LUCAS BASTOS FRANCO – 2310622

# Resumo dos Métodos Sort Árvores

Matéria: Árvores e Grafos

Docente: Willian Júnior

2025 – Anápolis

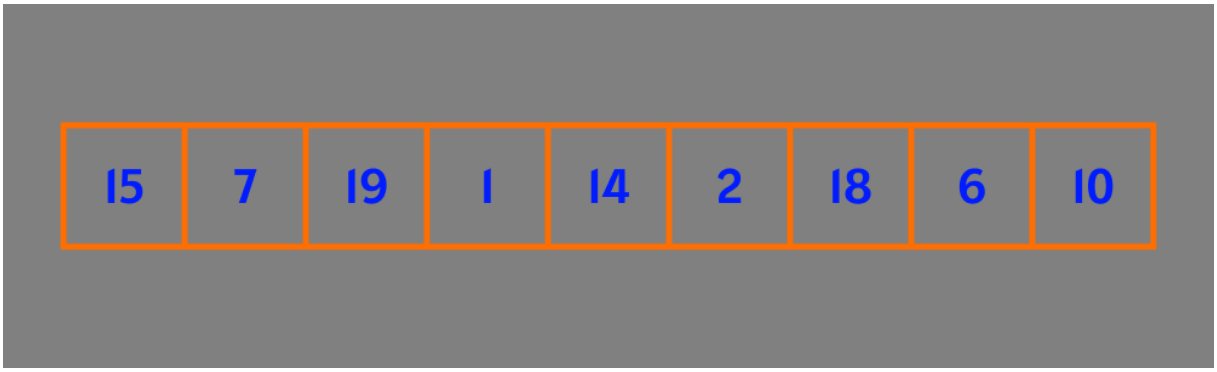
# SUMÁRIO

# INTRODUÇÃO

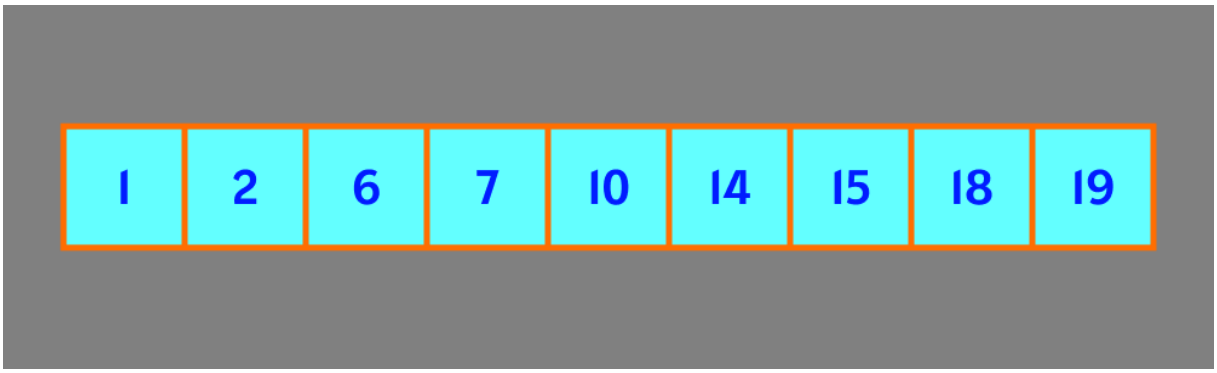
Hoje vamos ver a diferença e sua aplicação em um algoritmo de ordenação. O algoritmo vai efetuar sua ordenação em uma certa ordem. O objetivo da ordenação é facilitar a recuperação dos dados de uma lista. Os algoritmos escolhidos são, Bubble Sort, Selection Sort, Quick Sort e Insertion Sort.

# PADRÃO DE DADOS

Para a sequência de dados, para você verem a aplicação de cada método, vamos utilizar 1d20 e colocar os dados numa lista de 9 espaços:




Como exemplo, vamos utilizar da ordenação crescente, logicamente, a lista no final têm que ficar assim:




# BUBBLE SORT

## EXPLICAÇÃO

Bubble sort é o algoritmo mais simples, mas o menos eficientes. Neste algoritmo, ele vai olhar para cada posição da lista, e vai comparar com o próximo de sua posição. Ou seja, ele vai olhar o elemento da 2º posição e vai comparar com o elemento da 3º

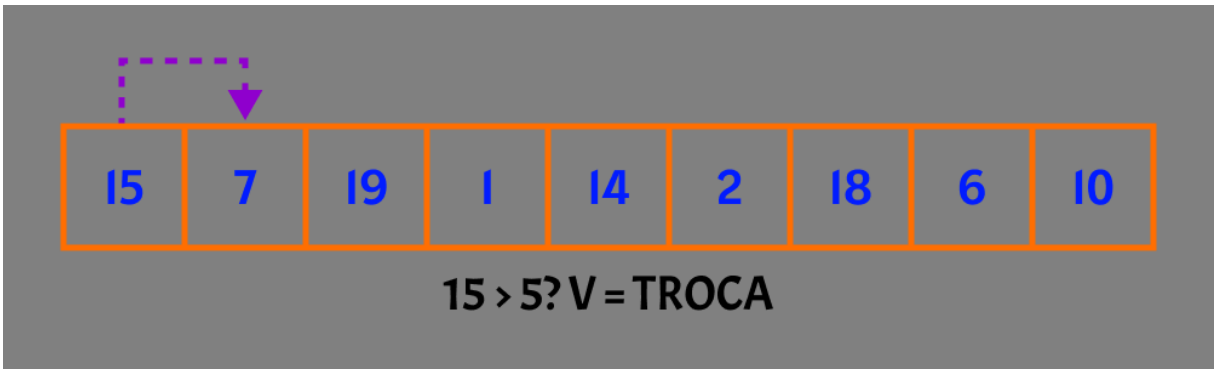
posição, se caso o elemento da 2º posição for maior que o elemento da 3º posição, eles realizam a troca.

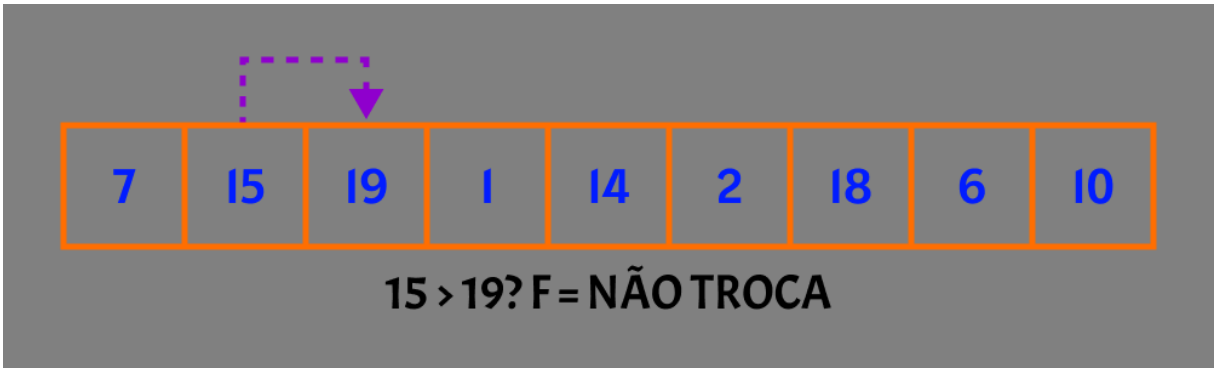
**CODIGO EM C**

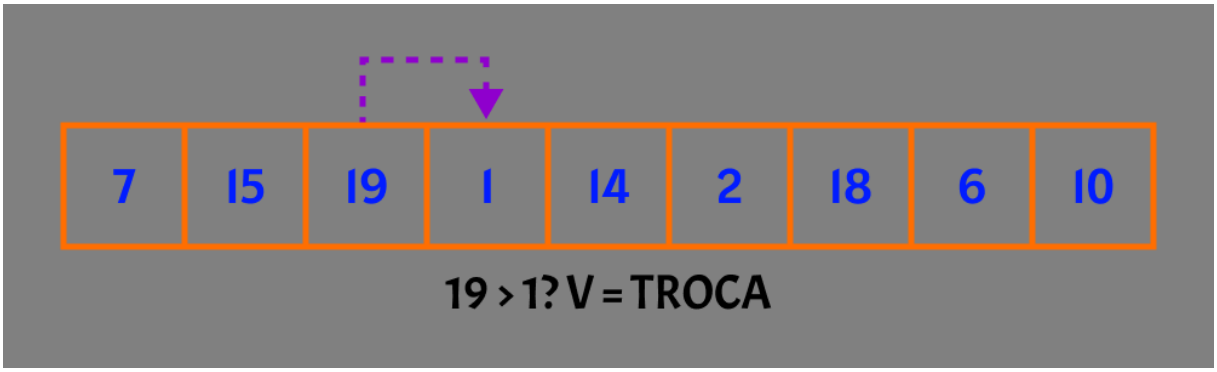
```
do {
    int trocou = 0;
    for(int i = 0; i < TAMANHO_MAX_LISTA - 1; i++) {
        if (lista[i] > lista[i + 1]) {
            int temp = lista[i];
            lista[i] = lista[i + 1];
            lista[i + 1] = temp;
            trocou = 1;
        }
    }
} while (trocou);
```

**TESTE DE PECORRIDAS**

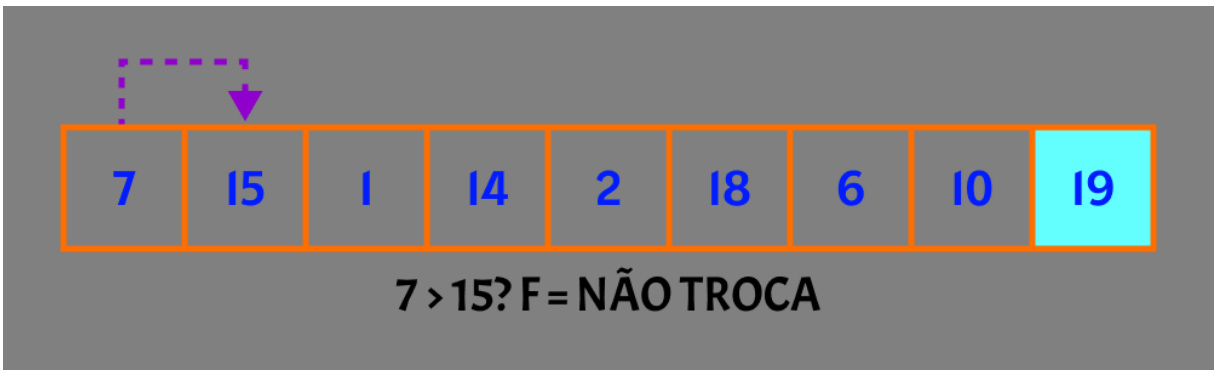
Com a base na lista pré-definida de dados, ele vai percorrer pela sua primeira vez

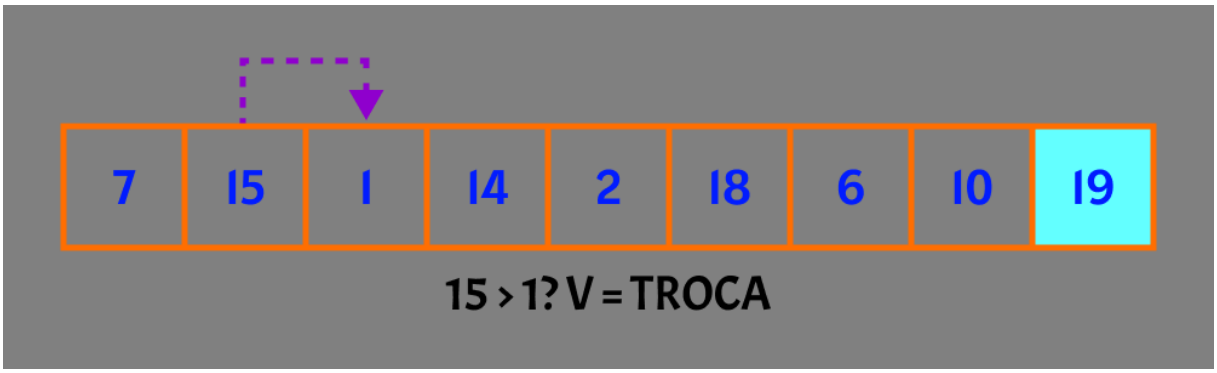


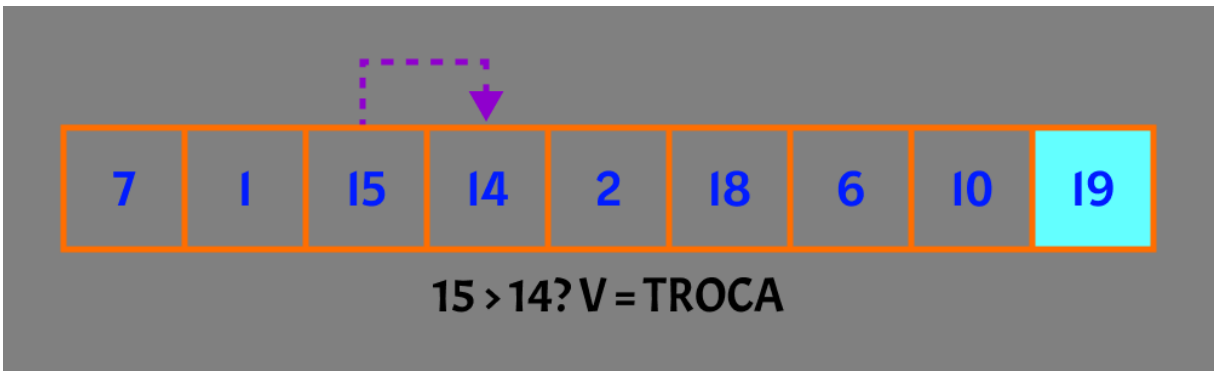






Agora ele irá comparar cada posição e vai realizar a troca, pois todos os números são menores que 19. Quando ele colocar o 19 no final, ele vai percorrer a lista novamente.

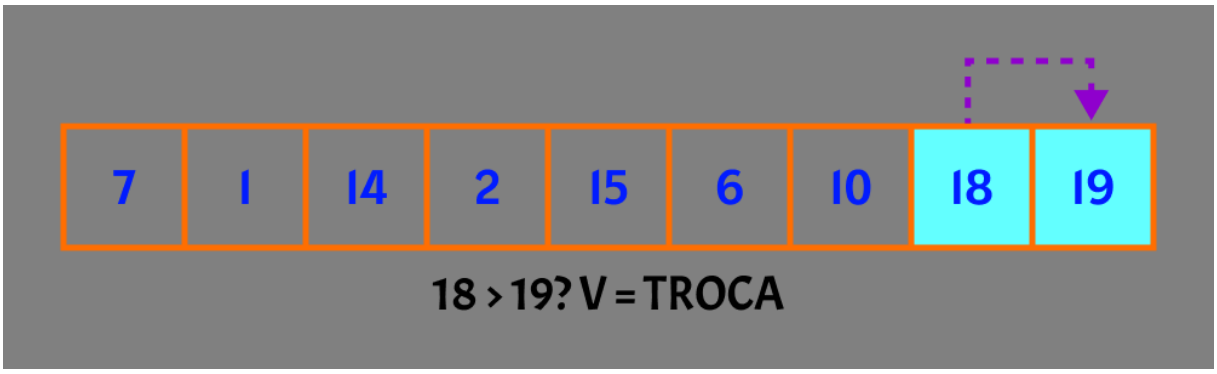


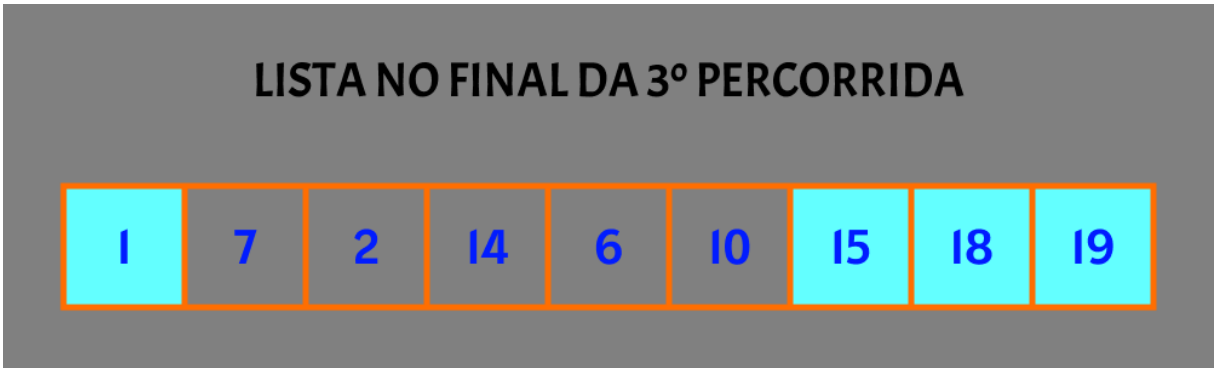




Temos o adendo para acrescenta, antes de realizar a segunda percorrida, o 19 já estava na posição correta, e quando foi realizada a segunda percorrida, o 18 também

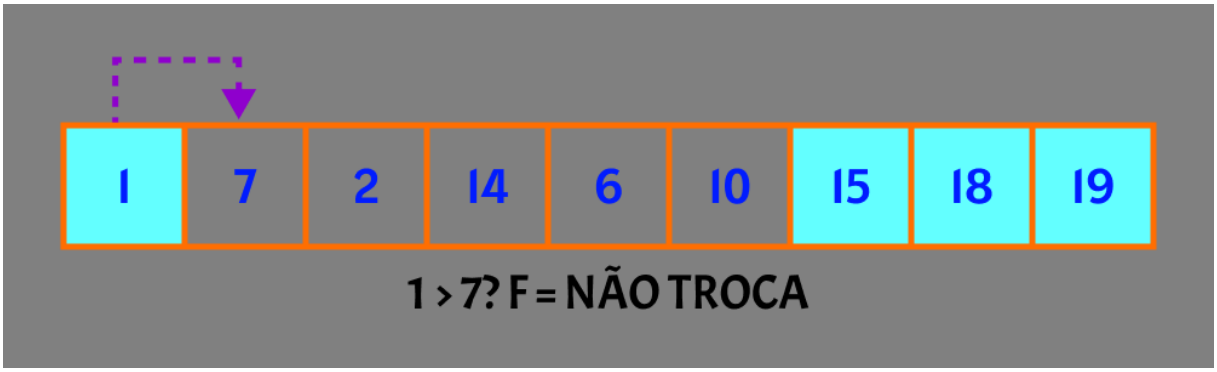
chegou em sua posição. Mesmo o 19 já estando em sua posição correta, e o 18 também, ele irá fazer uma comparação dos dois elementos em todas percorridas que tiverem.



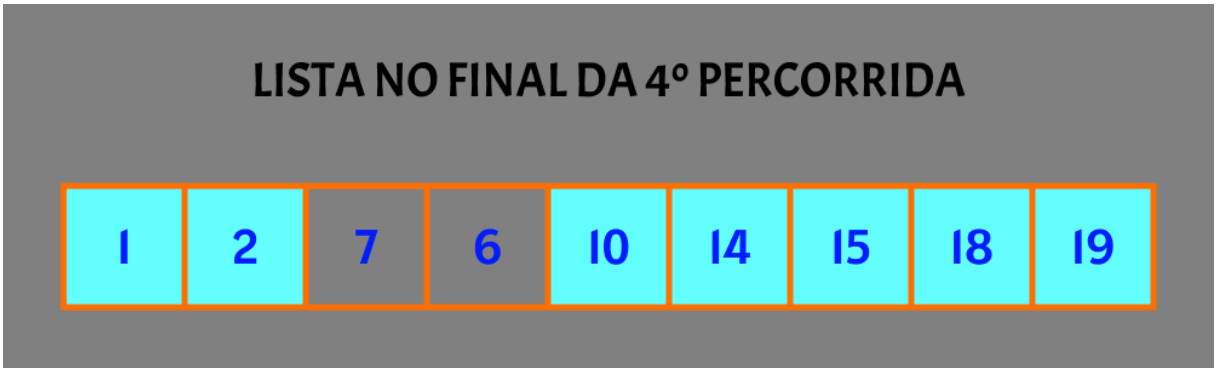

Agora finalizado a sua segunda percorrida, ele irá fazer isso até ordenar todos os elementos:

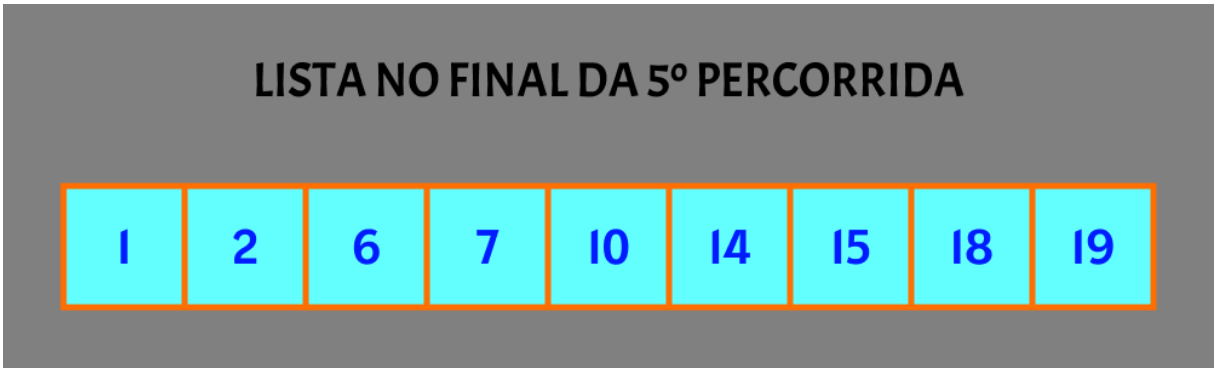



Pausei aqui para afirmar novamente: do mesmo jeito que ele vai comparar 18 e 19 em todas as percorridas, ele vai comparar o 1 com a próxima posição e o 15 com 18 em todas as percorridas.




Agora vamos continuar com a percorridas:




Pronto? Não, mesmo na 5º percorrida tendo ordenado os últimos que faltava, ele vai percorrer mais uma vez para verificar se realmente todos foram colocados.

**RESULTADO DO TEXTO**

| BUBBLE SORT | |
|---|---|
| PECORRIDAS | 6 |
| TEMPO | 2.000 milissegundos |
| COMPARAÇÃOS | 48 |
| MOVIMENTOS | 20 |

## SELECTION SORT

**EXPLICAÇÃO**

Este algoritmo é baseado em se passar sempre o menor valor do vetor para a primeira posição (ou o maior dependendo da ordem requerida), depois o segundo menor valor para a segunda posição e assim sucessivamente, até os últimos dois elementos.

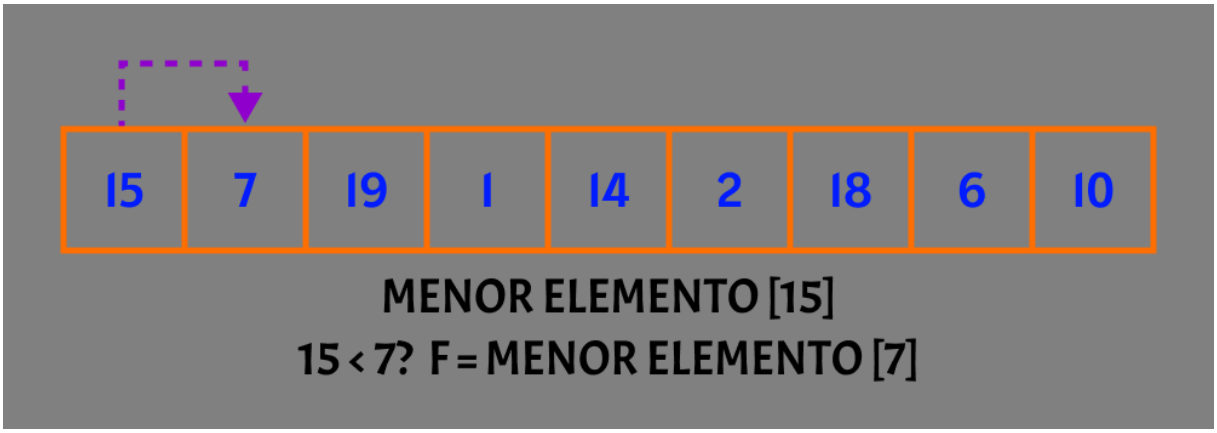
Basicamente, na primeira percorrida, ele pegar o elemento da 1º posição, e vai comparar com todos os outros elementos, e vai trocar com a posição que tiver o menor elemento; depois na segunda percorrida, ele vai pegar o elemento da 2º posição e vai comparar com todos os elementos, menos o da primeira posição, e vai trocar com a posição que tiver o menor elemento, e assim, suscetivelmente até ficar ordenado.

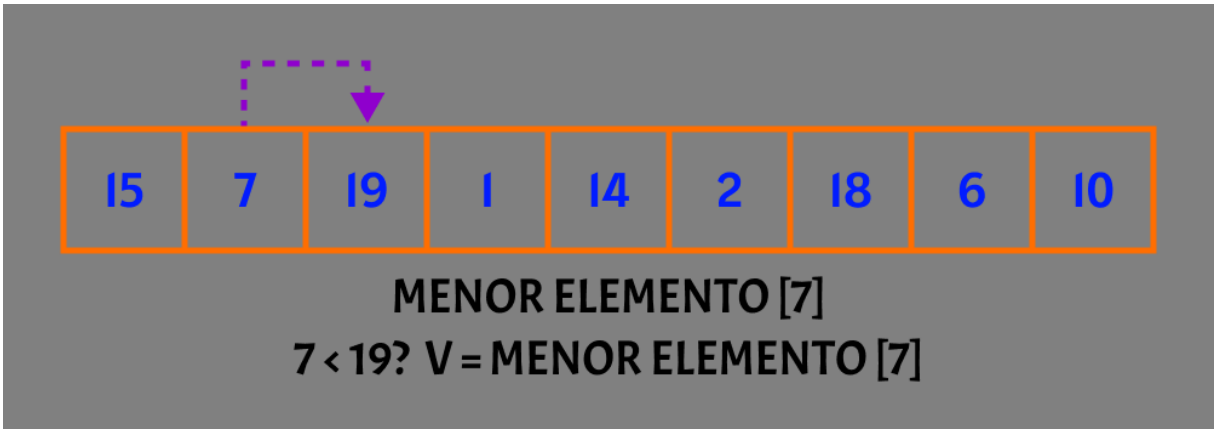
**CODIGO EM C**

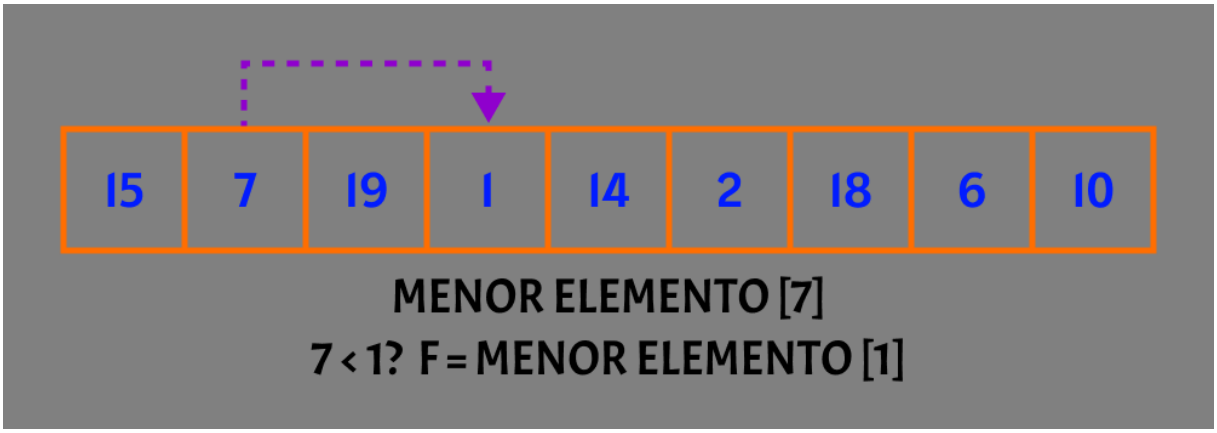
```
for(int i = 0; i < TAMANHO_MAX_LISTA - 1; i++) {
    int posicao = i;
    for(int j = i + 1; j < TAMANHO_MAX_LISTA; j++) {
        if (lista[j] < lista[posicao]) {
            posicao = j;
        }
    }
    if (posicao != i) {
        int temp = lista[i];
        lista[i] = lista[posicao];
        lista[posicao] = temp;
    }
}
```

**TESTE DE PECORRIDAS**

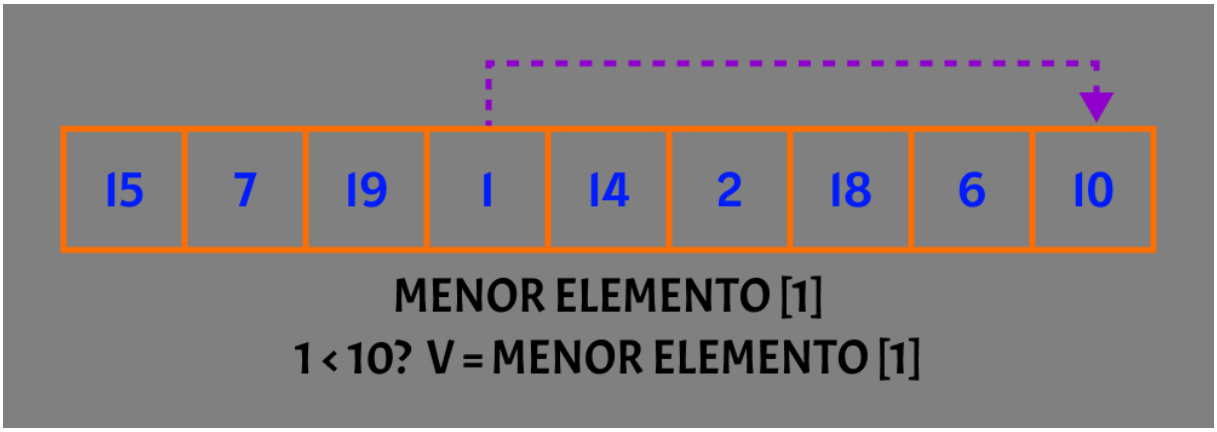
Com a base na lista pré-definida de dados, ele vai percorrer pela sua primeira vez.



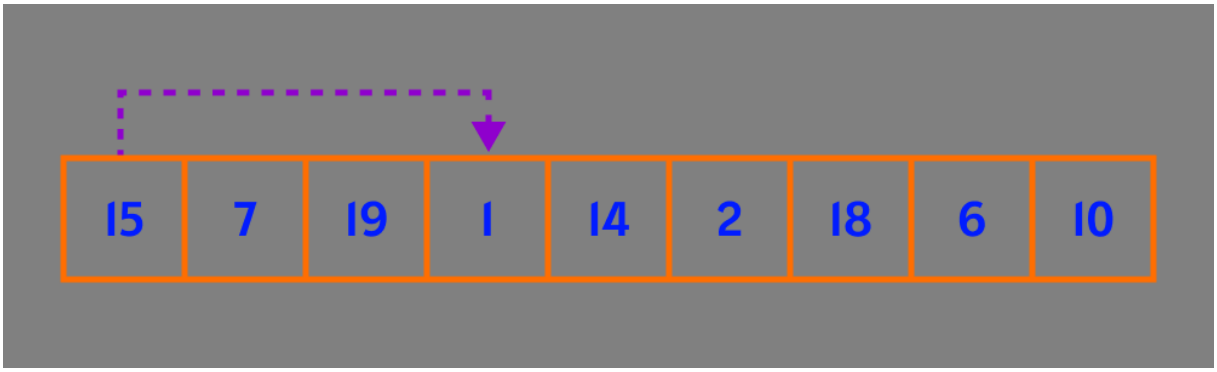


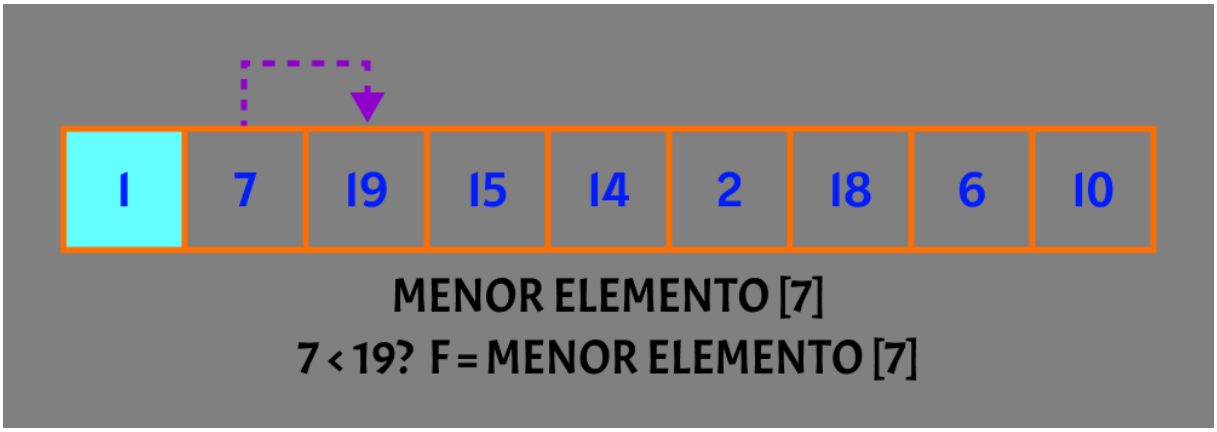

Depois que ele percorrer até o final, verificando se todos os elementos são menores do que o elemento que foi selecionado o menor

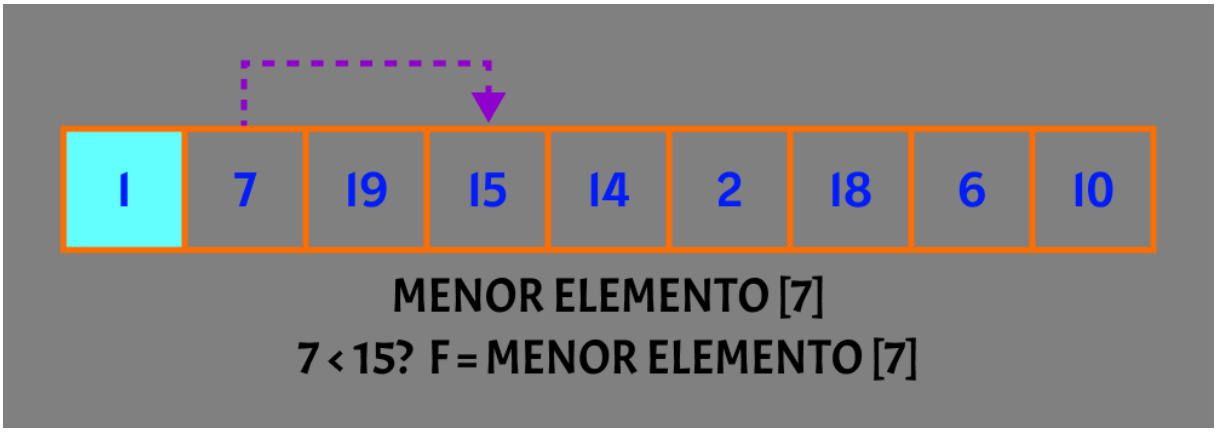


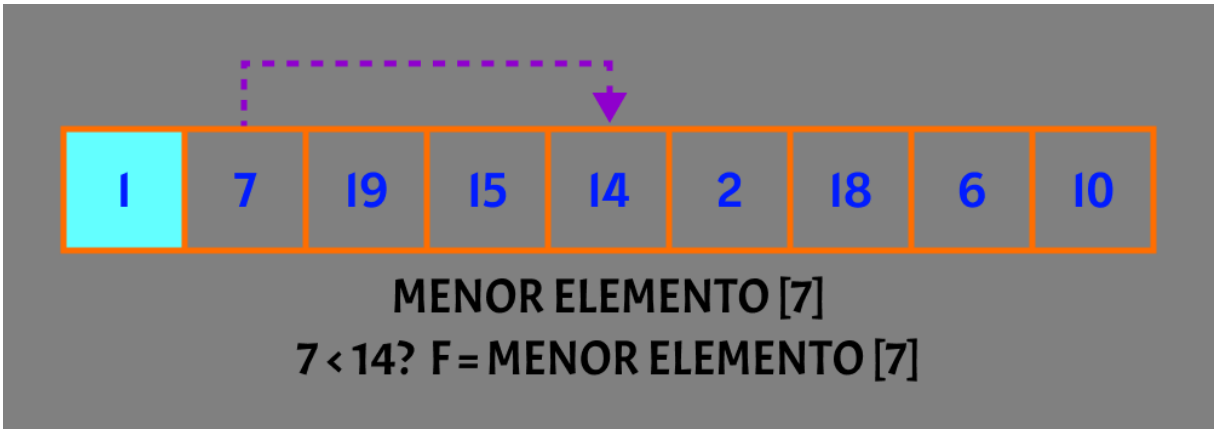

Depois ele vai fazer a troca com a posição do menor elemento.



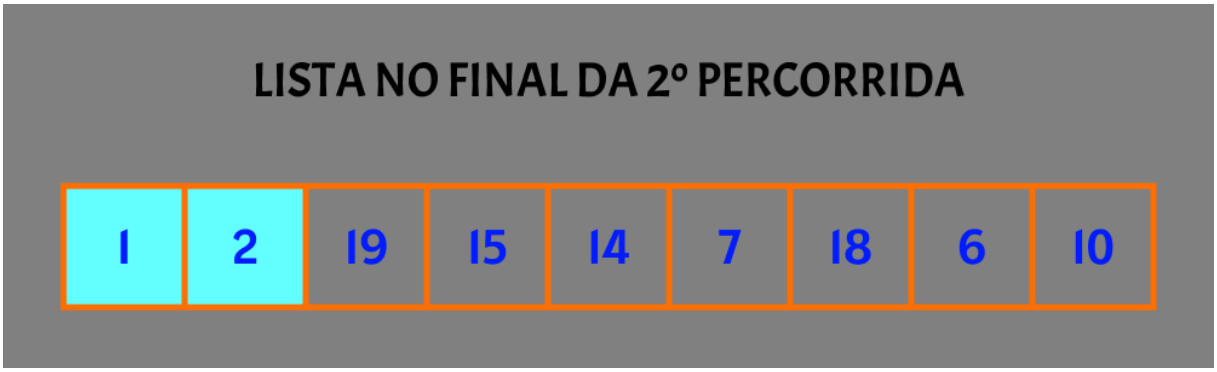

E assim, termina a primeiro percorrida, iniciando a segunda

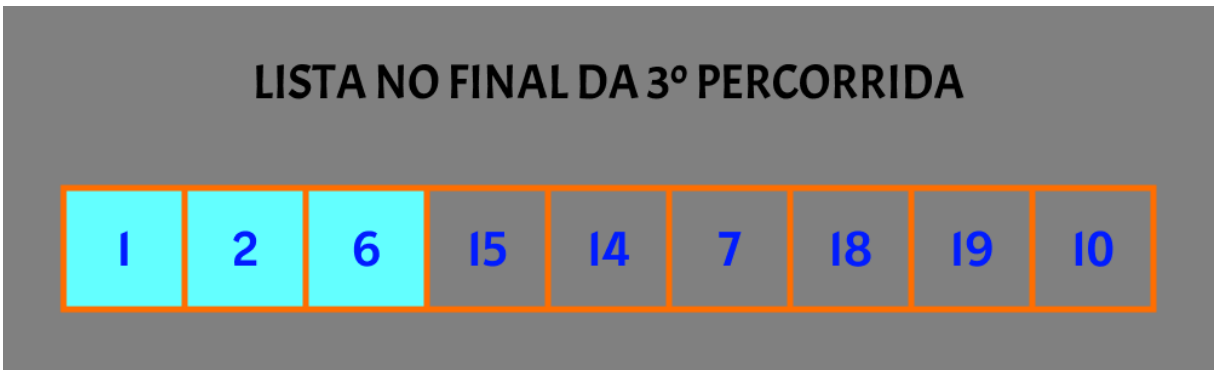


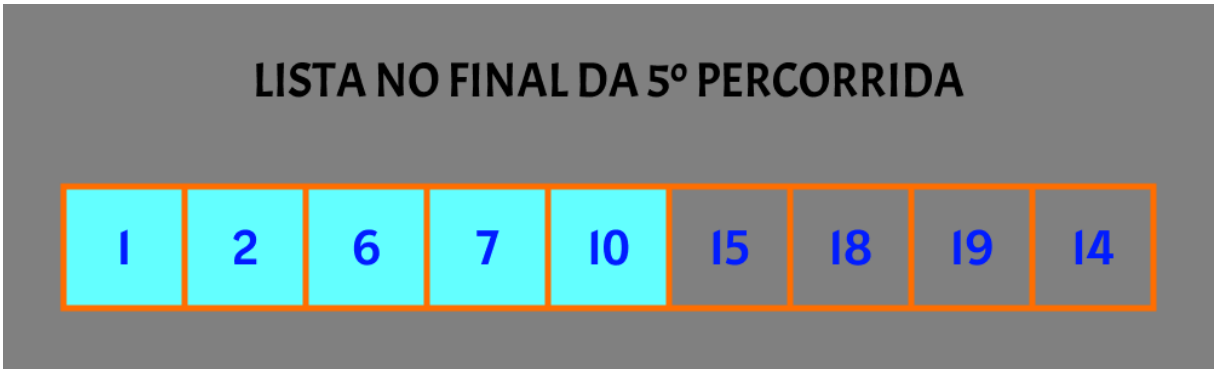



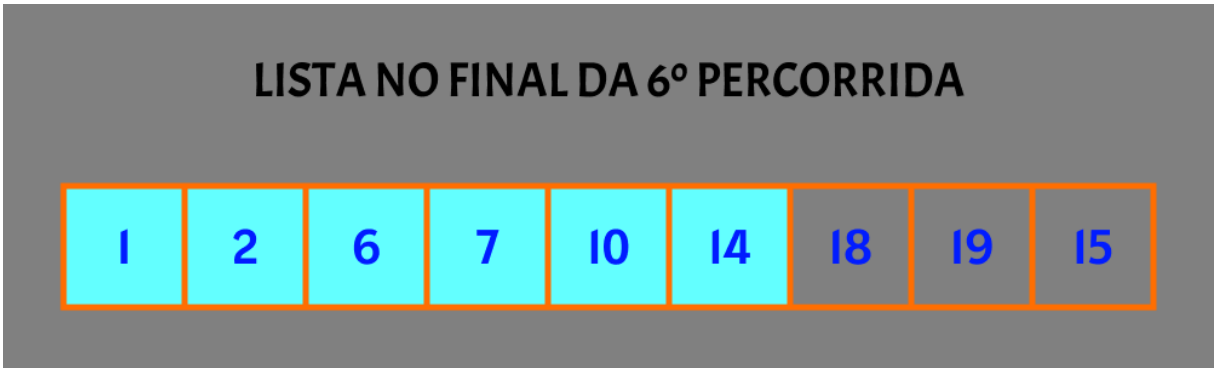
Agora estamos em sua segunda percorrida, ele vai fazer essas comparação e única troca até a lista ficar ordenada:

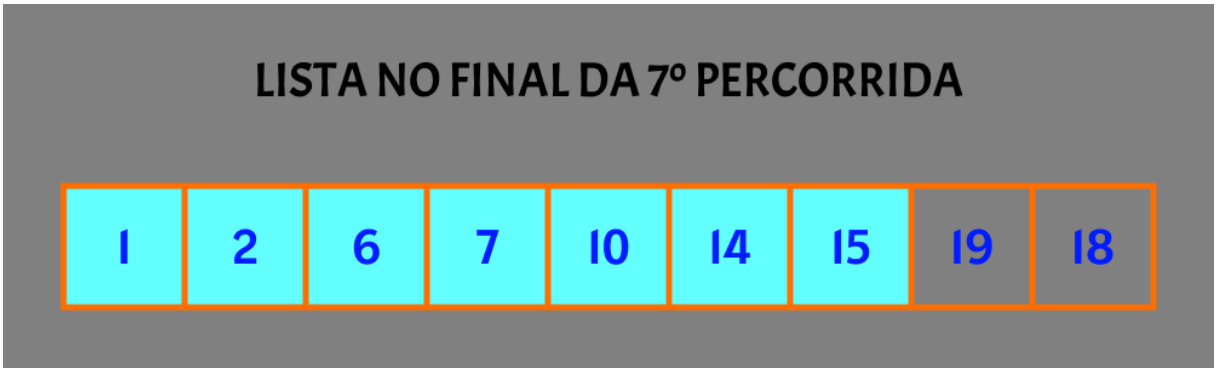


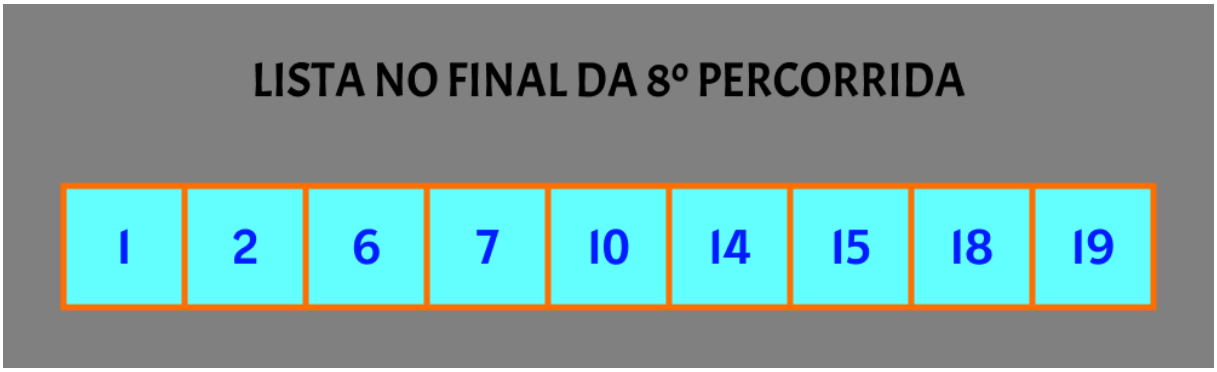










**RESULTADO DO TEXTO**

| SELECTION SORT | |
|---|---|
| PECORRIDAS | 8 |
| TEMPO | 5.000 milissegundos |
| COMPARAÇÃOS | 36 |
| MOVIMENTOS | 8 |

# INSERTION SORT

## EXPLICAÇÃO

O Insertion sort é um algoritmo simples e eficiente quando aplicado em pequenas listas. Neste algoritmo a lista é percorrida da esquerda para a direita, à medida que avança vai deixando os elementos mais à esquerda ordenados. O algoritmo funciona da mesma forma que as pessoas usam para ordenar cartas em um jogo de baralho.

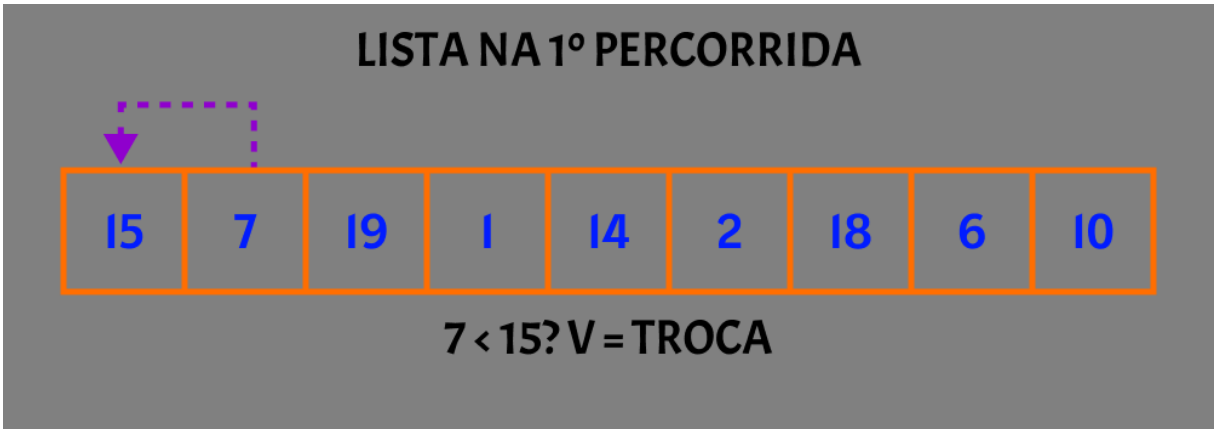
Basicamente, ele vai pegar uma posição da lista e vai comparar com todos que são antes dele ou depois dele, dependendo do sistema. Exemplo, o sistema vai ordenar em ordem crescente e ele pegar o elemento da 3º posição, ele vai comparar o elemento da 3º posição com o elemento da 2º posição, se o elemento da 3º for menor que o elemento da 2º, ele também vai comparar o elemento da 3º com o elemento da 1º, se o elemento da 3º for menor que o elemento da 1º, o elemento da 2º vai ficar na posição do 3º, o elemento da 1º vai ficar na posição do 2º, o elemento da 3º vai ficar na posição do 1º; se caso o elemento da 3º não for menor que o elemento da 1º, os elementos da 3º e da 2º trocam de lugares.

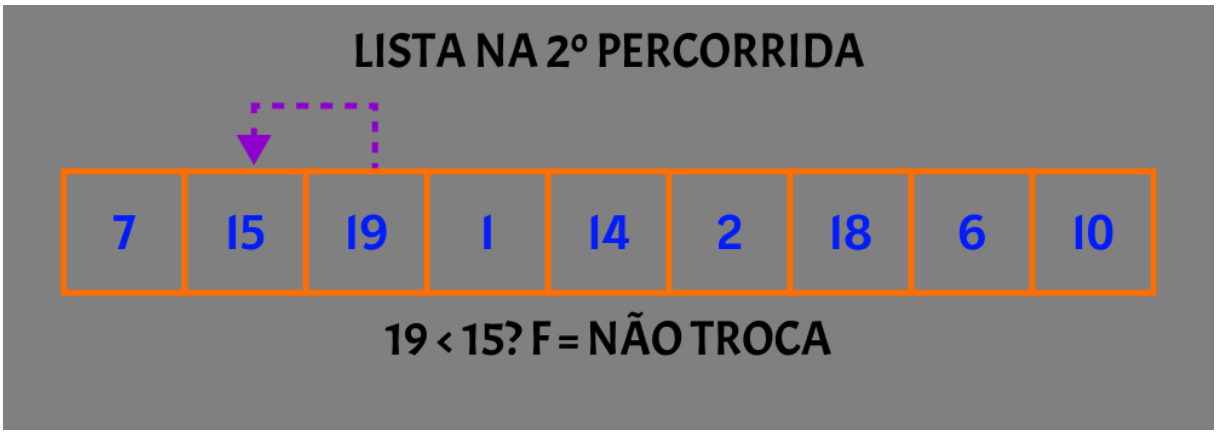
## CODIGO EM C

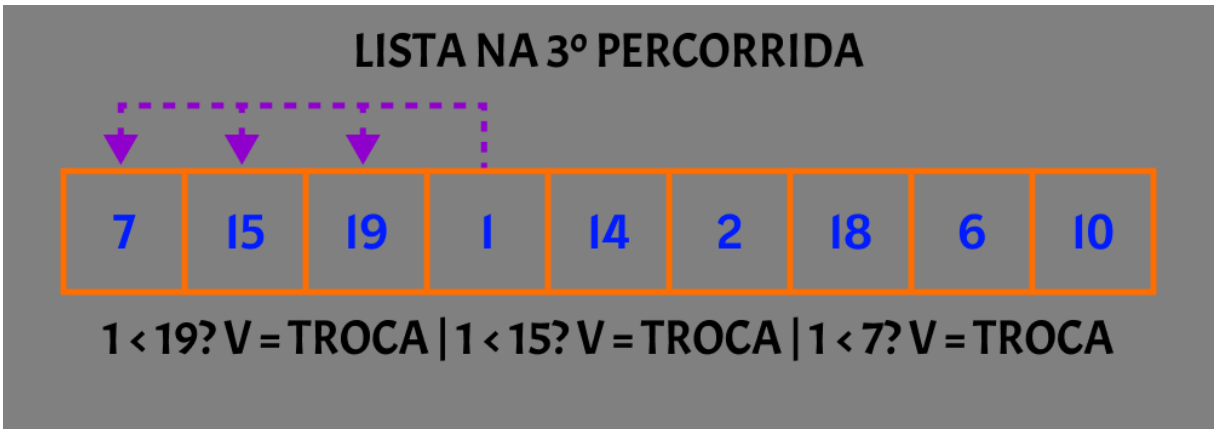
```
for (int i = 1; i < TAMANHO_MAX_LISTA; i++) {
    int chave = lista[i];
    int j = i - 1;
    while (j >= 0 && lista[j] > chave) {
        lista[j + 1] = lista[j];
        j = j - 1;
    }
    lista[j + 1] = chave;
}
```

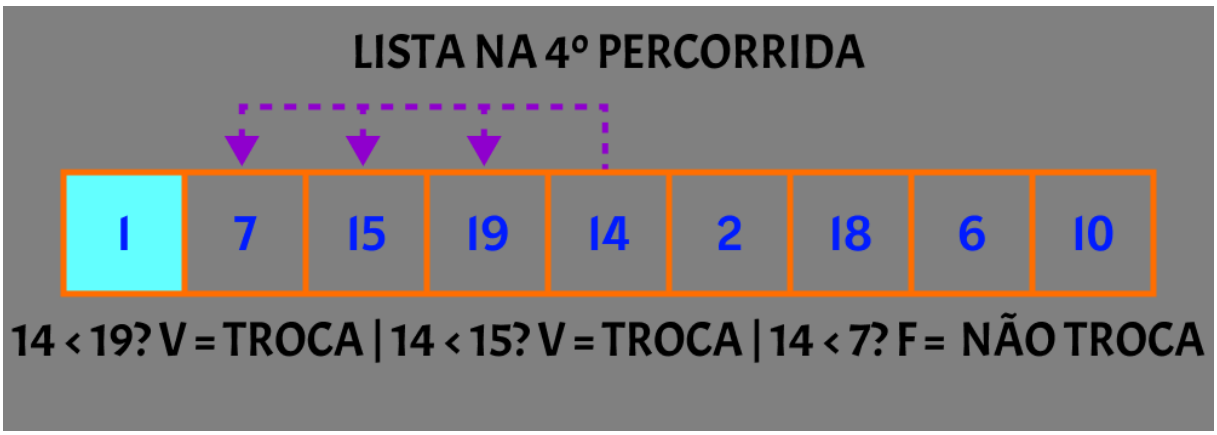
## TESTE DE PECORRIDAS

Com a base na lista pré-definida de dados, ele vai percorrer até os dados serem ordenados:

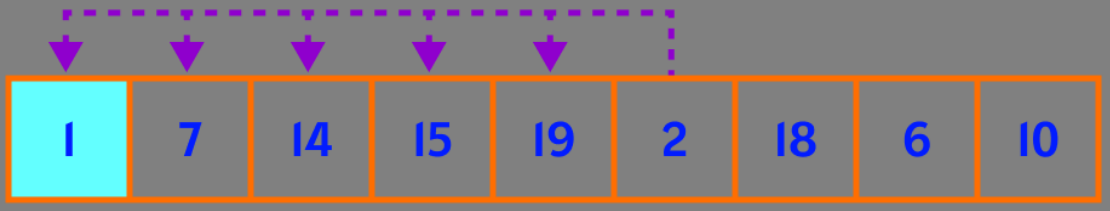

LISTA NA 1º PERCORRIDA
15 7 19 1 14 2 18 6 10
7 < 15? V = TROCA


LISTA NA 2º PERCORRIDA
7 15 19 1 14 2 18 6 10
19 < 15? F = NÃO TROCA


LISTA NA 3º PERCORRIDA
7 15 19 1 14 2 18 6 10
1 < 19? V = TROCA | 1 < 15? V = TROCA | 1 < 7? V = TROCA

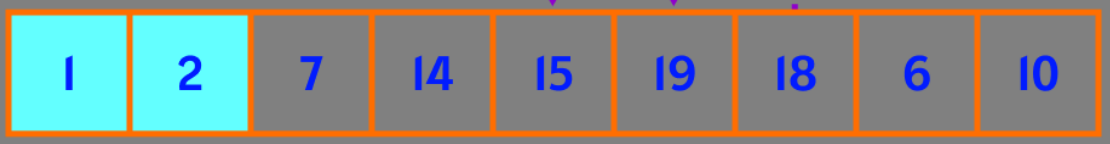

LISTA NA 4º PERCORRIDA
1 7 15 19 14 2 18 6 10
14 < 19? V = TROCA | 14 < 15? V = TROCA | 14 < 7? F = NÃO TROCA

LISTA NA 5º PERCORRIDA



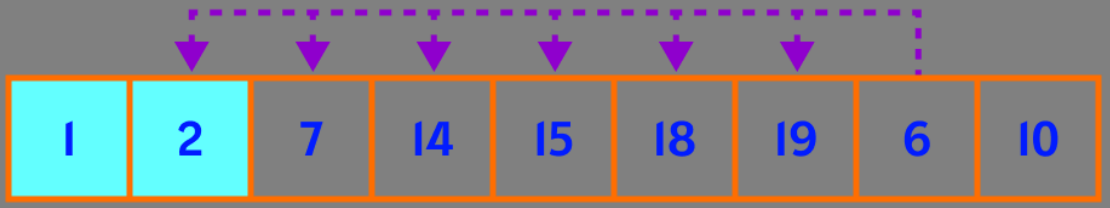

2 < 19? V = TROCA | 2 < 15? V = TROCA | 2 < 14? V = TROCA
2 < 7? V = TROCA | 2 < 1? F = NÃO TROCA

LISTA NA 6º PERCORRIDA



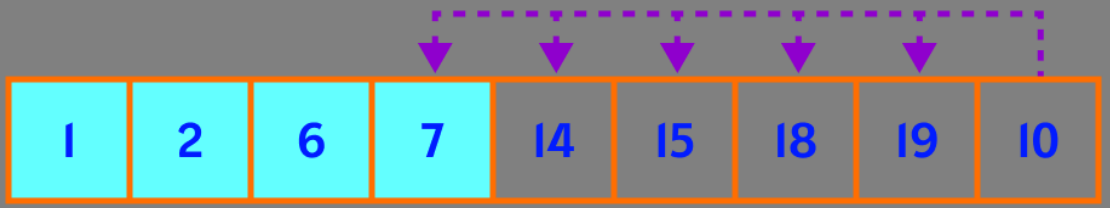

18 < 19? V = TROCA | 18 < 15? F = NÃO TROCA

LISTA NA 7º PERCORRIDA

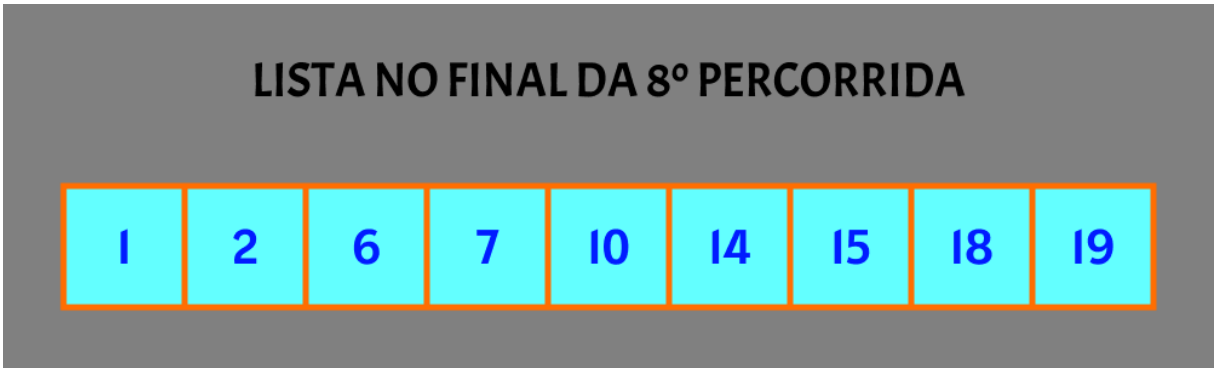



6 < 19? V = TROCA | 6 < 18? V = TROCA | 6 < 15? V = TROCA
6 < 14? V = TROCA | 6 < 7? V = TROCA | 6 < 2? F = NÃO TROCA

LISTA NA 8º PERCORRIDA




10 < 19? V = TROCA | 10 < 18? V = TROCA | 10 < 15? V = TROCA
10 < 14? V = TROCA | 10 < 7? F = NÃO TROCA




**RESULTADO DO TEXTO**

| INSERTION SORT | |
|---|---|
| PECORRIDAS | 8 |
| TEMPO | 6.000 milissegundos |
| COMPARAÇÃOS | 26 |
| MOVIMENTOS | 20 |

## QUICK SORT

### EXPLICAÇÃO

O Quicksort é o algoritmo mais eficiente na ordenação por comparação. Nele se escolhe um elemento chamado de pivô, a partir disto é organizada a lista para que todos os números anteriores a ele sejam menores que ele, e todos os números posteriores a ele sejam maiores que ele. Ao final desse processo o número pivô já está em sua posição final. Os dois grupos desordenados recursivamente sofreram o mesmo processo até que a lista esteja ordenada.

### CODIGO EM C

Para a realização do Quick, você precisa criar 3 funções

```
void trocar(int *a, int *b) {
    int temp = *a;
    *a = *b;
```

```
        *b = temp;
    }
```

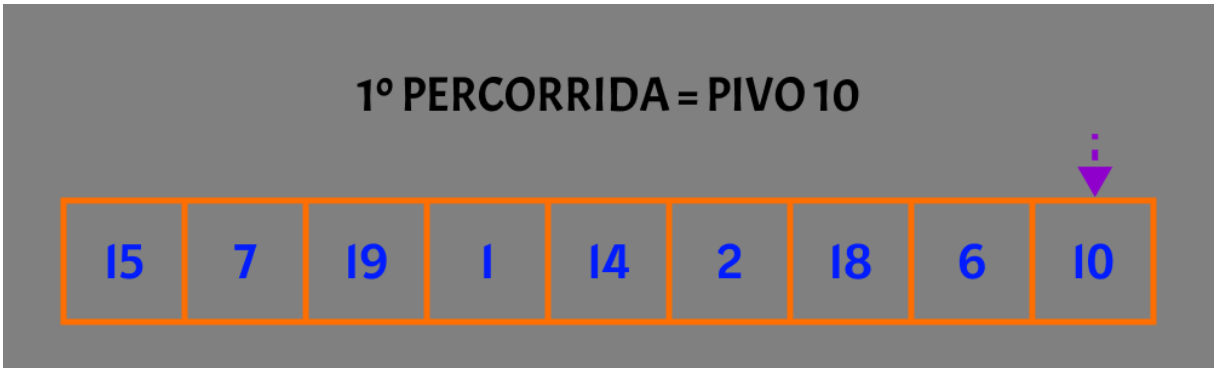
```
int particionar(int lista[], int PONTO_INICIAL_LISTA, int TAMANHO_MAX_LISTA) {
    int pivo = lista[TAMANHO_MAX_LISTA];
    int i = (PONTO_INICIAL_LISTA - 1);

    for (int j = PONTO_INICIAL_LISTA; j < TAMANHO_MAX_LISTA; j++) {
        if (lista[j] < pivo) {
            i++;
            trocar(&lista[i], &lista[j]);
        }
    }
    trocar(&lista[i + 1], &lista[TAMANHO_MAX_LISTA]);
    return (i + 1);
}
```

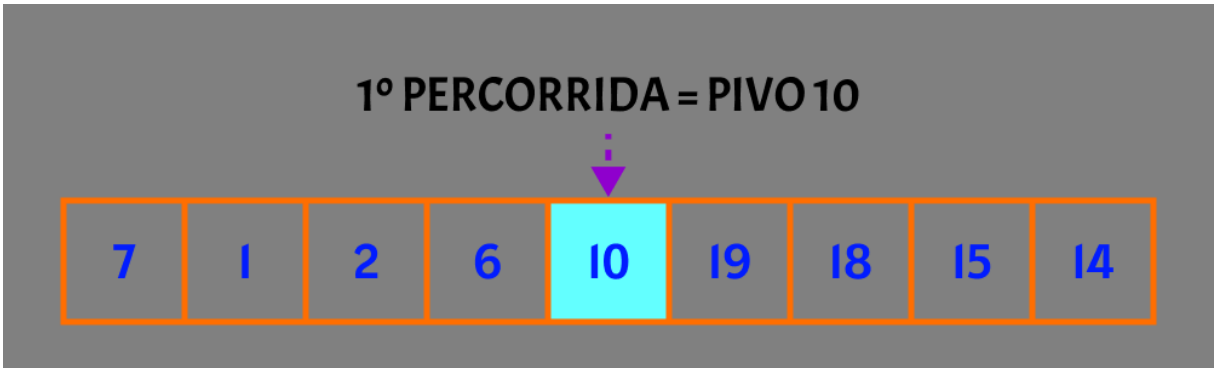
```
void quickSort(int lista[], int PONTO_INICIAL_LISTA, int TAMANHO_MAX_LISTA) {
    if (PONTO_INICIAL_LISTA < TAMANHO_MAX_LISTA) {
        int posicao_pivo = particionar(lista, PONTO_INICIAL_LISTA, TAMANHO_MAX_LISTA);
        quickSort(lista, PONTO_INICIAL_LISTA, posicao_pivo - 1);
        quickSort(lista, posicao_pivo + 1, TAMANHO_MAX_LISTA);
    }
}
```

## TESTE DE PECORRIDAS

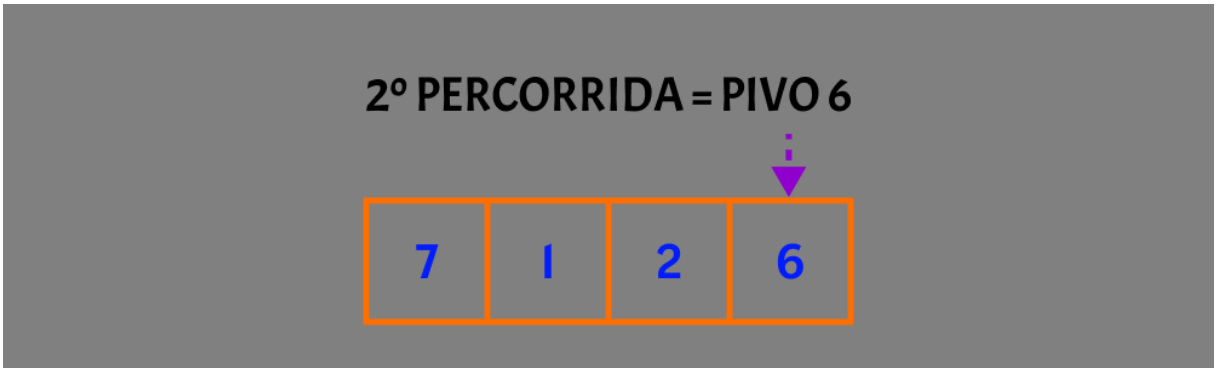
Com a base na lista pré-definida de dados, ele vai percorrer pela sua primeira vez. Neste sistema ele vai escolher o elemento que estar na última posição como o seu pivô:

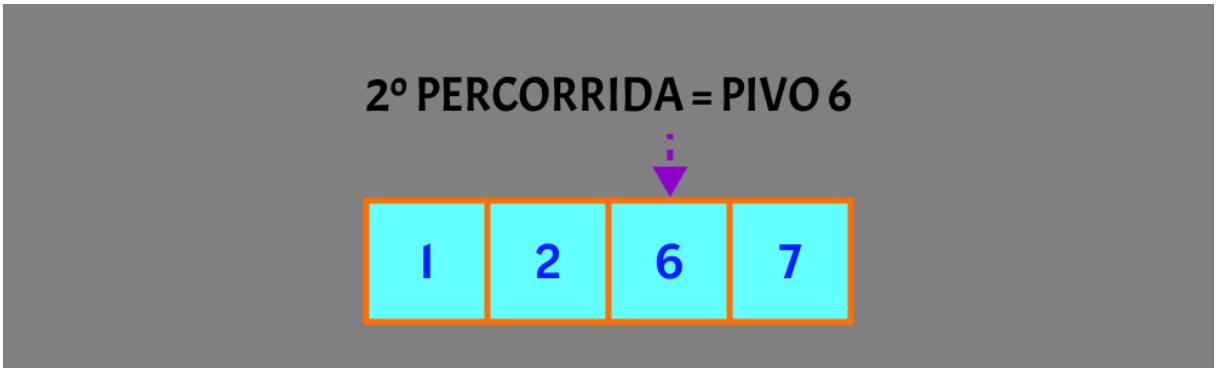



Depois dele escolher o pivo, ele vai separar os números, deixando os maiores ao seu lado direito e os menos ao seu esquerdo.

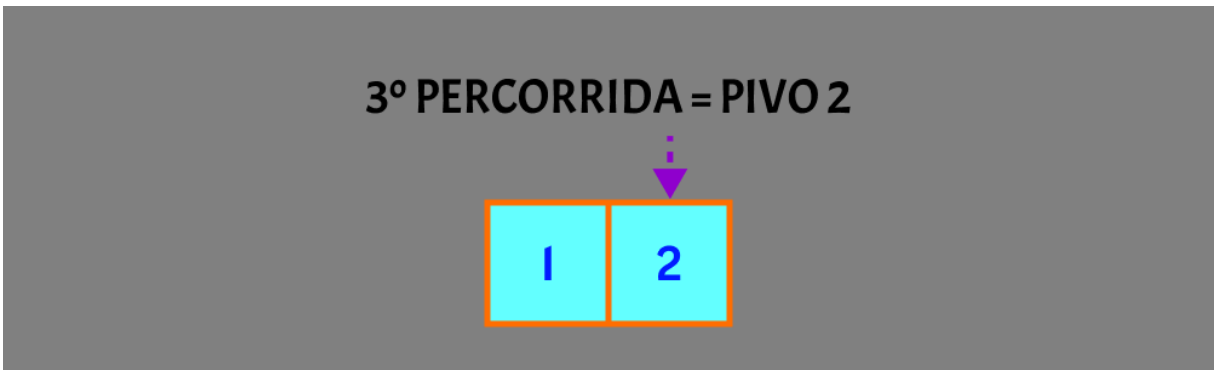



Depois que ele separa ele vai pegar cada parte separadamente para pode repetir [ escolher pivô, separa os números e seguir com a próxima lista]

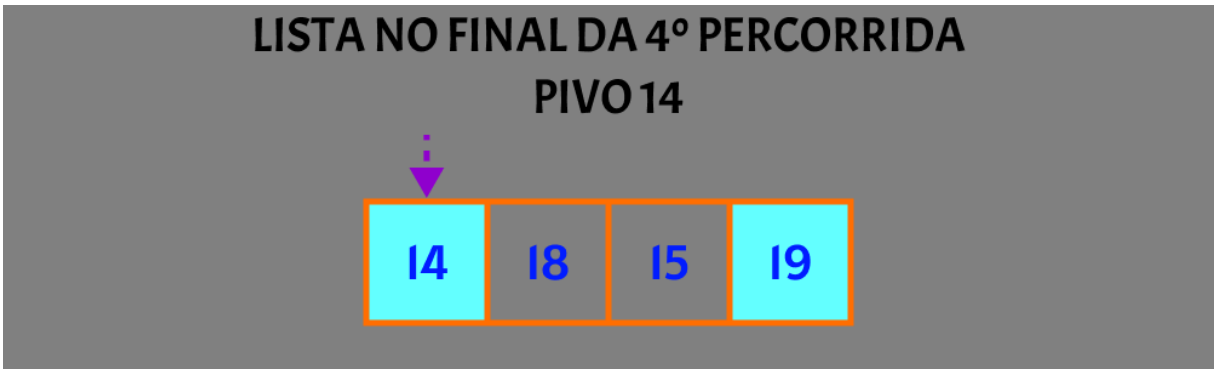


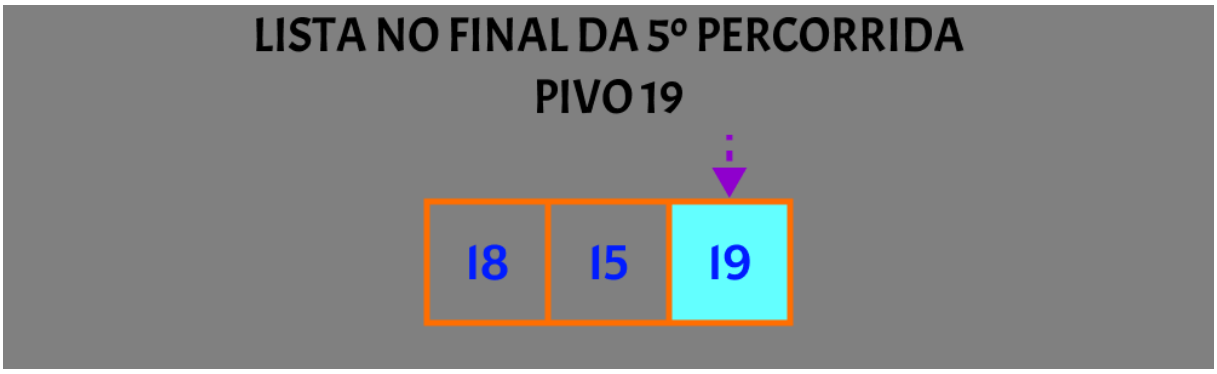


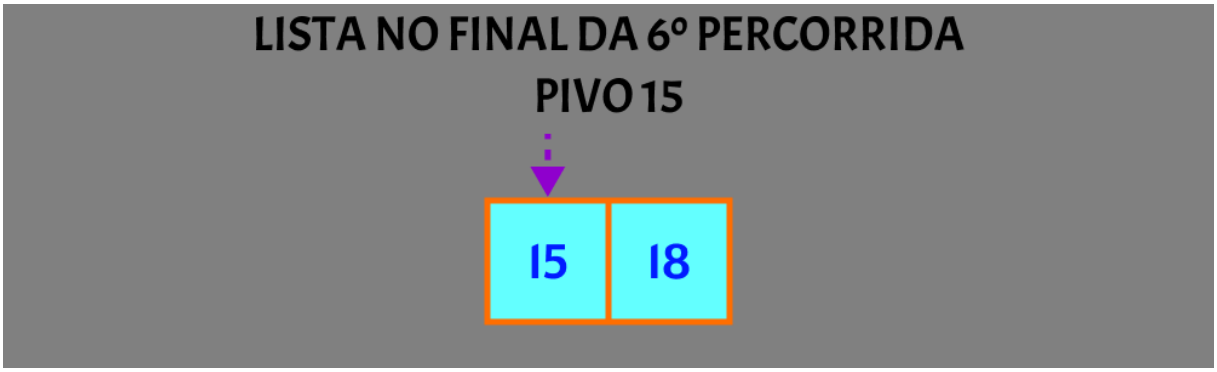

Observação: mesmo que os elementos 1 e 2 já esteja ordenado, ele vai fazer a percorrida para verificar ele:



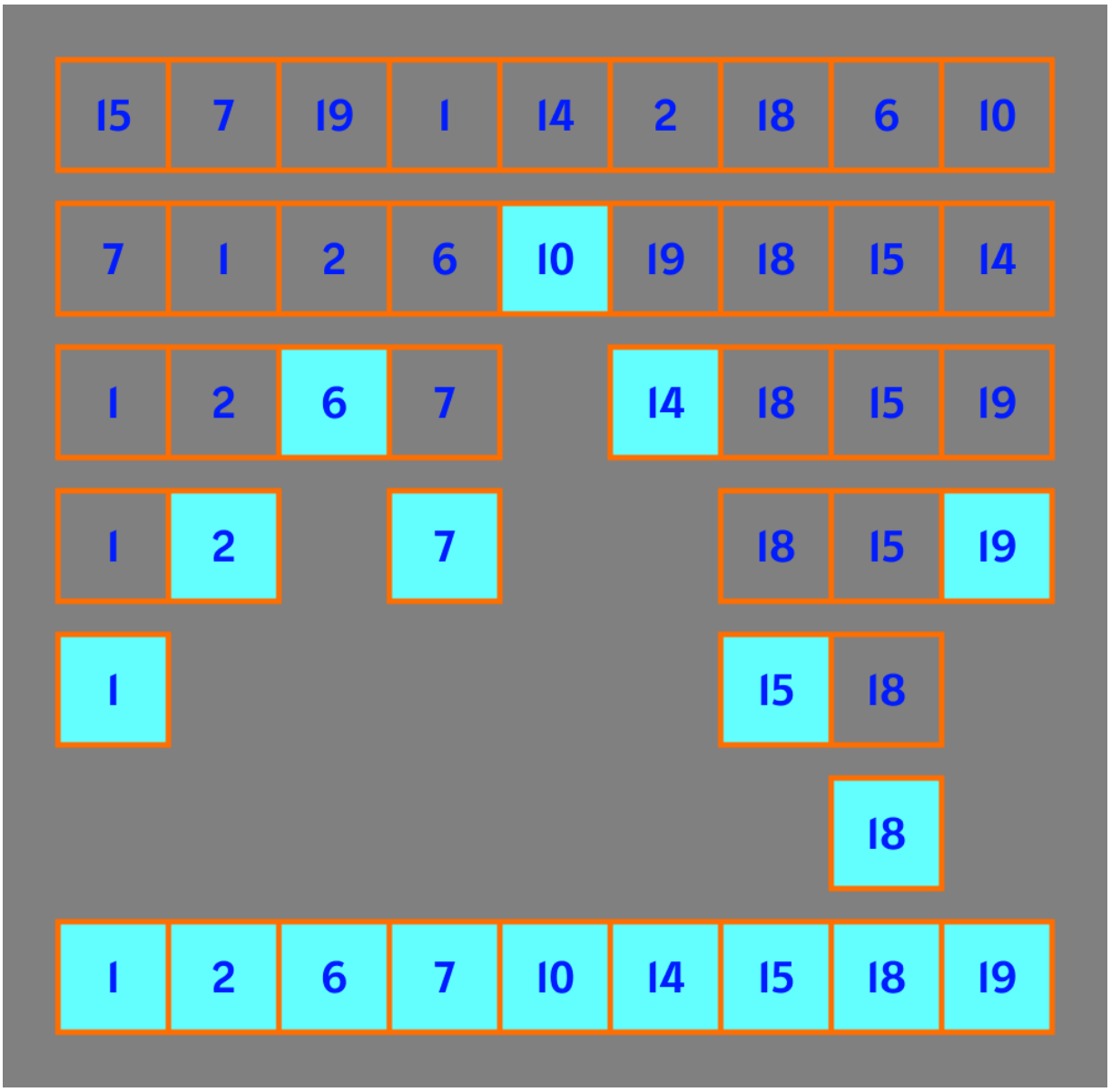

Agora entendido isso, vamos prosseguir






Vendo superficial, as trocas vão ficar assim:



**RESULTADO DO TEXTO**

| QUICK SORT | |
|---|---|
| PECORRIDAS | 6 |
| TEMPO | 3.000 milissegundos |
| COMPARAÇÃOS | 18 |
| MOVIMENTOS | 15 |