Busca e Ordenação

Algoritmos de Ordenação Linearítmicos

Prof. Edson Alves - UnB/FGA

2020

Sumário

MergeSort

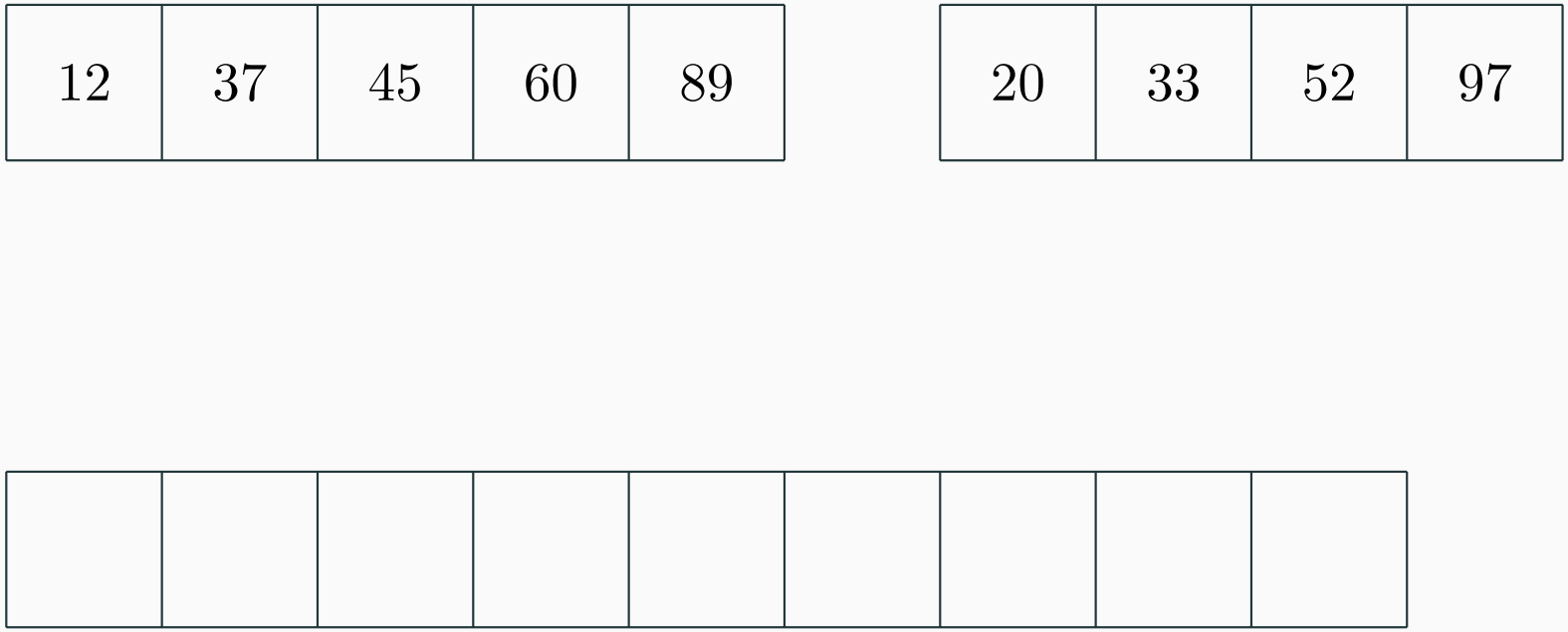
MergeSort

- O *MergeSort* é um algoritmo de ordenação antigo, já conhecido por John von Neumann em 1945
- Ele usa o paradigma dividir-e-conquistar para ordenar os elementos de um vetor
- Ele divide o vetor em duas metades, ordena cada uma delas e, em seguida, funde ambas partes em um todo ordenado
- O algoritmo é replicado em cada uma das metades, até que o tamanho de cada parte seja trivialmente ordenável
- A complexidade é linearítmica, isto é, $O(N \log N)$, onde N é o número de elementos no vetor

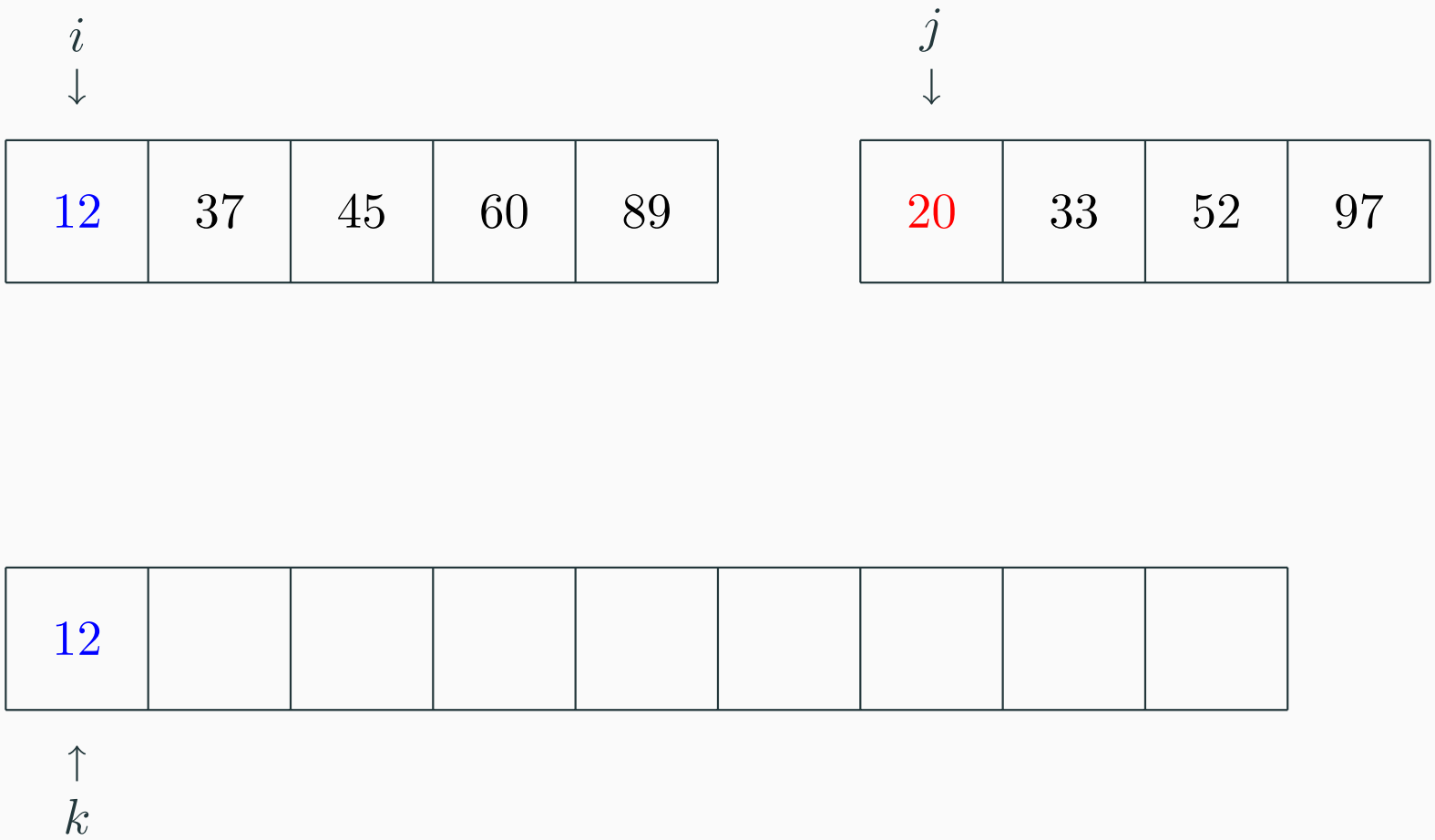
Fusão de dois vetores ordenados

- A divisão do vetor em partes cada vez menores corresponde à etapa **dividir** do algoritmo
- Se $N = 1$, o vetor já está trivialmente ordenado
- A fusão de duas partes ordenadas consiste na etapa **conquistar** do paradigma
- Esta não é uma etapa trivial, e é linear em relação à soma do número de elementos de cada parte
- O procedimento consiste em inicializar um ponteiro para o primeiro elemento de cada parte e, sucessivamente, escolher o menor dentre os elementos disponíveis
- Este procedimento não pode ser feito *in-place*, gerando um custo de memória $O(N)$ adicional ao algoritmo

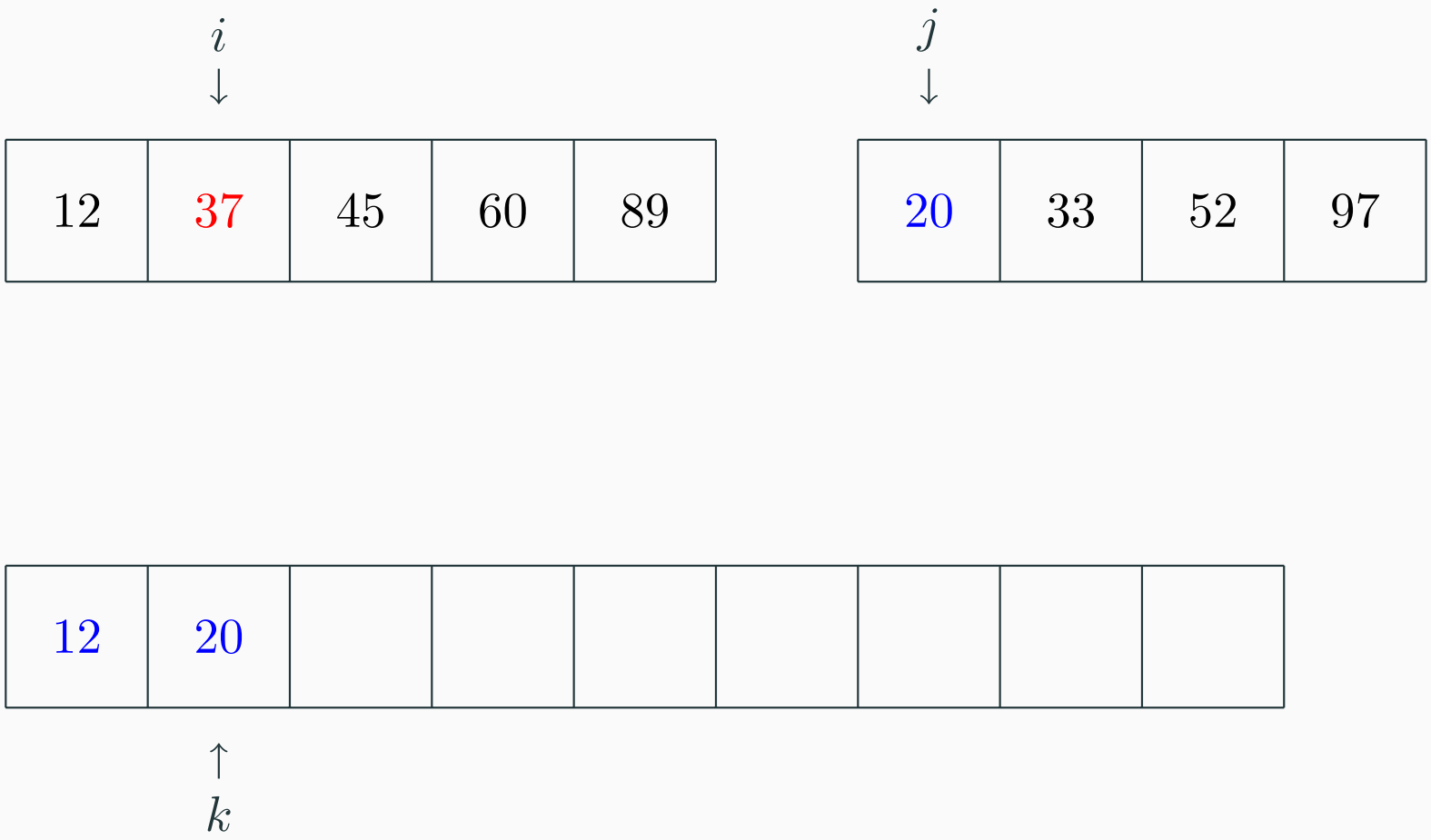
Visualização da rotina de fusão



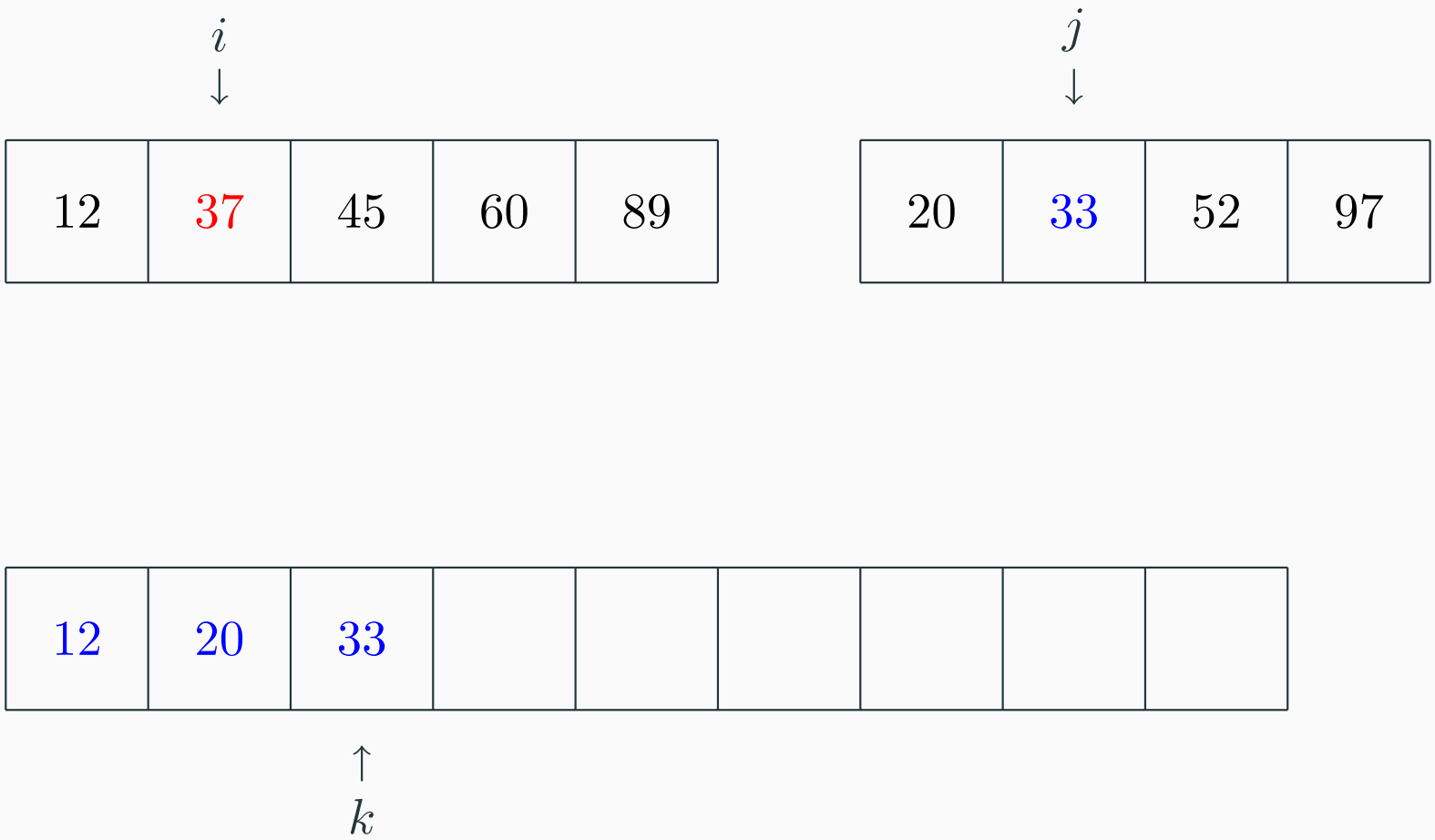
Visualização da rotina de fusão



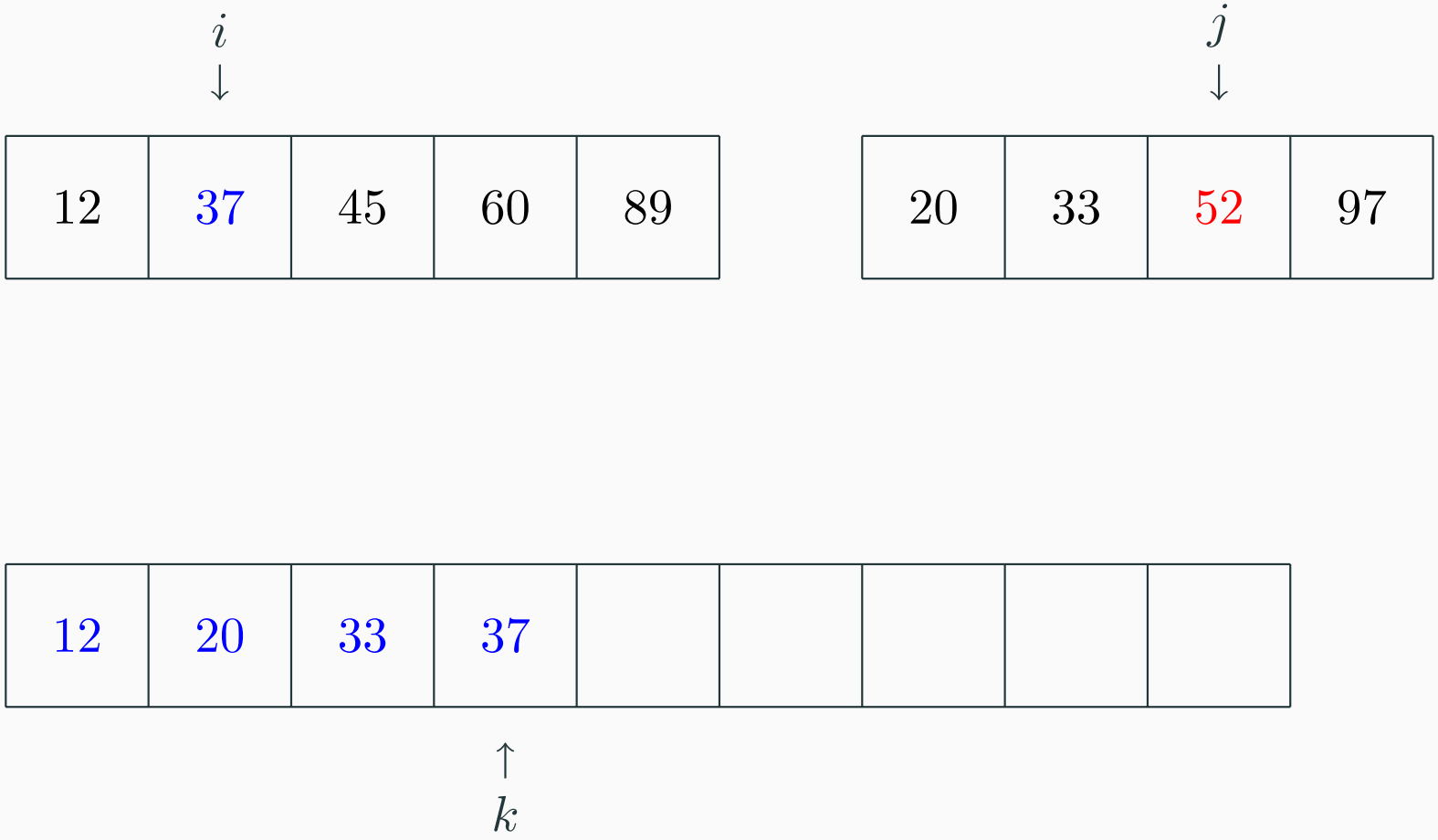
Visualização da rotina de fusão



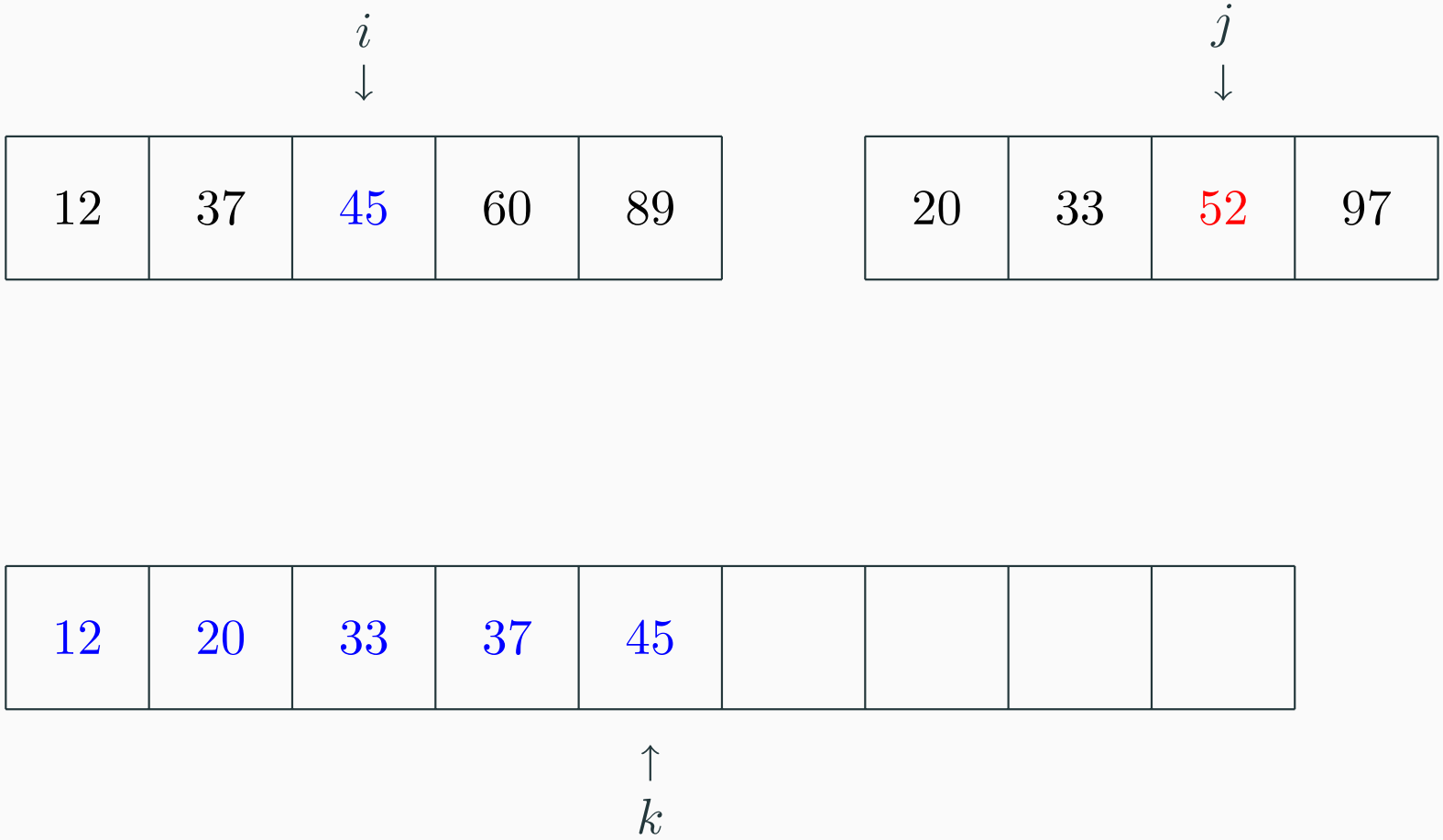
Visualização da rotina de fusão



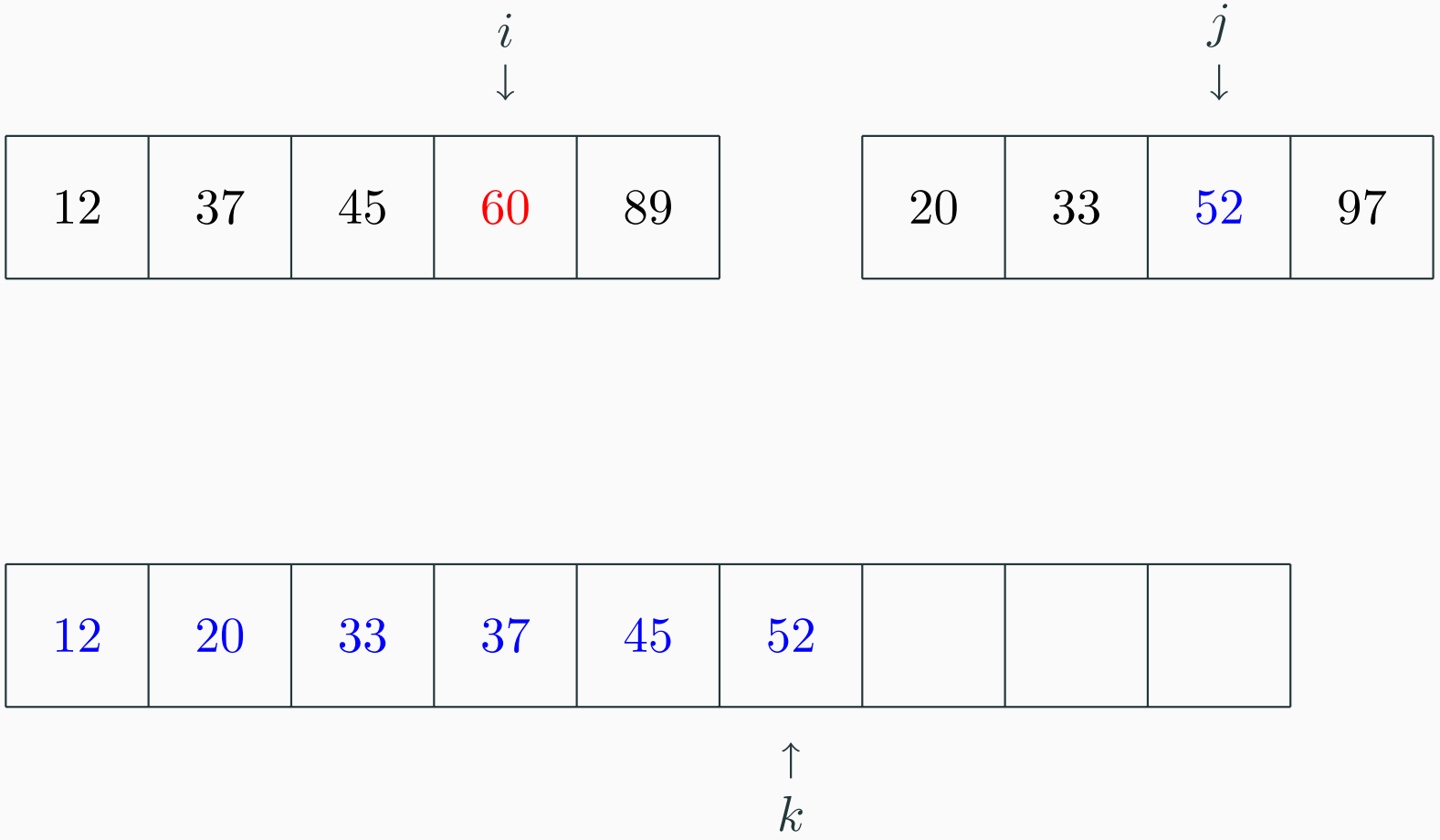
Visualização da rotina de fusão



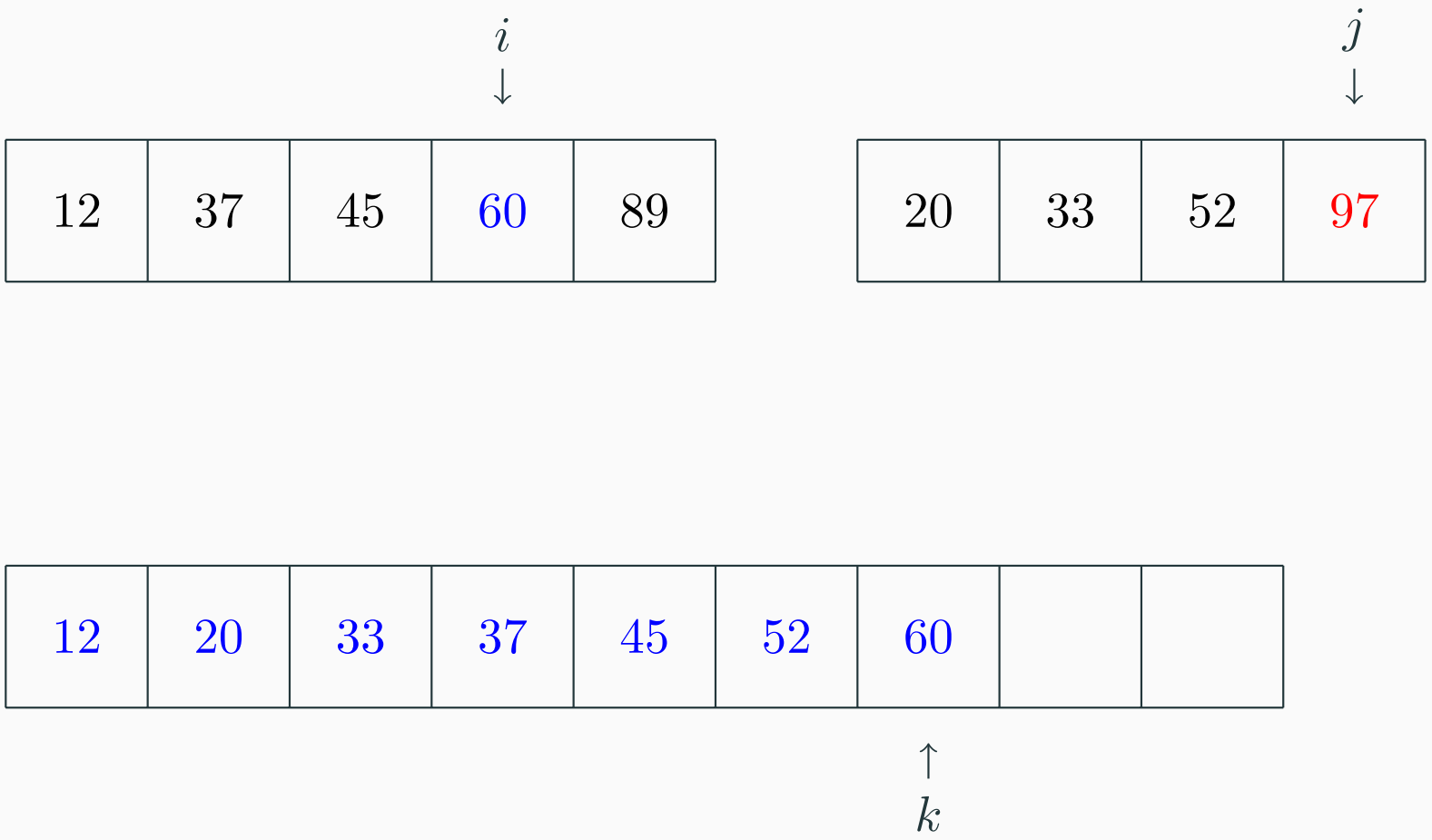
Visualização da rotina de fusão



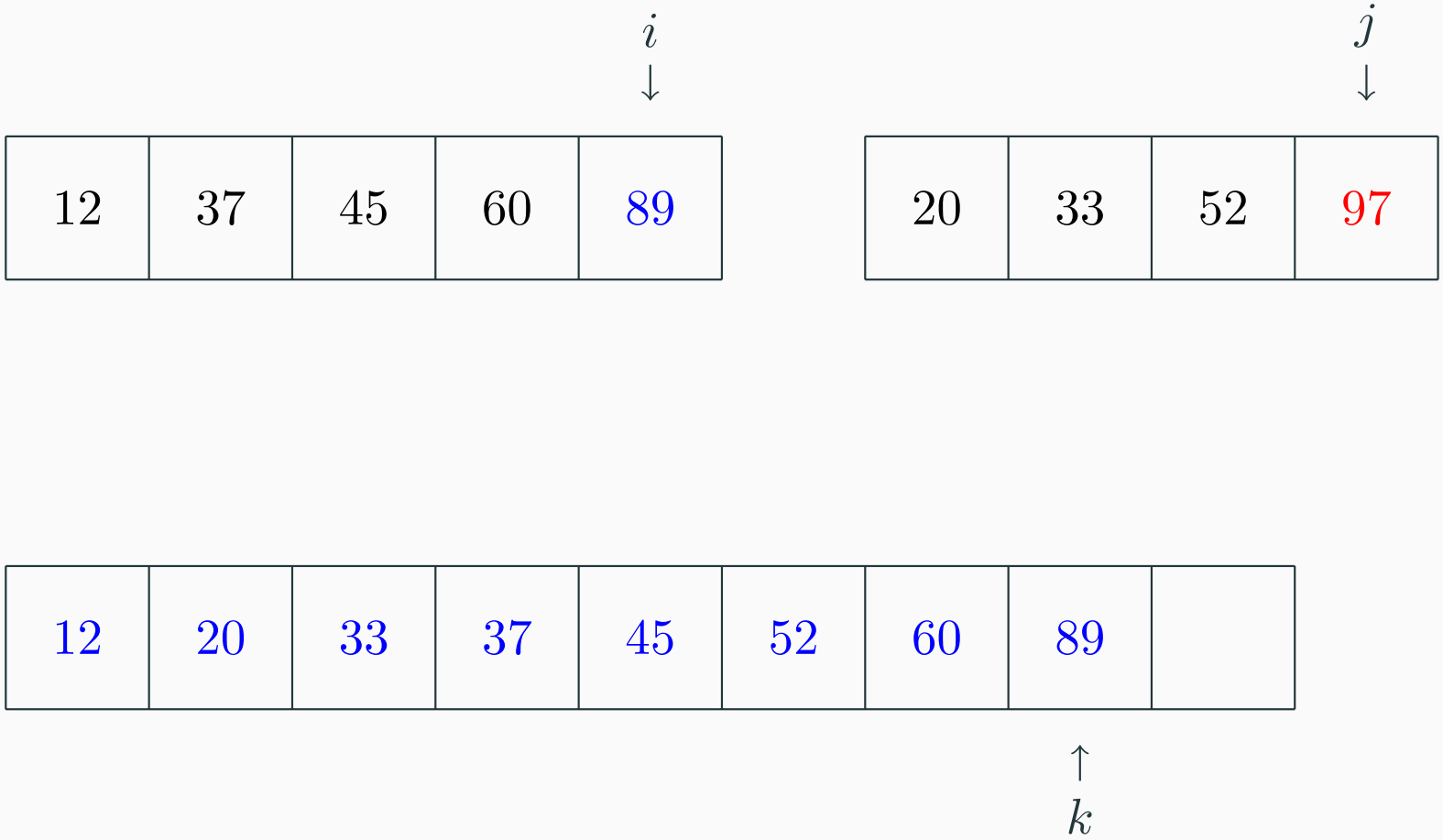
Visualização da rotina de fusão



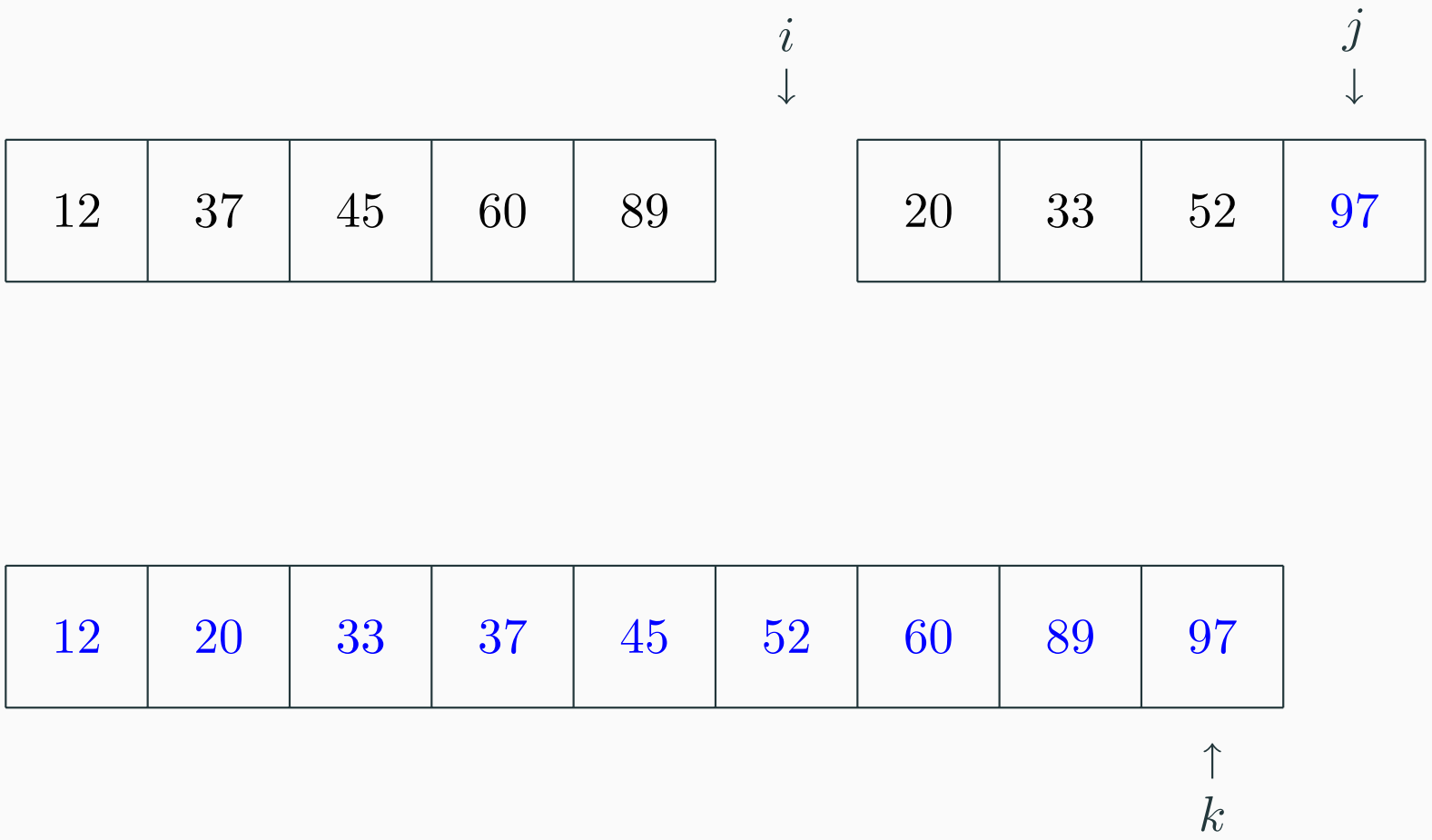
Visualização da rotina de fusão



Visualização da rotina de fusão



Visualização da rotina de fusão



Implementação da rotina de fusão

```
5  template<typename RandIt>
6  void merge(size_t N, RandIt first, RandIt middle, RandIt last)
7  {
8      vector<typename iterator_traits<RandIt>::value_type> temp(N);
9      auto it = first, jt = middle;
10     auto k = 0;
11
12     while (it != middle and jt != last) {
13         temp[k++] = min(*it, *jt);
14         temp[k - 1] == *it ? ++it : ++jt;
15     }
16
17     while (it != middle)
18         temp[k++] = *it++;
19
20     while (jt != last)
21         temp[k++] = *jt++;
22
23     for (const auto& elem : temp)
24         *first++ = elem;
25 }
```

Visualização do `mergesort`

89	60	12	45	37	52	33	97	20

Visualização do `mergesort`

89	60	12	45	37	52	33	97	20

89	60	12	45	37

Visualização do mergesort

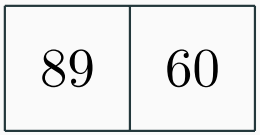
89	60	12	45	37	52	33	97	20

89	60	12	45	37

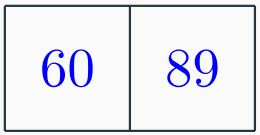
89	60	12

Visualização do mergesort

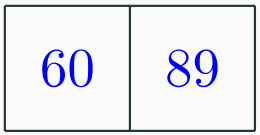
Visualização do mergesort



Visualização do mergesort

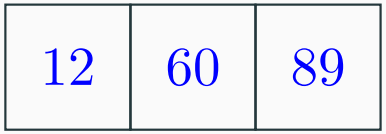


Visualização do mergesort

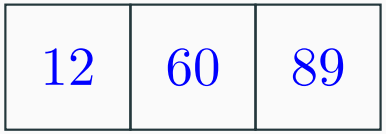


Visualização do mergesort

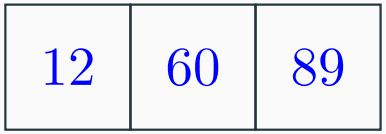
Visualização do mergesort

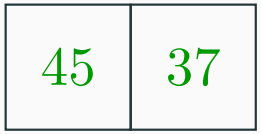


Visualização do mergesort

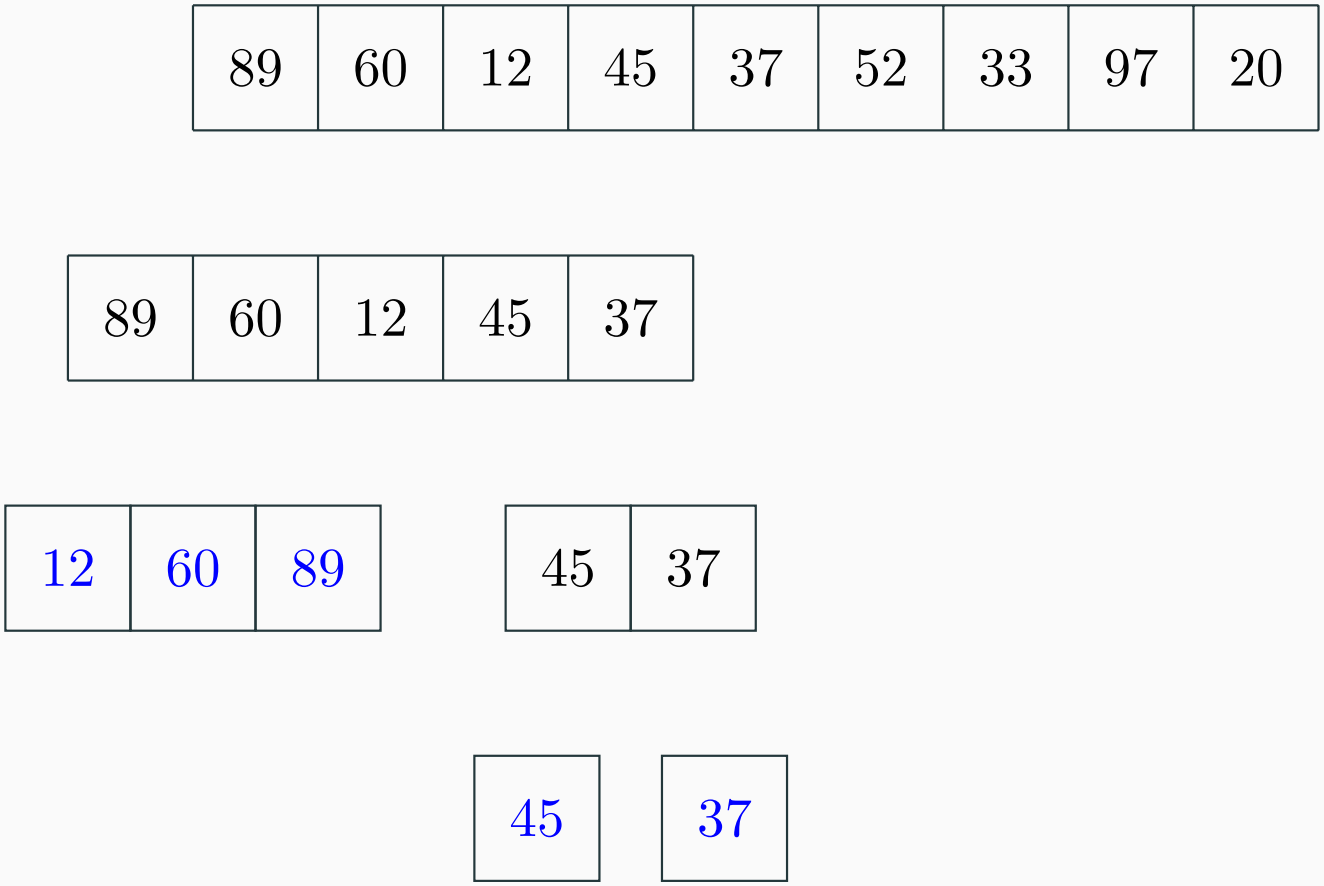


Visualização do mergesort

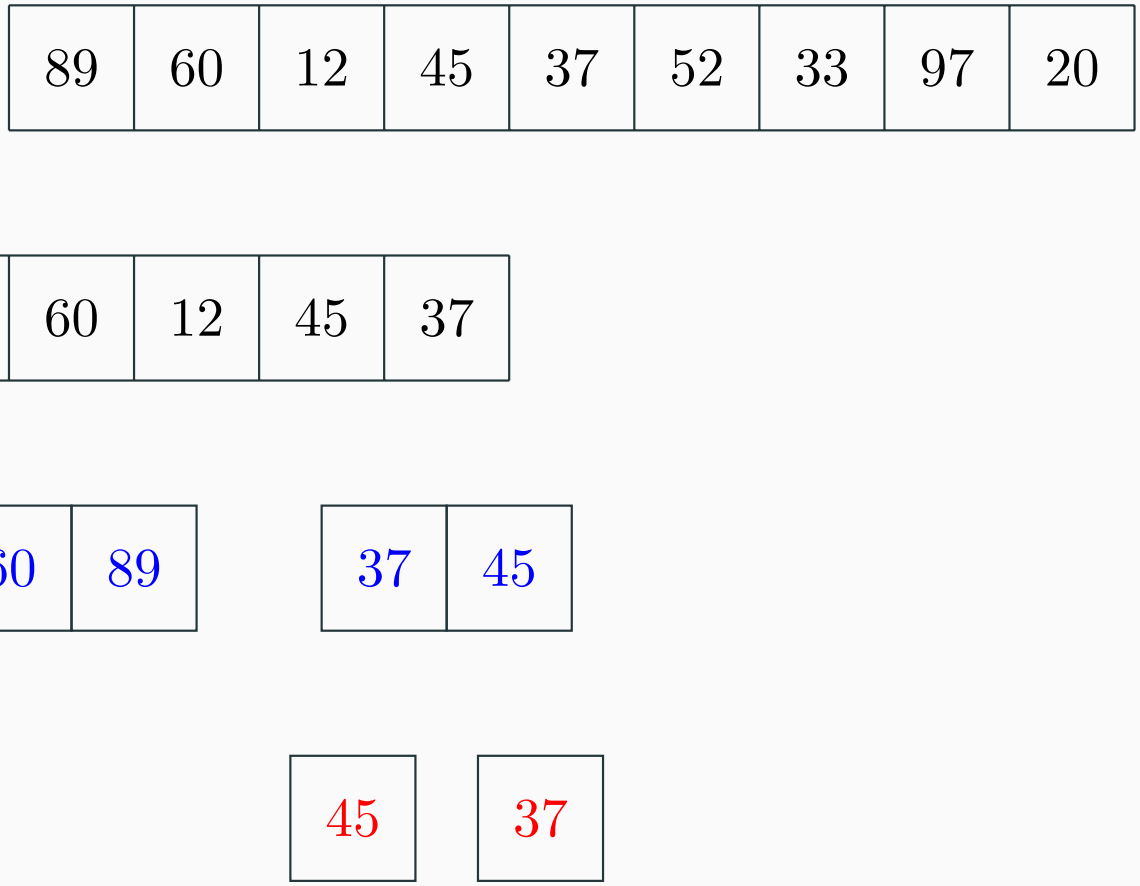




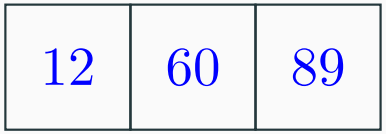
Visualização do mergesort

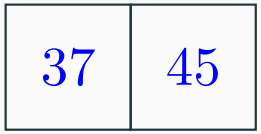


Visualização do mergesort

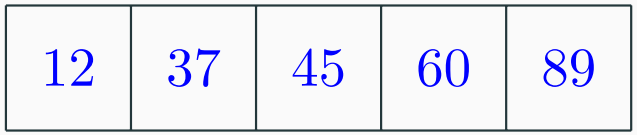


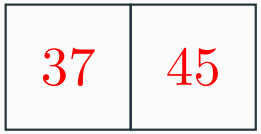
Visualização do mergesort





Visualização do mergesort



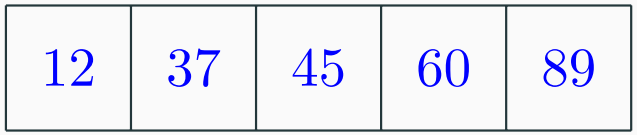


Visualização do mergesort

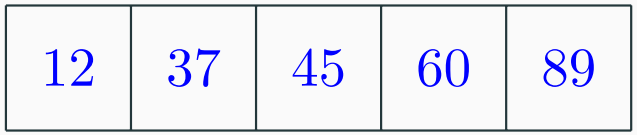
89	60	12	45	37	52	33	97	20

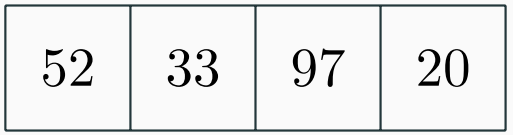
12	37	45	60	89

Visualização do mergesort

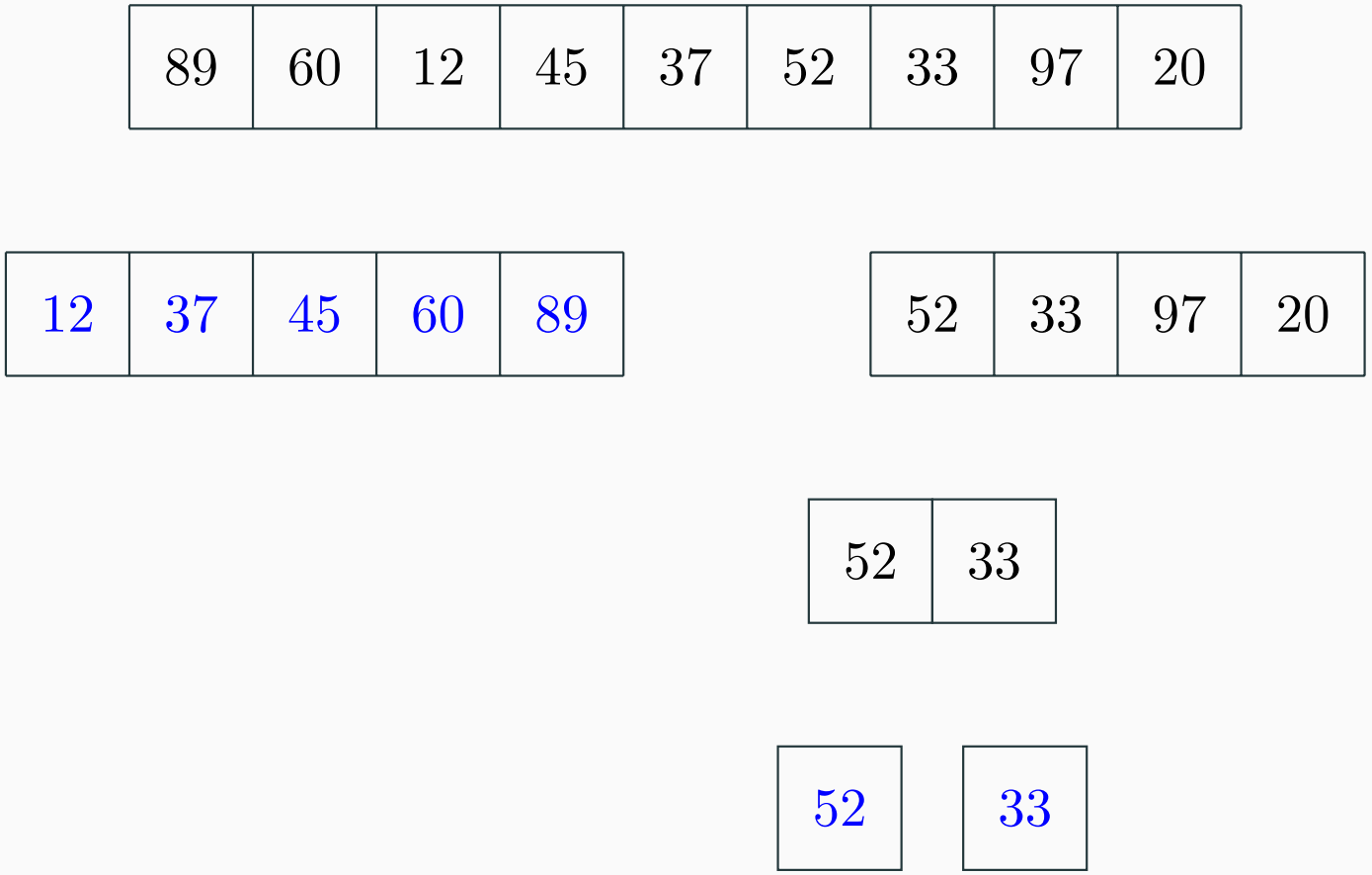


Visualização do mergesort

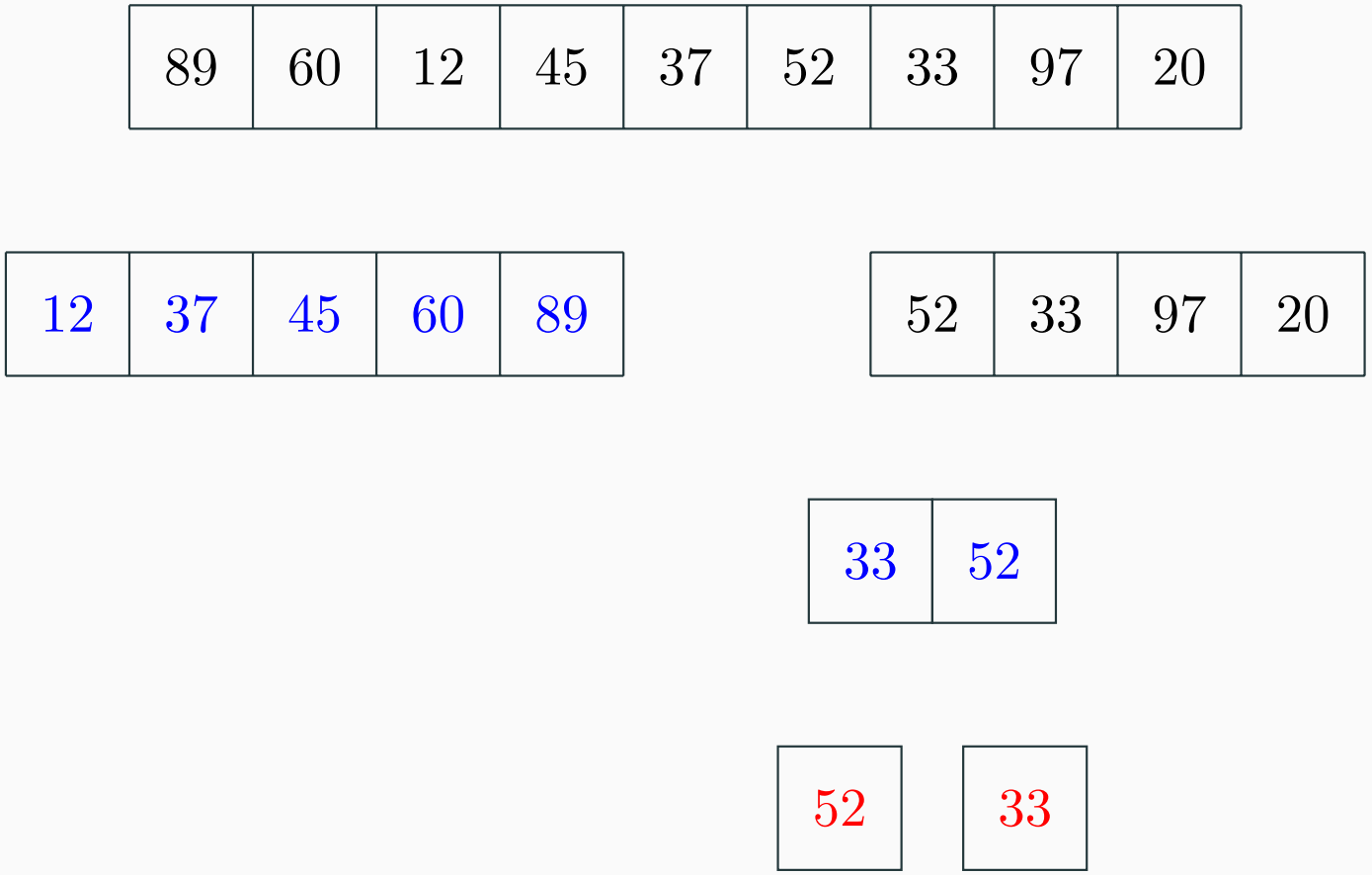




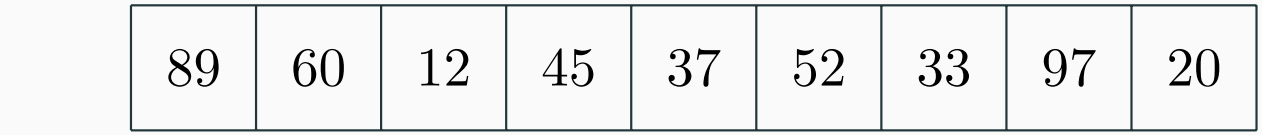
Visualização do mergesort

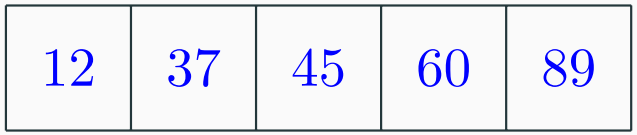


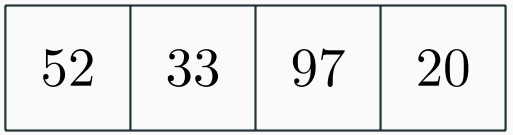
Visualização do mergesort

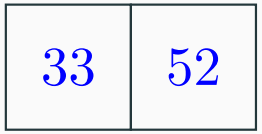


Visualização do mergesort

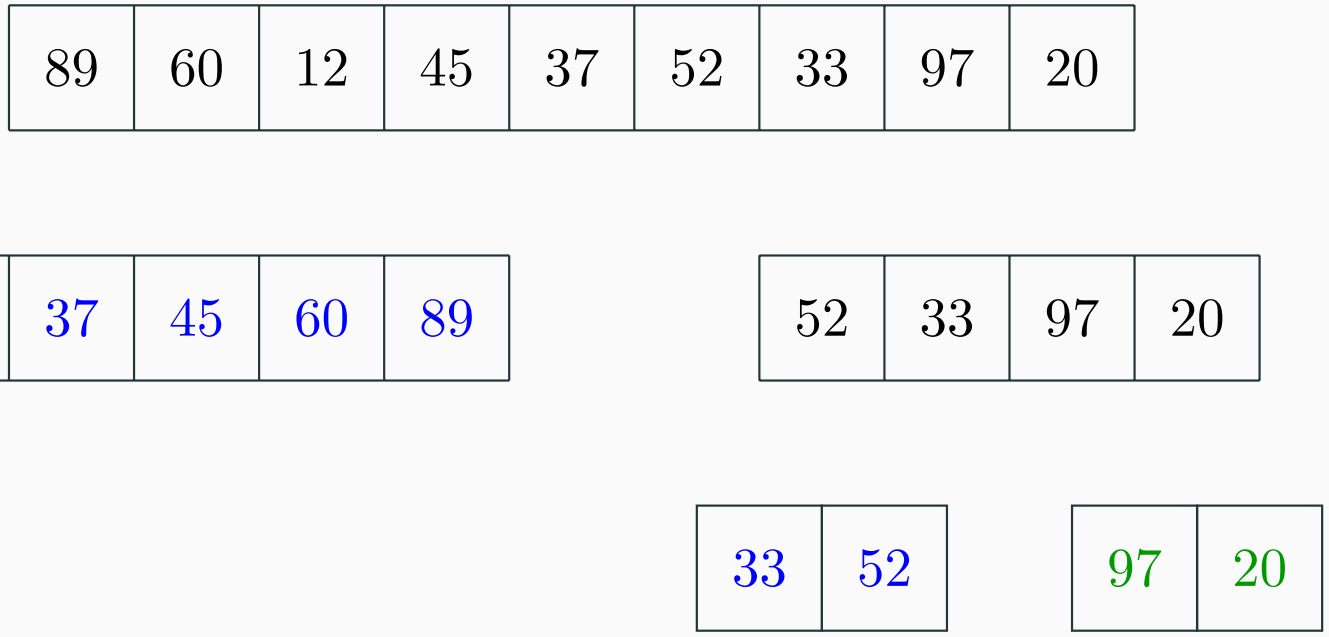




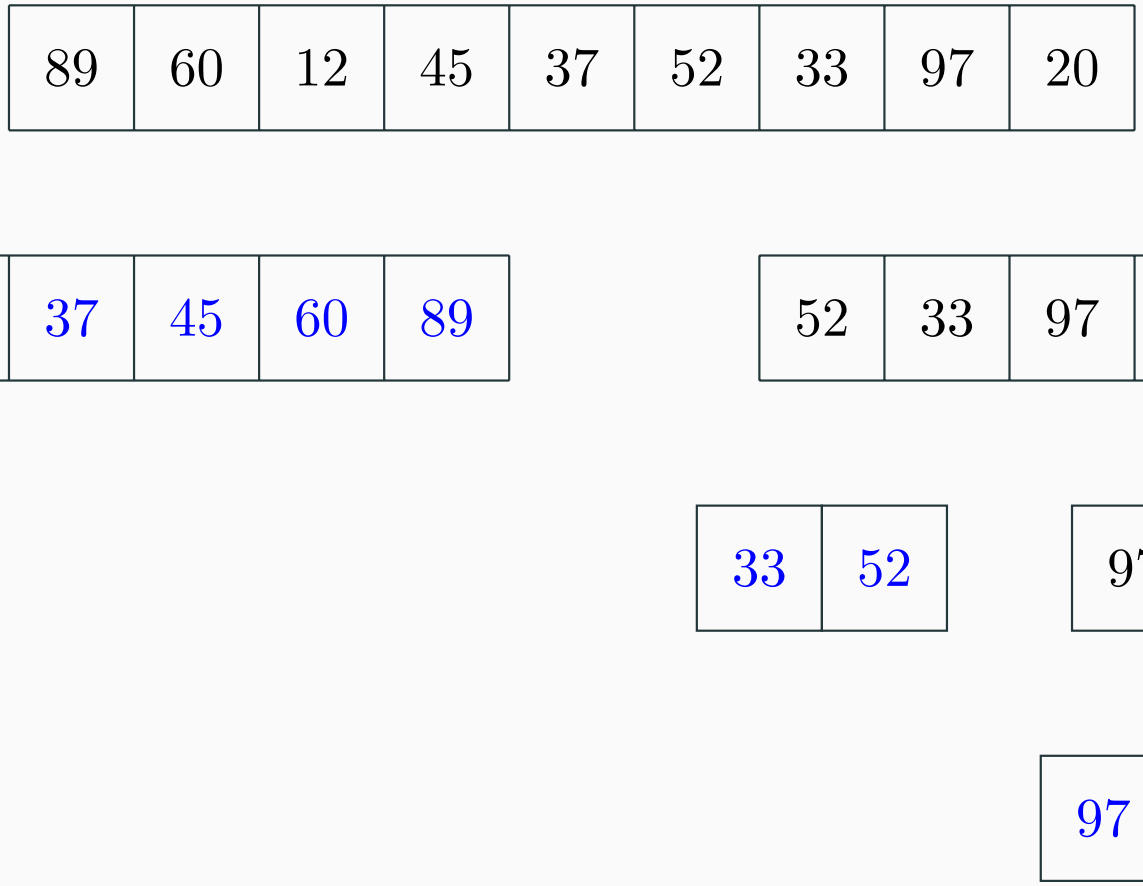




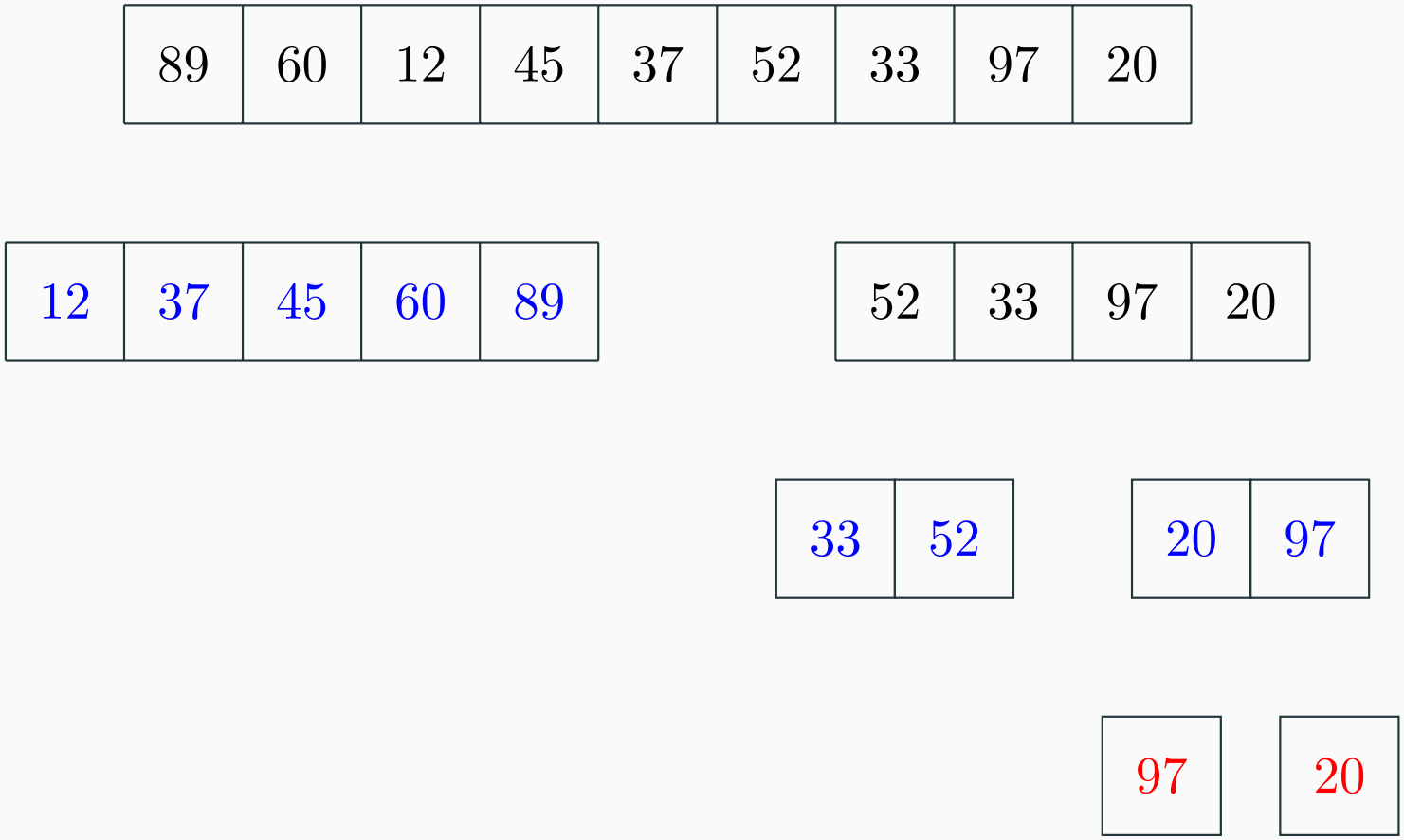
Visualização do mergesort



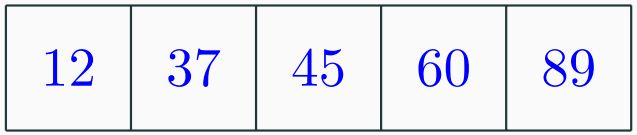
Visualização do mergesort

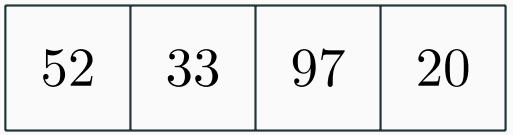


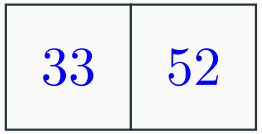
Visualização do mergesort

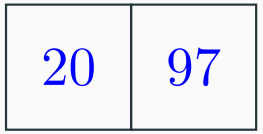


Visualização do mergesort

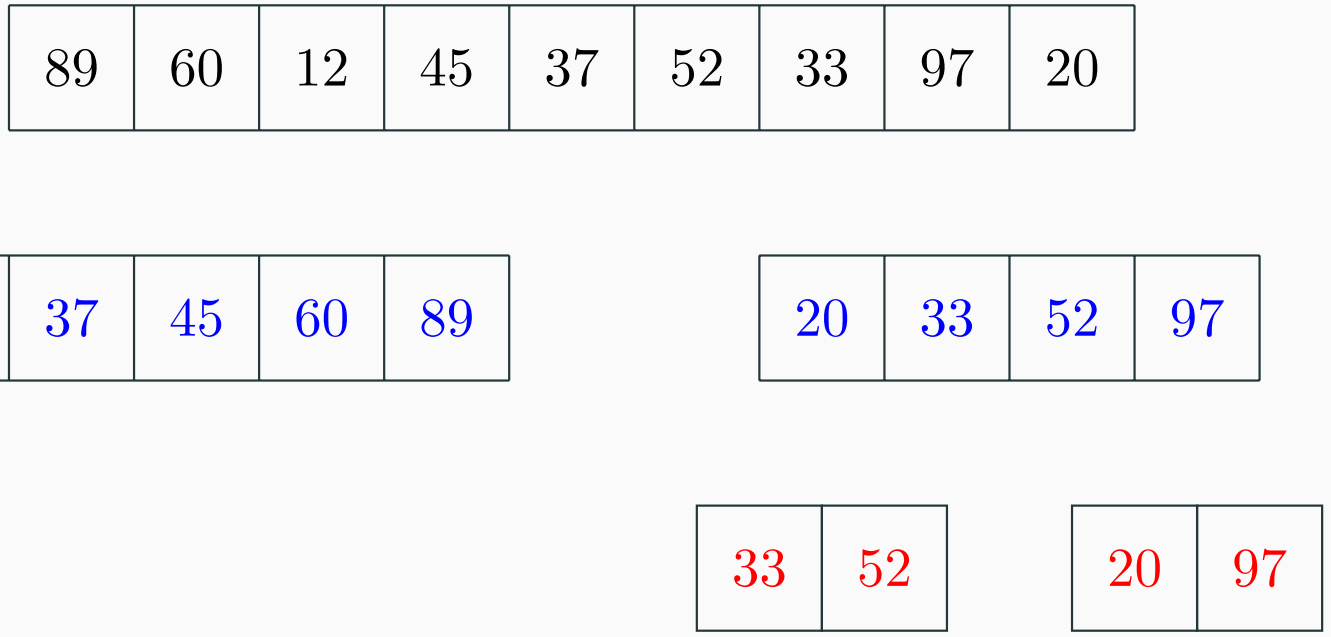




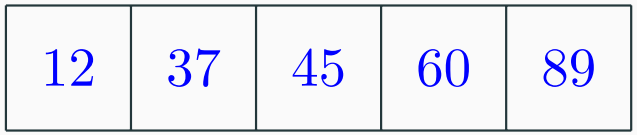


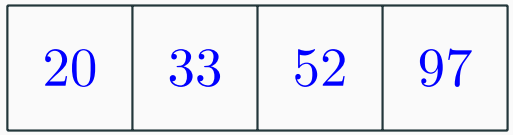


Visualização do mergesort

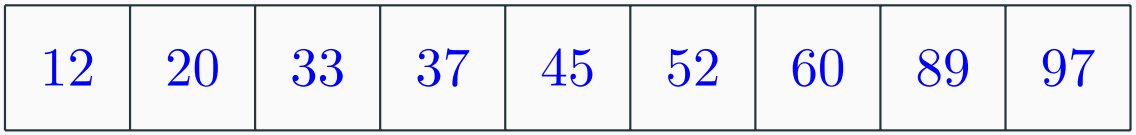


Visualização do mergesort





Visualização do mergesort



Visualização do `mergesort`

12	20	33	37	45	52	60	89	97

Implementação do mergesort

```
26
27 template<typename RandomAccessIterator>
28 void mergesort(RandomAccessIterator first, RandomAccessIterator last)
29 {
30     auto N = last - first;
31
32     if (N == 1)
33         return;
34
35     auto middle = first + (N + 1)/2;
36
37     mergesort(first, middle);
38     mergesort(middle, last);
39
40     merge(N, first, middle, last);
41 }
42
```

QuickSort

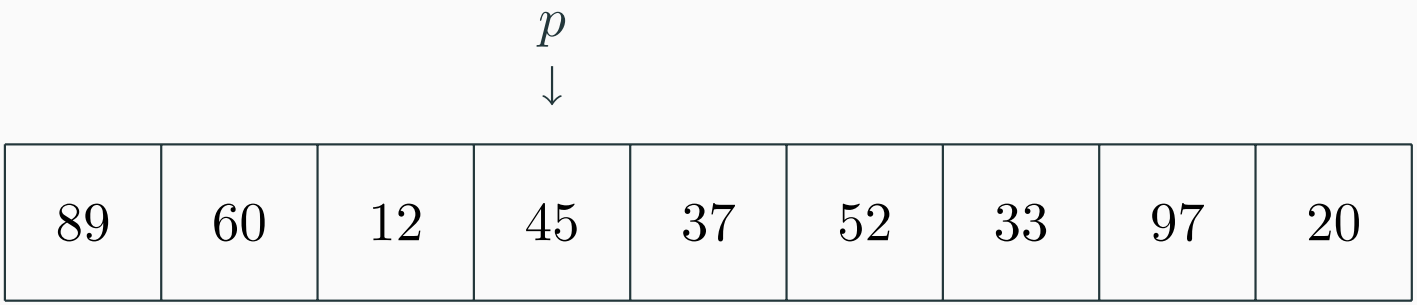
Motivação

- Embora o *MergeSort* seja um algoritmo que atinja a limite inferior $O(N \log N)$ para algoritmos de ordenação baseados em comparações, ele demanda uma memória adicional $O(N)$, não sendo portando um algoritmo *in-place*
- A ideia do *QuickSort* é aproveitar a ideia da divisão do vetor em subvetores menores, como é feito no *MergeSort*
- Contudo, a ideia é que o algoritmo seja *in-place*
- Assim, a divisão do vetor não será posicional, mas sim baseada no valor de um elemento escolhido arbitrariamente, denominado pivô
- O pivô permite um rearranjo dos elementos usando a própria memória do vetor, tornando o algoritmo *in-place*
- Embora o *QuickSort* tenha complexidade média $O(N \log N)$, no pior caso ele pode se degenerar para $O(N^2)$

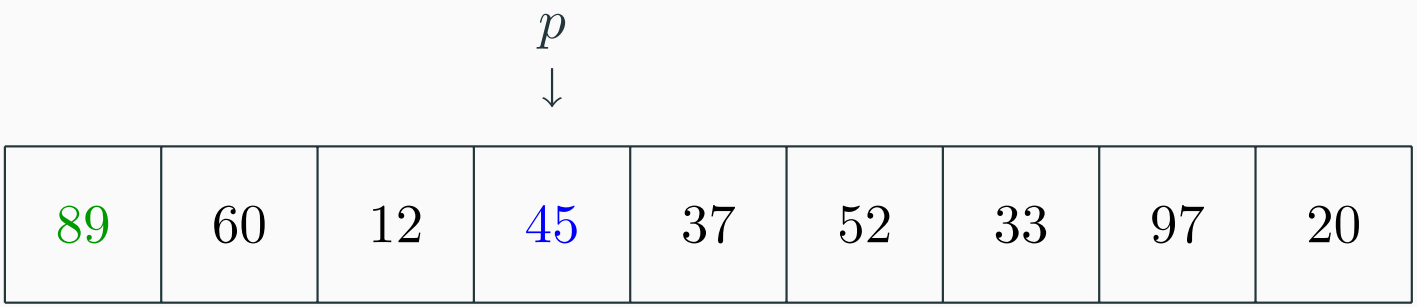
Pivoteamento

- Pivoteamento é o processo de reposicionamento dos elementos do vetor de acordo com o valor x do elemento pivô que ocupa o índice p
- Ao final do pivoteamento, todos elementos com valores menores que x estarão à esquerda do pivô, e os demais à direita
- O pivô já estará na posição correta em relação ao ordenamento global, de modo que o *QuickSort* pode prosseguir recursivamente nas duas partes separadas pelo pivô
- Para simplificar a rotina, no início do pivoteamento o pivô troca de posição com o primeiro elemento do vetor
- Ao final, o pivô se move para a posição adequada e esta posição é retornada
- Para evitar o pior caso, a escolha do pivô deve ser aleatória entre todos os índices possíveis

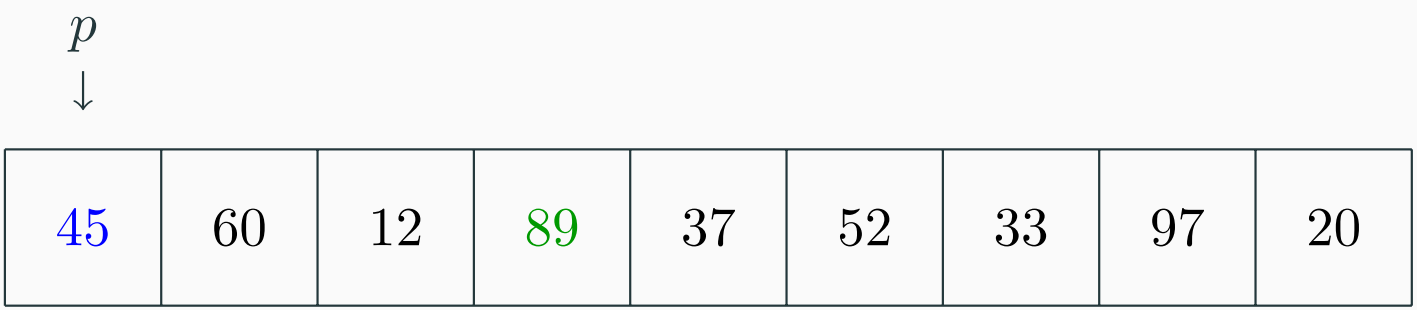
Visualização da rotina de pivoteamento



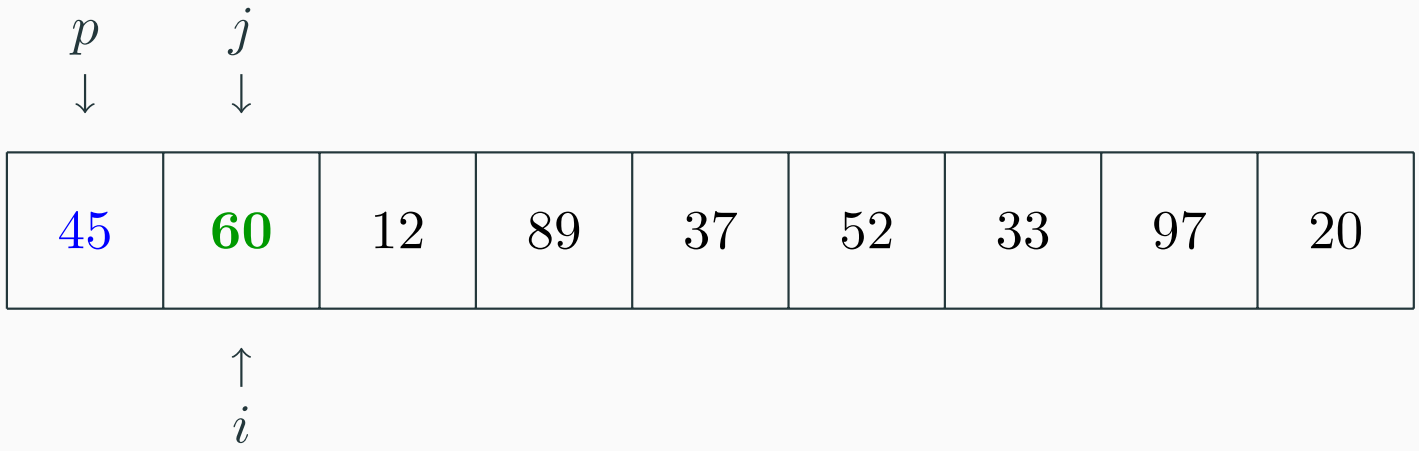
Visualização da rotina de pivoteamento



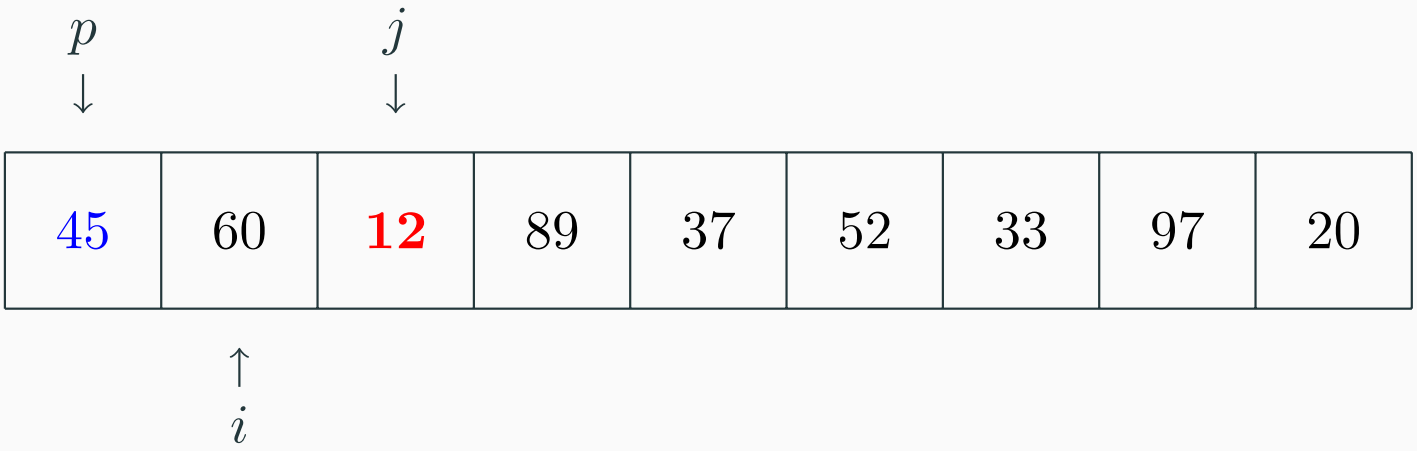
Visualização da rotina de pivoteamento



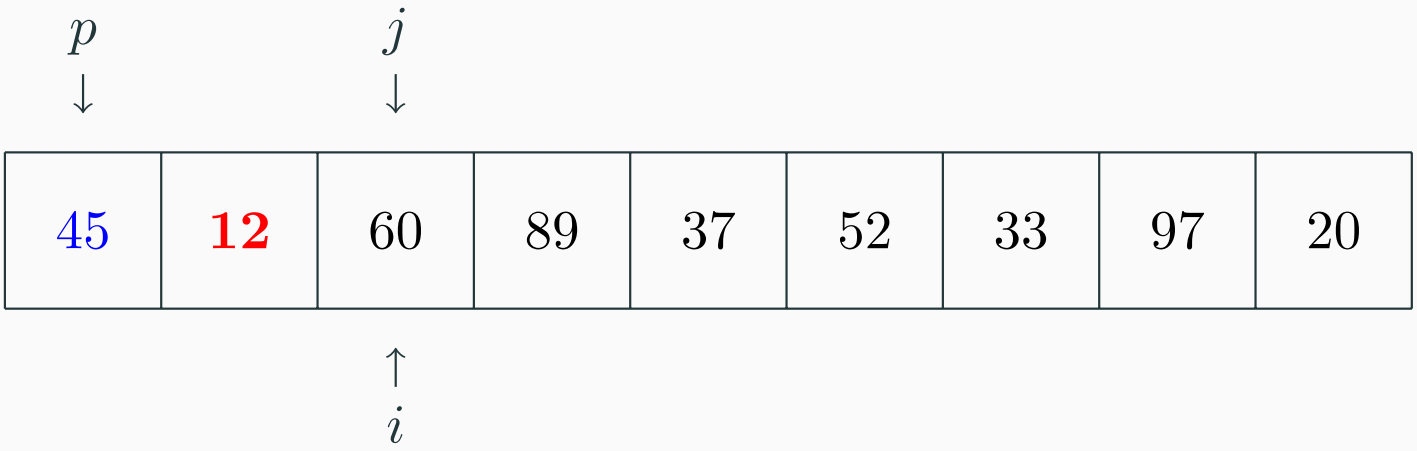
Visualização da rotina de pivoteamento



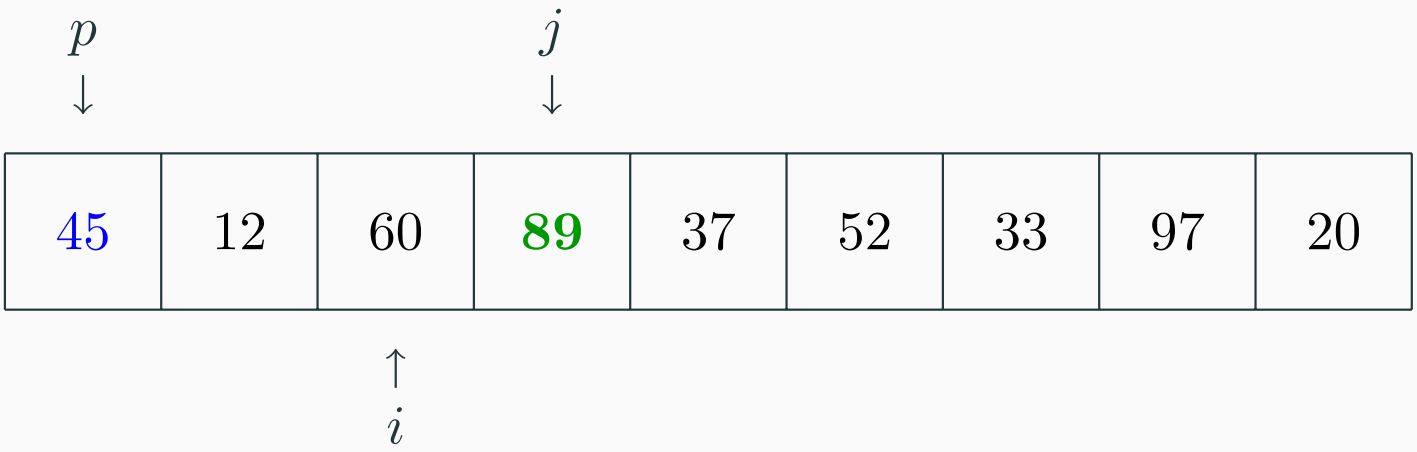
Visualização da rotina de pivoteamento



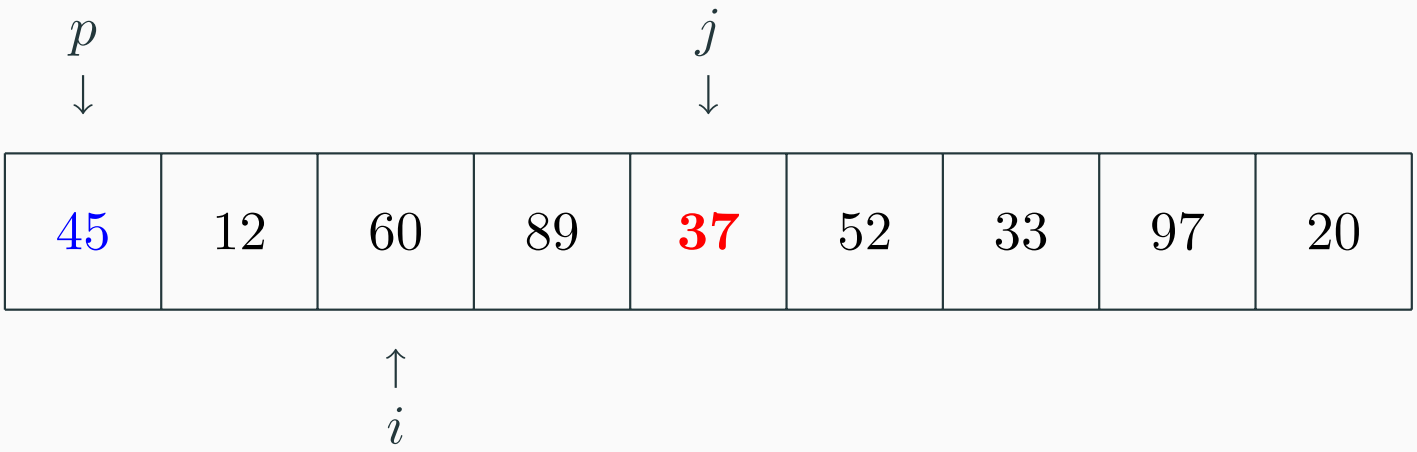
Visualização da rotina de pivoteamento



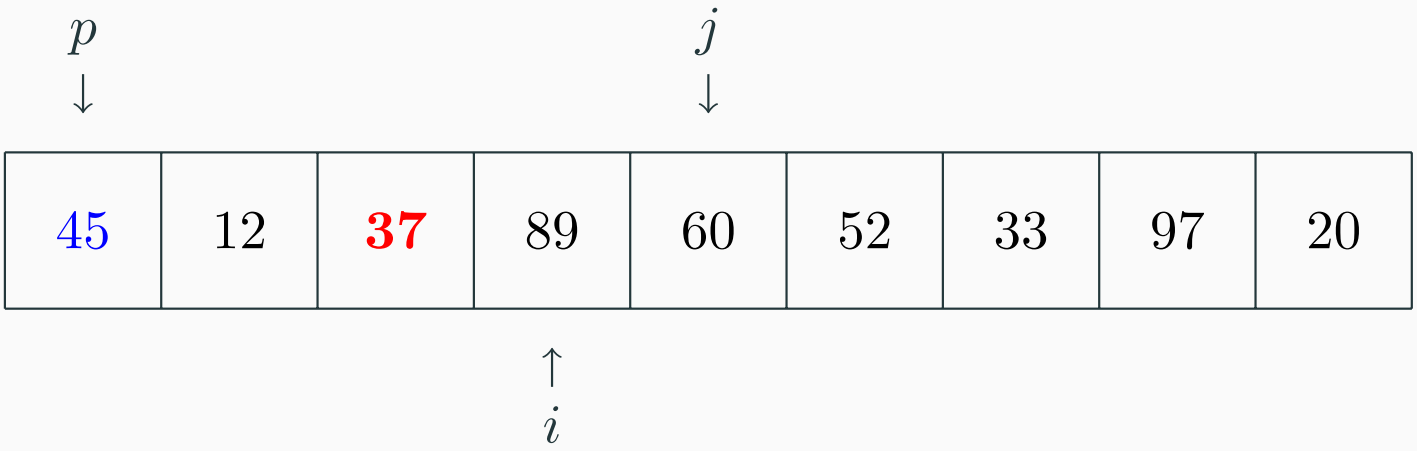
Visualização da rotina de pivoteamento



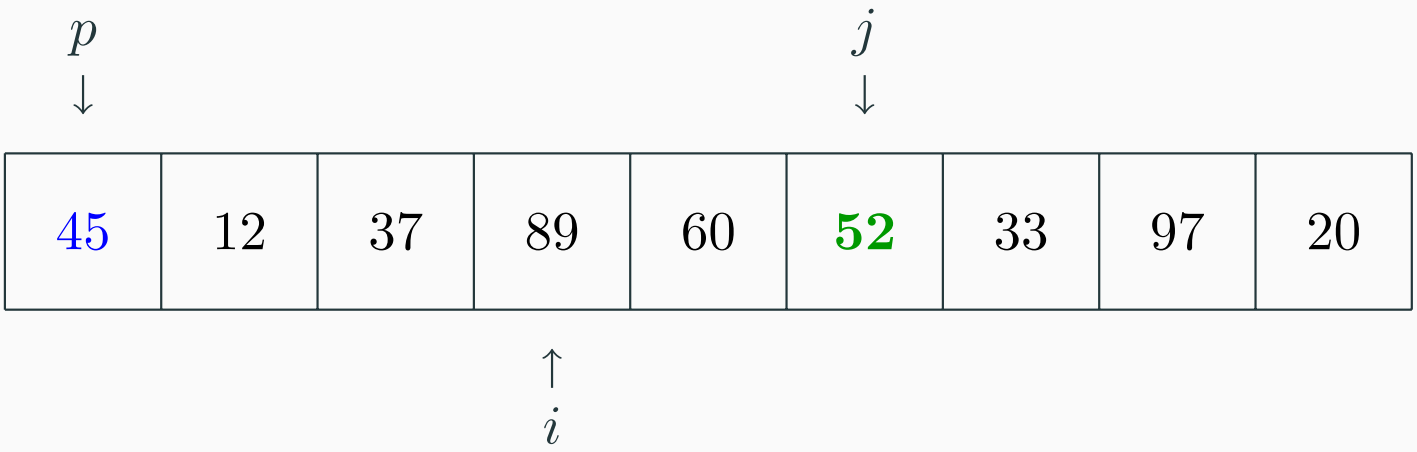
Visualização da rotina de pivoteamento



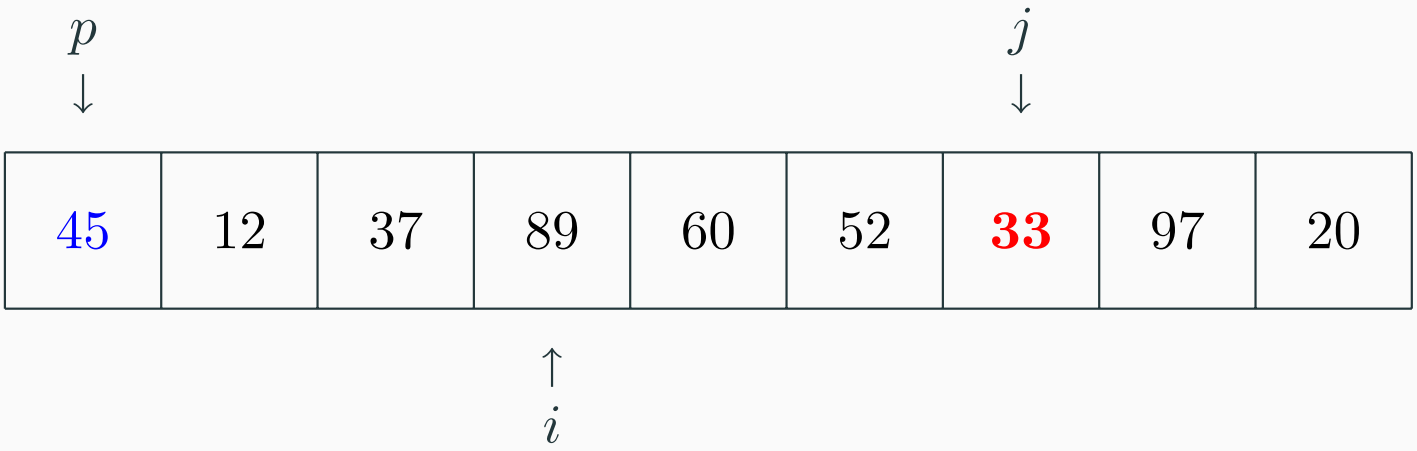
Visualização da rotina de pivoteamento



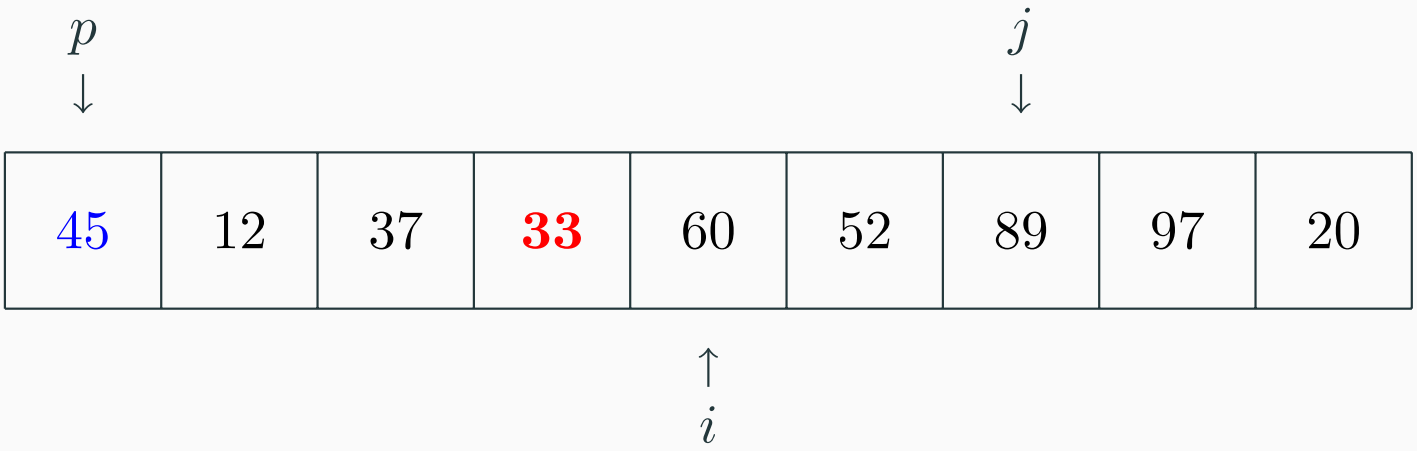
Visualização da rotina de pivoteamento



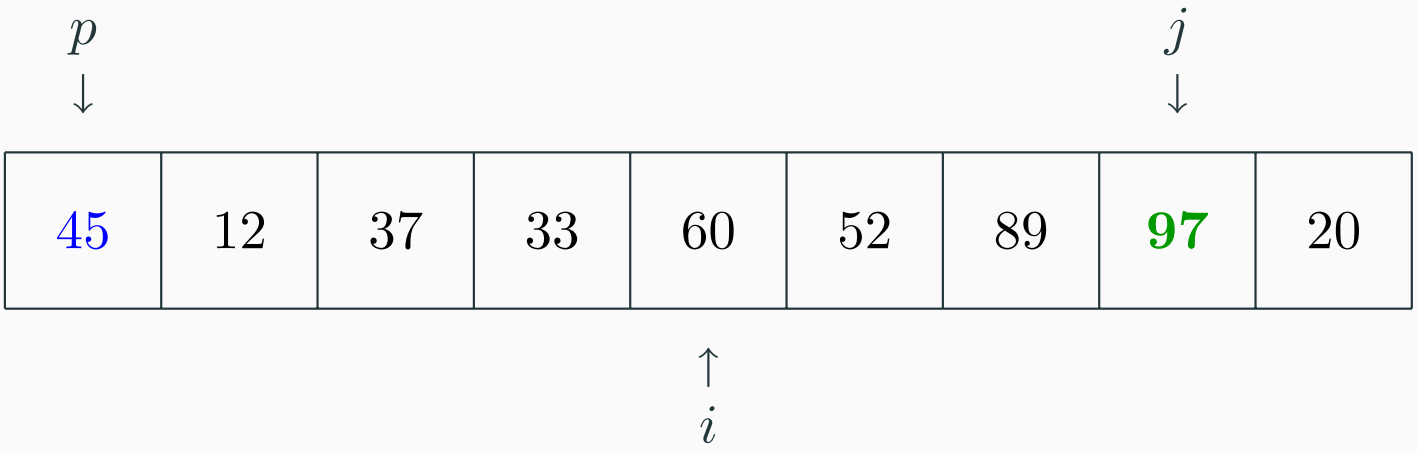
Visualização da rotina de pivoteamento



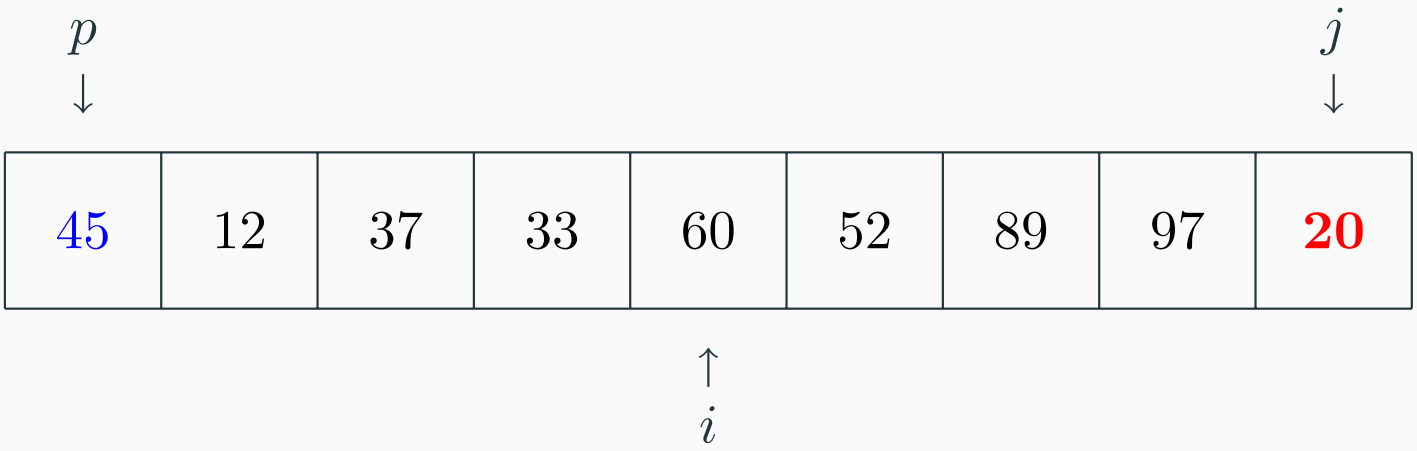
Visualização da rotina de pivoteamento



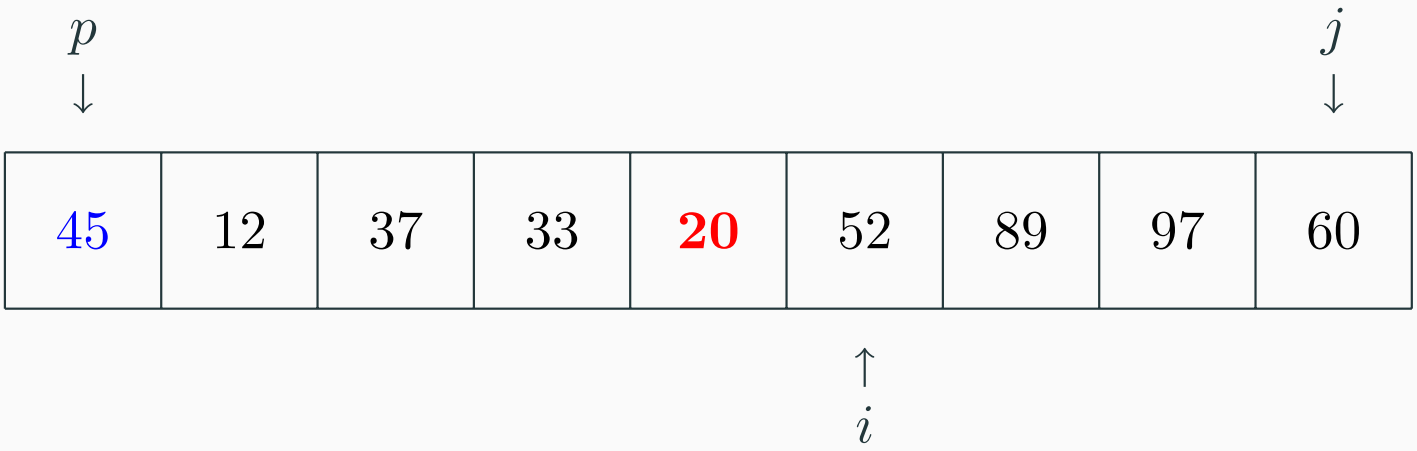
Visualização da rotina de pivoteamento



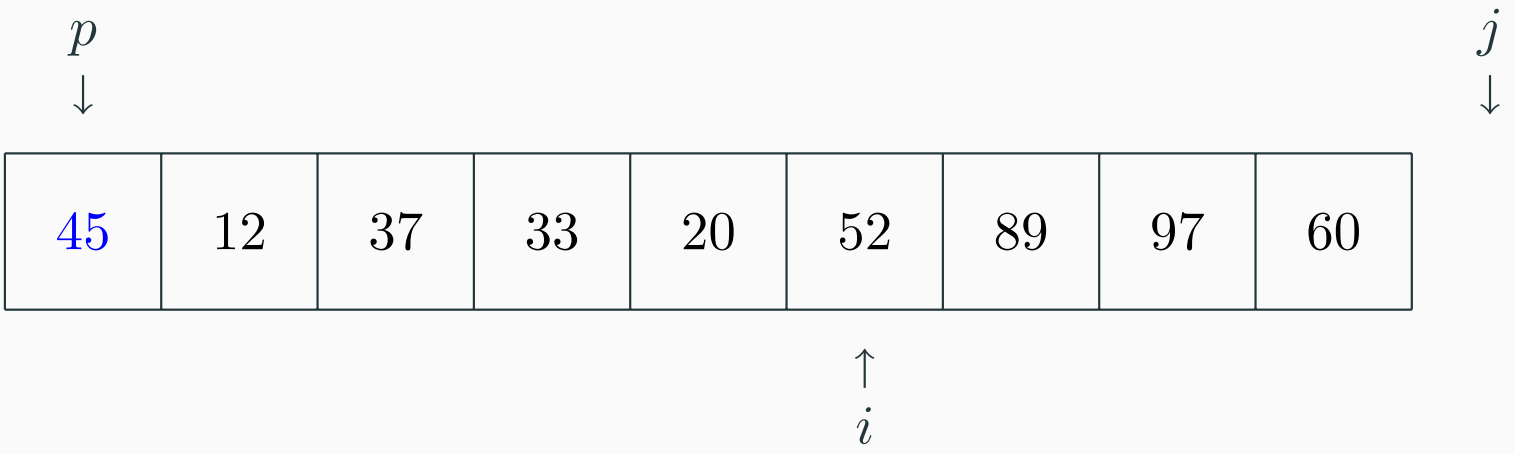
Visualização da rotina de pivoteamento



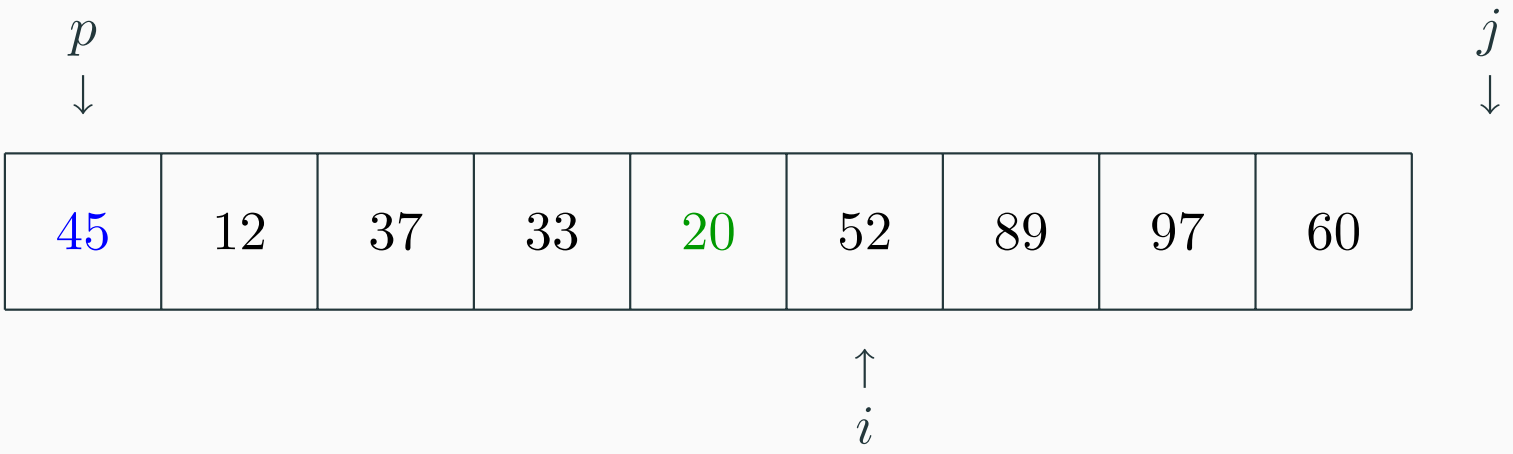
Visualização da rotina de pivoteamento



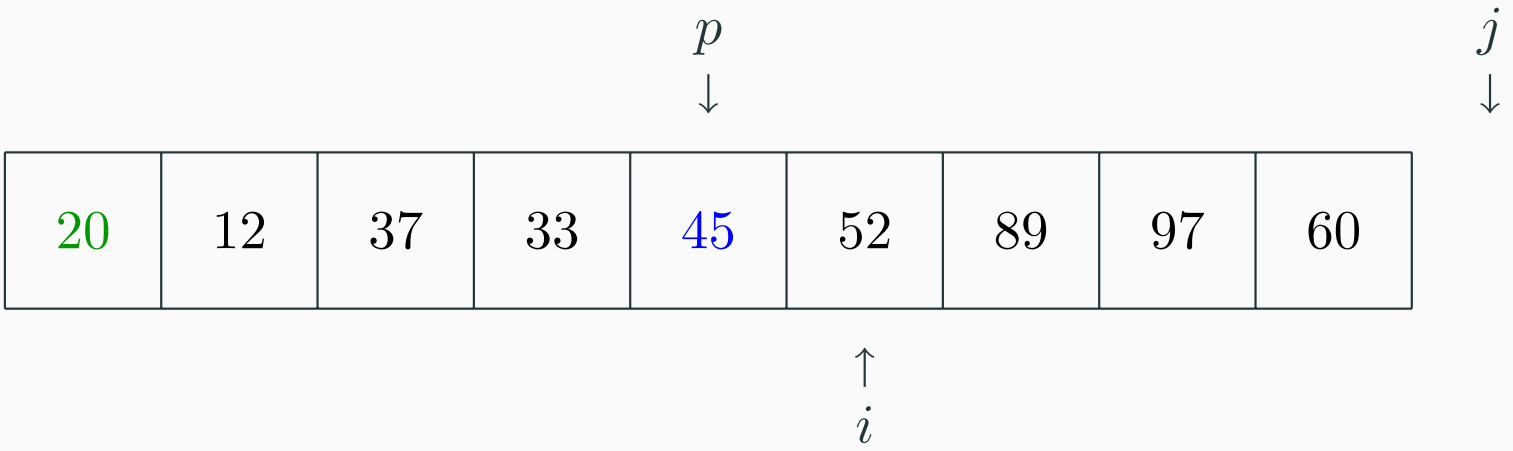
Visualização da rotina de pivoteamento



Visualização da rotina de pivoteamento



Visualização da rotina de pivoteamento



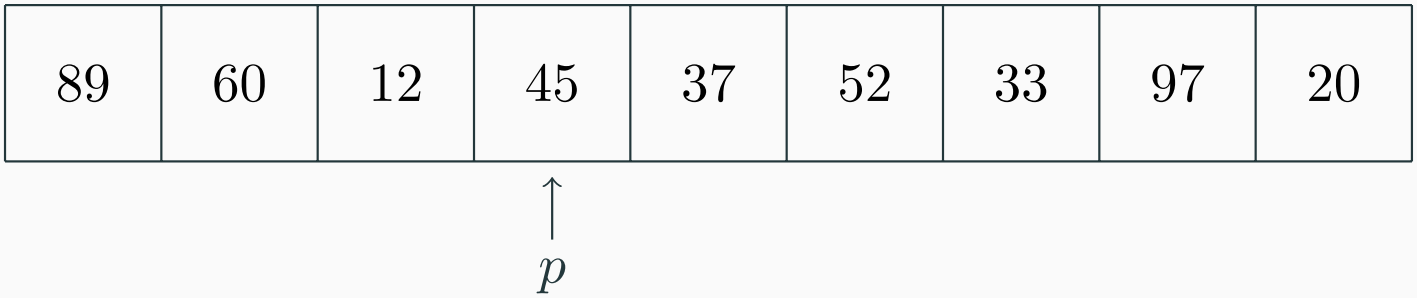
Implementação da rotina de pivoteamento

```
5  template<typename RandIt>
6  RandIt partitioning(RandIt first, RandIt last)
7  {
8      auto N = last - first;
9      RandIt p = first + (rand() % N);    // slide: RandIt p = first + 3;
10
11     swap(*first, *p);
12     p = first;
13     RandIt i = first + 1;
14
15     for (RandIt j = first + 1; j < last; ++j)
16         if (*j < *p)
17         {
18             swap(*j, *i);
19             ++i;
20         }
21
22     swap(*p, *(--i));
23
24     return i;
25 }
```

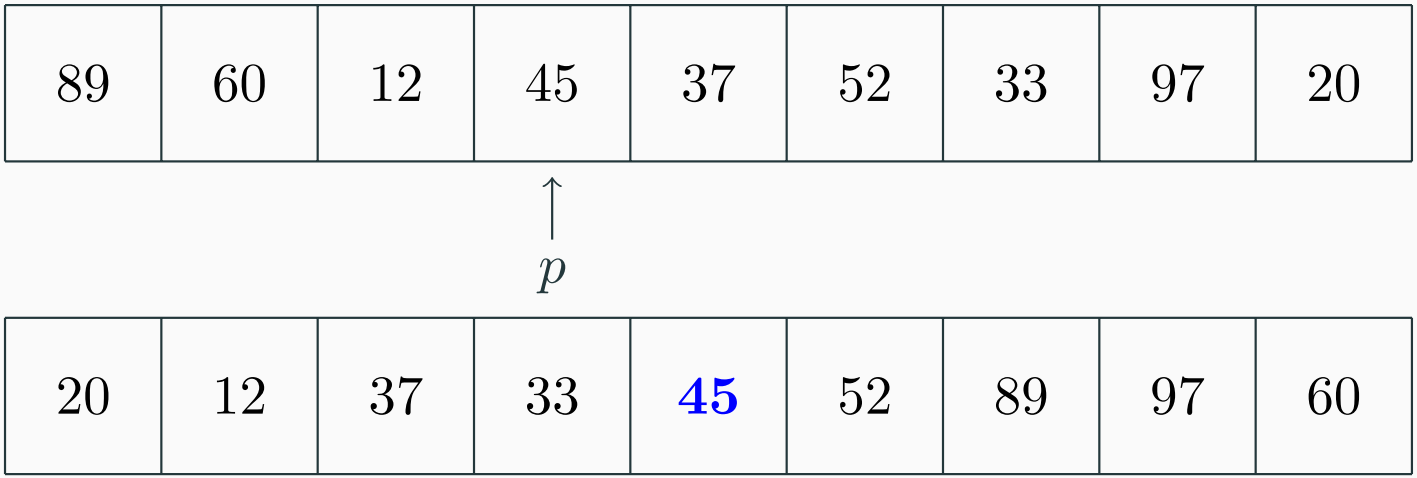
Visualização do *quicksort*

89	60	12	45	37	52	33	97	20

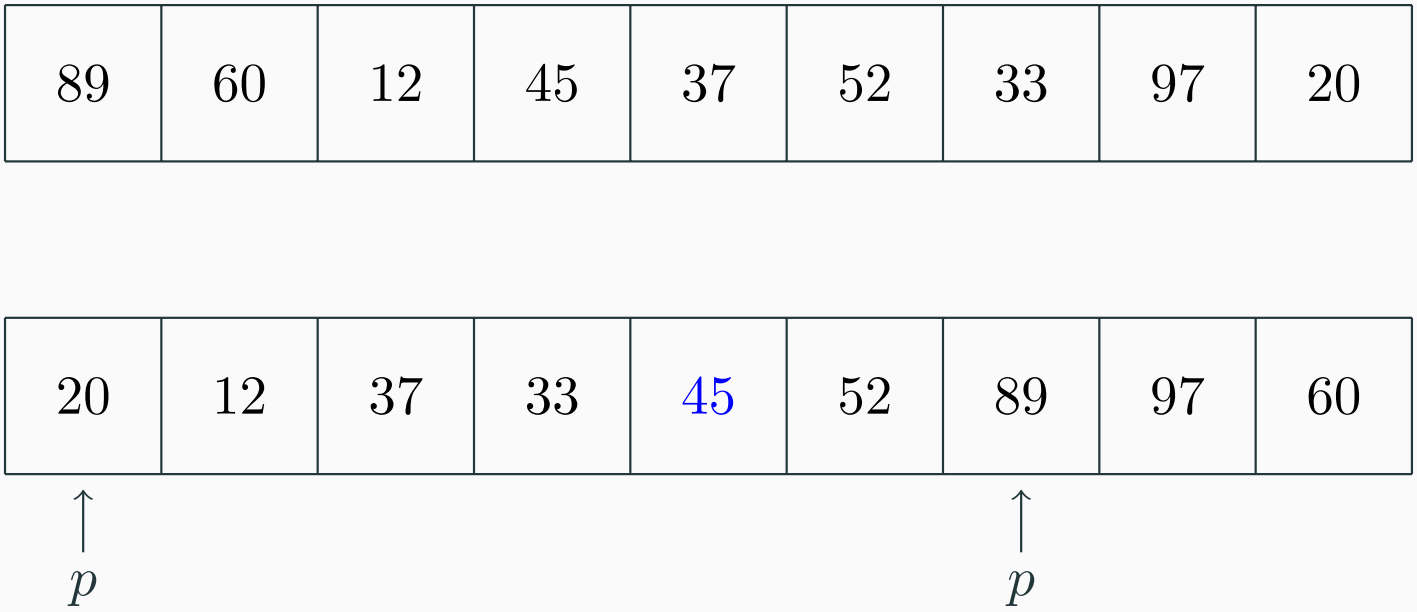
Visualização do *quicksort*



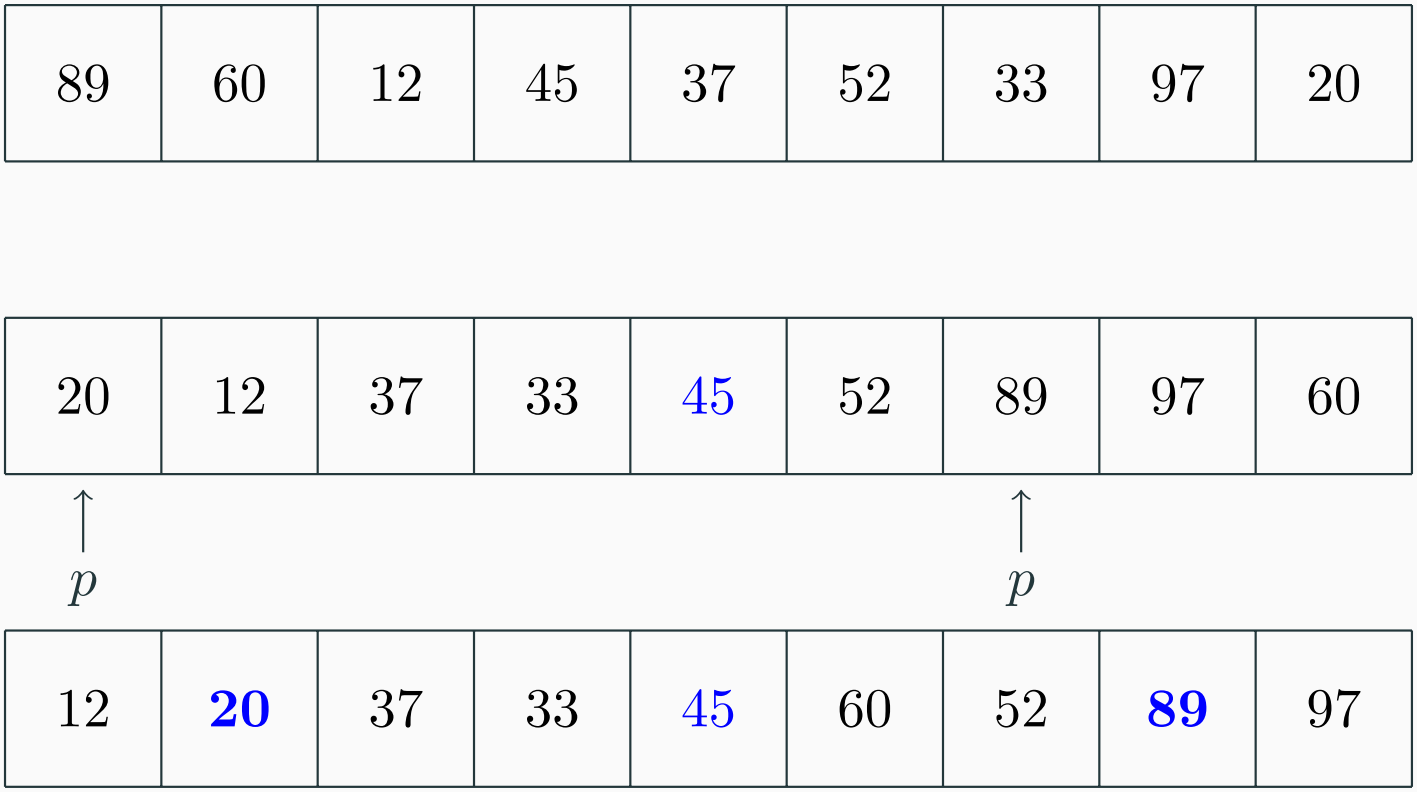
Visualização do *quicksort*



Visualização do *quicksort*



Visualização do *quicksort*



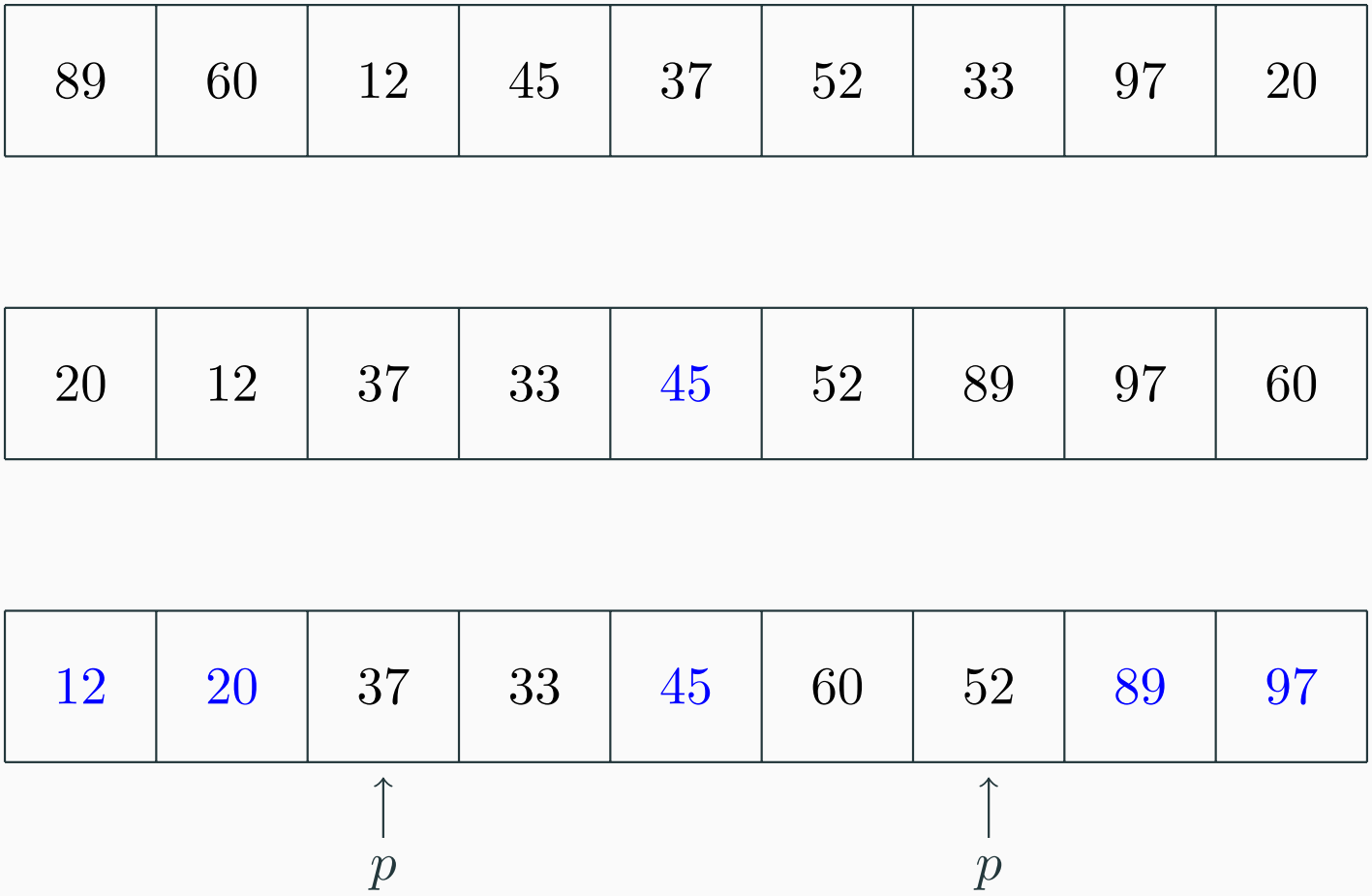
Visualização do *quicksort*

89	60	12	45	37	52	33	97	20

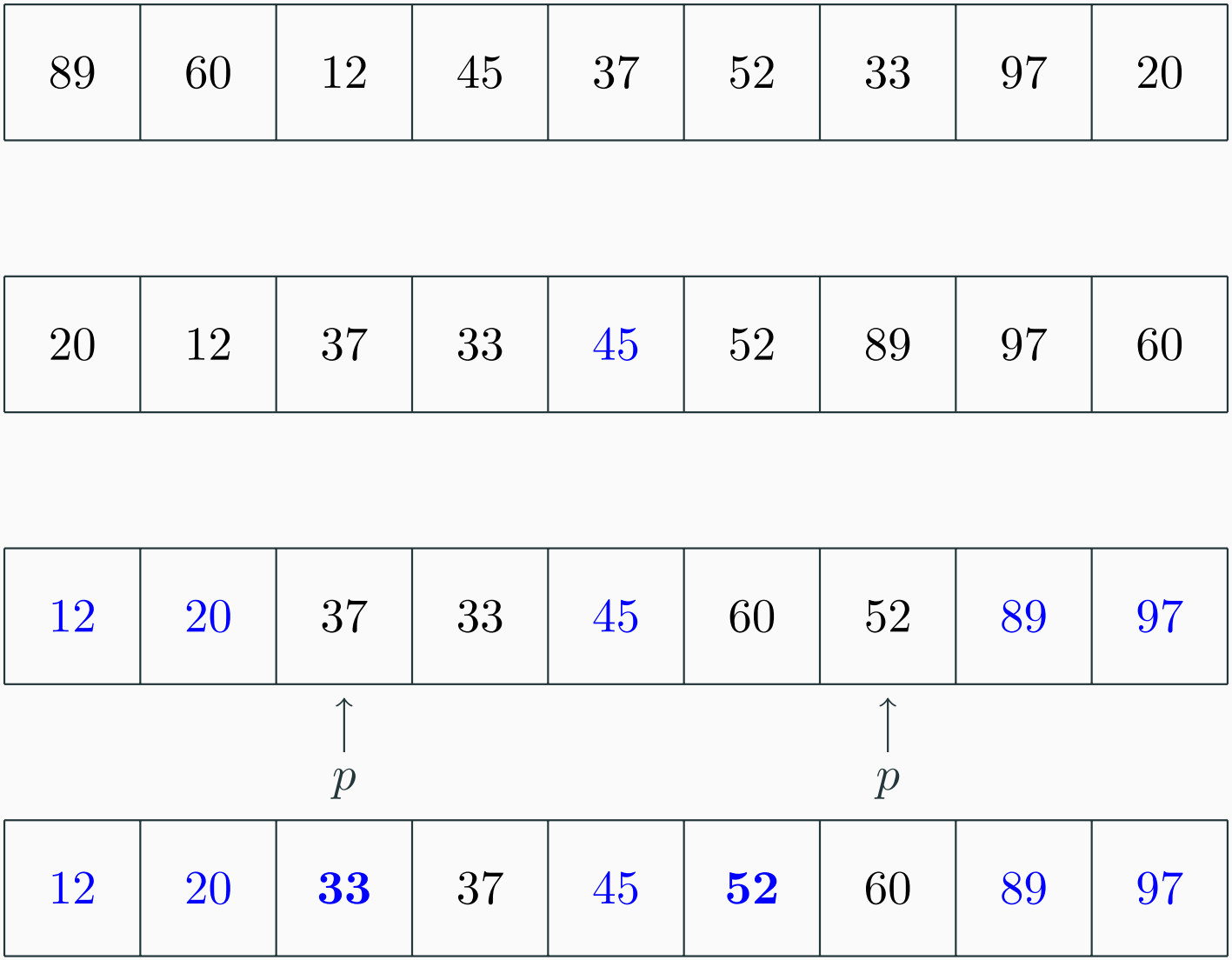
20	12	37	33	45	52	89	97	60

12	20	37	33	45	60	52	89	97

Visualização do *quicksort*



Visualização do *quicksort*



Visualização do *quicksort*

89	60	12	45	37	52	33	97	20

20	12	37	33	45	52	89	97	60

12	20	37	33	45	60	52	89	97

12	20	33	37	45	52	60	89	97

Implementação do *quicksort*

```
27 template<typename RandomAccessIterator>
28 void quicksort(RandomAccessIterator first, RandomAccessIterator last)
29 {
30     auto p = partitioning(first, last);
31
32     if (p - first > 1)
33         quicksort(first, p);
34
35     if (last - (p + 1) > 1)
36         quicksort(p + 1, last);
37 }
```

Referências

1. **DROZDEK**, Adam. *Algoritmos e Estruturas de Dados em C++*, 2002.
2. **KERNIGHAN**, Bryan; **RITCHIE**, Dennis. *The C Programming Language*, 1978.
3. **ROUGHGARDEN**, Tim. *Algorithms Illuminated (Part 1): The Basics*, LLC, 2018.
4. **STROUSTROUP**, Bjarne. *The C++ Programming Language*, 2013.