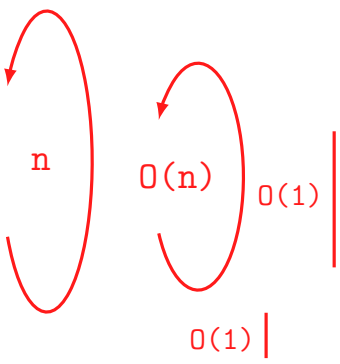
Agora considere a seguinte tarefa

→ *Verificar se existem dois elementos na lista cuja soma é igual a k*

Na primeira aula nós apresentamos dois algoritmos para esse problema: um para o caso não ordenado, e um para o caso ordenado.

A análise do algoritmo para o caso não ordenado é quase imediata.

Quer dizer, assumindo que o laço interno dá n voltas, nós obtemos a seguinte versão anotada do algoritmo




```
Para  i <-- 1  Até  n-1
  Para  j <-- i+1  Até  n
   {
      Se ( L[i] + L[j] = k )
       {
          Imprime ("Sim");   Retorna
       }
   }
Imprime ("Não");
```

(*Nós anotamos o laço mais interno como $O(n)$ para indicar que isso é uma aproximação*)

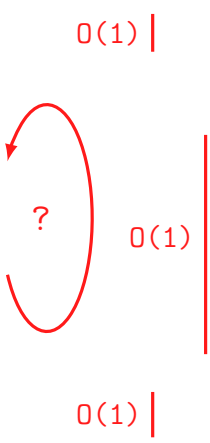
Agora com base nessas anotações nós obtemos a seguinte estimativa para o tempo de execução do algoritmo

$$O(1) \quad + \quad n \times O(n) \times O(1) \quad + \quad O(1) \qquad = \qquad O(n^2)$$

Certo.

Mas a análise do algoritmo para o caso ordenado não é tão imediata assim.

Quer dizer, esse algoirtmo apresenta mais uma novidade: o laço `Enquanto`, que não possui um número fixo, pré-determinado de voltas.




```
i  <--  1;   j  <--  n
Enquanto ( i < j )
{
   Se ( L[i] + L[j] = k )
    {
       Imprime("Sim");   Retorna
    }
   Se ( L[i] + L[j] < k )  i++
   Se ( L[i] + L[j] > k )  j--
}
Imprime("Não")
```

Daí que, parte da análise consiste em estimar o número de voltas que o laço dá.

Nesse caso, nós podemos fazer as seguintes observações

- no início as variáveis `i`, `j` são inicializadas como

  `i  <--  1;` `j  <--  n`

  de modo que a diferença entre elas é $(\texttt{j} - \texttt{i}) = \texttt{n} - 1$.
- daí, a cada volta do laço
  - ou a variável `i` é incrementada
  - ou a variável `j` é decrementada

  de modo que a diferença sempre diminui de 1

Logo, o laço termina no máximo após n voltas.

Portanto, o algoritmo do caso ordenado executa em tempo

$$O(1) \quad + \quad O(n) \times O(1) \quad + \quad O(1) \qquad = \qquad O(n)$$

Abaixo nós temos os resultados das duas análises

- `caso não ordenado`: $O(n^2)$
- `caso ordenado`: $O(n)$

E aqui nós temos um primeiro exemplo onde o algoritmo para o caso ordenado é significativamente mais eficiente do que o algoritmo para o caso não ordenado.

◇