Trabalhando com dedo manualmente
Professor Wladimir

Na aula anterior, fizemos os seguintes problemas:

1. Contando a quantidade de números ímpares
2. Encontrando o tamanho da maior sequência ordenada
3. Lista particionada
4. Lista dentro da lista (ordenada)

Busca em lista ordenada

Agora imagine que nós temos uma lista ordenada

	0	1	···						N-1
L	2	5	6	9	11	15	18	20	21	25

E imagine que a tarefa é Verificar se o número `k` está na lista ou não
Bom, a gente pode resolver esse problema com o for.
Reescrevendo com o while `while`, temos

```c
#include <stdio.h>
#include <stdlib.h>
#define  N  10
int  k = 18;
int L[N] = { 2, 5, 6, 9, 11, 15, 18, 20, 21, 25 };
int main()
{
 int  i, achei = 0;

 i = 0;
 while ( i < N )
  {
      if ( L[i] == k )
       {
          achei = 1;   break;
       }
      i ++;
  }

 if ( achei == 1 )     printf ("Achei!");
 else                  printf ("Não está lá ...");
}
```

Mas, a gente podia ter sido mais esperto.
Quer dizer, a lista está ordenada.
Daí que, não faz sentido continuar examinando os elementos quando eles ficam maiores do que `k`.
Logo, a gente pode reescrever o código assim

```c
int main()
{
 int  i, achei = 0;

 i = 0;
 while ( i < N  &&  L[i] <= k )
  {
      if ( L[i] == k )
       {
          achei = 1;
       }
      i ++;
  }
}
```

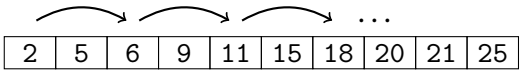
Agora, o mais legal mesmo é escrever o programa assim

```c
    int main()
     {
        int  i;

        i = 0;
        while ( i < N  &&  L[i] < k )    i++;

        if ( i == N )            achei = 0;

        else if ( L[i] == k )    achei = 1;

        else                     achei = 0;
     }
```

Quer dizer, primeiro a gente desliza o dedo até não poder mais.
E depois a gente vê o que aconteceu ...

Busca mais esperta

Imagine outra vez que nós temos uma lista ordenada
E imagine outra vez que a tarefa é Verificar se o número `k` está na lista ou não
A esperteza aqui consiste em explorar o controle manual do dedo no comando `while`.
E ir pulando na lista de 2 em 2

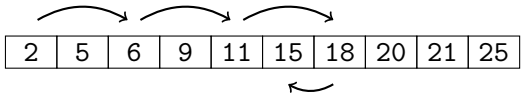


Fazendo isso, a gente percorre a lista duas vezes mais rápido.

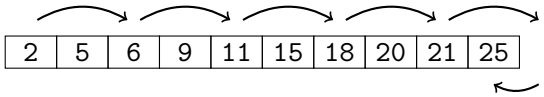
Mas, às vezes a gente passa do ponto.

E daí, é preciso dar um passo atrás

`k = 15:`



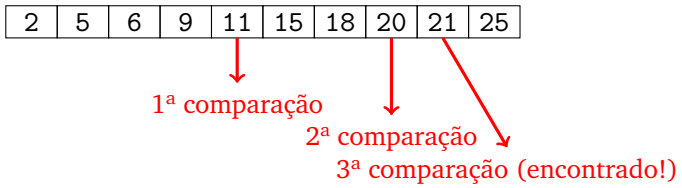
`k = 25:`



A coisa fica assim

```c
int main()
{
 int  i = 0;

 while ( i < N  &&  L[i] < k )   i = i + 2;

 if ( i == N + 1 )          achei = 0;

 else if ( i == N )         // pulou o último elemento ...
  {
    if ( L[N-1] == k )      achei = 1;
    else                    achei = 0;
  }

 else if ( L[i] == k )      achei = 1;

 else if ( L[i] > k )
  {
    if ( i > 0 && L[i-1] == k )     achei = 1;
    else                    achei = 0;
  }
}
```

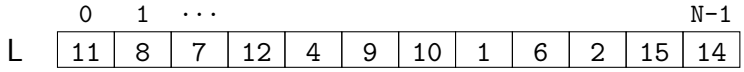
Busca super esperta



A ideia da busca binária é descartar, a cada passo, metade dos elementos do vetor. Imagine que estamos procurando o valor 21 na lista L. Começamos calculando o elemento central do vetor L[0..9], que está na posição 4. Nesse caso, L[4] = 11, e como 11 ¡ 21, podemos eliminar a metade esquerda e continuar a busca na sublista L[5..9]. Calculamos então o novo meio, que é L[7] = 20. Como 20 ¡ 21, novamente descartamos a parte à esquerda e seguimos com L[8..9]. O novo meio agora é L[8] = 21, que é exatamente o valor procurado.

```c
#include <stdio.h>
#include <stdlib.h>
#define  N  10
int  k = 21;
int L[N] = { 2, 5, 6, 9, 11, 15, 18, 20, 21, 25 };
int achei;
int main()
{
 int  inicio, fim, meio;

 inicio = 0;
 fim    = N-1;
 achei = 0;
 // Analisando a lista [inicio..fim]
 while ( inicio <=  fim )
  {
      meio = (inicio+fim)/2;

      if(L[meio] == k){
        achei = 1;
        break;
      }else if(L[meio] < k){
        inicio = meio+1;
      }else{ // k  < L[meio]
        fim = meio-1;
      }
  }

 if ( achei == 1 )     printf ("Achei!");
 else                  printf ("Não está lá ...");
}
```

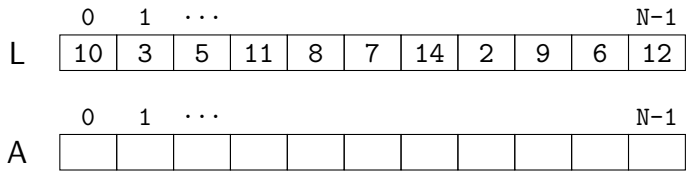
Separando em pares e ímpares



E imagine que a tarefa consiste em Mover os elementos ímpares para o fim da lista, e os elementos pares para o início da lista

Trabalhando com 3 dedos

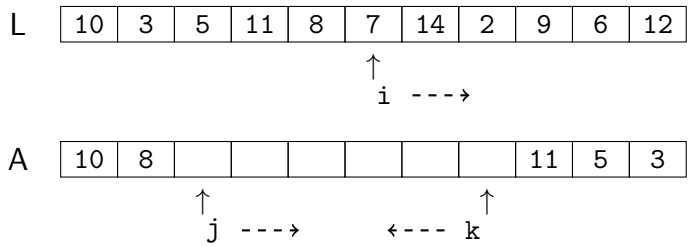
Podemos resolver esse problema utilizando uma lista auxiliar.



Daí, a gente percorre a lista L da esquerda para a direita.
E daí,

- quando a gente encontra um número par, a gente copia ele para o início da lista A
- quando a gente encontra um número ímpar, a gente copia ele para o final da lista A

Para fazer a coisa funcionar, a gente coloca dois dedos na lista A



Quer dizer,

- `j` indica a próxima posição vazia no início da lista A
- e `j` indica a próxima posição vazia no final da lista A

No final, a lista A já contém os elementos pares e ímpares separados.
E daí, basta copiar o seu conteúdo de volta para a lista L.
A coisa fica assim

```c
#include <stdio.h>
#include <stdlib.h>

#define  N  11

int  L[N] = { 10, 3, 5, 11, 8, 7, 14, 2, 9, 6, 12 };

int main()
{
 int  i, j, k;       // três dedos

 j = 0;    k = N-1;

 for ( i=0; i<N; i++ )
  {
```

```
16      if ( L[i] % 2 == 0 )          // Se L[i] é par
17       {
18         A[j]  =  L[i];             // copia p/ o início
19         j++;
20       }
21      else                          // Se L[i] é ímpar
22       {
23         A[k]  =  L[i];             // copia p/ o final
24         k--;
25       }
26   }
27
28  for ( i=0; i<N; i++ )             // copia elementos de volta p/ L
29     L[i]  =  A[i];
30 }
```

Trabalhando com 2 dedos

Podemos resolver esse problema utilizando uma lista auxiliar.

A gente percorre a lista L da esquerda para a direita.

Quando a gente encontra um número par, a gente copia ele para a lista A na posição dedo e incrementa a posição dedo.

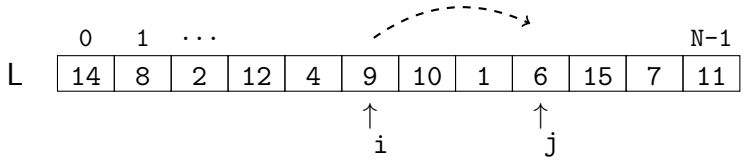
Em seguida, percorremos a lista L da esquerda para a direita.

Quando a gente encontra um número ímpar, a gente copia ele para a lista A na posição dedo e incrementa a posição dedo.

```c
#include <stdio.h>
#include <stdlib.h>

#define  N  11

int  L[N] = { 10, 3, 5, 11, 8, 7, 14, 2, 9, 6, 12 };

int main()
{
 int  i, dedo;       // três dedos
 int A[N];

 dedo = 0;

 for ( i=0; i<N; i++ )
  {
     if ( L[i] % 2 == 0 )          // Se L[i] é par
      {
        A[dedo]  =  L[i];             // copia p/ a posição dedo
        dedo++;
      }
  }

  for ( i=0; i<N; i++ )
  {
     if ( L[i] % 2 == 1 )          // Se L[i] é par
      {
        A[dedo]  =  L[i];             // copia p/ a posição dedo
        dedo++;
      }
  }

 for ( i=0; i<N; i++ )             // copia elementos de volta p/ L
    L[i]  =  A[i];
}
```

Jogando os ímpares para o final

Daí, a gente percorre a lista da esquerda para a direita.
E sempre que encontra um número ímpar, a gente joga ele para o final da lista



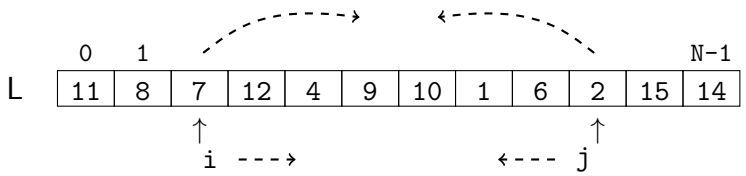
A coisa fica assim

```c
#include <stdio.h>
#include <stdlib.h>

#define  N  10

int  L[N] = { 11, 8, 7, 12, 4, 9, 10, 1, 6, 2, 15, 14 };

int main()
{
    int  i, j;      // dois dedos
    int  aux;

    i = 0;    j = N-1;
    while ( i < j )
    {
        if ( L[i] % 2 == 1 )          // Se L[i] é ímpar
        {
            aux  =  L[i];
            L[i]  =  L[j];             // troca
            L[j]  =  aux;
            j--;
        }
        else
            i++;
    }
}
```

Trabalhando nas duas pontas

Mas, essa não é a única maneira de fazer as coisas.
Quer dizer, a gente também pode trabalhar nas duas pontas da lista ao mesmo tempo.
A ideia é a seguinte

- `i` avança da esquerda para a direita procurando um elemento ímpar
- `j` avança da direita para a esquerda procurando um elemento par
- quando os dois já encontraram o seu elemento, daí a troca acontece



A repetição pára quando o `i` ultrapassa o `j`.
E a coisa fica assim

```c
#include <stdio.h>
#include <stdlib.h>

#define  N  10

int  L[N] = { 11, 8, 7, 12, 4, 9, 10, 1, 6, 2, 15, 14 };

int main(){
    int  i, j;      // dois dedos
    int  aux;

    i = 0;    j = N-1;
    while ( i < j ){
        if ( L[i] % 2 == 1   &&  L[j] % 2 == 0 )
        {
            aux  =  L[i];
            L[i]  =  L[j];            // troca
            L[j]  =  aux;
        }

        if ( L[i] % 2 == 0 )  i++;

        if ( L[j] % 2 == 1 )  j--;
    }
}
```