Recursividade

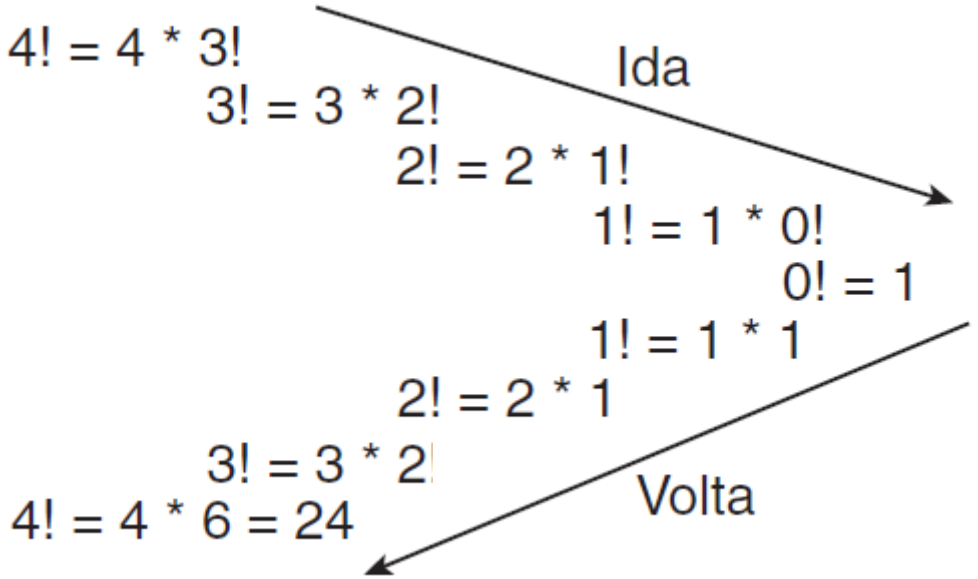
ECM404

Recursividade

- Na linguagem C, uma função pode chamar outra função. A função *main( )* pode chamar qualquer função, seja ela da biblioteca da linguagem (como a função *printf( )*) ou alguma definida pelo programador.
- Entretanto, uma função pode chamar a si própria, o que chamamos de **função recursiva**.

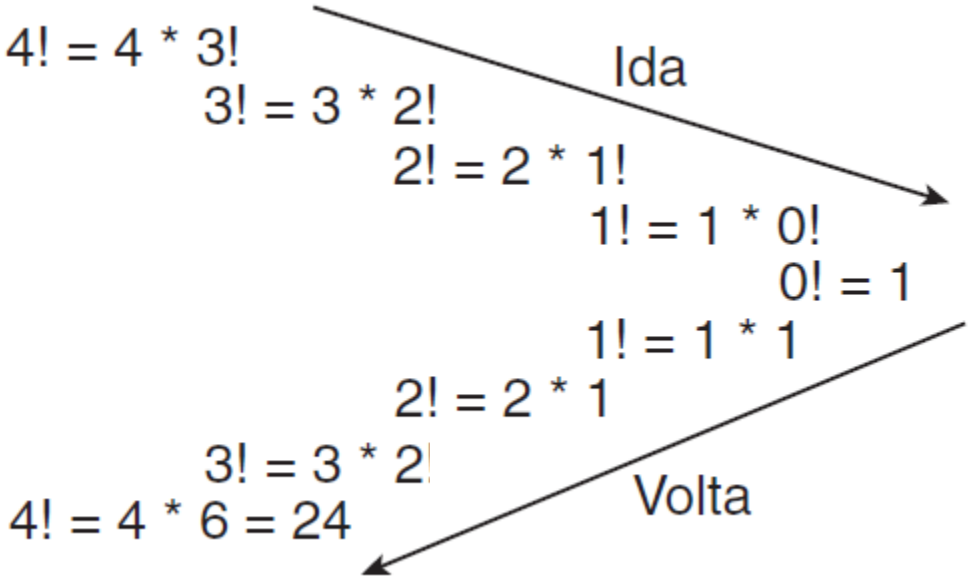
Recursividade

- Um exemplo clássico de função recursiva é o cálculo do fatorial de um número.



Recursividade

- Para isso, precisamos determinar a fórmula geral e o caso base:
  - Fórmula geral: $n! = n \cdot (n-1)!$
  - Caso base: $0! = 1$



Recursividade

- Iterativo

```
int fatorial(int n){
  if(n==0){
    return 1;
  }
  else{
    int i, f=1;
    for(i=1;i<=n;i++){
      f = f * i;
    }
    return f;
  }
}
```

- Recursivo

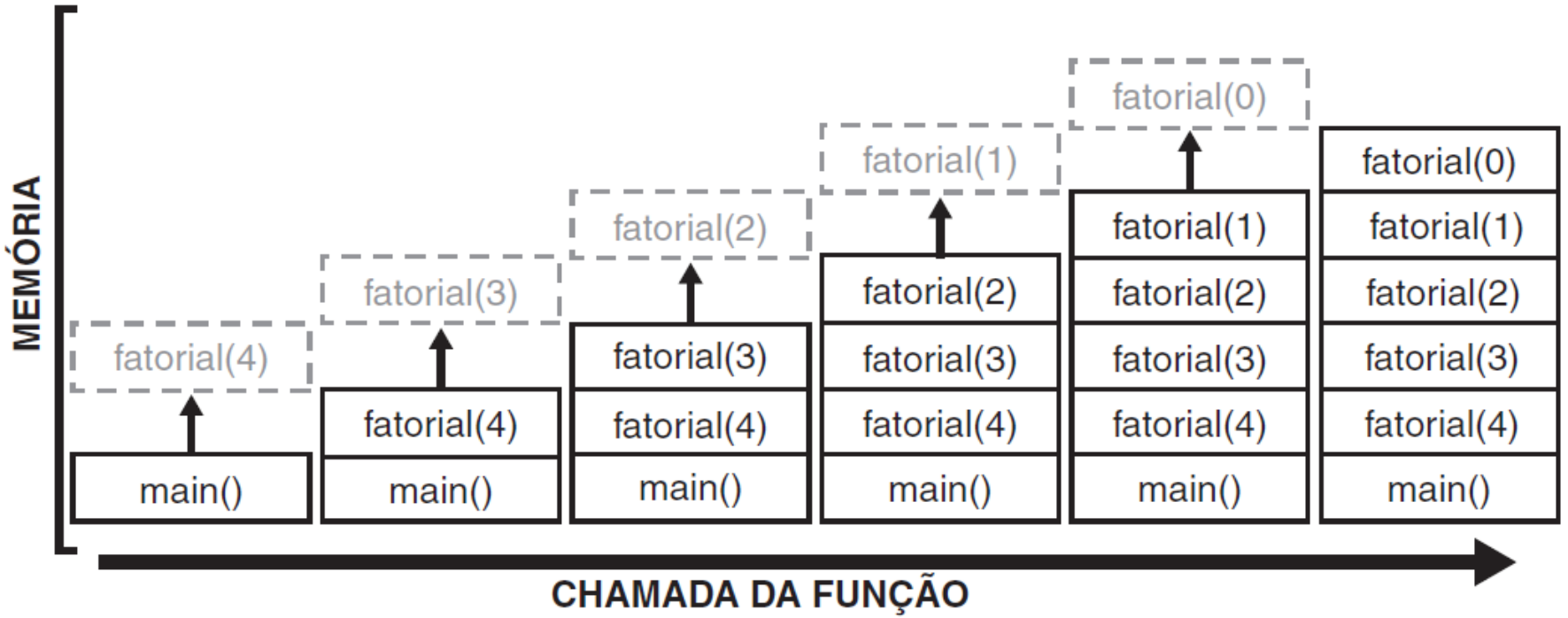
```
int fatorial(int n){
  if(n==0){
    return 1;
  }
  else{
    return n * fatorial(n-1);
  }
}
```

Recursividade

- Em geral, soluções recursivas são consideradas “mais enxutas” ou “mais elegantes” do que soluções iterativas.
- Porém, os algoritmos recursivos tendem a necessitar de mais recurso computacional do que os iterativos.

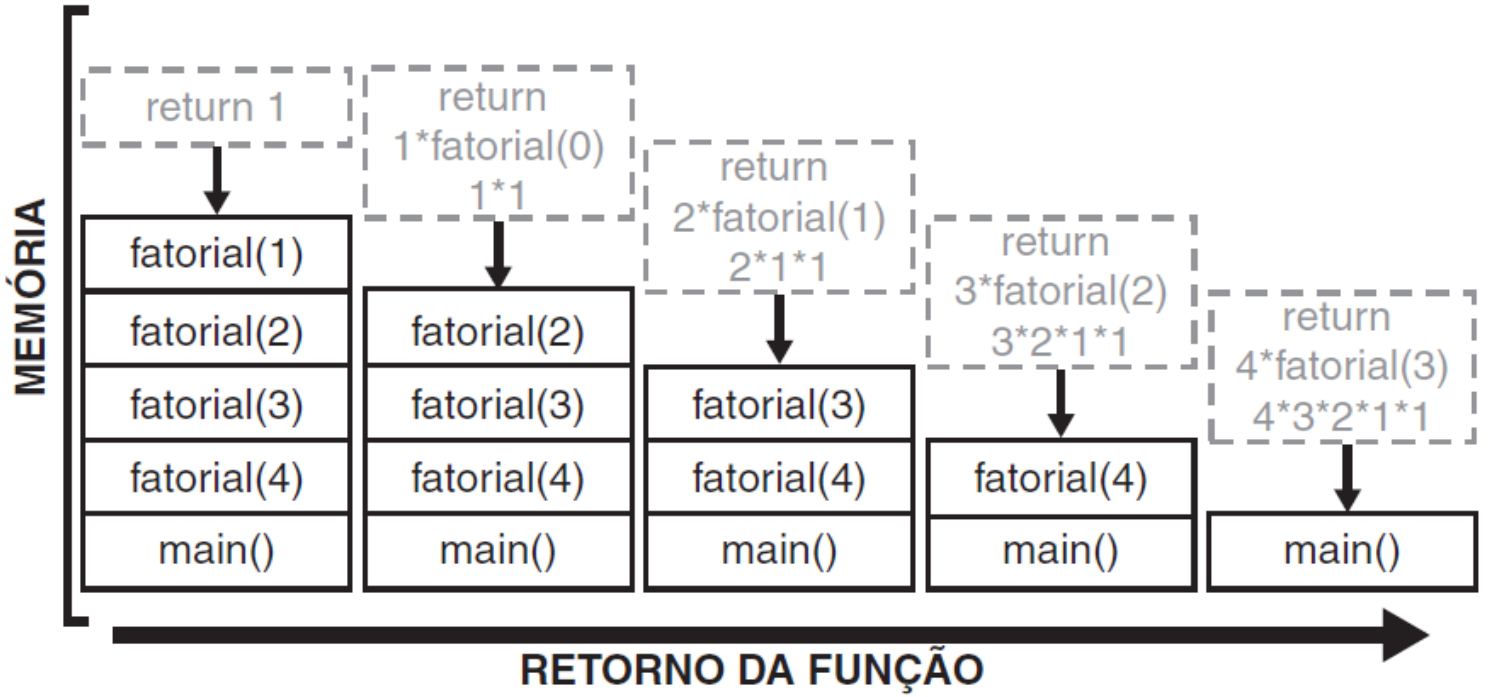
Recursividade

- O que acontece com a chamada da função fatorial durante o cálculo de 4!



Recursividade

- Uma vez que alcançamos o caso base, é hora de fazer o caminho de volta da recursão.

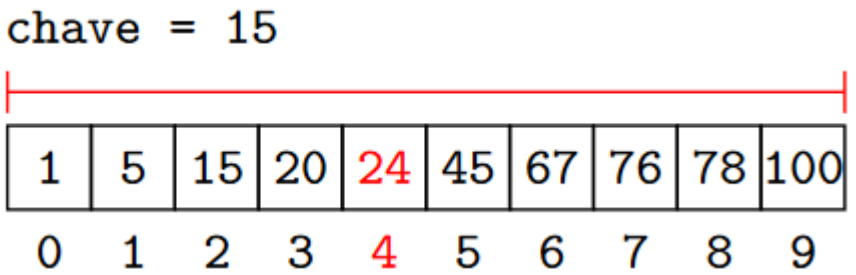


Busca Binária Recursiva

- A ideia do algoritmo é a seguinte:
  - Verifique se a chave de busca é igual ao valor da posição do meio do array;
  - Caso seja igual, devolva esta posição;
  - Caso o valor central seja maior que a chave, então repita o processo, mas considere um array reduzido, com os elementos do começo do array até a posição anterior a do meio;
  - Caso o valor central seja menor que a chave, então repita o processo, mas considere um array reduzido, com os elementos da posição seguinte a do meio até o fim do array;

Busca Binária – Exemplo

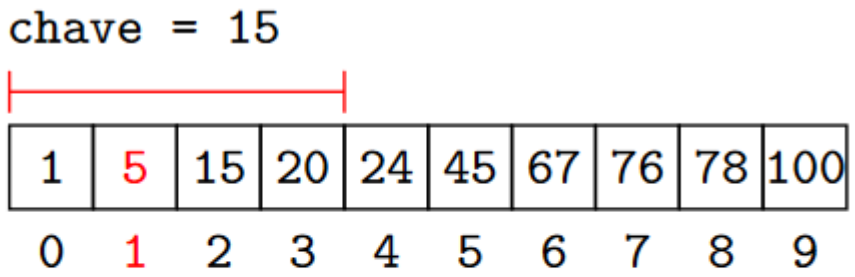
- Considere que desejamos buscar a chave 15.



- Como [pos_meio] > chave, devemos continuar a busca na primeira metade da região.

Busca Binária – Exemplo

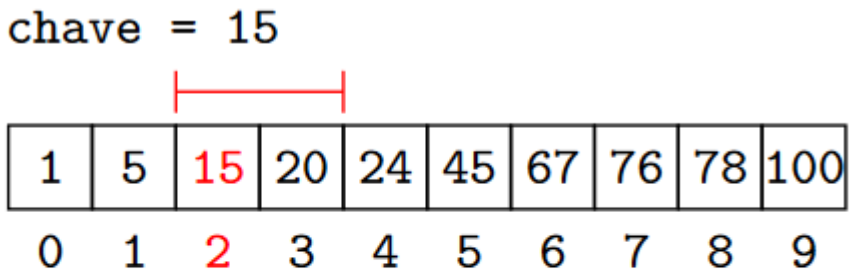
- Considere que desejamos buscar a chave 15.



- Como [pos_meio] < chave, devemos continuar a busca na segunda metade da região.

Busca Binária – Exemplo

- Considere que desejamos buscar a chave 15.



- Finalmente, encontramos a chave ([pos_meio] = chave). Assim, devolvemos sua posição no array (pos_meio).

Busca Binária Iterativa

```
int busca_binaria_iterativa(int v[], int chave, int pos_ini, int pos_fim){
  int pos = -1;
  int pos_meio;
  do{
    pos_meio = (pos_ini+pos_fim)/2;
    if(chave < v[pos_meio]){
      pos_fim = pos_meio-1;
    }
    else{
      pos_ini = pos_meio+1;
    }
  } while (!((chave == v[pos_meio]) || (pos_ini>pos_fim)));
  if(chave == v[pos_meio]){
    pos = pos_meio;
  }
  return pos;
}
```

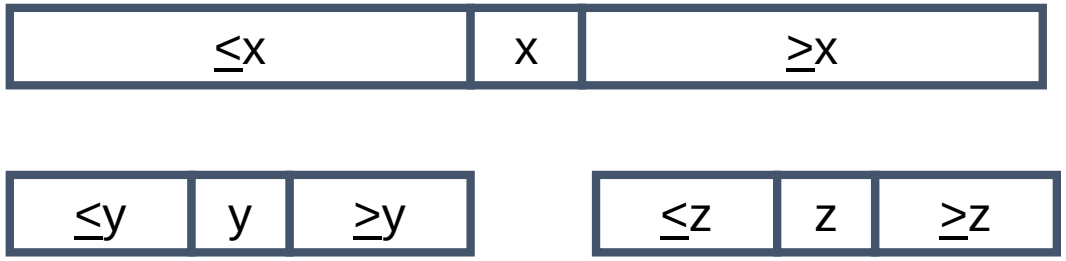
Busca Binária Recursiva

```
int busca_binaria_recursiva(int v[], int chave, int pos_ini, int pos_fim){
  int pos_meio;
  if(pos_ini > pos_fim){
    return -1;
  }
  pos_meio = (pos_ini+pos_fim)/2;
  if(v[pos_meio] == chave){
    return pos_meio;
  }
  else{
    if(v[pos_meio] > chave){
      return busca_binaria_recursiva(v,chave,pos_ini,pos_meio-1);
    }
    else{
      return busca_binaria_recursiva(v,chave,pos_meio+1,pos_fim);
    }
  }
}
```

Ordenação – QuickSort

❑ Escolher, de forma aleatória, um elemento x da sequência v_0, v_1, ..., v_{n-1}, de n elementos.

❑ Percorrer sequência deixando os valores menores que x à sua esquerda e os maiores à sua direita;



❑ Repetir o processo de partição nas duas listas geradas, até que as novas listas possuam apenas um elemento;

Ordenação – Quicksort

```
Inicio do particionar...
L = 0 Pivo = 15 R = 10  Vetor:35 33 52 10 14 15 27 44 26 31 5

Jogando o pivo pra ultima posicao...
L = 0 Pivo = 15 R = 10  Vetor:35 33 52 10 14 5 27 44 26 31 15

Colocando os maiores pra direita e menores pra esquerda do pivo...

********************************** for **********************************
j = 0 i = 0 L = 0 Pivo = 15 R = 10      Vetor:35 33 52 10 14 5 27 44 26 31 15
j = 0 i = 1 L = 0 Pivo = 15 R = 10      Vetor:35 33 52 10 14 5 27 44 26 31 15
j = 0 i = 2 L = 0 Pivo = 15 R = 10      Vetor:35 33 52 10 14 5 27 44 26 31 15
j = 0 i = 3 L = 0 Pivo = 15 R = 10      Vetor:35 33 52 10 14 5 27 44 26 31 15
j = 1 i = 4 L = 0 Pivo = 15 R = 10      Vetor:10 33 52 35 14 5 27 44 26 31 15
j = 2 i = 5 L = 0 Pivo = 15 R = 10      Vetor:10 14 52 35 33 5 27 44 26 31 15
j = 3 i = 6 L = 0 Pivo = 15 R = 10      Vetor:10 14 5 35 33 52 27 44 26 31 15
j = 3 i = 7 L = 0 Pivo = 15 R = 10      Vetor:10 14 5 35 33 52 27 44 26 31 15
j = 3 i = 8 L = 0 Pivo = 15 R = 10      Vetor:10 14 5 35 33 52 27 44 26 31 15
j = 3 i = 9 L = 0 Pivo = 15 R = 10      Vetor:10 14 5 35 33 52 27 44 26 31 15

****************************** fim do for ******************************

Movendo o pivo para a posicao j = 3
L = 0 Pivo = 15 R = 10  Vetor:10 14 5 15 33 52 27 44 26 31 35
```

Ordenação – Quicksort

```
Inicio do particionar...
L = 0 Pivo = 14 R = 2   Vetor:10 14 5 15 33 52 27 44 26 31 35

Jogando o pivo pra ultima posicao...
L = 0 Pivo = 14 R = 2   Vetor:10 5 14 15 33 52 27 44 26 31 35

Colocando os maiores pra direita e menores pra esquerda do pivo...

********************************** for ***********************************
j = 0 i = 0 L = 0 Pivo = 14 R = 2       Vetor:10 5 14 15 33 52 27 44 26 31 35
j = 1 i = 1 L = 0 Pivo = 14 R = 2       Vetor:10 5 14 15 33 52 27 44 26 31 35

******************************** fim do for ******************************

Movendo o pivo para a posicao j = 2
L = 0 Pivo = 14 R = 2   Vetor:10 5 14 15 33 52 27 44 26 31 35
```

Ordenação – Quicksort

```
Inicio do particionar...
L = 0 Pivo = 10 R = 1   Vetor:10 5 14 15 33 52 27 44 26 31 35

Jogando o pivo pra ultima posicao...
L = 0 Pivo = 10 R = 1   Vetor:5 10 14 15 33 52 27 44 26 31 35

Colocando os maiores pra direita e menores pra esquerda do pivo...

********************************** for **********************************
j = 0 i = 0 L = 0 Pivo = 10 R = 1       Vetor:5 10 14 15 33 52 27 44 26 31 35

******************************** fim do for ********************************

Movendo o pivo para a posicao j = 1
L = 0 Pivo = 10 R = 1   Vetor:5 10 14 15 33 52 27 44 26 31 35
```

Ordenação – Quicksort

```
Inicio do particionar...
L = 4 Pivo = 44 R = 10  Vetor:5 10 14 15 33 52 27 44 26 31 35

Jogando o pivo pra ultima posicao...
L = 4 Pivo = 44 R = 10  Vetor:5 10 14 15 33 52 27 35 26 31 44

Colocando os maiores pra direita e menores pra esquerda do pivo...

********************************** for **********************************
j = 4 i = 4 L = 4 Pivo = 44 R = 10      Vetor:5 10 14 15 33 52 27 35 26 31 44
j = 5 i = 5 L = 4 Pivo = 44 R = 10      Vetor:5 10 14 15 33 52 27 35 26 31 44
j = 5 i = 6 L = 4 Pivo = 44 R = 10      Vetor:5 10 14 15 33 52 27 35 26 31 44
j = 6 i = 7 L = 4 Pivo = 44 R = 10      Vetor:5 10 14 15 33 27 52 35 26 31 44
j = 7 i = 8 L = 4 Pivo = 44 R = 10      Vetor:5 10 14 15 33 27 35 52 26 31 44
j = 8 i = 9 L = 4 Pivo = 44 R = 10      Vetor:5 10 14 15 33 27 35 26 52 31 44

******************************** fim do for ********************************

Movendo o pivo para a posicao j = 9
L = 4 Pivo = 44 R = 10  Vetor:5 10 14 15 33 27 35 26 31 44 52
```

Ordenação – Quicksort

```
Inicio do particionar...
L = 4 Pivo = 35 R = 8   Vetor:5 10 14 15 33 27 35 26 31 44 52

Jogando o pivo pra ultima posicao...
L = 4 Pivo = 35 R = 8   Vetor:5 10 14 15 33 27 31 26 35 44 52

Colocando os maiores pra direita e menores pra esquerda do pivo...

********************************** for **********************************
j = 4 i = 4 L = 4 Pivo = 35 R = 8       Vetor:5 10 14 15 33 27 31 26 35 44 52
j = 5 i = 5 L = 4 Pivo = 35 R = 8       Vetor:5 10 14 15 33 27 31 26 35 44 52
j = 6 i = 6 L = 4 Pivo = 35 R = 8       Vetor:5 10 14 15 33 27 31 26 35 44 52
j = 7 i = 7 L = 4 Pivo = 35 R = 8       Vetor:5 10 14 15 33 27 31 26 35 44 52

******************************** fim do for ********************************

Movendo o pivo para a posicao j = 8
L = 4 Pivo = 35 R = 8   Vetor:5 10 14 15 33 27 31 26 35 44 52
```

Ordenação – Quicksort

```
Inicio do particionar...
L = 4 Pivo = 27 R = 7   Vetor:5 10 14 15 33 27 31 26 35 44 52

Jogando o pivo pra ultima posicao...
L = 4 Pivo = 27 R = 7   Vetor:5 10 14 15 33 26 31 27 35 44 52

Colocando os maiores pra direita e menores pra esquerda do pivo...

********************************** for **********************************
j = 4 i = 4 L = 4 Pivo = 27 R = 7       Vetor:5 10 14 15 33 26 31 27 35 44 52
j = 4 i = 5 L = 4 Pivo = 27 R = 7       Vetor:5 10 14 15 33 26 31 27 35 44 52
j = 5 i = 6 L = 4 Pivo = 27 R = 7       Vetor:5 10 14 15 26 33 31 27 35 44 52

******************************** fim do for ********************************

Movendo o pivo para a posicao j = 5
L = 4 Pivo = 27 R = 7   Vetor:5 10 14 15 26 27 31 33 35 44 52
```

Ordenação – Quicksort

```
Inicio do particionar...
L = 6 Pivo = 31 R = 7   Vetor:5 10 14 15 26 27 31 33 35 44 52

Jogando o pivo pra ultima posicao...
L = 6 Pivo = 31 R = 7   Vetor:5 10 14 15 26 27 33 31 35 44 52

Colocando os maiores pra direita e menores pra esquerda do pivo...

********************************** for **********************************
j = 6 i = 6 L = 6 Pivo = 31 R = 7       Vetor:5 10 14 15 26 27 33 31 35 44 52

******************************** fim do for ********************************

Movendo o pivo para a posicao j = 6
L = 6 Pivo = 31 R = 7   Vetor:5 10 14 15 26 27 31 33 35 44 52
```

Atividade

- Abra a pasta Downloads no VS Code
  - Utilize o comando git clone https://github.com/ECM404/atividade3.git --recursive
  - Entre na pasta utilizando o comando cd atividade3
  - Inicialize o diretório com o comando make install