QUICKSORT, PARTICIONAMENTO E MODIFICAÇÕES

Universidade Federal de Uberlândia

UFU – FACOM

MARIA ADRIANA VIDIGAL DE LIMA

QUICKSORT – ORDENAÇÃO RÁPIDA

Proposto por Charles A. R. Hoare em 1960 e publicado em 1962.

É o algoritmo de ordenação interna mais rápido que se conhece para uma ampla variedade de situações.

A ordenação baseia-se em dividir para conquistar. Um conjunto com *n* itens deve dividir-se em dois conjuntos menores que devem ser ordenados de forma independente.

A ordenação acontece no próprio vetor e o método não garante estabilidade de ordem para valores iguais.

ESTRATÉGIA PARA A ORDENAÇÃO

- Definição de um pivô.
- Colocam-se os elementos menores que o pivô à esquerda.
- Os elementos maiores que o pivô são acomodados à direita.
- O pivô é colocado na sua posição correta (ordenada).
- Os lados esquerdo e o direito são em seguida ordenados independentemente.

ESTRATÉGIA PARA A ORDENAÇÃO

Definição do PIVÔ:

<u>Primeiro elemento do vetor</u>

Ordenação baseada em:

- Escolher como pivô o primeiro elemento

- <u>Particionar</u> o vetor:
  - Trazer para a esquerda os elementos menores que o pivô e empurrar para a direita os maiores
  - Posicionar o pivô ao final dos menores.

```
int particao(int v[],int esq,int dir){
  int i, pivo;

  pivo = esq;
  for(i=esq+1; i<=dir; i++)
     if(v[i] < v[esq]) {
          pivo = pivo + 1;
          troca(v,pivo,i);
     }
  troca(v,esq,pivo);
  return pivo;
}
```

PASSOS DE ORDENAÇÃO

Definição do PIVÔ:

<u>Primeiro elemento do vetor</u>

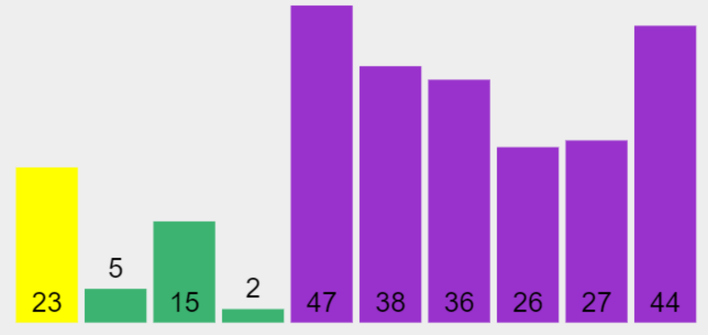
Ordenação baseada em:

- Escolher como pivô o primeiro elemento
- <u>Particionar</u> o vetor:
    - Trazer para a esquerda os elementos menores que o pivô e empurrar para a direita os maiores
    - Posicionar o pivô ao final dos menores
- Ordenar os menores
- Ordenar os maiores

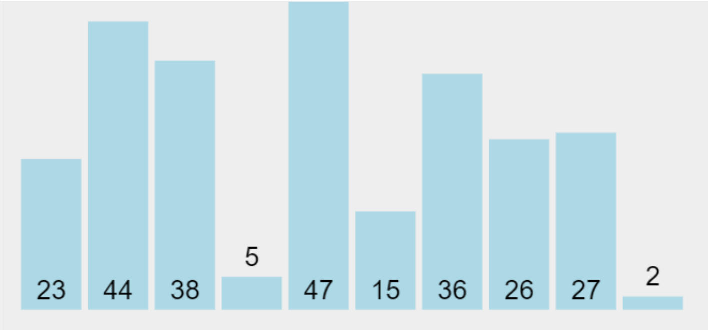
```
int particao(int v[],int esq,int dir){
  int i, pivo;
  pivo = esq;
  for(i=esq+1; i<=dir; i++)
     if(v[i] < v[esq]) {
         pivo = pivo + 1;
         troca(v,pivo,i);
     }
  troca(v,esq,pivo);
  return pivo;
}
void quicksort(int v[], int esq, int dir){
   int i;
   if(esq>=dir) return;
   i = particao(v,esq,dir);
   quicksort(v,esq,i-1);
   quicksort(v,i+1,dir);
}
```

VISUALIZAÇÃO

Vetor inicial:



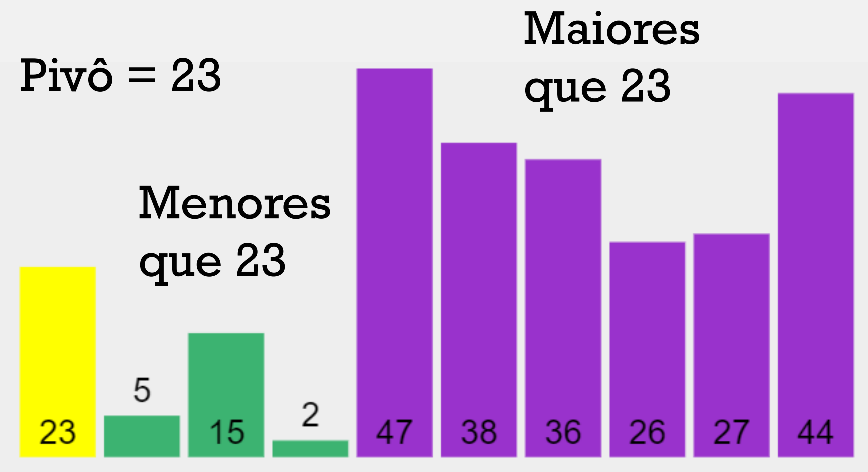
Pivô - Menores - Maiores



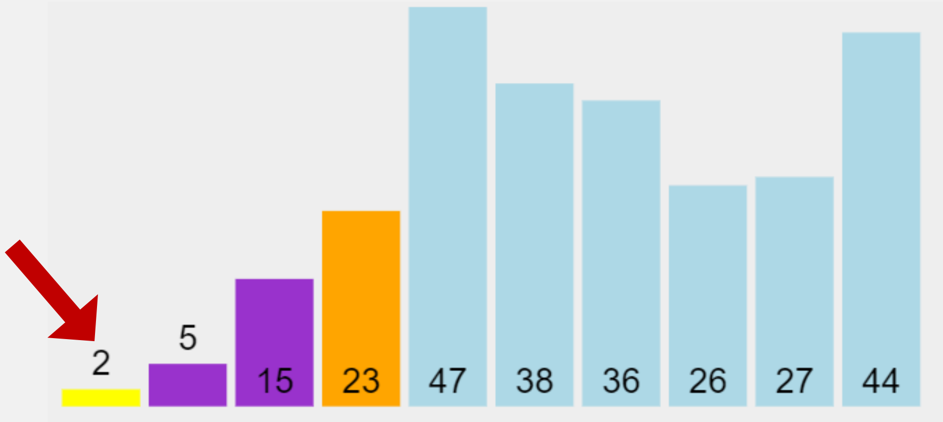

```
int particao(int v[],int esq,int dir){
   int i, pivo;

   pivo = esq;
   for(i=esq+1; i<=dir; i++)
      if(v[i] < v[esq]) {
          pivo = pivo + 1;
          troca(v,pivo,i);
      }
   troca(v,esq,pivo);
   return pivo;
}
void quicksort(int v[], int esq, int dir){
    int i;
    if(esq>=dir) return;
    i = particao(v,esq,dir);
    quicksort(v,esq,i-1);
    quicksort(v,i+1,dir);
}
```

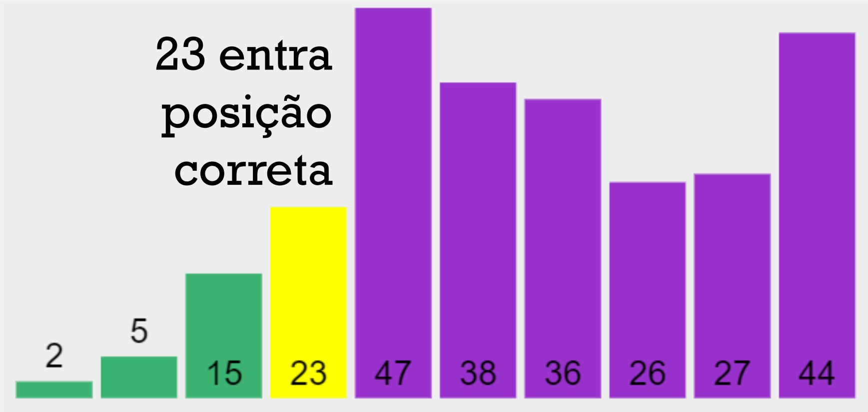
VISUALIZAÇÃO



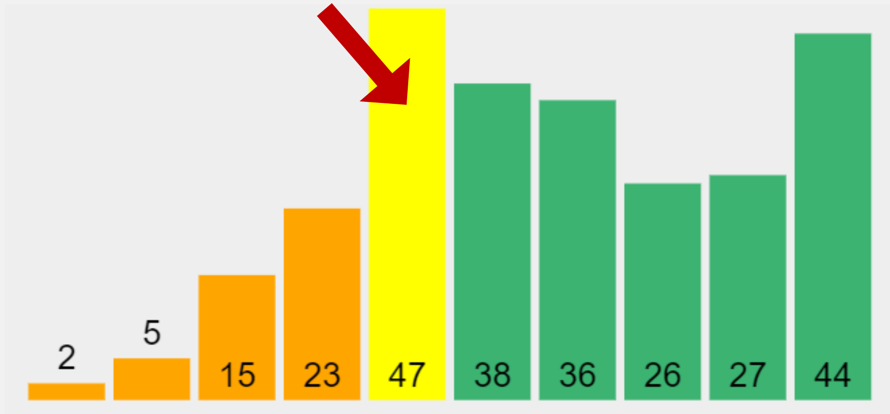
<u>Início da ordenação da esquerda</u>: Pivô = 2



<u>Ajuste do Pivô</u>



<u>Início da ordenação da direita</u>: Pivô = 47



TEMPO DE UMA PARTIÇÃO

A função *Partição* executa em tempo O(n), sendo *n* o tamanho do vetor.

O laço for realiza uma única varredura no vetor completo para a separação dos menores e maiores.

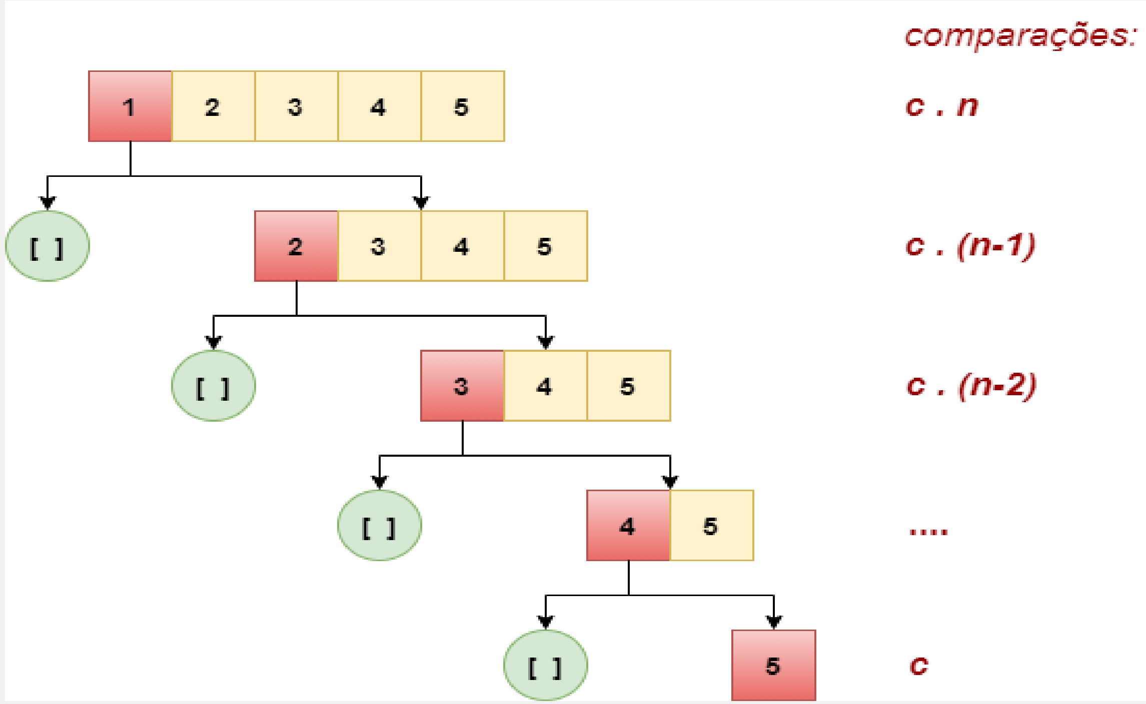
A função troca tem tempo constante O(1).

```
int particao(int v[],int esq,int dir){
  int i, pivo;

  pivo = esq;
  for(i=esq+1; i<=dir; i++)
     if(v[i] < v[esq]) {
         pivo = pivo + 1;
         troca(v,pivo,i);
     }
  troca(v,esq,pivo);
  return pivo;
}
```

PIOR CASO DO QUICKSORT

Uma entrada de dados ordenada representa o pior caso para o algoritmo *quicksort* quando a escolha do pivô se dá pelo primeiro elemento.

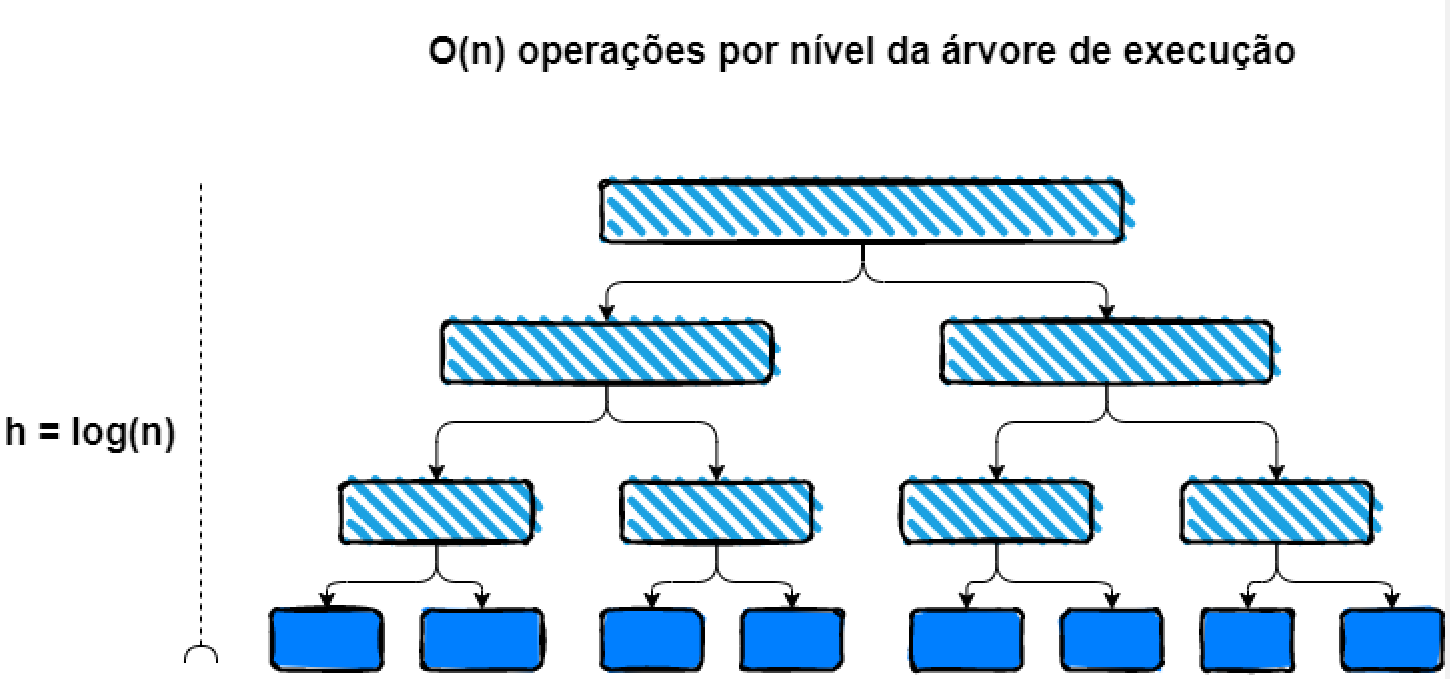


O tempo de execução do *quicksort* é quadrático neste caso:

$$c \cdot n + c \cdot (n-1) + \cdots + c = c \sum_{i=0}^{n-1} (n-i) = c \sum_{j=1}^{n} j = c \frac{n(n+1)}{2} = O(n^2)$$

CASO MÉDIO DO QUICKSORT

O tempo médio (esperado) do *quicksort* é *O(n log n)* quando o pivô particiona o vetor de forma equilibrada.



$$C(n) = 2 * C(n/2) + n = n \log n = O(n \log n)$$

QUICKSORT RECURSIVO: CARACTERÍSTICAS

Vantagens	Eficiente para ordenar arquivos.
	Requer O(n log n) comparações em média (caso médio) para ordenar n itens.
Desvantagens	Tem um pior caso com $O(n^2)$ comparações.
	O método não é estável.

MODIFICAÇÕES NO QUICKSORT

O pior caso pode ser evitado através da realização de pequenas modificações no algoritmo. Algumas opções são:

- Escolha aleatória do pivô
- Escolher três elementos quaisquer e usar a mediana dos três como pivô
- Antes de iniciar a ordenação, aplicar um algoritmo de embaralhamento, como o de Ficher-Yates (O(n)):

```
n = tamanho do vetor
para cada i entre n e 2
  sorteie j como um número entre 1 e i
  se i e j forem diferentes , troque os elementos i
    e j entre si
```

ALEATORIZAÇÃO DO PIVÔ

```
int pivo_aleatorio(int esq, int dir) {
   double r;
   r = (double) rand()/RAND_MAX;   // valor entre 0.01 e 0.99
   return (int)(esq + r*(dir-esq));
}
/* A função rand() gera um número pseudo-aleatório entre 0 e a
constante RAND_MAX. A constante RAND_MAX é 32762. */

void quicksort_aleatorizado(int *v, int esq, int dir) {
   int i;
  if (dir <= esq) return;
  troca(v, pivo_aleatorio(esq,dir), esq);
  i = particao(v, esq, dir);
  quicksort_aleatorizado(v, esq, i-1);
  quicksort_aleatorizado(v, i+1, dir);
}
```

O tempo de execução depende dos pivôs sorteados.

O tempo médio é O(n log n) – podendo ser rápido ou lento, mas não depende de como os elementos estão organizados no vetor.

MEDIANA DE TRÊS

Mediana:
Se o conjunto de informações for numérico e estiver organizado em ordem crescente ou decrescente, a sua **mediana** será o número que ocupa a posição central da lista.

Em lugar de fixar como pivô o elemento da esquerda, pode-se escolher o **elemento médio** de uma amostra de três elementos. Por exemplo: o da esquerda, o do meio, e o da direita.

3 4 9 6 5 1 8

3 6 8

6 será escolhido como pivô

MEDIANA DE TRÊS - IMPLEMENTAÇÃO

1. Recuperar os elementos primeiro, do meio e último
2. Trocar o elemento do meio com o segundo elemento
3. Escolher como pivô a mediana entre os elementos: primeiro, segundo e último
4. O pivô deve ser colocado na segunda posição, o menor na primeira o e maior deles ao final
5. Esta estratégia permite entrar na partição sem considerar os elementos maior e menor que o pivô.

3 4 9 6 5 1 8

3 6 9 4 5 1 8

6 será escolhido como pivô

3 6 9 4 5 1 8

Trecho pronto para iniciar a partição

MEDIANA DE TRÊS - IMPLEMENTAÇÃO

```
void quicksort_mediana_tres(int v[], int esq, int dir) {
  int i;
  if(dir <= esq) return;
  troca(v, (esq+dir)/2, esq+1);
  if(v[esq] > v[esq+1])
      troca(v, esq, esq+1);
  if(v[esq] > v[dir])
      troca(v, esq, dir);
  if(v[esq+1] > v[dir])
      troca(v, esq+1, dir);
  i = particao(v, esq+1, dir-1);
  quicksort_mediana_tres(v, esq, i-1);
  quicksort_mediana_tres(v, i+1, dir);
}
```

3 4 9 6 5 1 8

3 6 9 4 5 1 8

3 6 9 4 5 1 8

```
i = particao(v, 1, 5)
```

MEDIANA DE TRÊS - RESUMO

O particionamento mediana-de-três consiste em selecionar três elementos como pré-pivôs, sejam estes por exemplo, os três da esquerda, os três da direita ou os três do centro, ou como mostrado anteriormente, esquerda-meio-direita.

Seleciona-se então a mediana desses três elementos e então esse elemento será eleito o pivô, e portanto, acomodado no início do vetor (ou trecho) a ser particionado.

Esta estratégia diminui a probabilidade de que o pivô seja o maior ou o menor elemento da sequência.

PIVÔ: ELEMENTO FUNDAMENTAL

- A escolha de um pivô adequado é uma atividade crítica para o bom funcionamento do *quicksort*. Se pudermos garantir que o pivô está próximo à mediana dos valores do vetor, então o quicksort é muito eficiente.

- Uma técnica que pode ser utilizada para aumentar a chance de encontrar bons pivôs é escolher aleatoriamente três elementos do vetor e usar a mediana desses três valores como pivô para a partição.

- Em sequências com muitos elementos repetidos, ainda é grande a chance de não encontrarmos bons pivôs aleatoriamente ou com a ajuda da mediana de três.

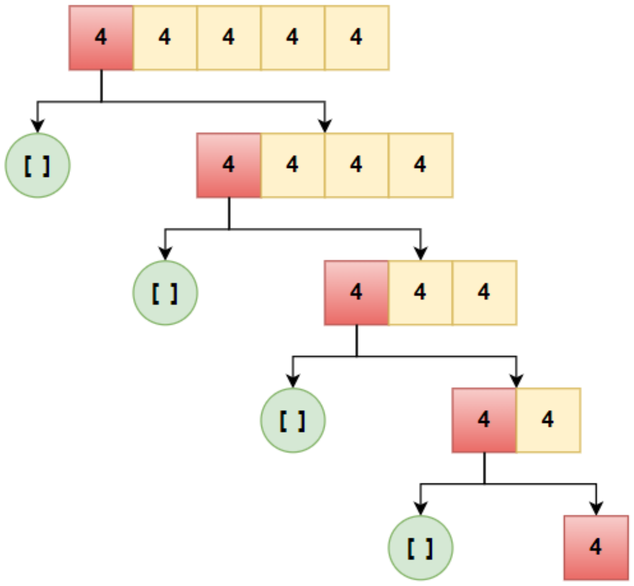
PROBLEMA DAS SEQUÊNCIAS COM MUITAS REPETIÇÕES

Seja o vetor V = [4, 4, 4, 4, 4, 4, ..., 4] contendo *n* elementos com valores iguais.

A função *partição*, mesmo precedida de embaralhamento, escolha prévia do pivô com mediana de três ou aleatória cria as seguintes partições:

- Uma vazia
- Outra com n-1 elementos

Ainda, cada chamada subsequente da função *partição* terá o mesmo comportamento. Este é o pior cenário para a função *quicksort*!



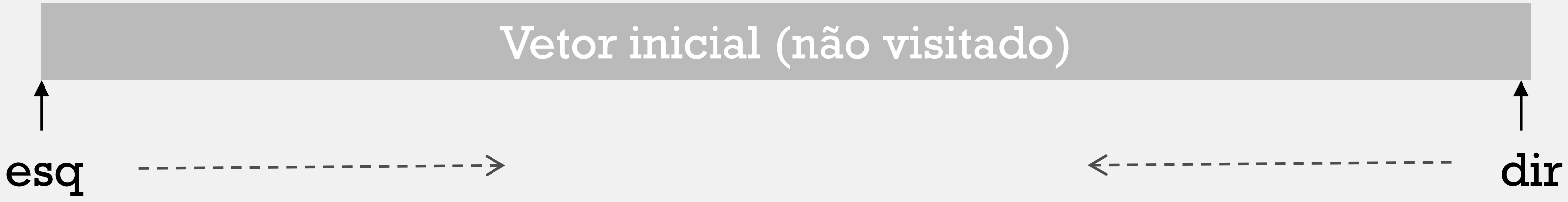
ESTRATÉGIA PARA SE ADEQUAR ÀS SEQUENCIAS COM MUITAS REPETIÇÕES

Jon Bentley e Douglas McIlroy co-produziram (em 1993) uma versão otimizada do *quicksort* para tratar entradas com muitos elementos repetidos utilizando um **particionamento em três vias**:

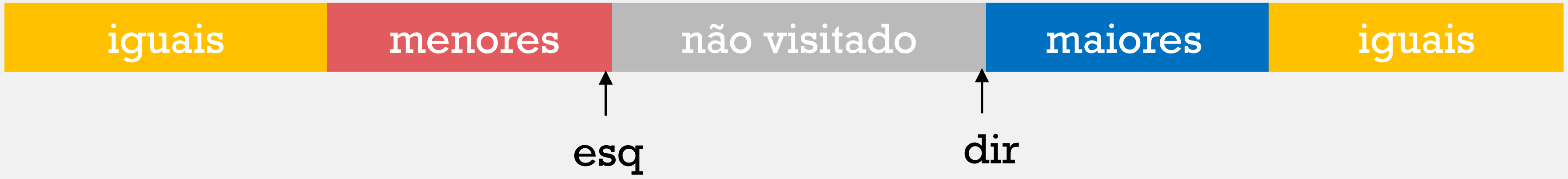
menores	iguais	maiores

O objetivo é que em uma varredura, tenhamos um valor de particionamento (pivô) e que tanto o pivô quanto os elementos iguais a ele sejam corretamente posicionados (ao meio) enquanto os menores ficam à esquerda e os maiores à direita.

PROCESSO DE PARTICIONAMENTO EM 3 VIAS



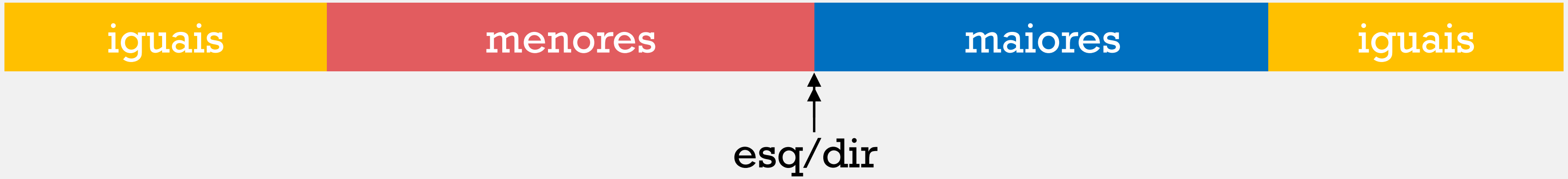
A construção do particionamento se inicia com dois ponteiros: um em cada extremidade, e o caminhamento é feito em direção ao meio buscando separar os menores, maiores e iguais ao elemento pivô.



- Nos dois extremos estão as regiões que mantêm os elementos iguais ao pivô.
- A região ainda não visitada fica no meio e vai diminuindo a cada iteração.
- À esquerda dos não visitados estão os menores que o pivô.
- Ao lado direito dos não visitados está a região que mantém os elementos maiores que o pivô.

PROCESSO DE PARTICIONAMENTO EM 3 VIAS

Ao final da primeira visita completa, tem-se:



Em seguida, todos os elementos iguais, acomodados nas extremidades, são movidos para o centro (apontado por esq/dir):

Ao encerrar a visita e organizar as três vias (ou partições), a mesma abordagem será utilizada para particionar as regiões dos menores e maiores.

IMPLEMENTAÇÃO DA VISITA E CONSTRUÇÃO DAS TRÊS VIAS

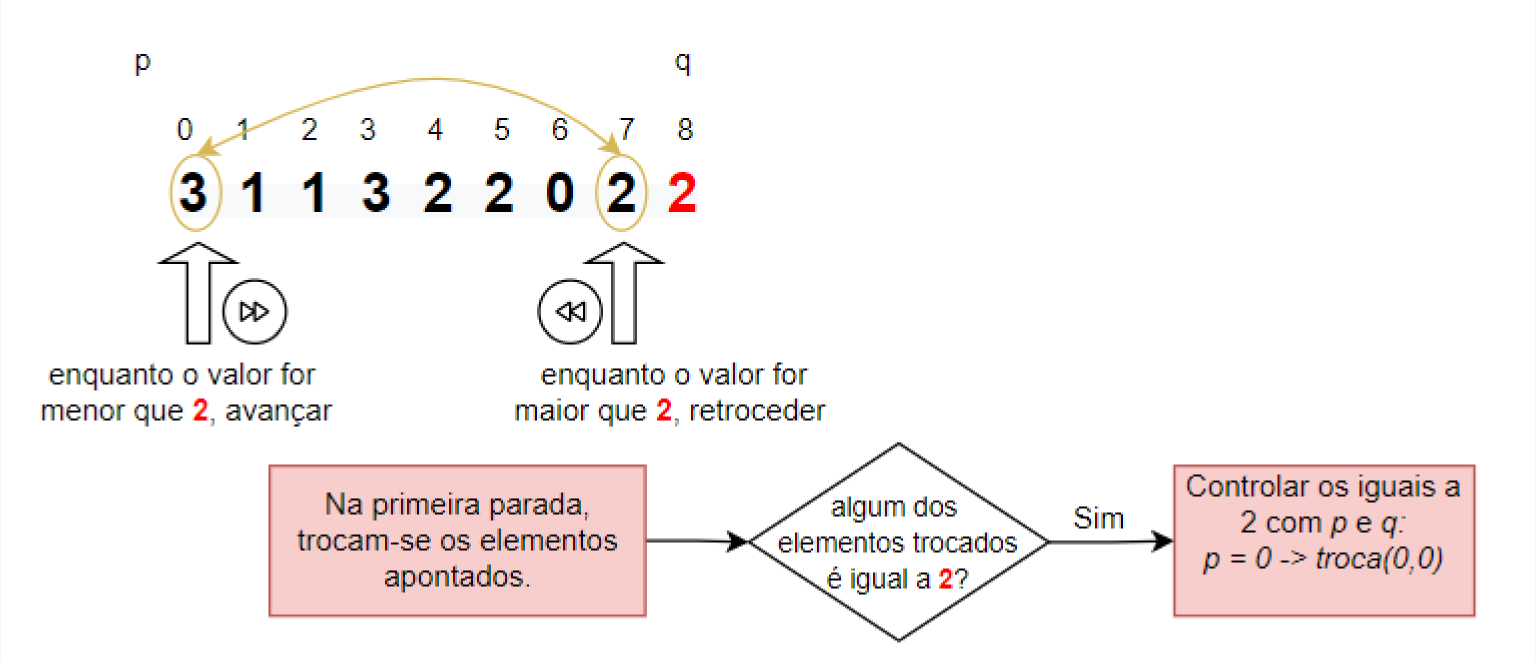
- Definir pivô: o pivô será o elemento mais à direita.
- Apontar para a esquerda e caminhar até achar um elemento que não seja menor que o pivô
- Apontar para a direita-1 (não considerar o pivô) e caminhar até achar um elemento que não seja maior que o pivô.
- Parar se os ponteiros se cruzarem.
- Trocar os elementos nas posições de parada.
- Se os elementos trocados forem iguais ao pivô, enviá-los para as extremidades correspondentes.

EXEMPLO:

Sequência Inicial: 3 1 1 3 2 2 0 2 2

Pivô : 2 (elemento mais à direita)

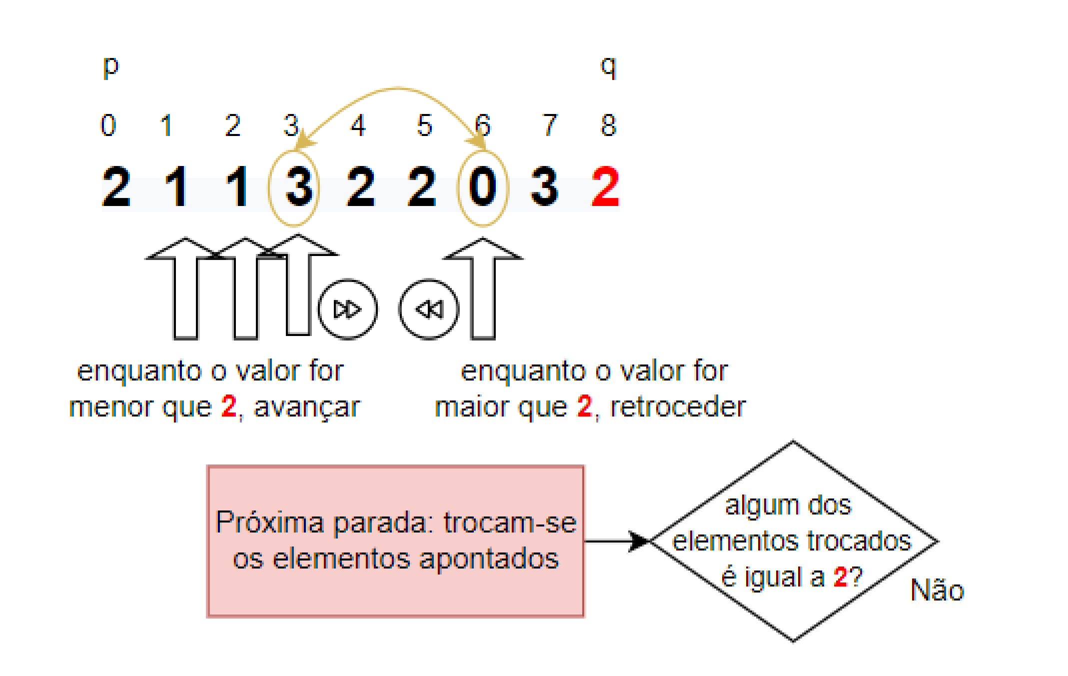
Início do percurso: primeira iteração



Resultado: **2 1 1 3 2 2 0 3 2**

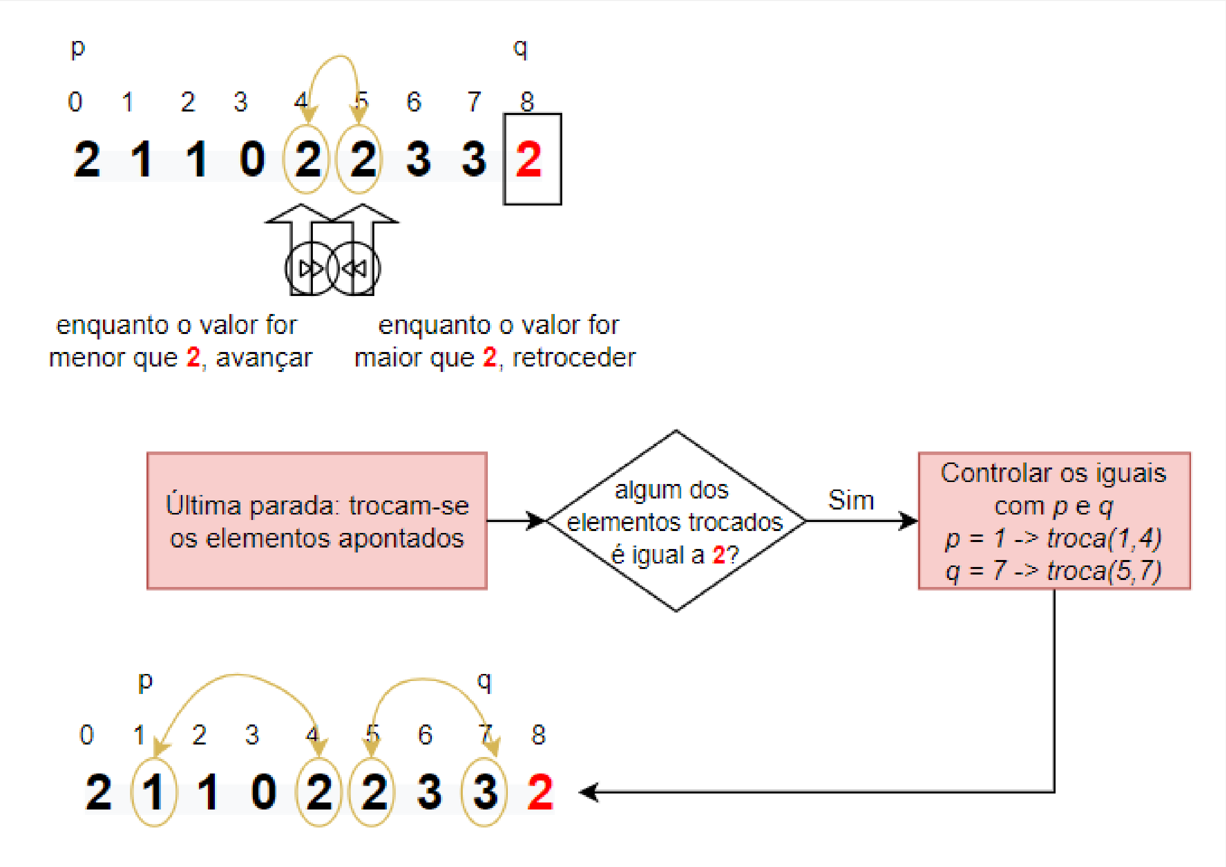
EXEMPLO:

Segunda iteração:



Resultado: 2 1 1 0 2 2 3 3 2

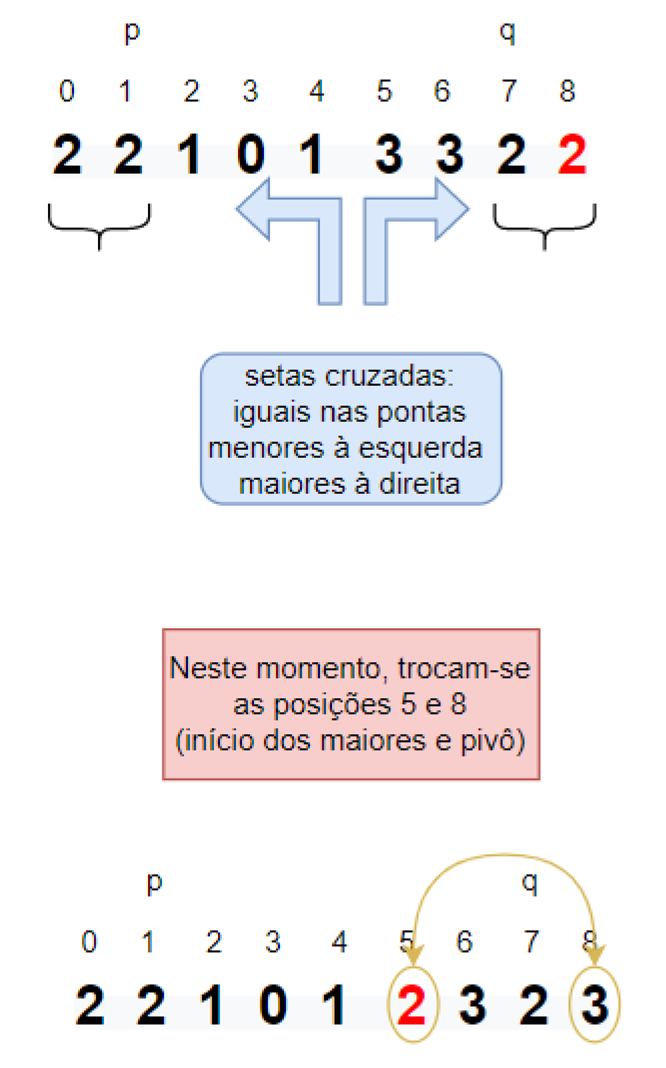
Terceira iteração:



Resultado: 2 2 1 0 1 3 3 2 2

EXEMPLO:

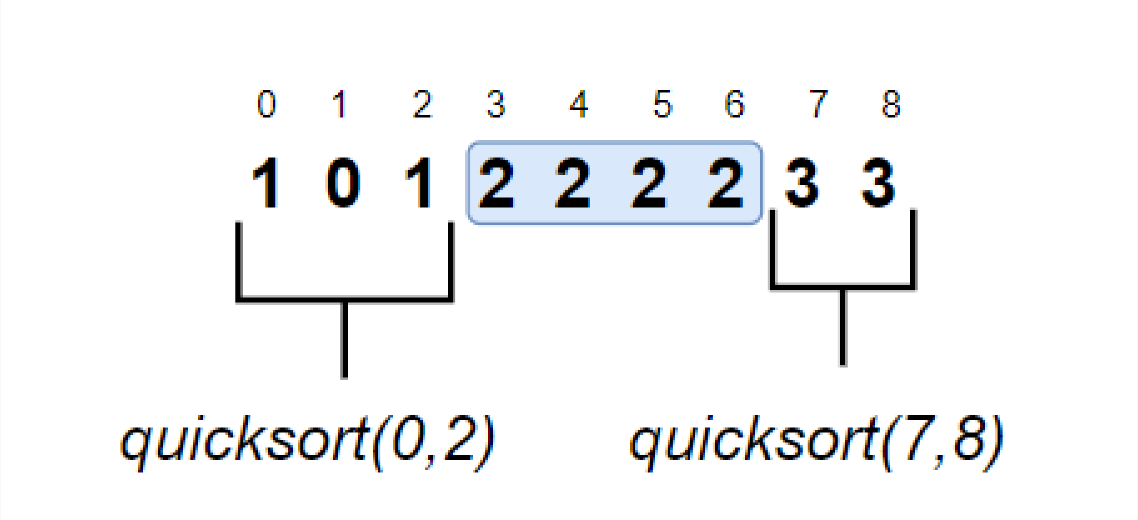
Final do percurso e movimentação do pivô



Movimentação de todos os iguais ao pivô para o centro:

EXEMPLO:

Pivô e seus iguais já acomodados nas posições corretas para a primeira varredura (pivô = 2).
Chamadas recursivas na sequência para partições esquerda e direita.



IMPLEMENTAÇÃO:

Linha 5: Definição do pivô

Linhas 7 a 16: Varredura do vetor usando ponteiros para iniciar da esquerda (i) e da direita (j).

Linhas 14 e 15: Verifica se o elemento trocado é igual ao pivô. Se sim, levar para um dos extremos.

Linhas 20 e 21: Colocam os iguais ao pivô no meio.

```
3   void quicksort_tres_partes(int a[], int l, int r){
4       int k, i = l-1, j = r, p = l-1, q = r;
5       int v = a[r];
6       if (r <= l) return;
7       for (;;)
8       {
9           while (a[++i] < v) ;
10          while (v < a[--j])
11              if (j == l) break;
12          if (i >= j) break;
13          troca(a,i,j);
14          if (a[i] == v) { p++; troca(a,p,i); }
15          if (v == a[j]) { q--; troca(a,j,q); }
16      }
17      troca(a,i,r);
18      j = i-1;
19      i = i+1;
20      for (k = l; k <= p; k++, j--) troca(a,k,j);
21      for (k = r-1; k >= q; k--, i++) troca(a,i,k);
22      quicksort_tres_partes(a, l, j);
23      quicksort_tres_partes(a, i, r);
24  }
```

COMPARAÇÃO ENTRE ALGORITMOS: TESTES EXPERIMENTAIS

Implementar os algoritmos
- *Quicksort* básico
- *Quicksort* aleatorizado
- *Quicksort* com mediana de três
- *Quicksort* com partição em três vias

Criar quatro vetores:
- Sequência aleatória
- Sequência ordenada
- Sequência invertida
- Sequência com muitas repetições

Tamanho do vetor: 5000

	Aleatorio	Ordenado	Invertido	Repetidos
Básico	0,00094	0,04559	0,09331	0,00169
Aleatorizado	0,00088	0,00107	0,00088	0,00167
Mediana de 3	0,00053	0,00051	0,00052	0,00080
Partição em 3 vias	0,02063	0,02815	0,02658	0,00039

Tamanho do vetor: 50000

	Aleatorio	Ordenado	Invertido	Repetidos
Básico	0,01130	3,92205	7,97799	0,09105
Aleatorizado	0,01034	0,01066	0,01109	0,10022
Mediana de 3	0,00703	0,00640	0,00703	0,04711
Partição em 3 vias	0,51811	2,51951	2,48866	0,00364

TESTES EXPERIMENTAIS

Tamanho do vetor: 5000

	Aleatorio	Ordenado	Invertido	Repetidos
Básico	0,00094	0,04559	0,09331	0,00169
Aleatorizado	0,00088	0,00107	0,00088	0,00167
Mediana de 3	0,00053	0,00051	0,00052	0,00080
Partição em 3 vias	0,02063	0,02815	0,02658	0,00039

Tamanho do vetor: 50000

	Aleatorio	Ordenado	Invertido	Repetidos
Básico	0,01130	3,92205	7,97799	0,09105
Aleatorizado	0,01034	0,01066	0,01109	0,10022
Mediana de 3	0,00703	0,00640	0,00703	0,04711
Partição em 3 vias	0,51811	2,51951	2,48866	0,00364