COUNTING SORT E PIGEONHOLE SORT

Kauã Lucas, Anderson Rodrigues, Humberto Henrique

Sumário

Introdução

Algoritmos de ordenação assintótica são métodos para organizar dados de forma eficiente, avaliando o desempenho em relação ao tamanho da entrada. Nesta apresentação, exploraremos dois algoritmos: Counting Sort e Pigeonhole Sort, detalhando suas características, aplicações e comparações

Counting Sort

Ordenação pela Frequência dos Elementos

• O que é o Counting Sort ?

O Counting Sort é um algoritmo de ordenação de inteiros criado por Harold Seward, um cientista da computação, engenheiro e inventor, em 1954 no Instituto de Tecnologia de Massachusetts.

Ele é um algoritmo de Programação Dinâmica, que funciona construindo uma contagem de frequência das chaves do vetor de entrada e usando essa estrutura para construir um vetor de saída ordenado.

• Como fuinciona ?

Sua implementação utiliza uma lista de entrada (A), de tamanho n, com elementos que estão no intervalo de 0 a k, onde k é um número inteiro e positivo, representando o maior número da lista.

O algoritmo ainda possui a Lista de Contagem (C), de tamanho k+1, que fornece temporariamente o armazenamento das contagens cumulativas da Lista A. E a Lista de Saída (B), de tamanho N, é a saída ordenada .

Sua implementação pode ser separada em 3 etapas:

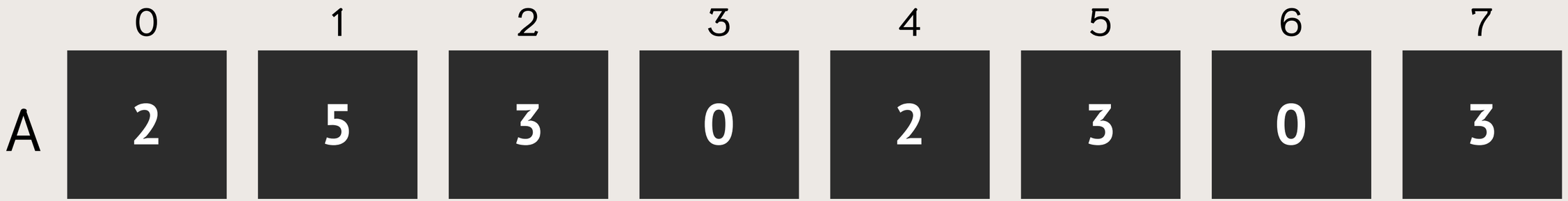
1. Contagem
2. Marcação de Posição
3. Ordenação

1. Contagem

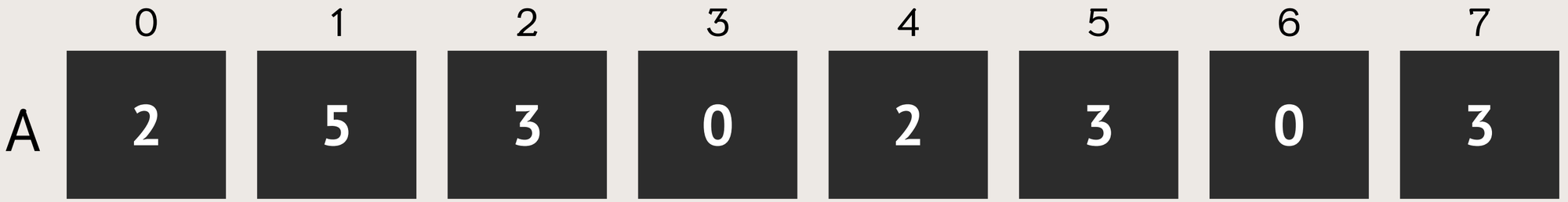
Nesta etapa, realiza-se a contagem dos elementos da lista de entrada A e os resultados são armazenados na lista C, onde cada posição corresponde ao número encontrado em A.

Contagem

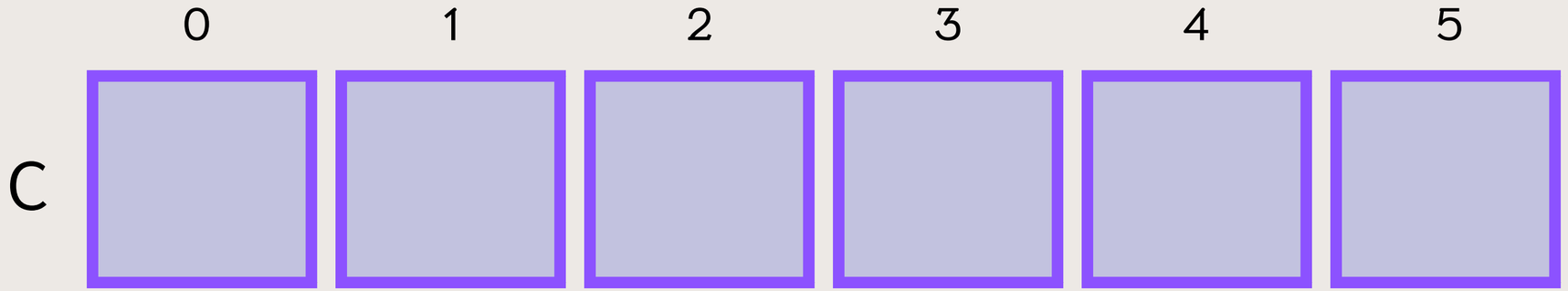
Exemplo de entrada:



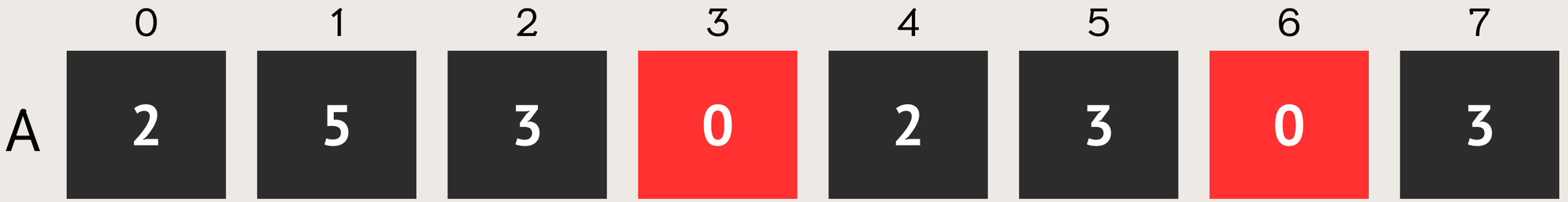
Contagem

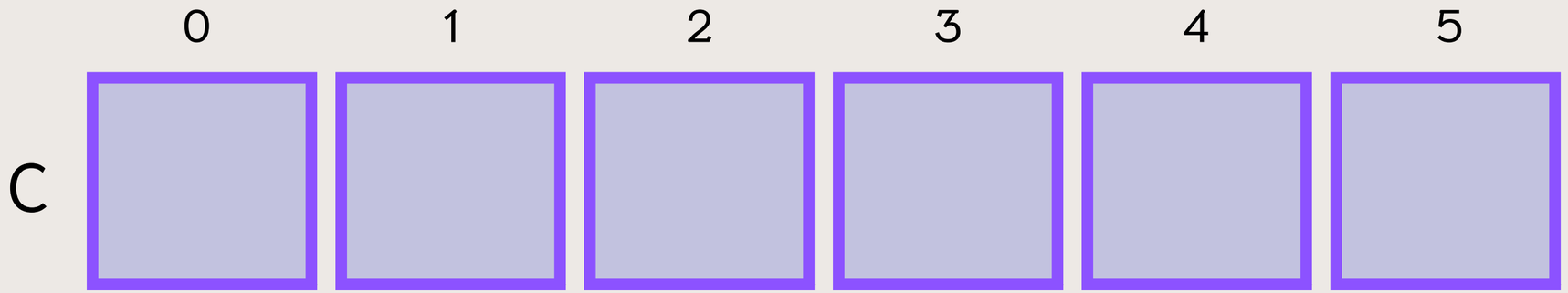


K = 5

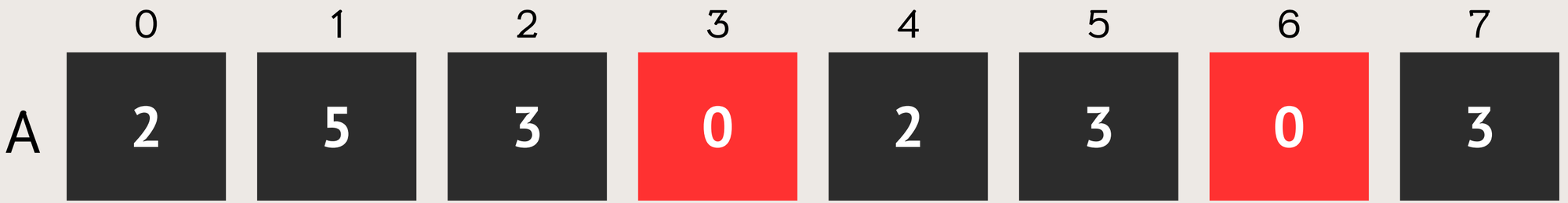


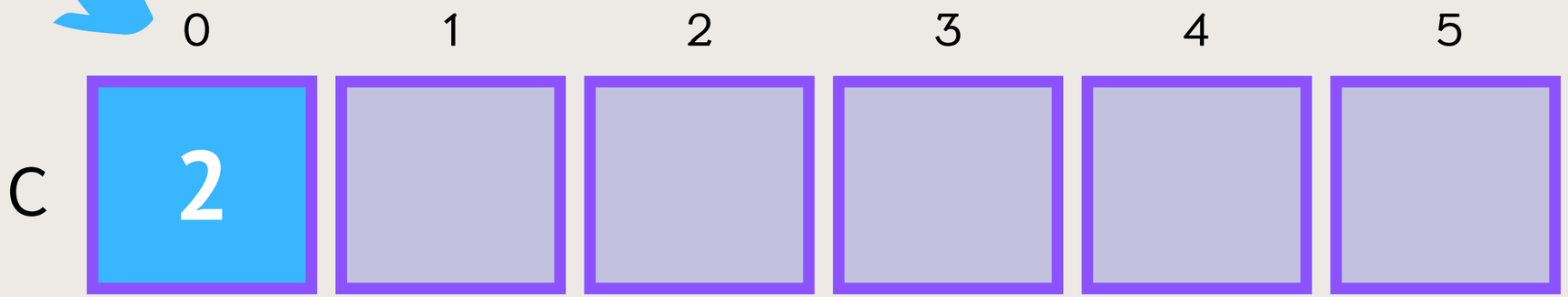
Contagem



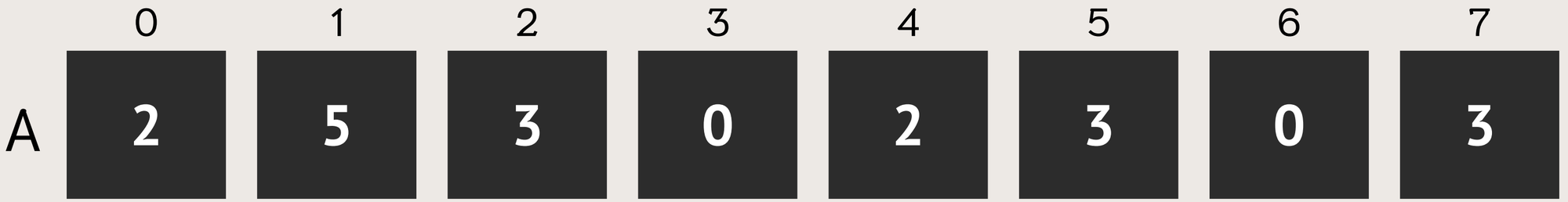


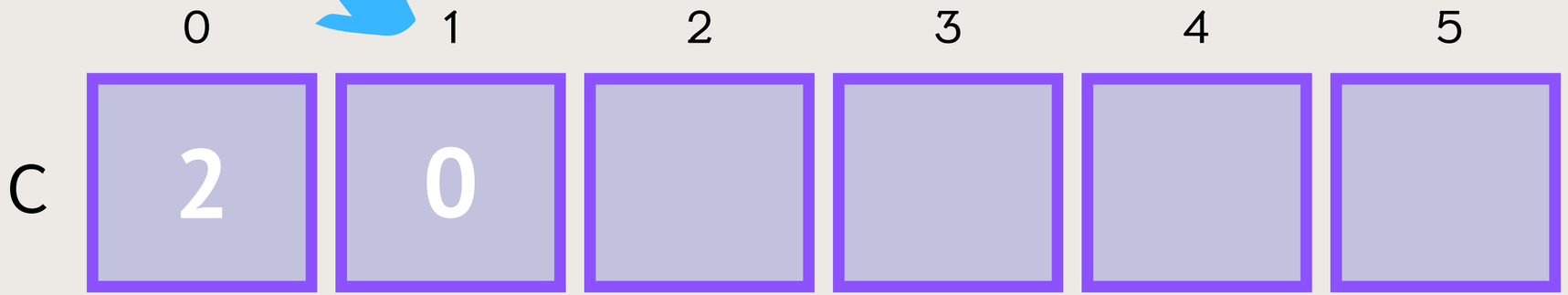
Contagem



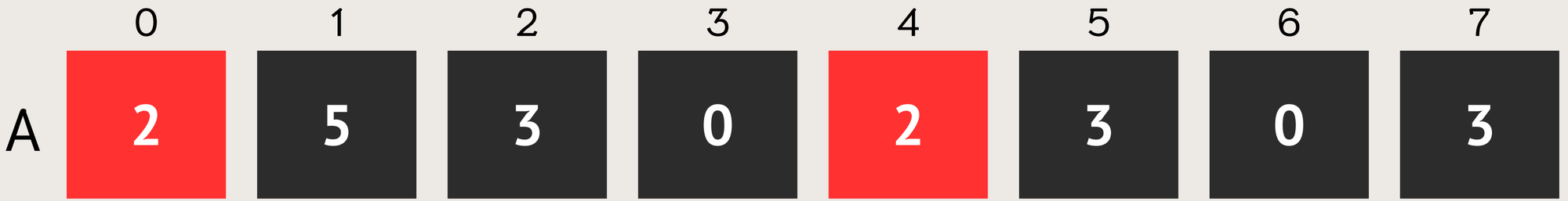


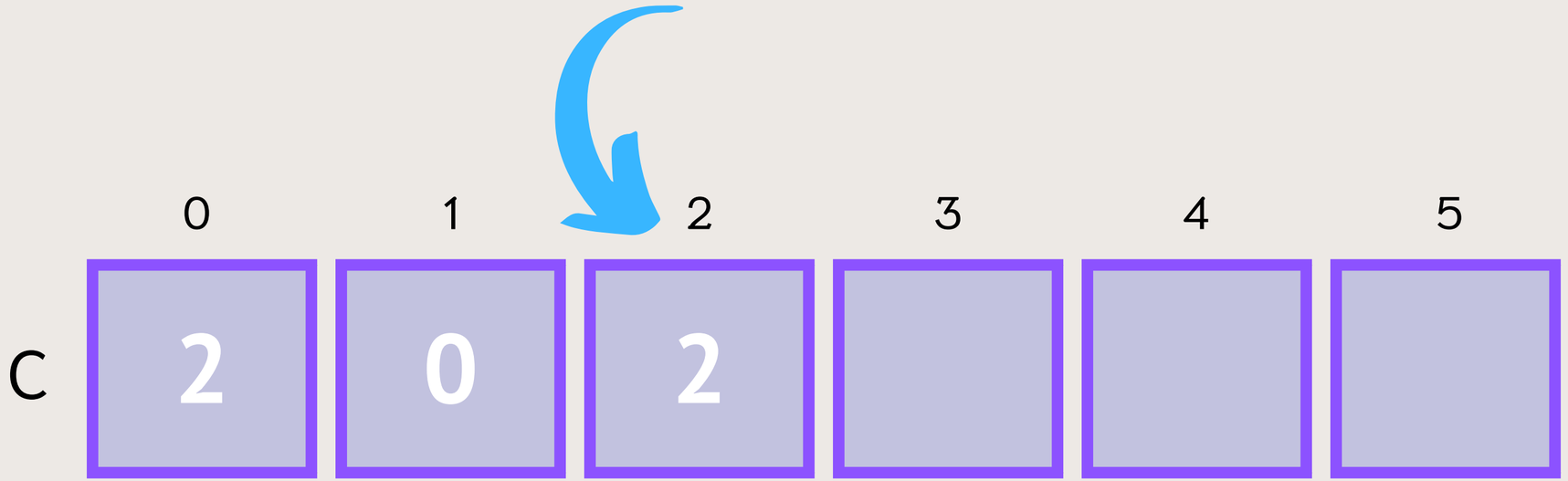
Contagem



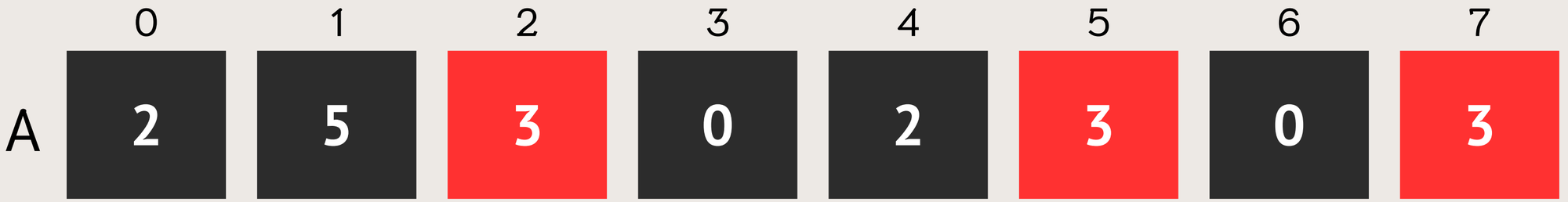


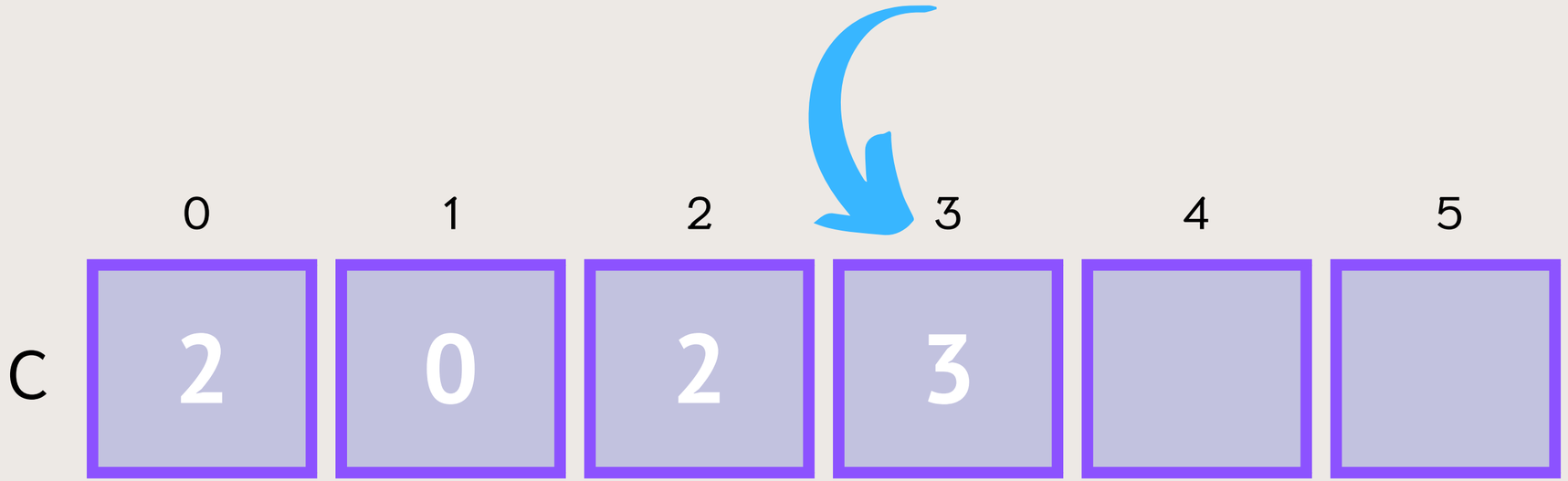
Contagem



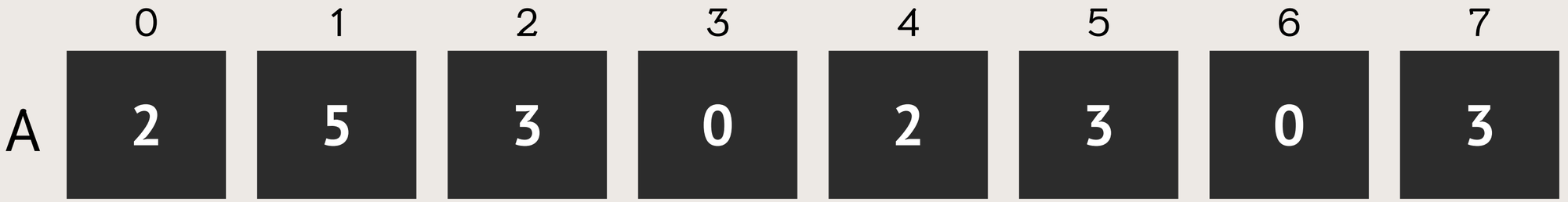


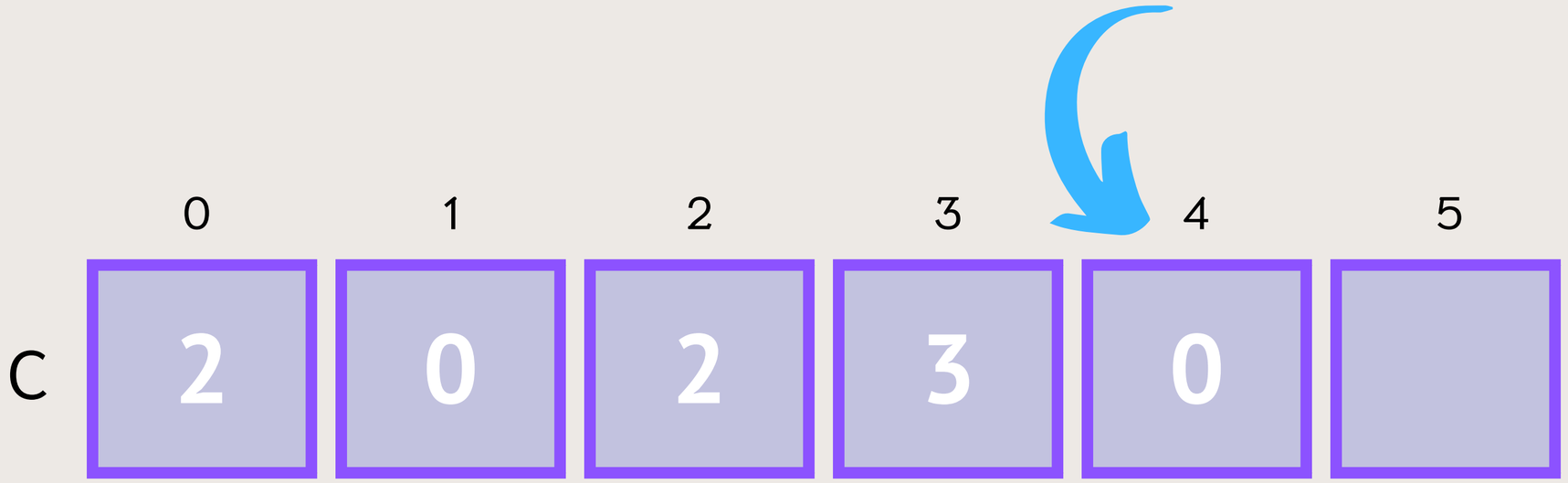
Contagem



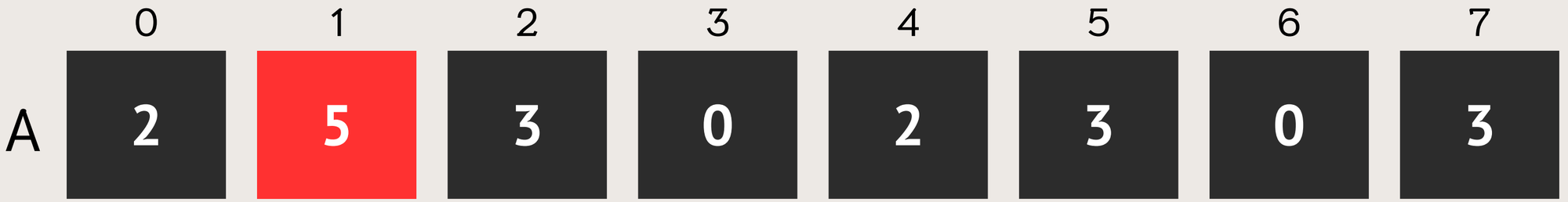


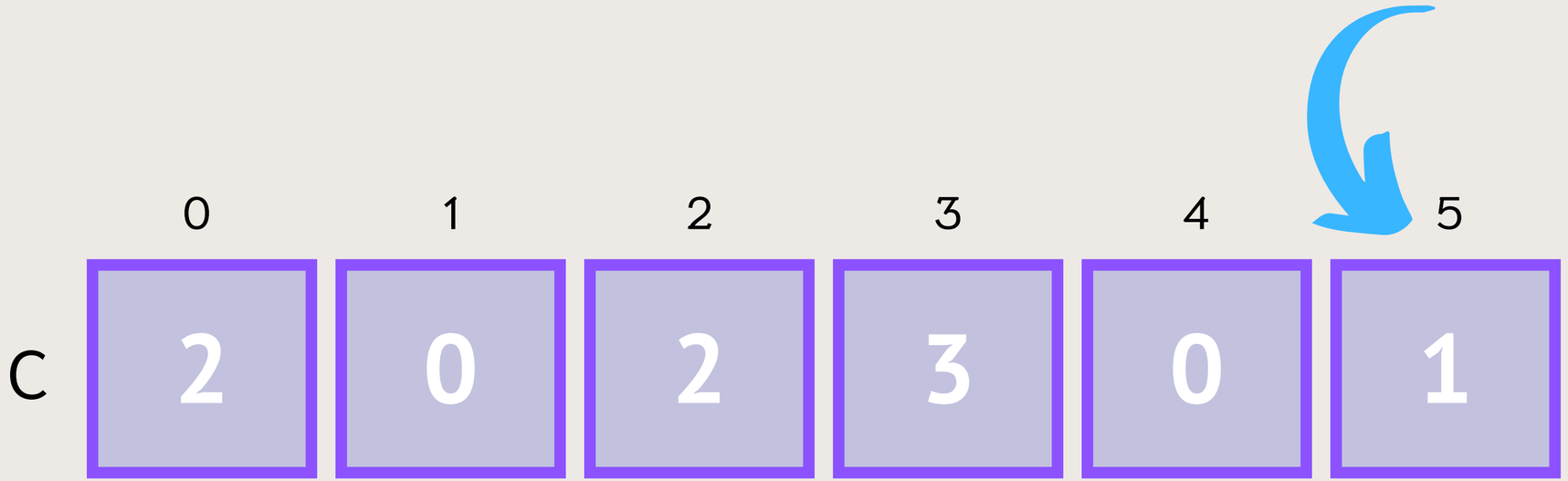
Contagem





Contagem

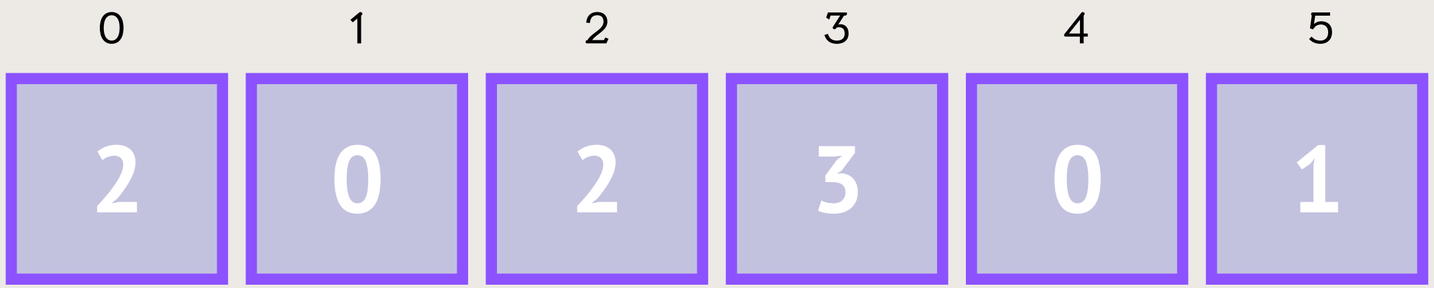


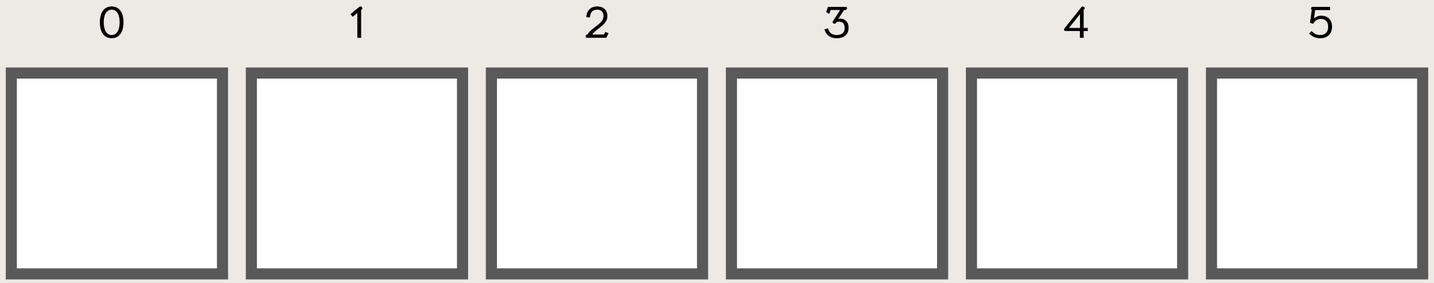


2. Marcação de Posição

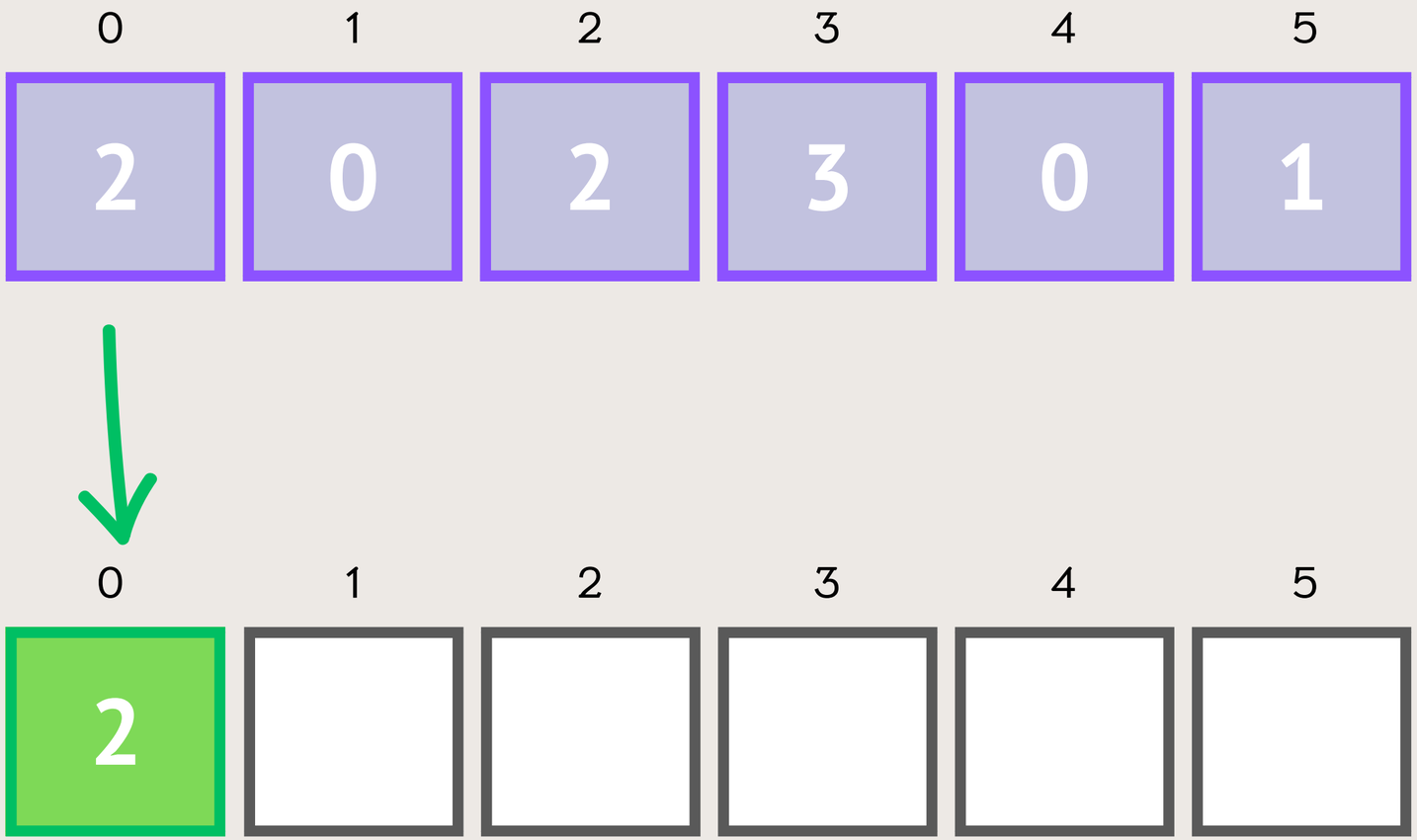
Na marcação de posição, cada posição na lista C é atualizada somando-se ao valor da posição anterior. Isso garante que cada posição em C represente a contagem acumulada dos elementos da lista A, refletindo a posição correta na lista ordenada.

Marcação de Posição

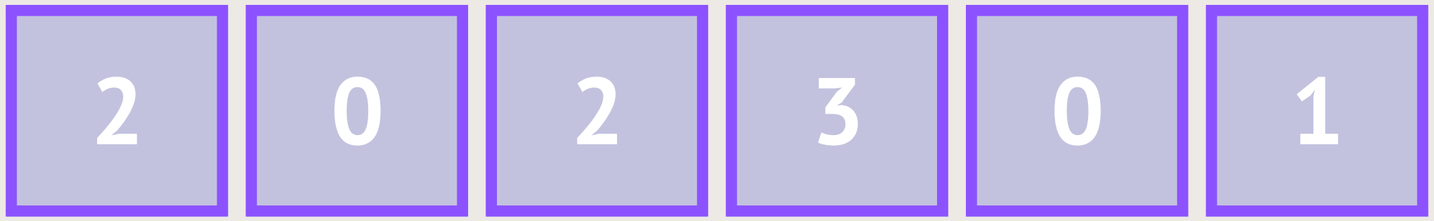


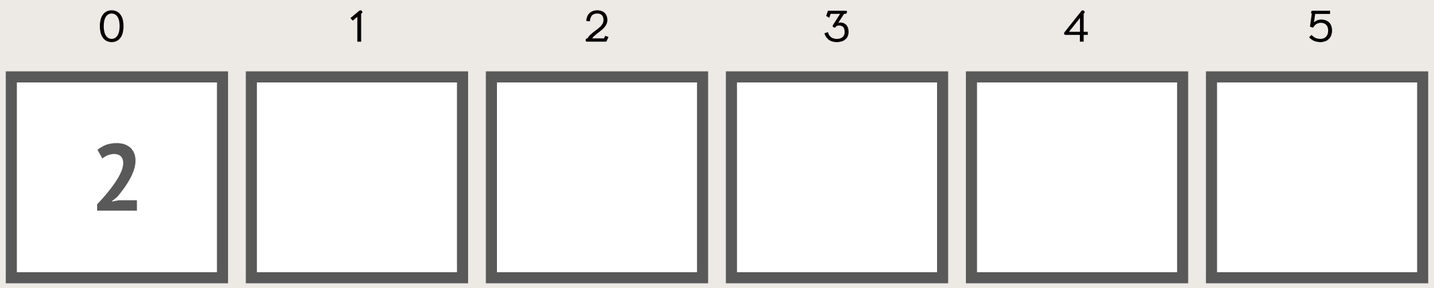


Marcação de Posição

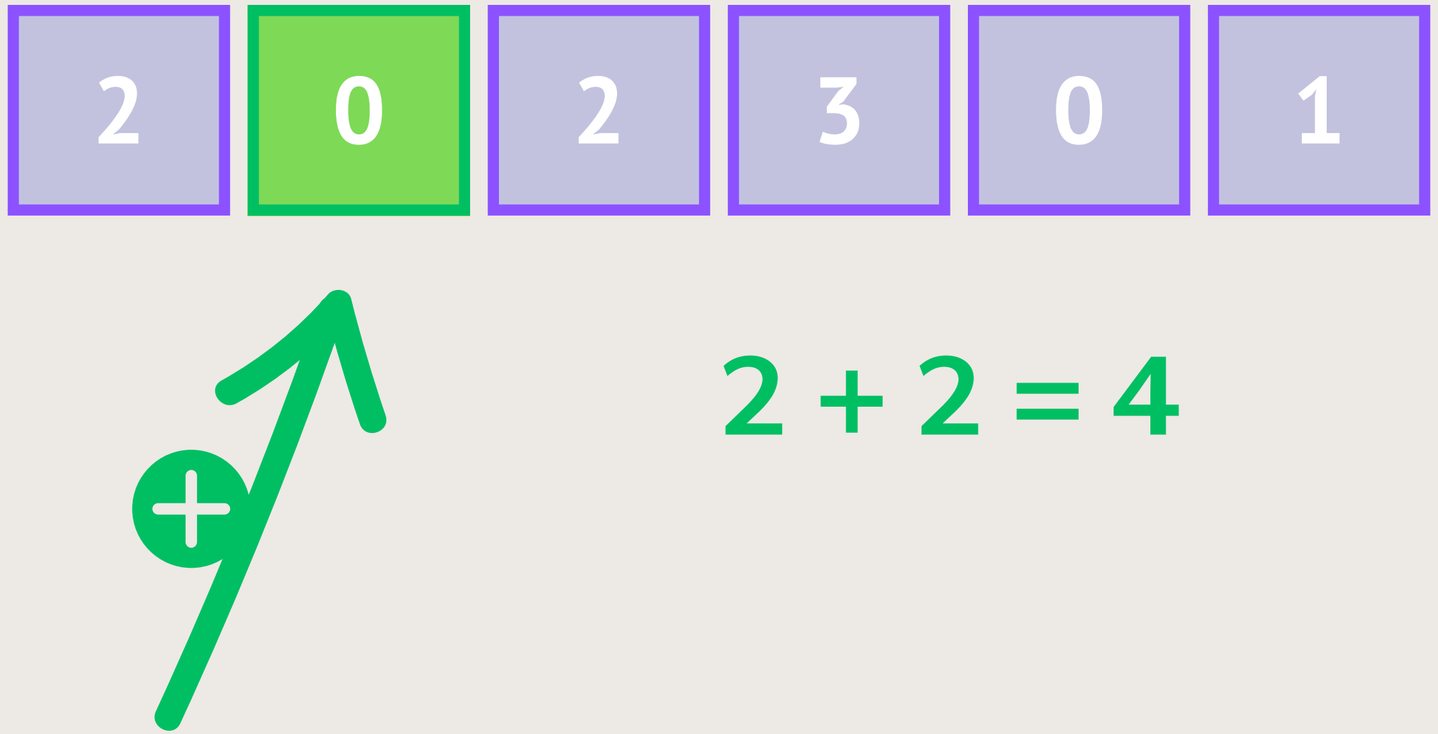


Marcação de Posição

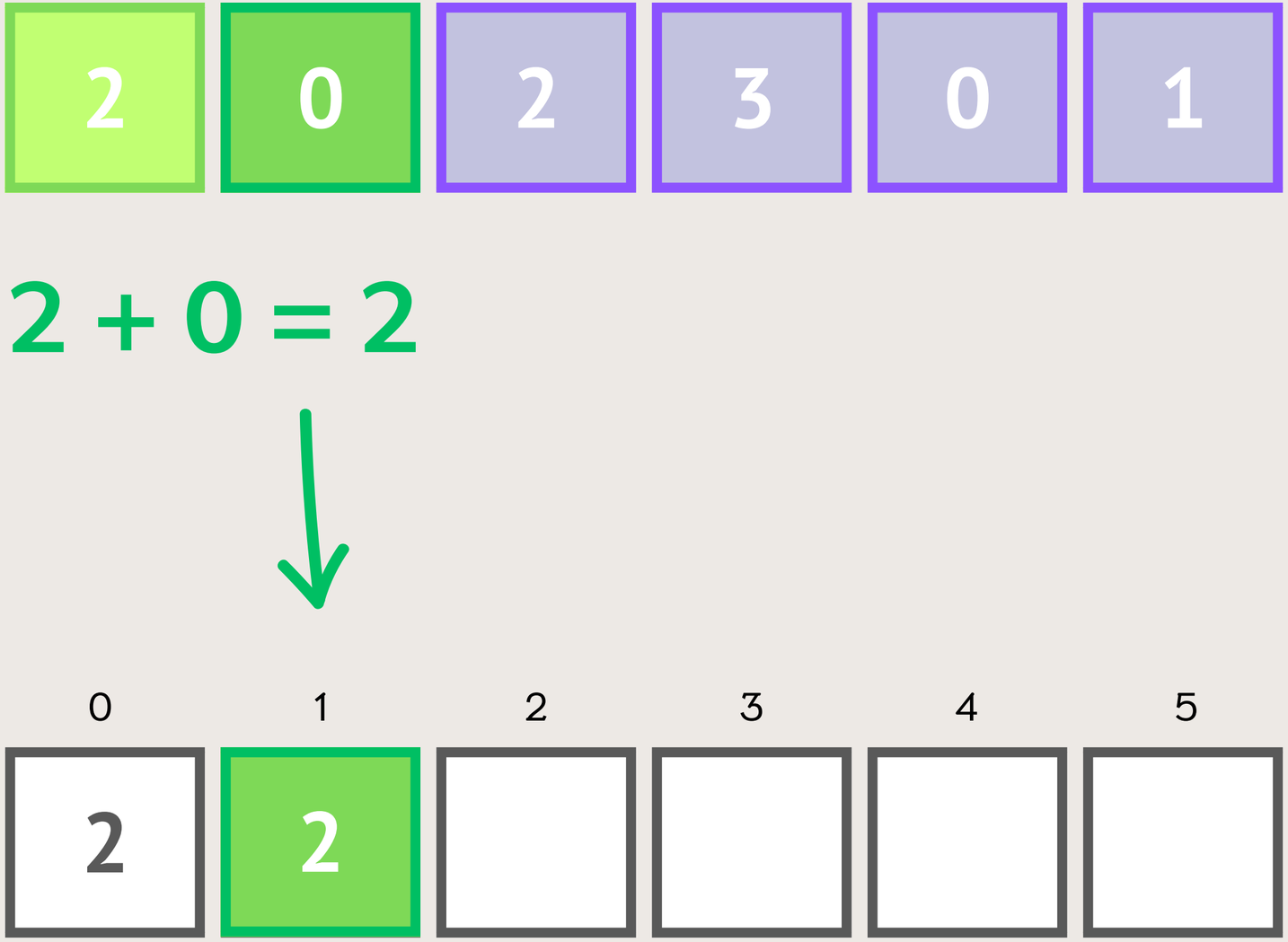




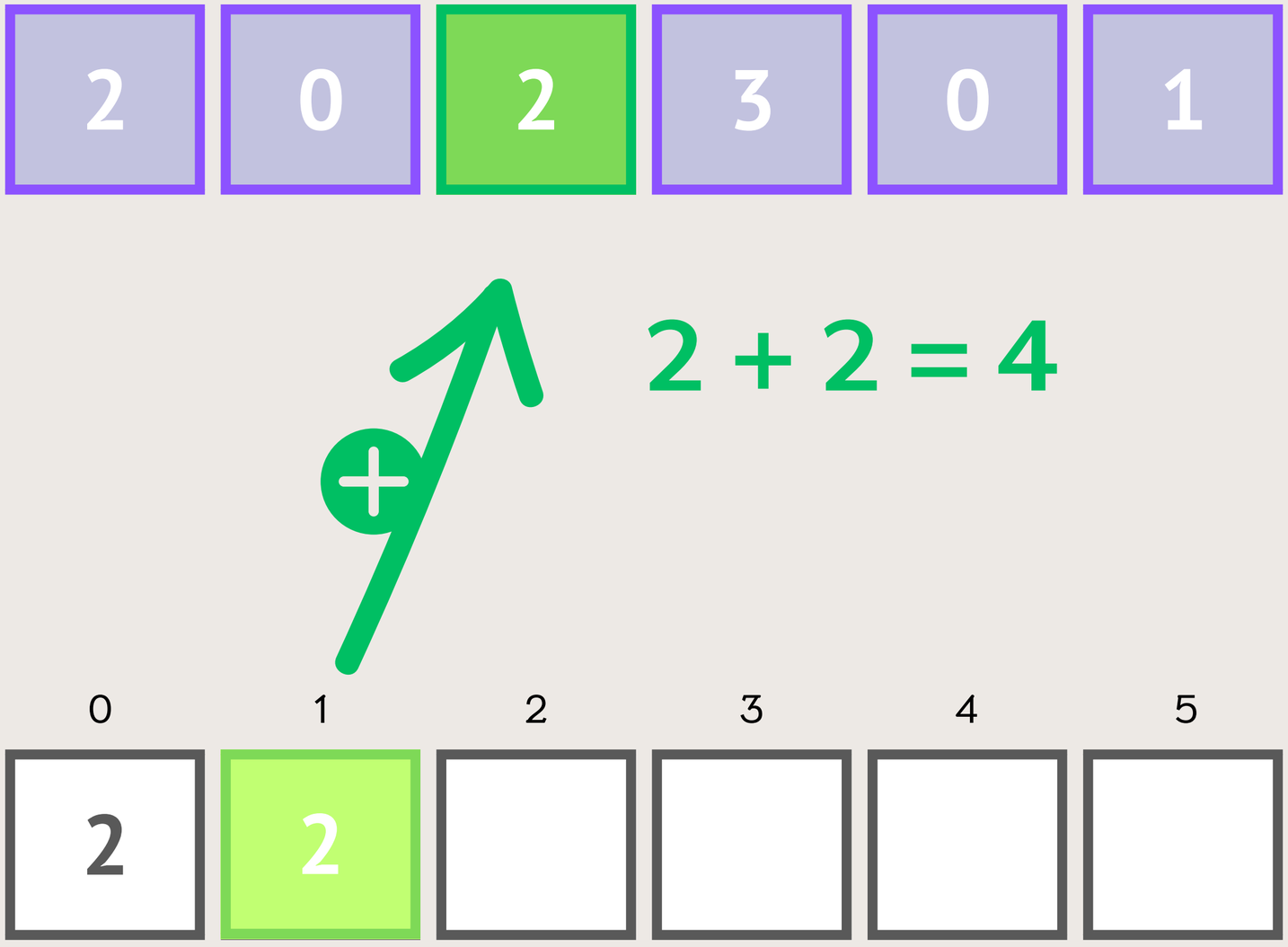
Marcação de Posição



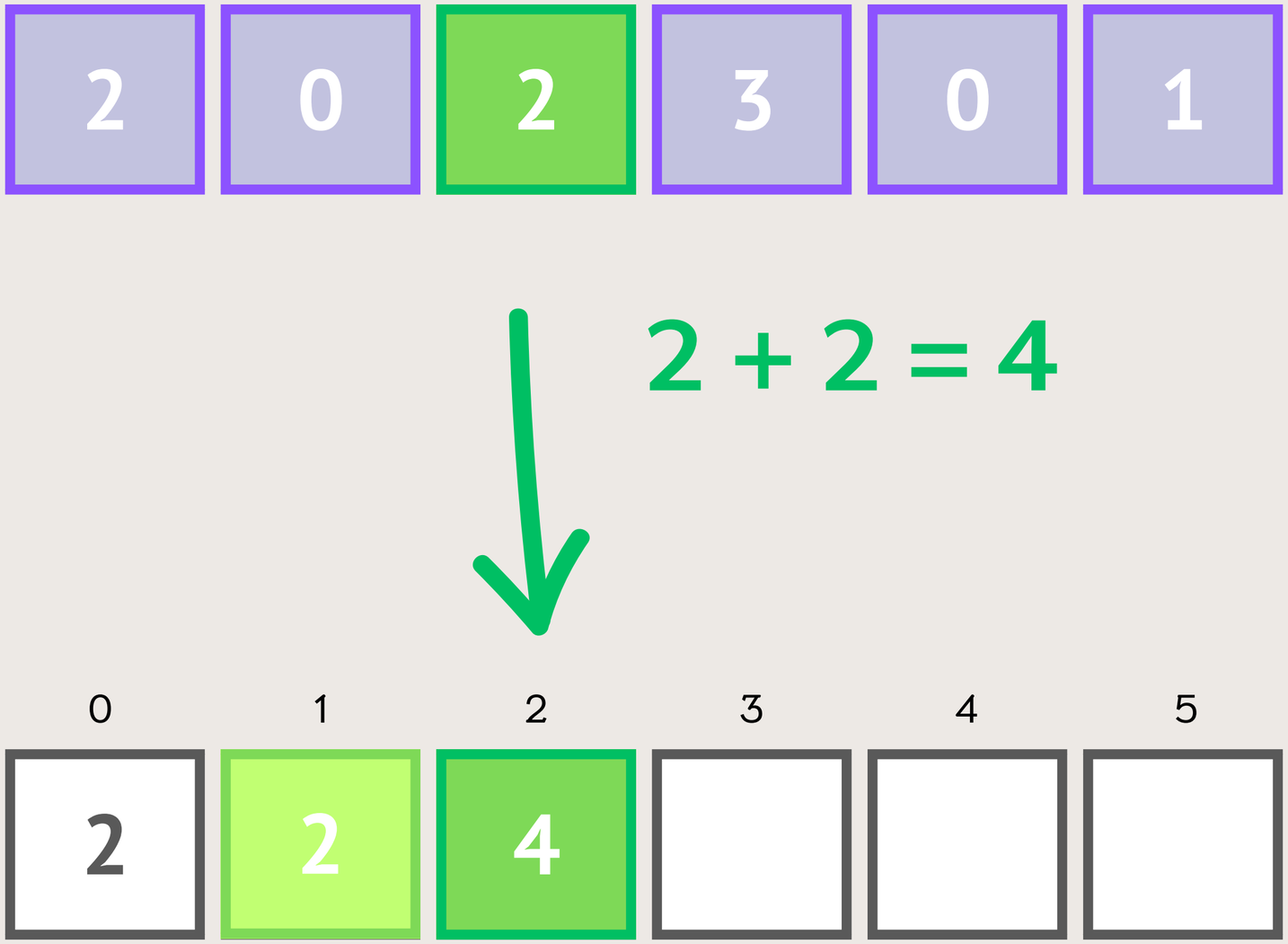
Marcação de Posição



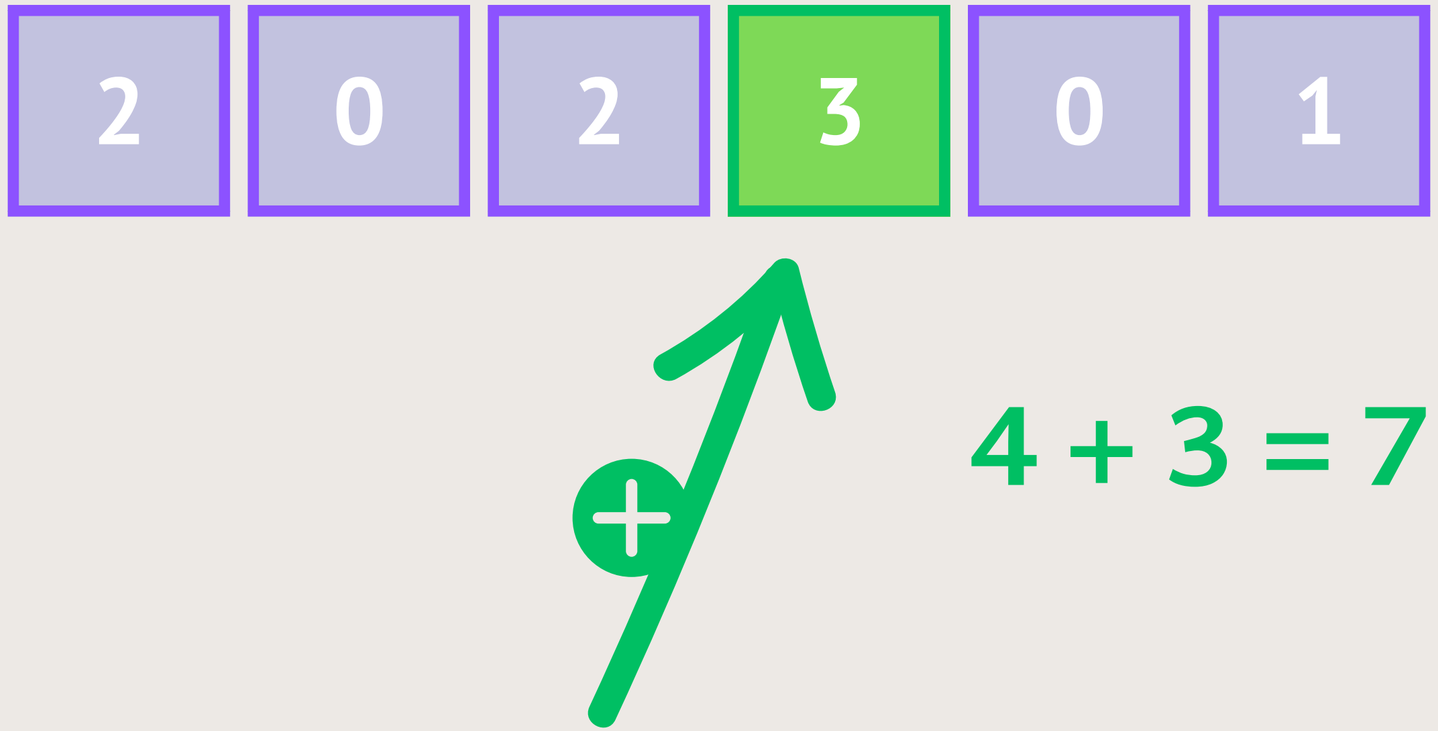
Marcação de Posição



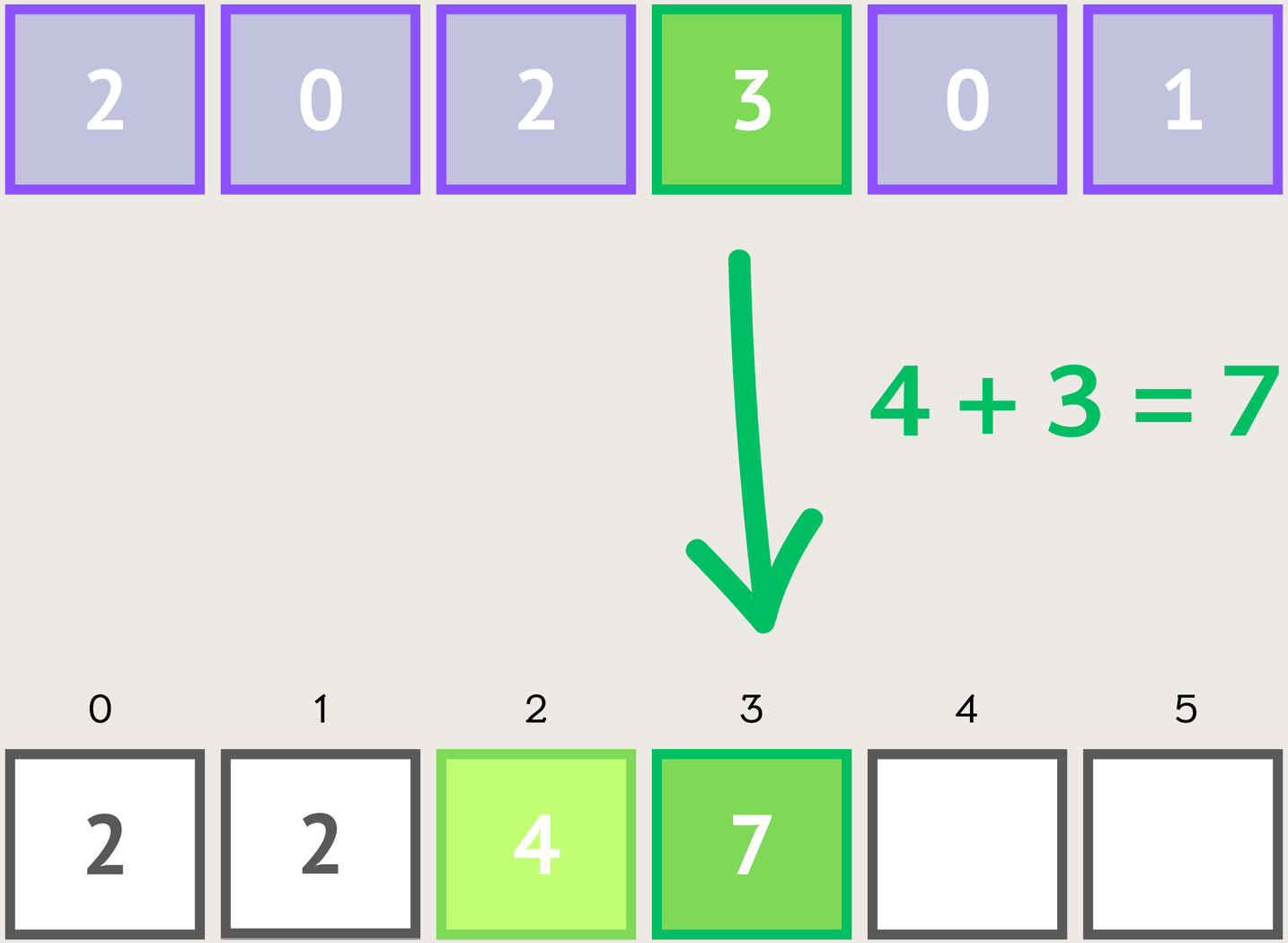
Marcação de Posição



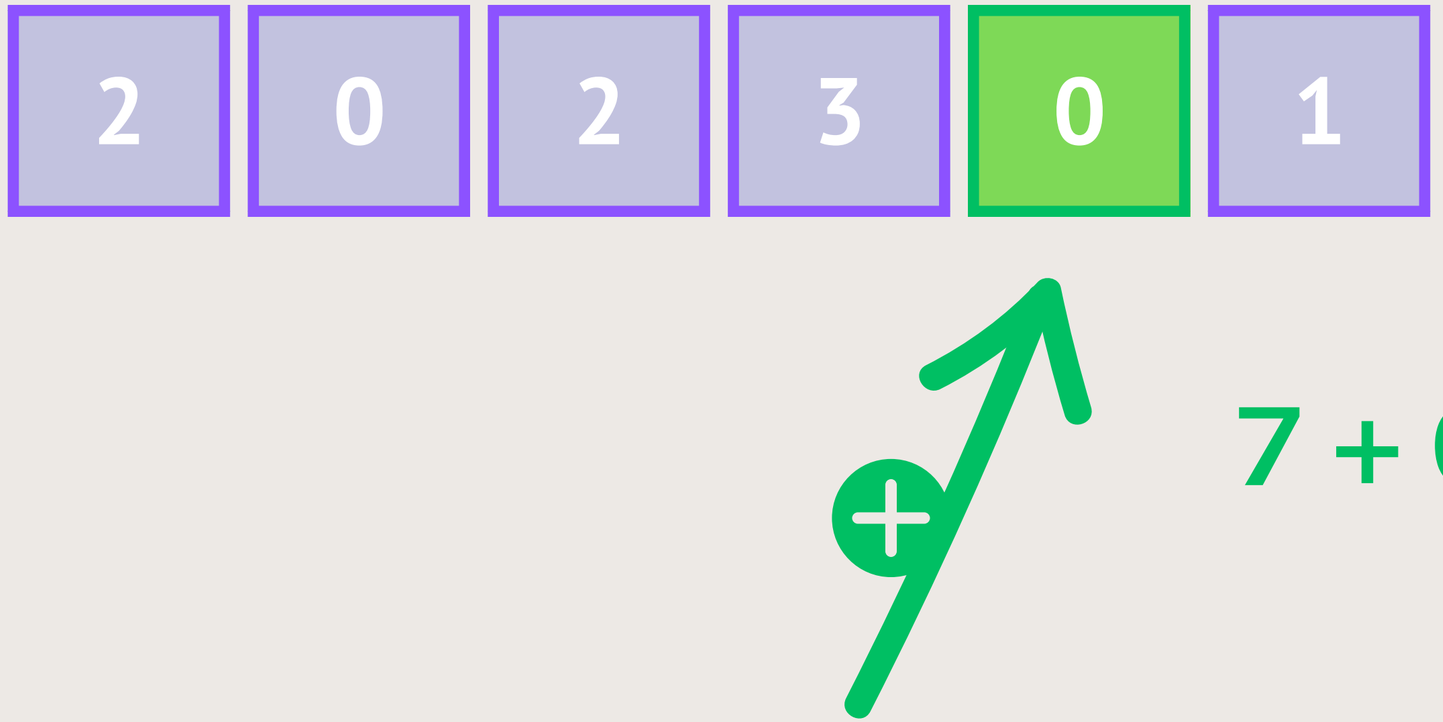
Marcação de Posição



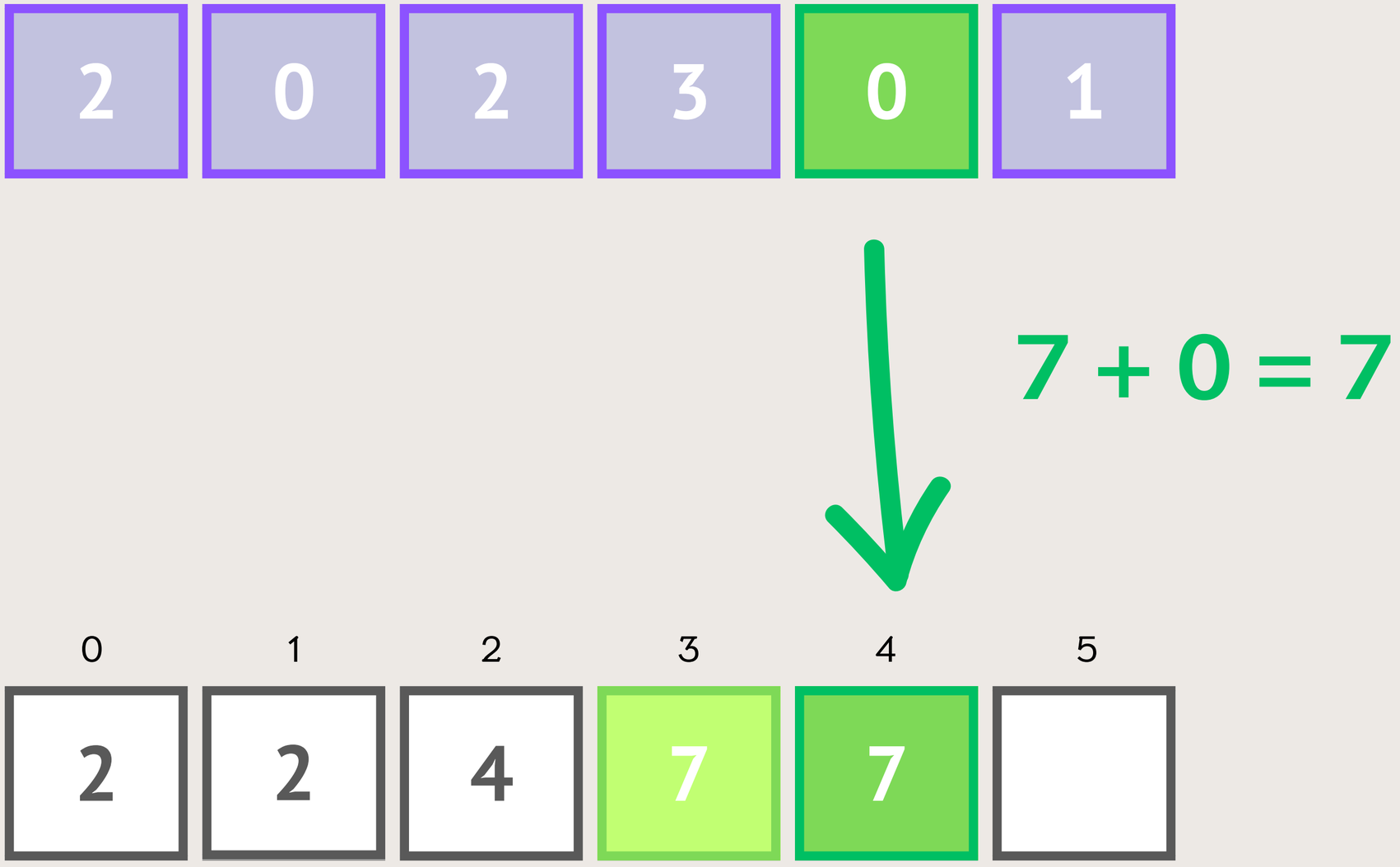
Marcação de Posição



Marcação de Posição



Marcação de Posição



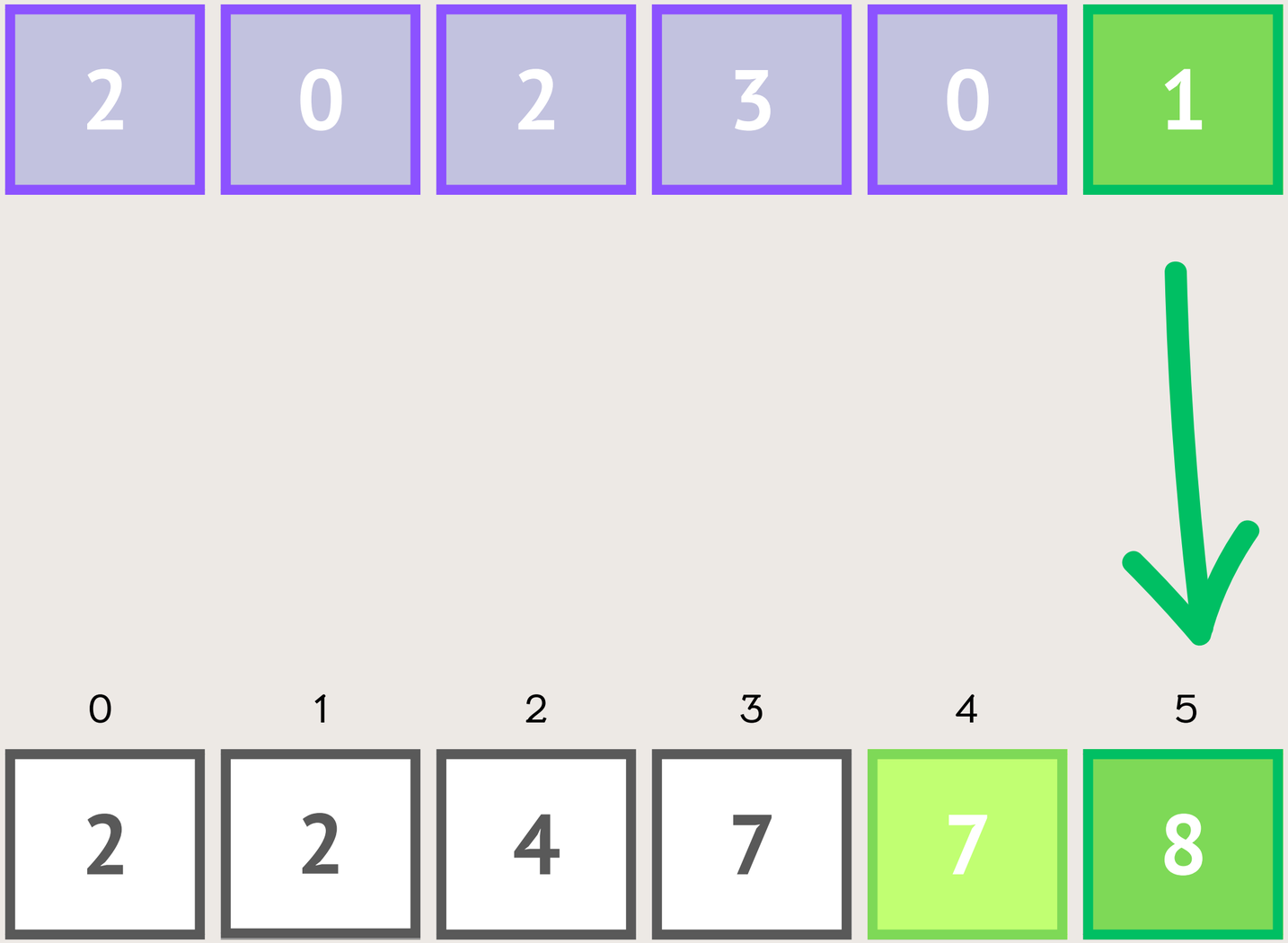
Marcação de Posição

2	0	2	3	0	1

7 + 1 = 8

0	1	2	3	4	5
2	2	4	7	7	

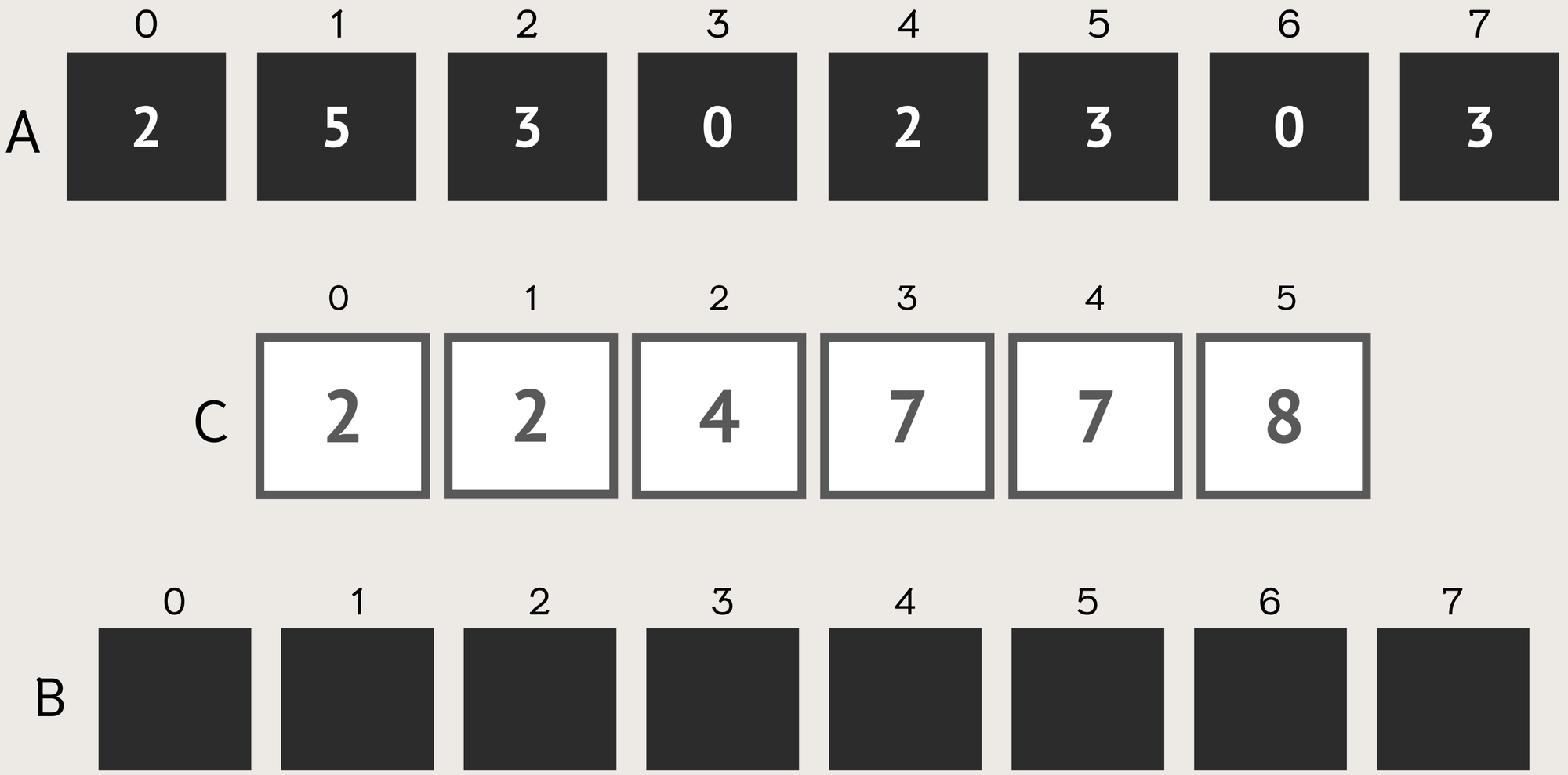
Marcação de Posição



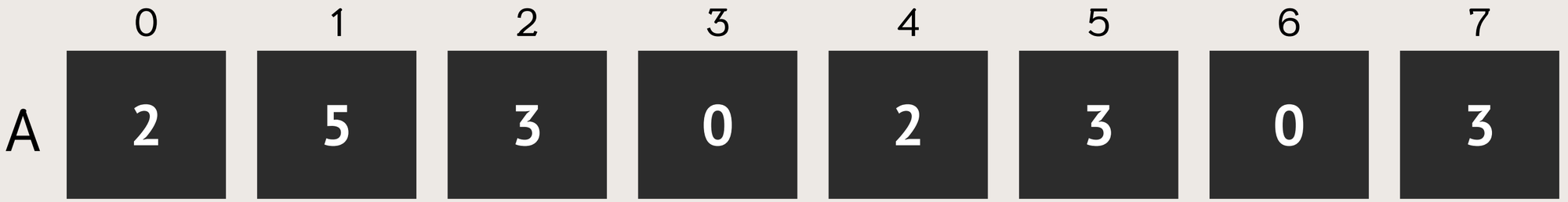
3. Ordenação

Nessa última etapa, ocorre a ordenação dos elementos de A utilizando os dados de C. A ordenação surge quando é feito o percurso da última posição para a primeira de A, e cada valor do elemento em percurso será utilizado para buscar a posição determinante em C que será determinada a posição que o elemento de A ficará em B.

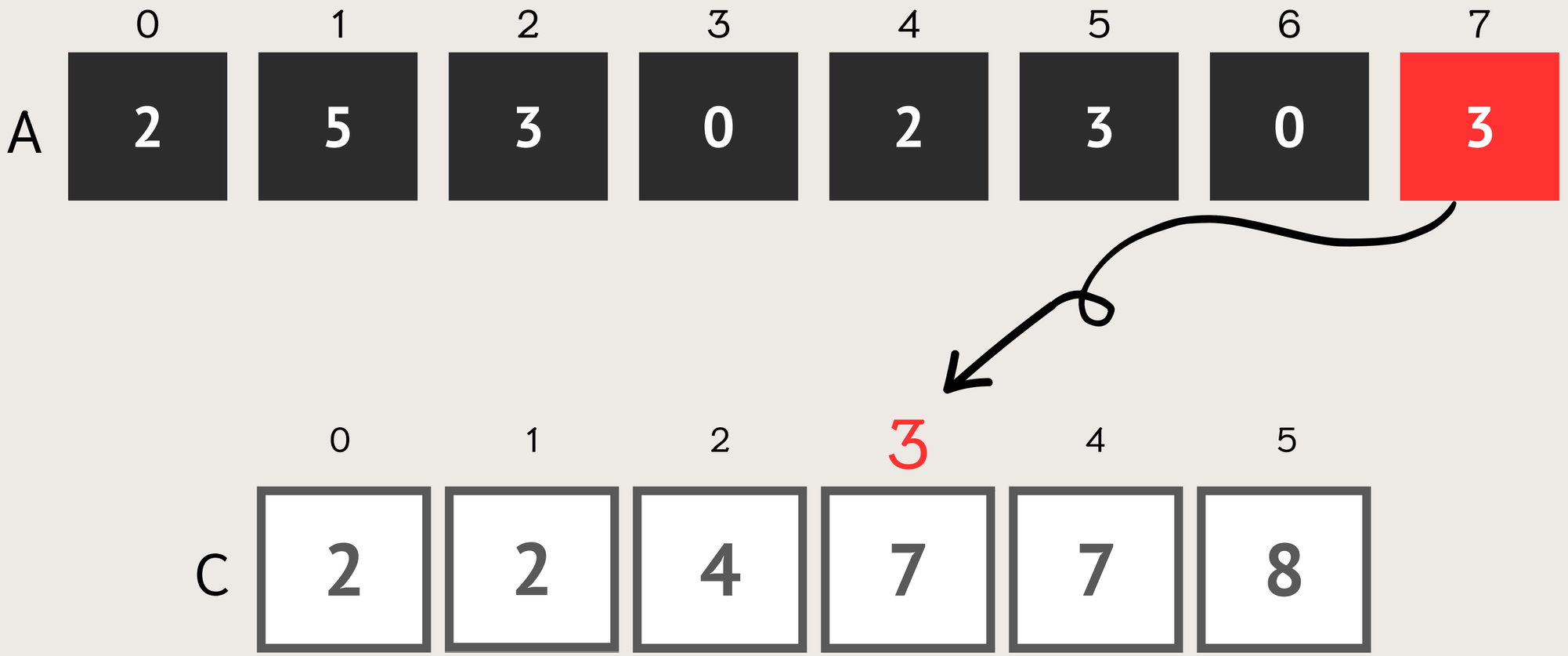
Ordenação



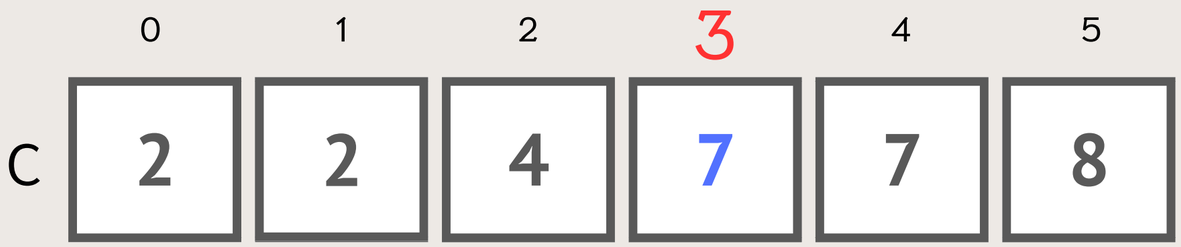
Ordenação



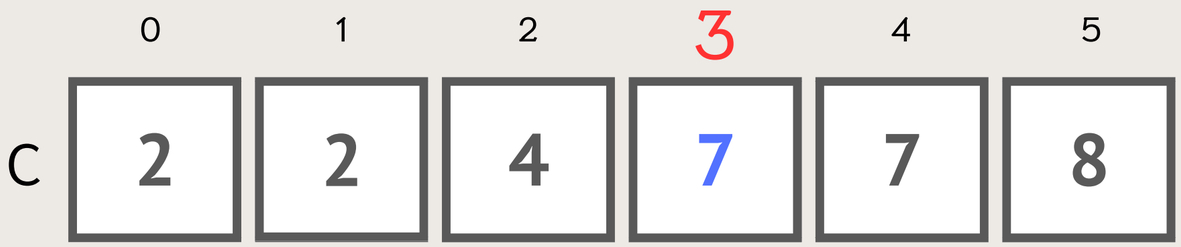
Ordenação



Ordenação

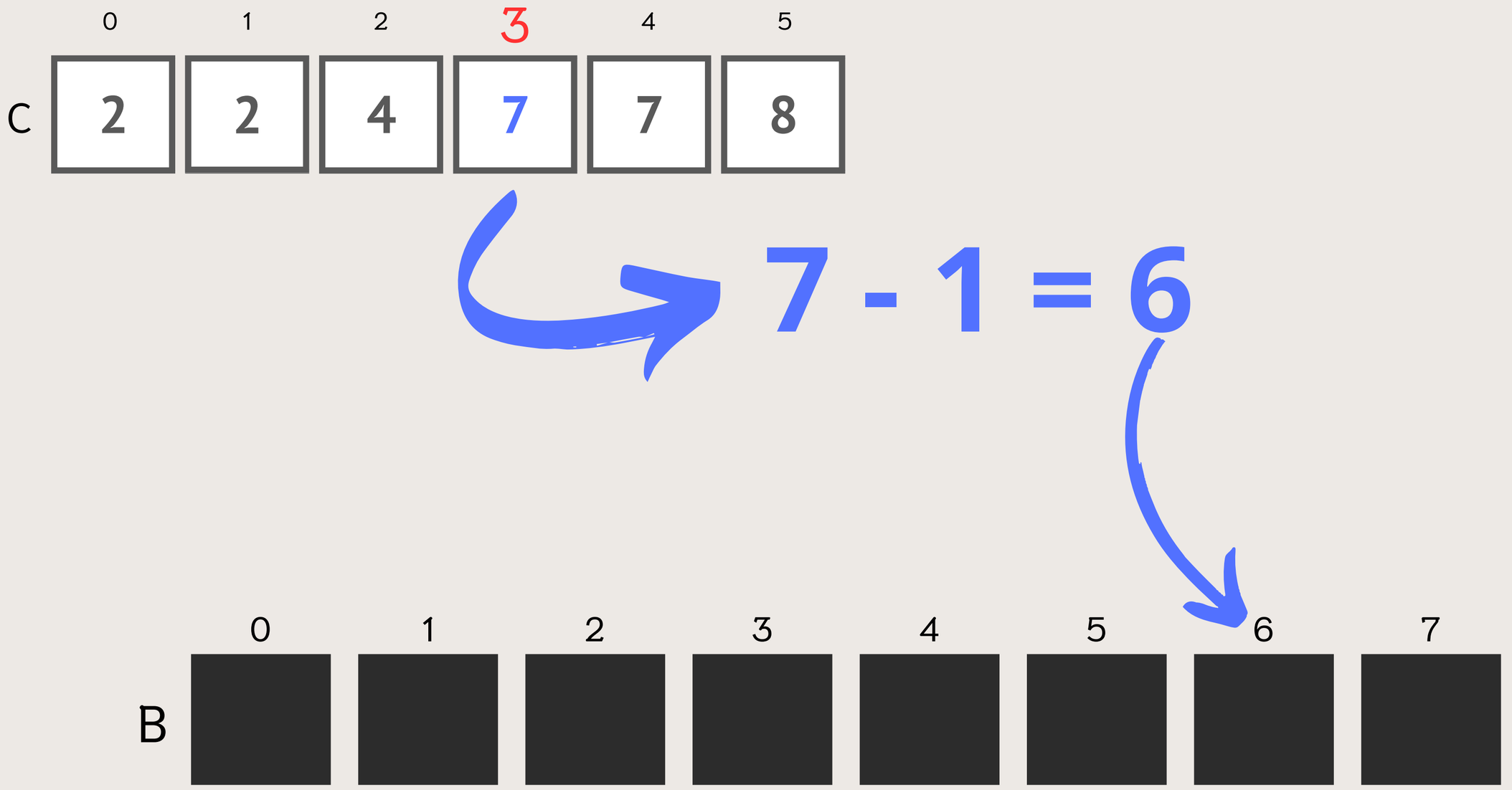


Ordenação

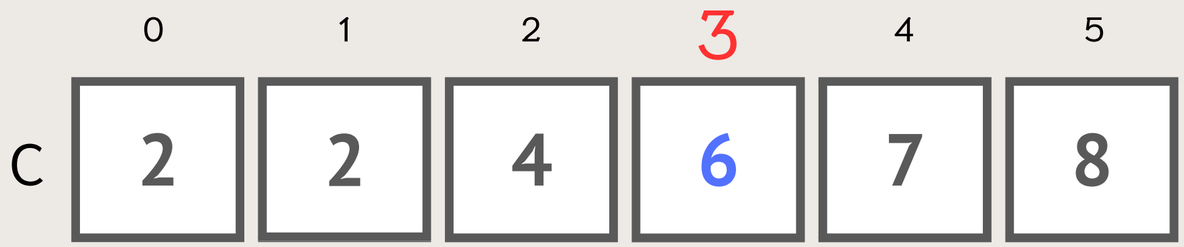


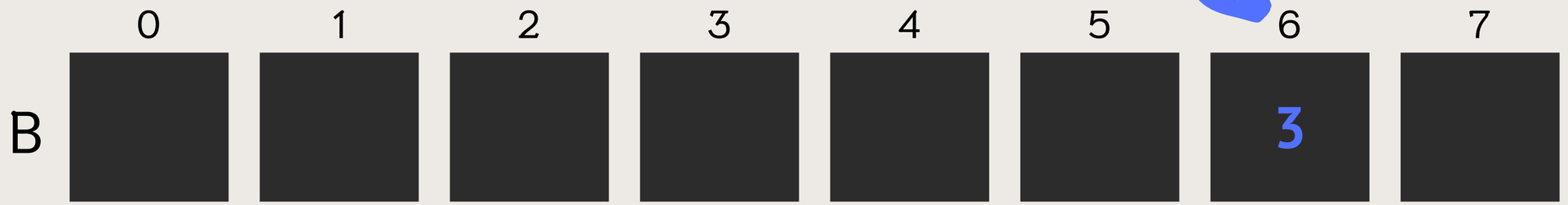
7 - 1 = 6

Ordenação

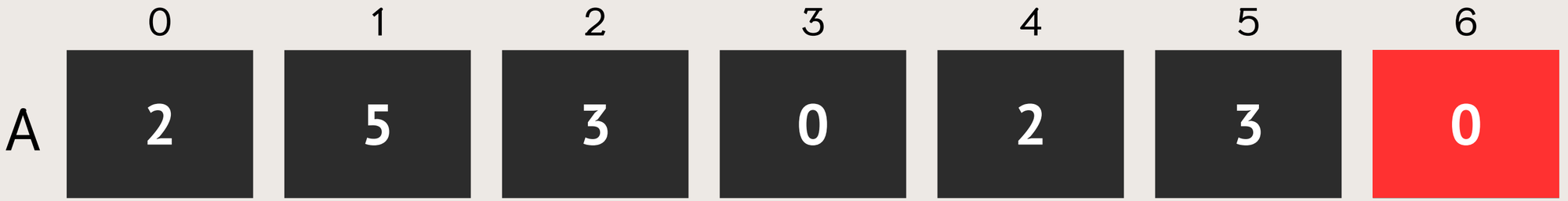


Ordenação

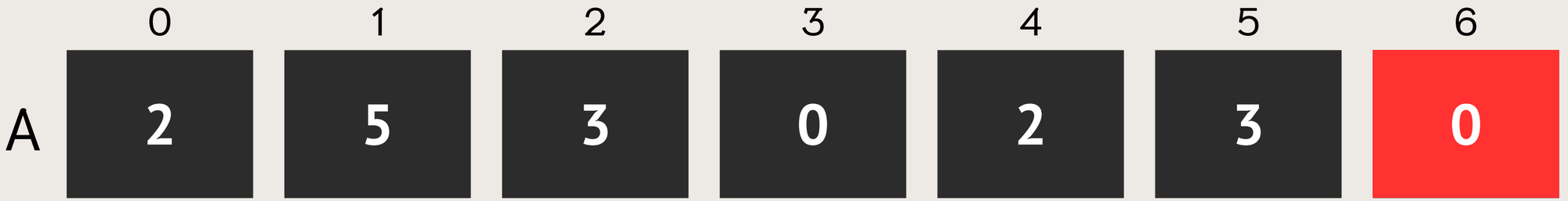


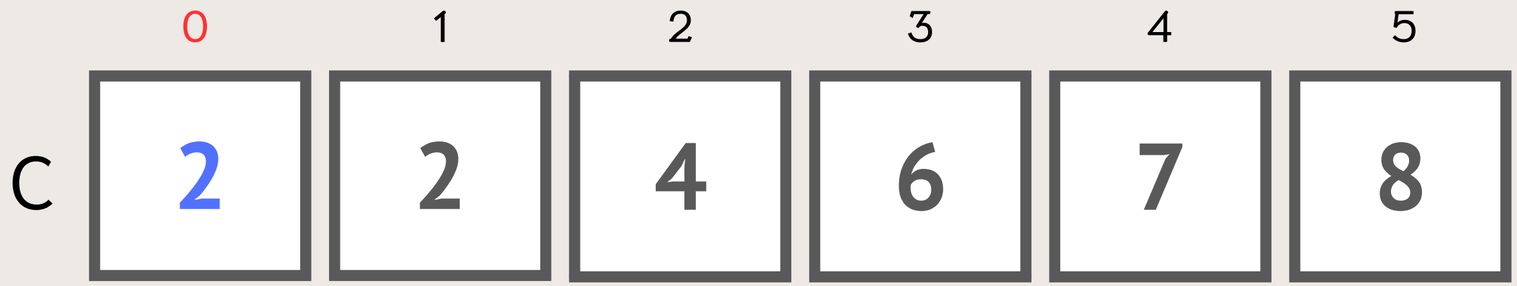


Ordenação

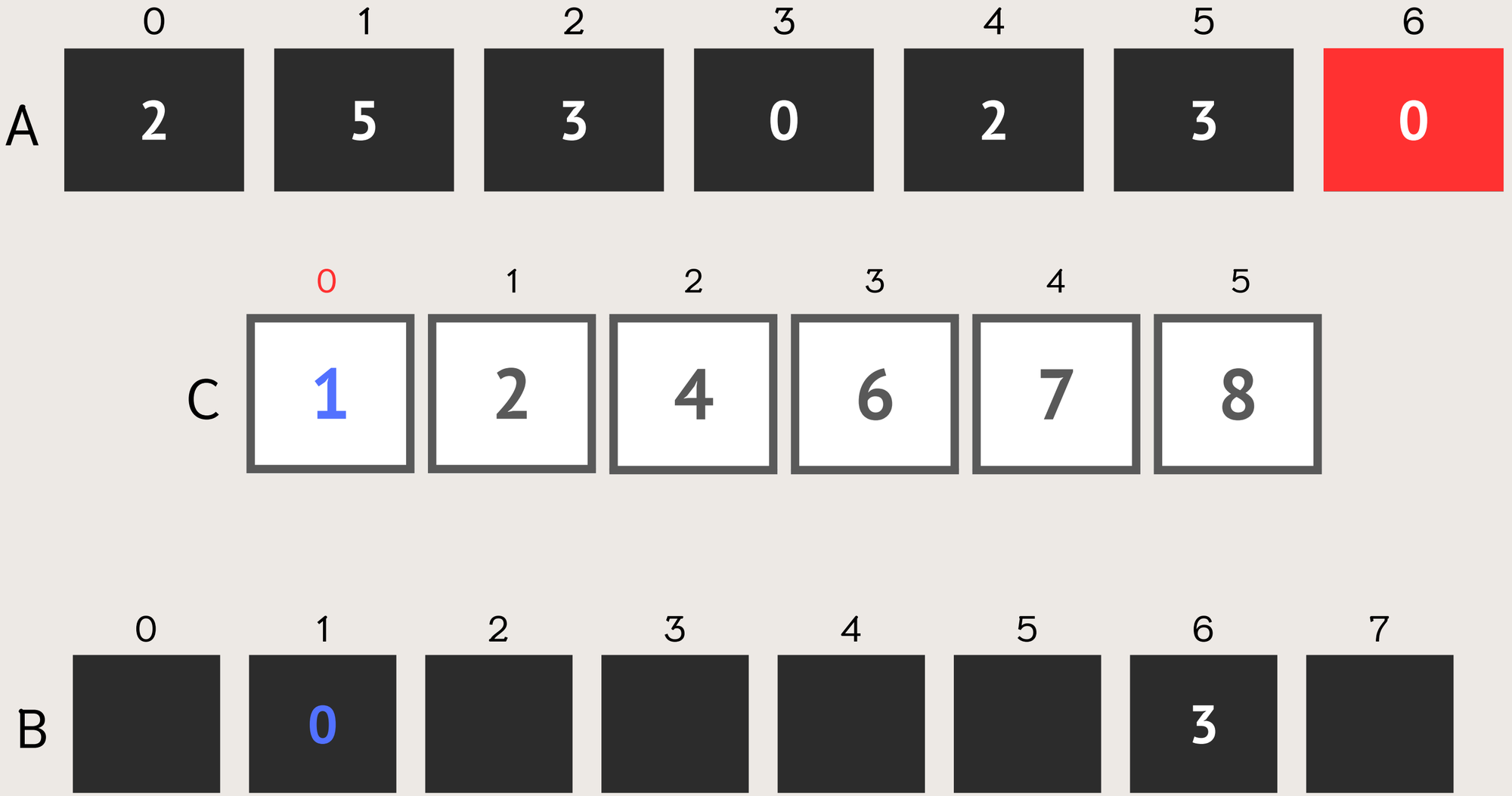


Ordenação

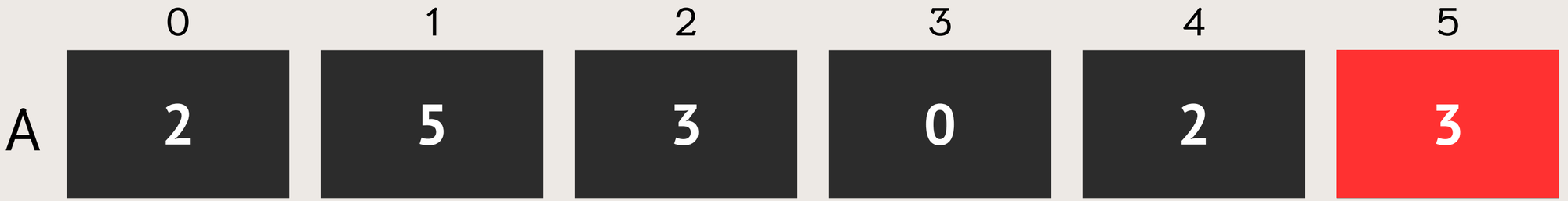




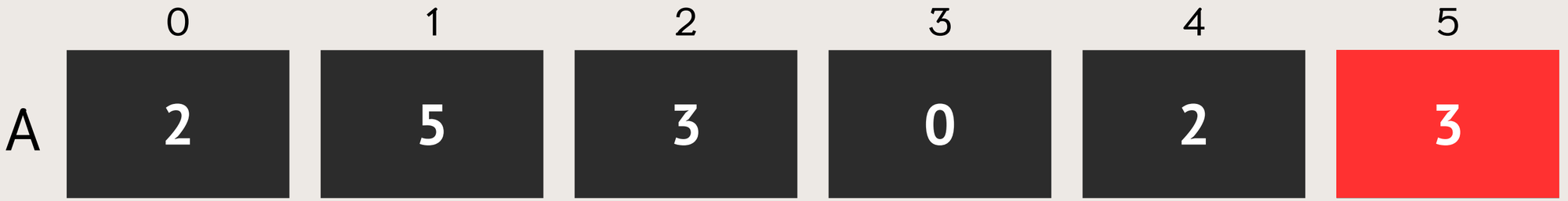
Ordenação

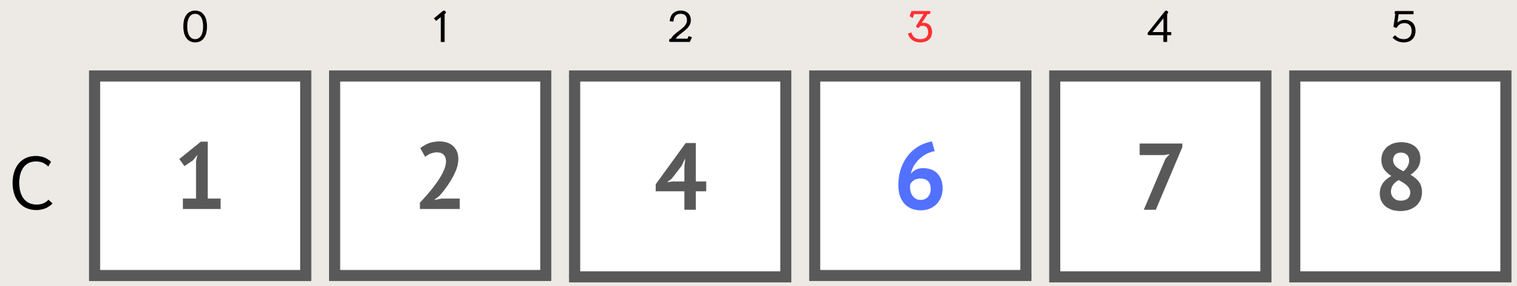


Ordenação

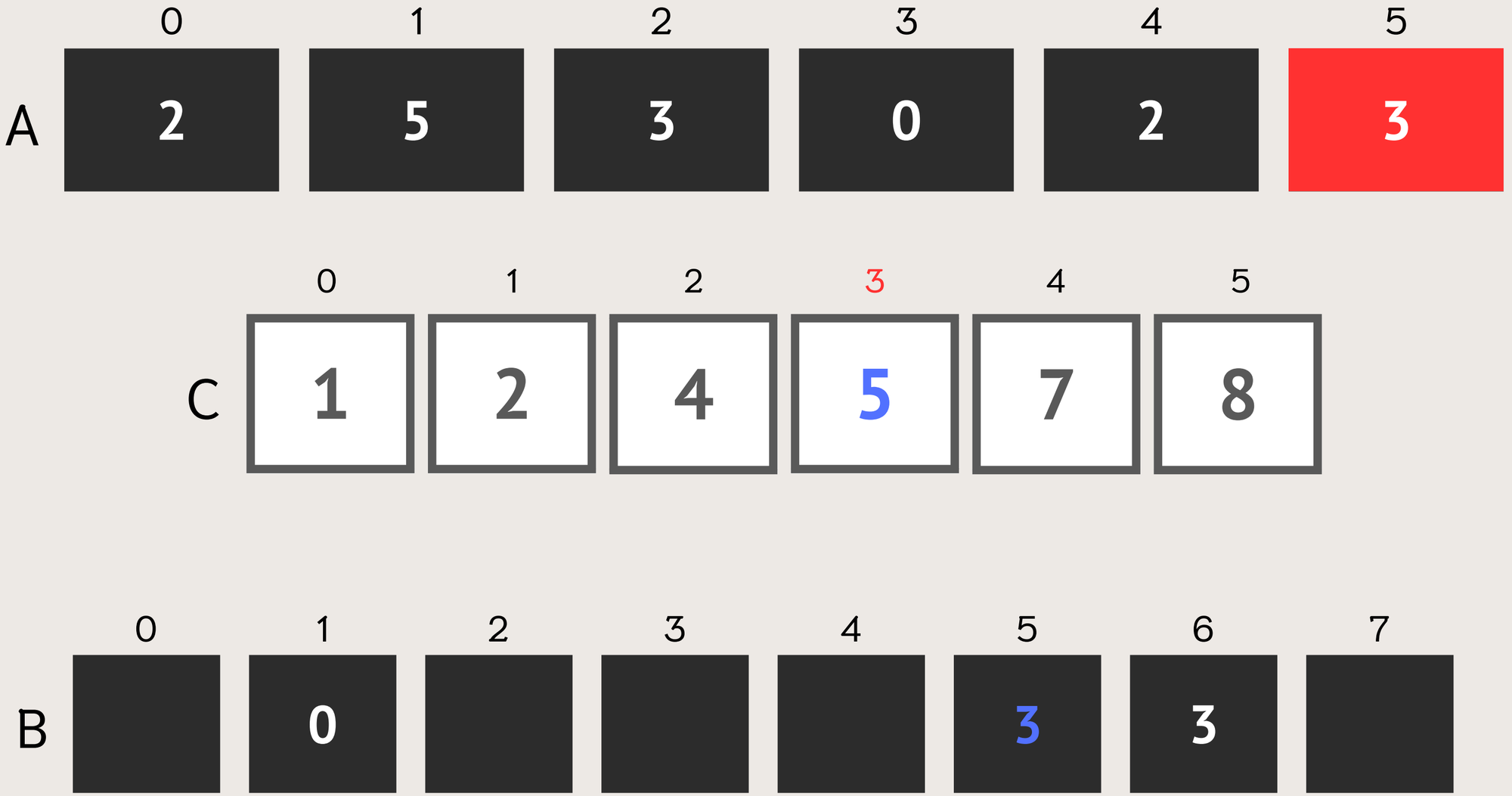


Ordenação

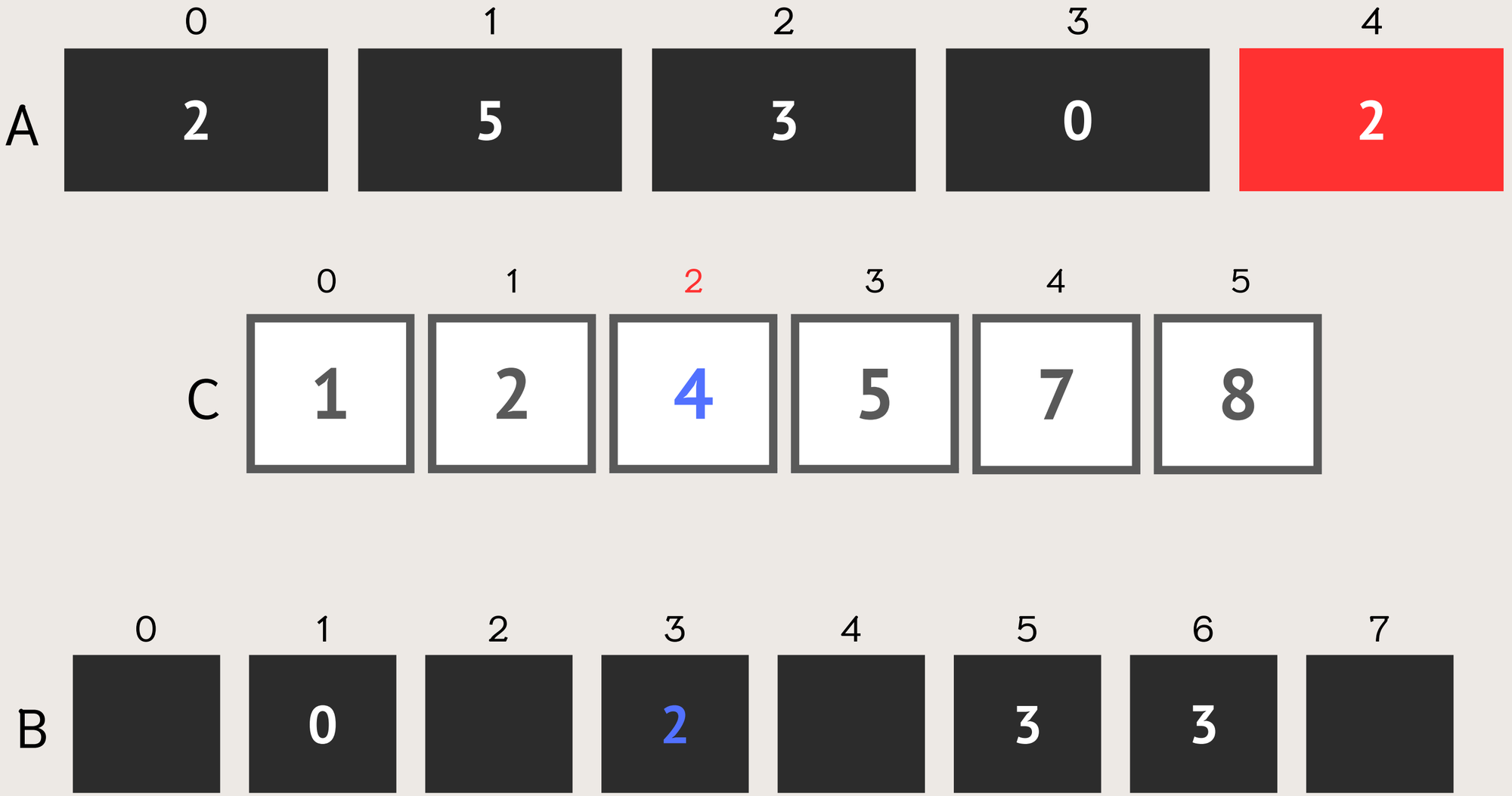




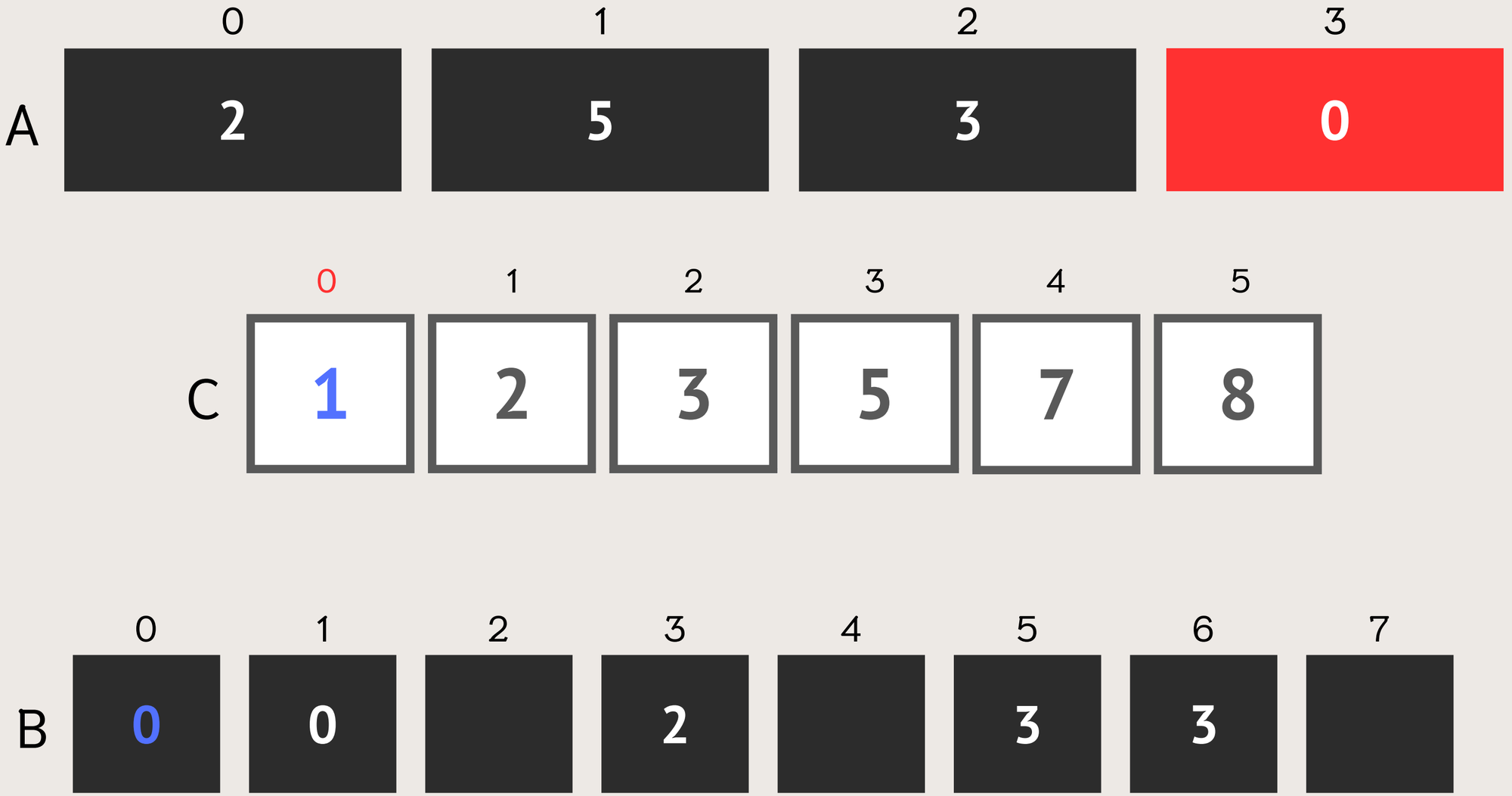
Ordenação



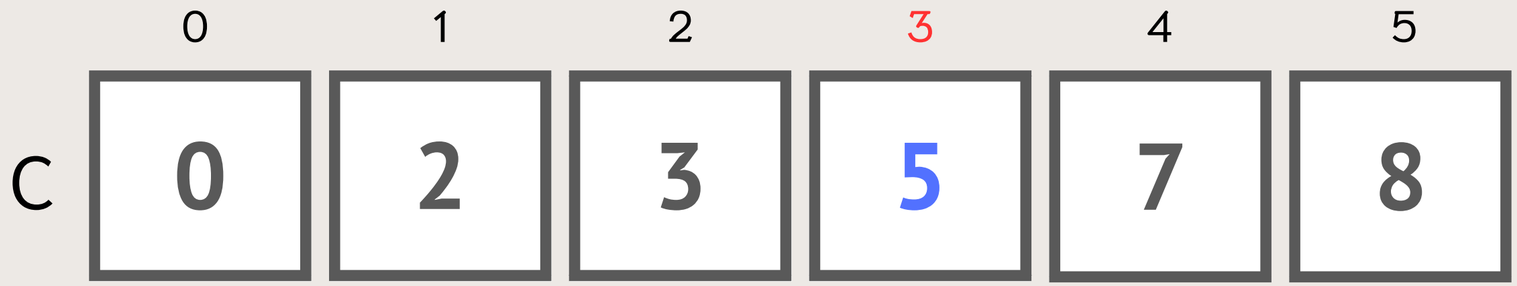
Ordenação

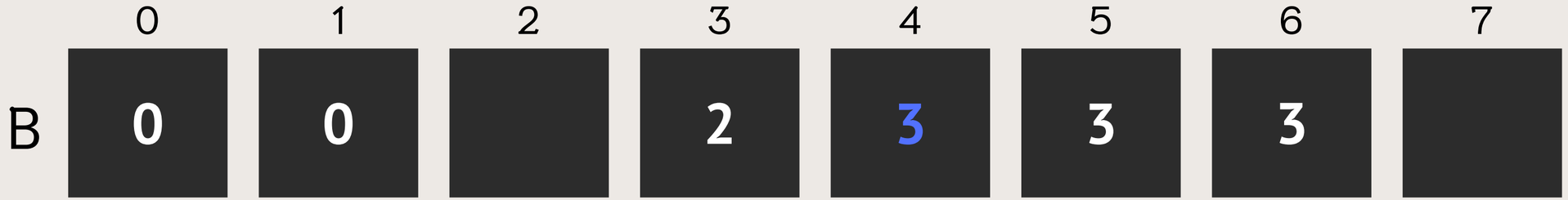


Ordenação

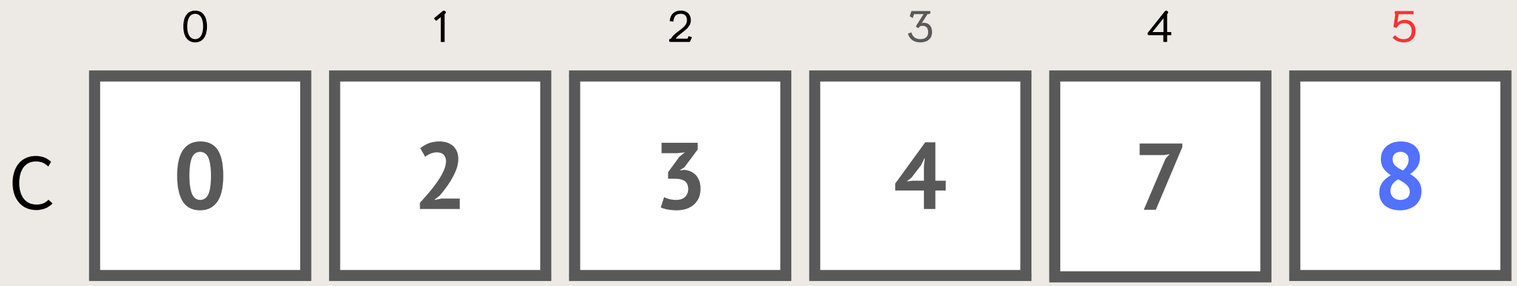


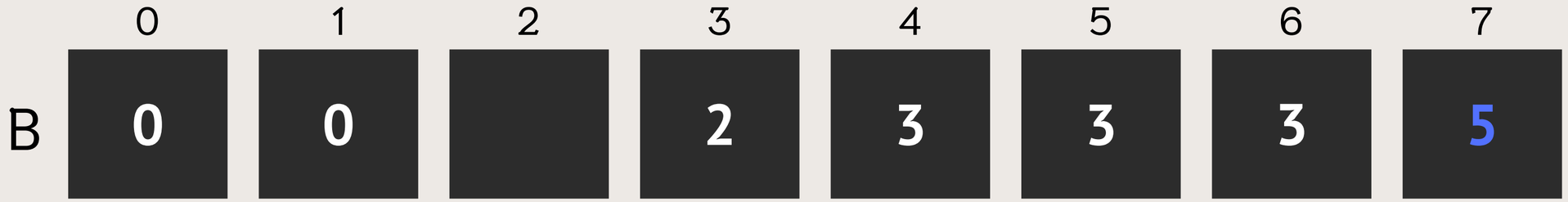
Ordenação



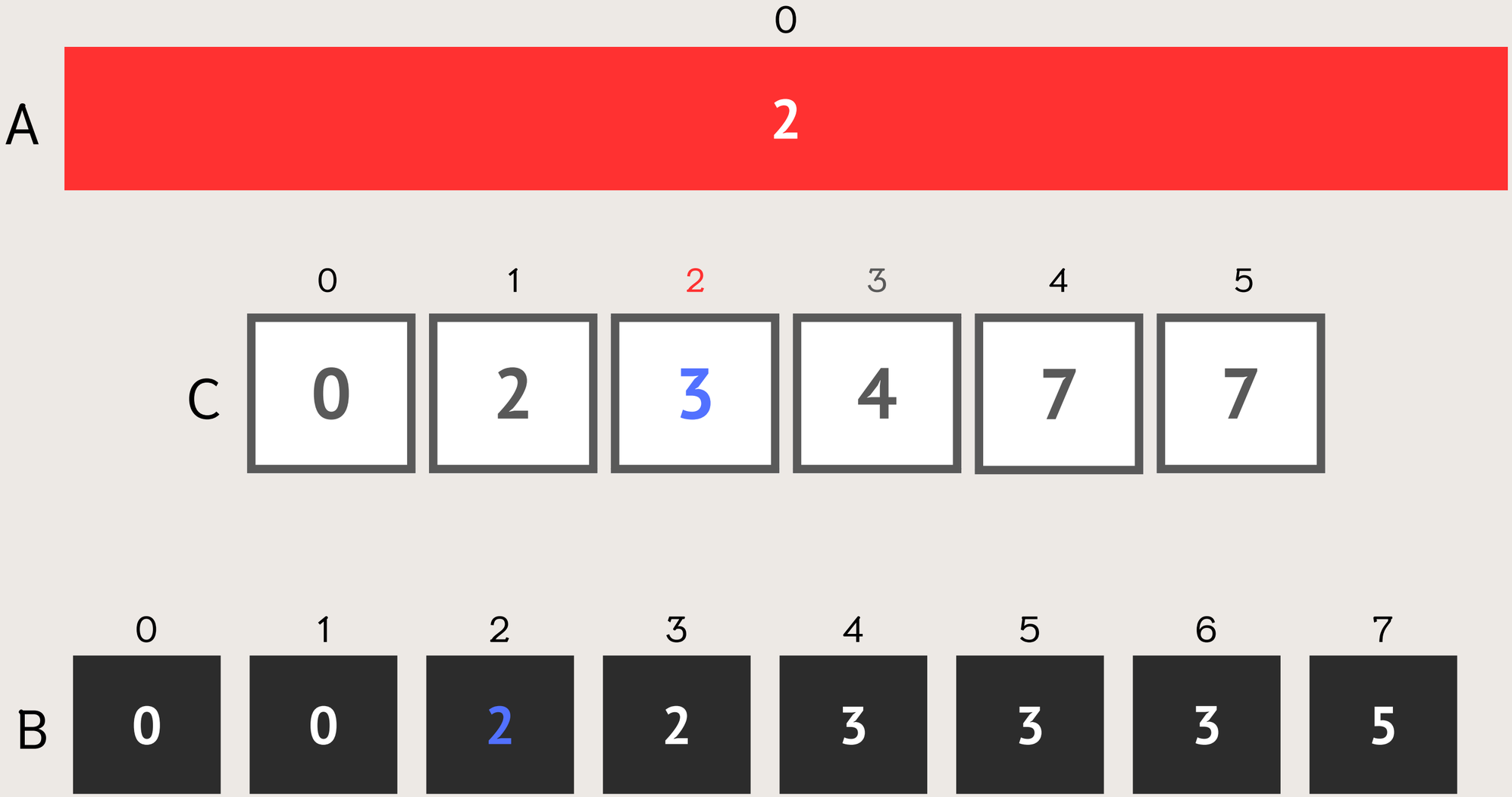


Ordenação





Ordenação



Ordenação

Pigeonhole Sort

Ordenação pela Frequência dos Elementos

• O que é o Pigeonhole Sort ?

Pigeonhole Sort é uma variação do Counting Sort, com a diferença de este algoritmo de ordenação é baseado no conceito dos "buracos de pombo", uma técnica eficiente para ordenar listas onde o número de elementos (n) e o intervalo de valores possíveis (N) são aproximadamente do mesmo tamanho.

• Como funciona?

Primeiro, identifica-se o valor mínimo e máximo da lista. Em seguida, cria-se um vetor de "buracos" com tamanho igual à diferença entre o valor máximo e mínimo, mais um. Cada elemento da lista é colocado no "buraco" correspondente ao seu valor. Por fim, os elementos são recolhidos dos "buracos" em ordem, resultando em uma lista ordenada.

Sua implementação pode ser separada em 3 etapas:

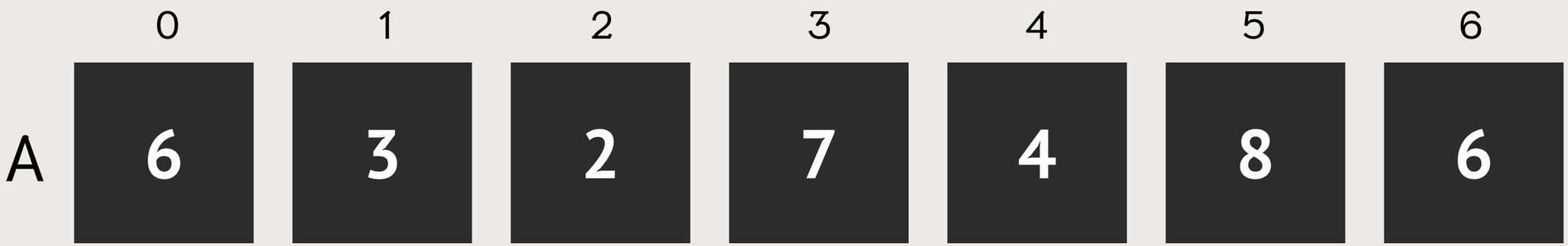
1. Determinar o tamanho
2. Criar os Pigeonholes
3. Coletar e Converter

1. Determinar o Tamanho

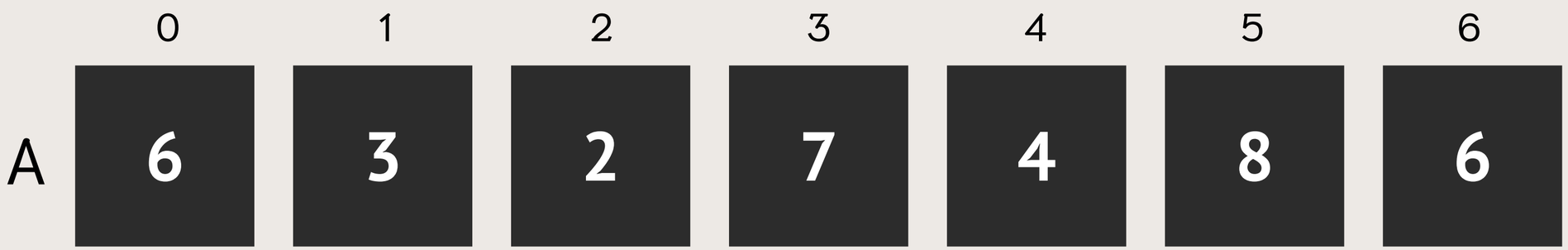
Nesta etapa, o algoritmo percorre a lista de valores de entrada e encontra o maior (max) e menor (min) valor, utilizando-os como base para criar um vetor com n posições (intervalo entre os valores). A lógica para a definição do tamanho do vetor é Max - Min + 1.

Determinar o Tamanho

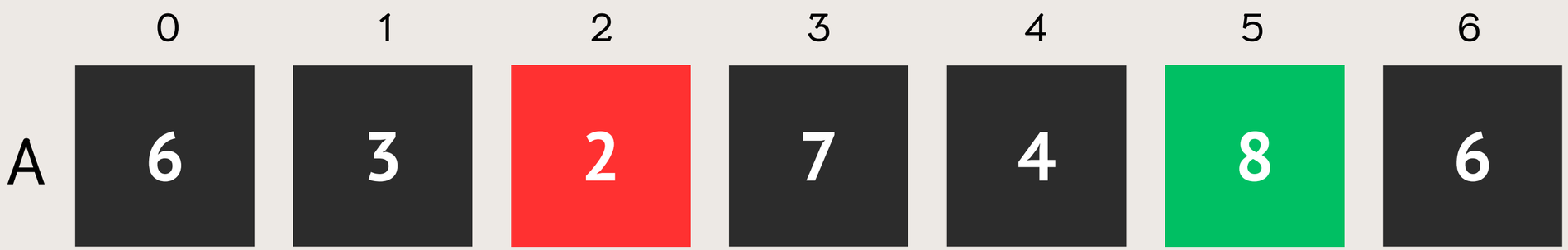
Exemplo de entrada:



Determinar o Tamanho



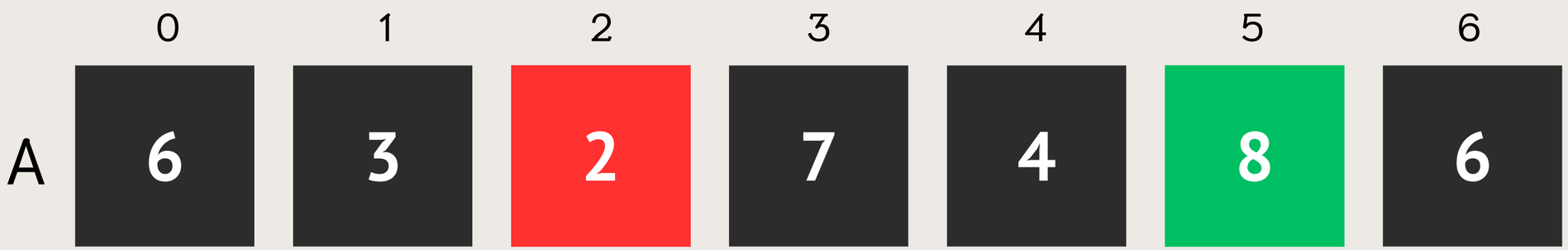
Determinar o Tamanho



Max = 8

Min = 2

Determinar o Tamanho



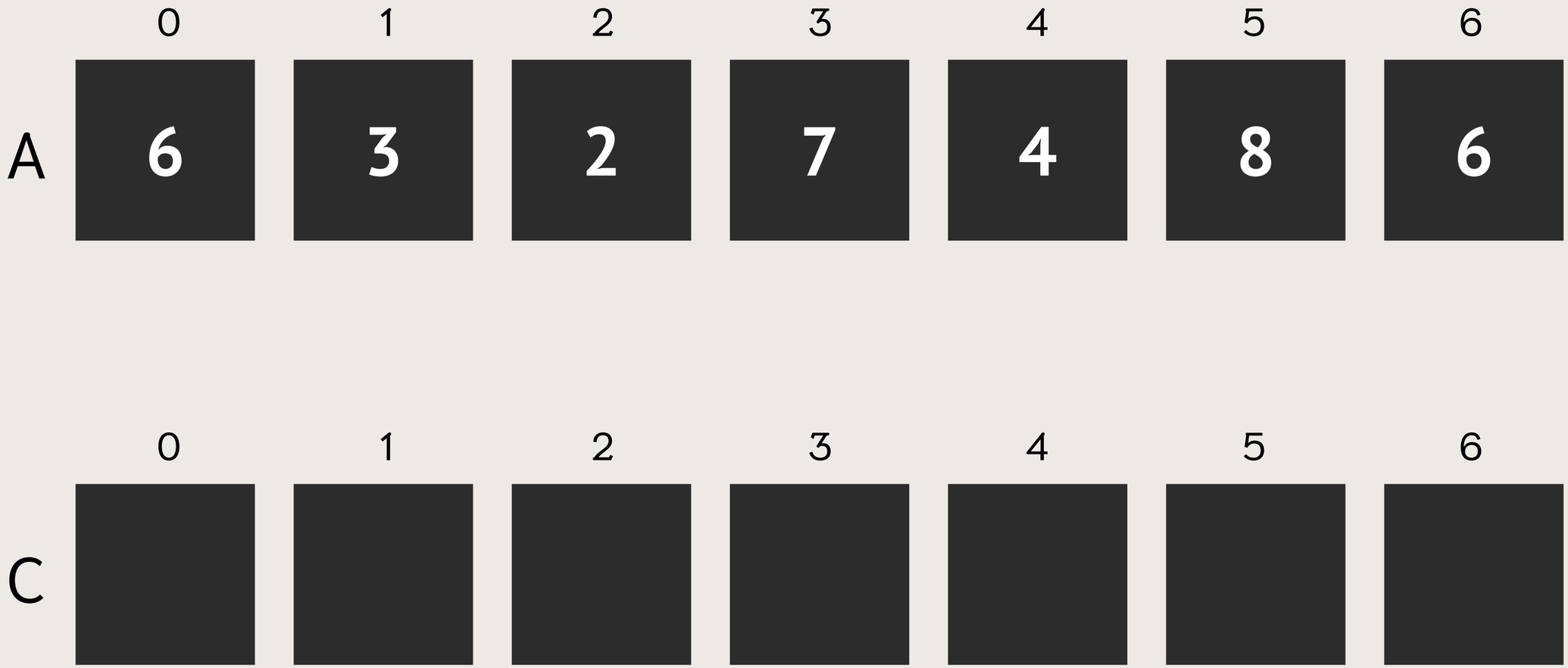
8 - 2 + 1 = 7

O vetor terá 7 posições

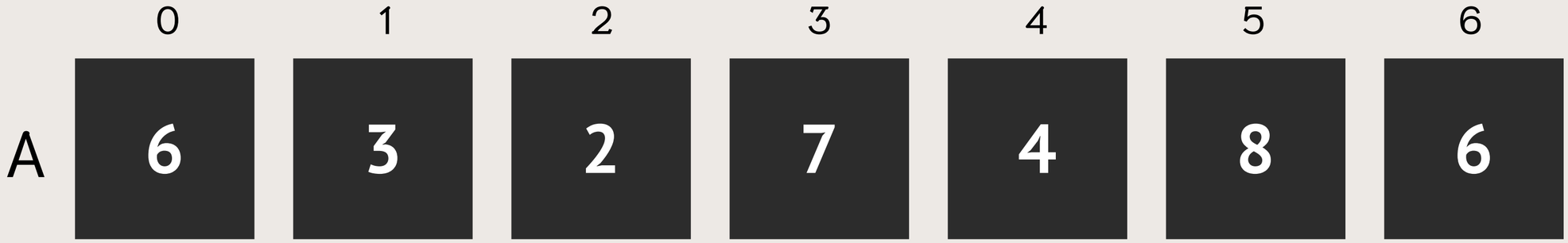
2. Criar os Pigeonholes

Crie um array de pigeonholes (buracos de pombo). Cada índice deste array representará um valor possível na lista original. O tamanho do array de pigeonholes será igual ao intervalo (range) determinado no passo anterior.

Criar os pigeonholes



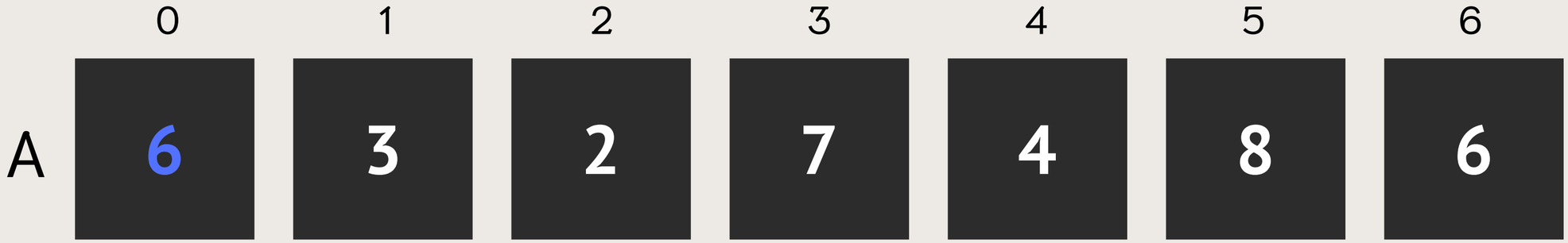
Criar os pigeonholes



Elemento - Mínimo = Índice

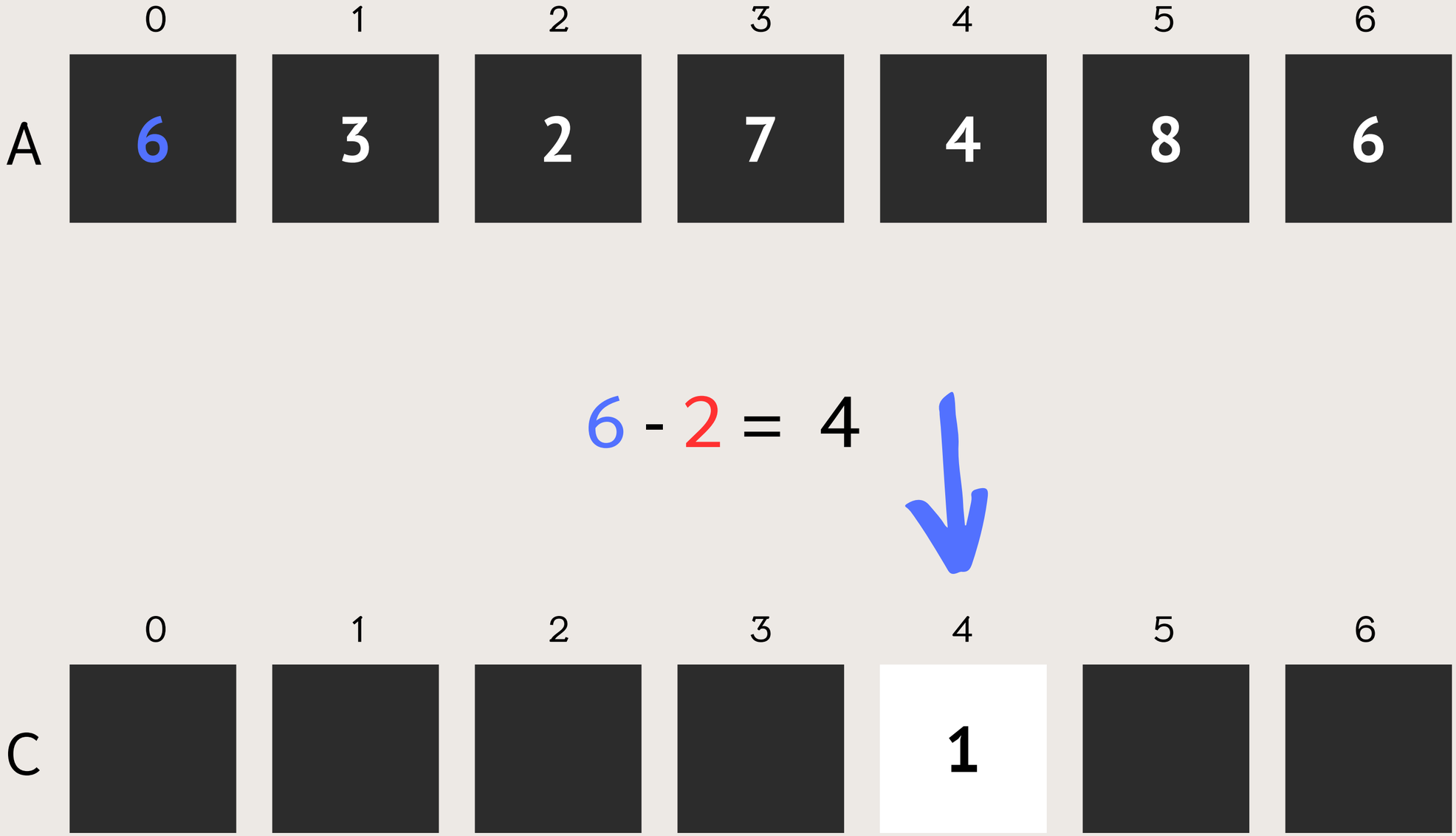
Criar os pigeonholes

Criar os pigeonholes

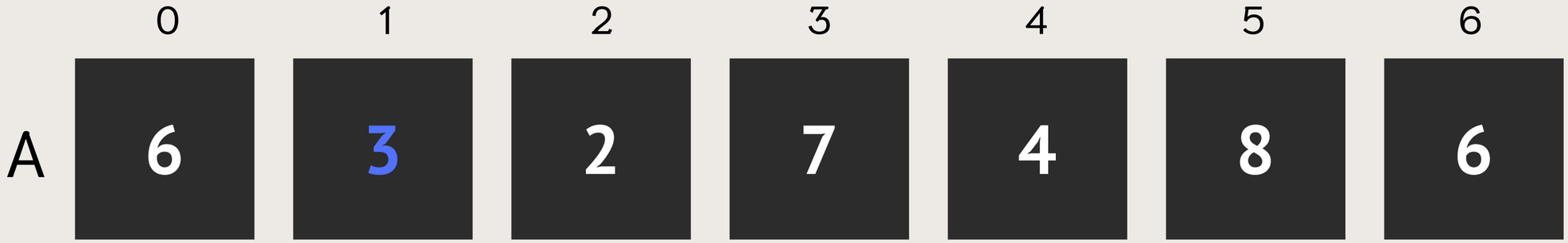


6 - 2 = 4

Criar os pigeonholes

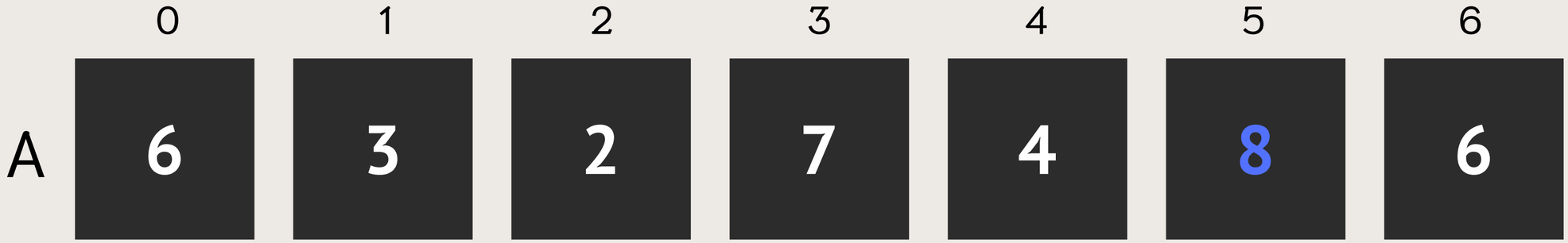


Criar os pigeonholes



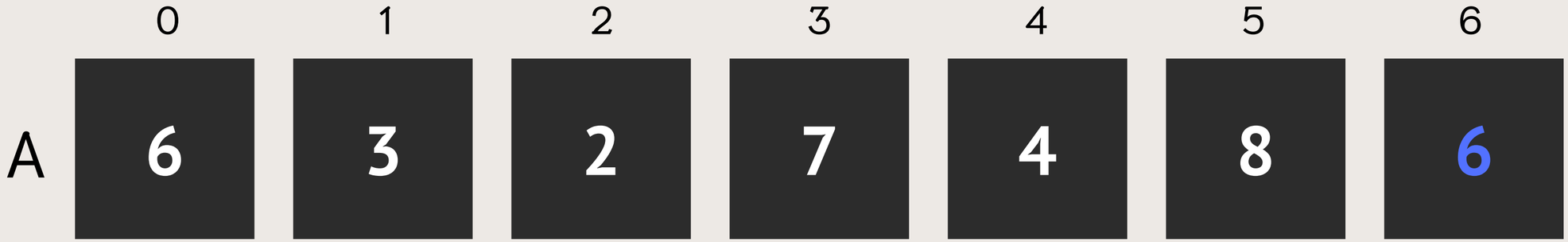
$$3 - 2 = 1$$

Criar os pigeonholes



$$8 - 2 = 6$$

Criar os pigeonholes

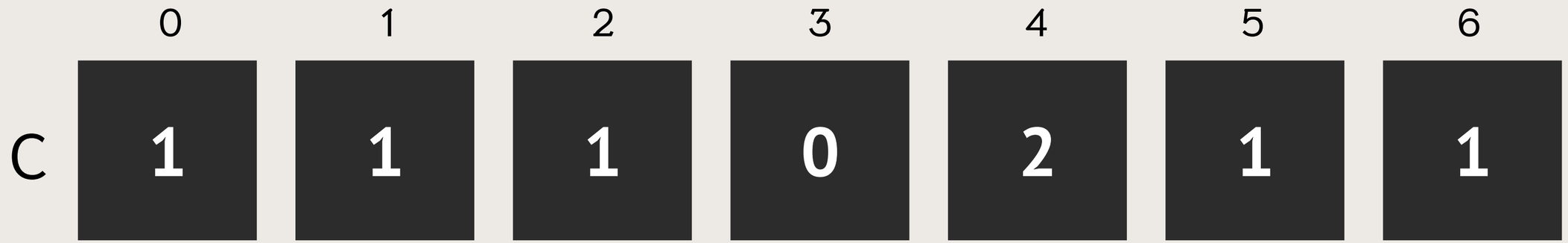


$$6 - 2 = 4$$

3. Coletar e Converter

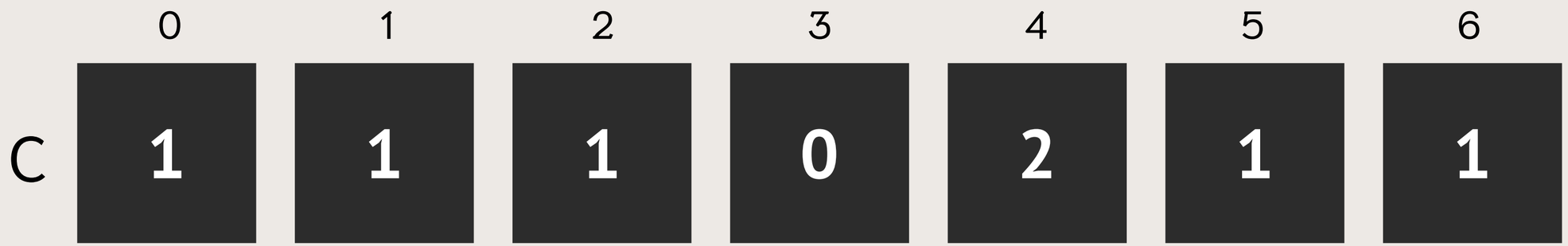
Percorra os pigeonholes em ordem crescente de índice e colete os elementos, reconstruindo a lista ordenada. Esta reconstrução se dá por meio de um for que acessa cada posição do vetor e soma essa posição à entrada de menor valor, alocando cada valor em sua respectiva posição dentro do array.

Coletar e Converter



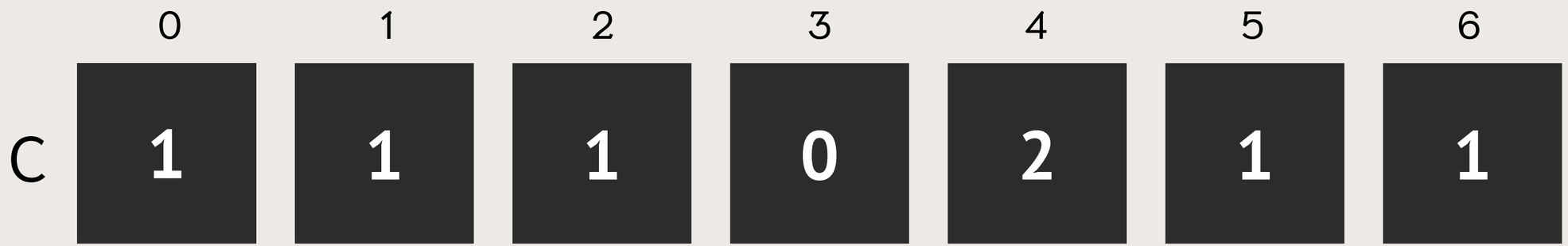
Índice + Mínimo = Elemento

Coletar e Converter



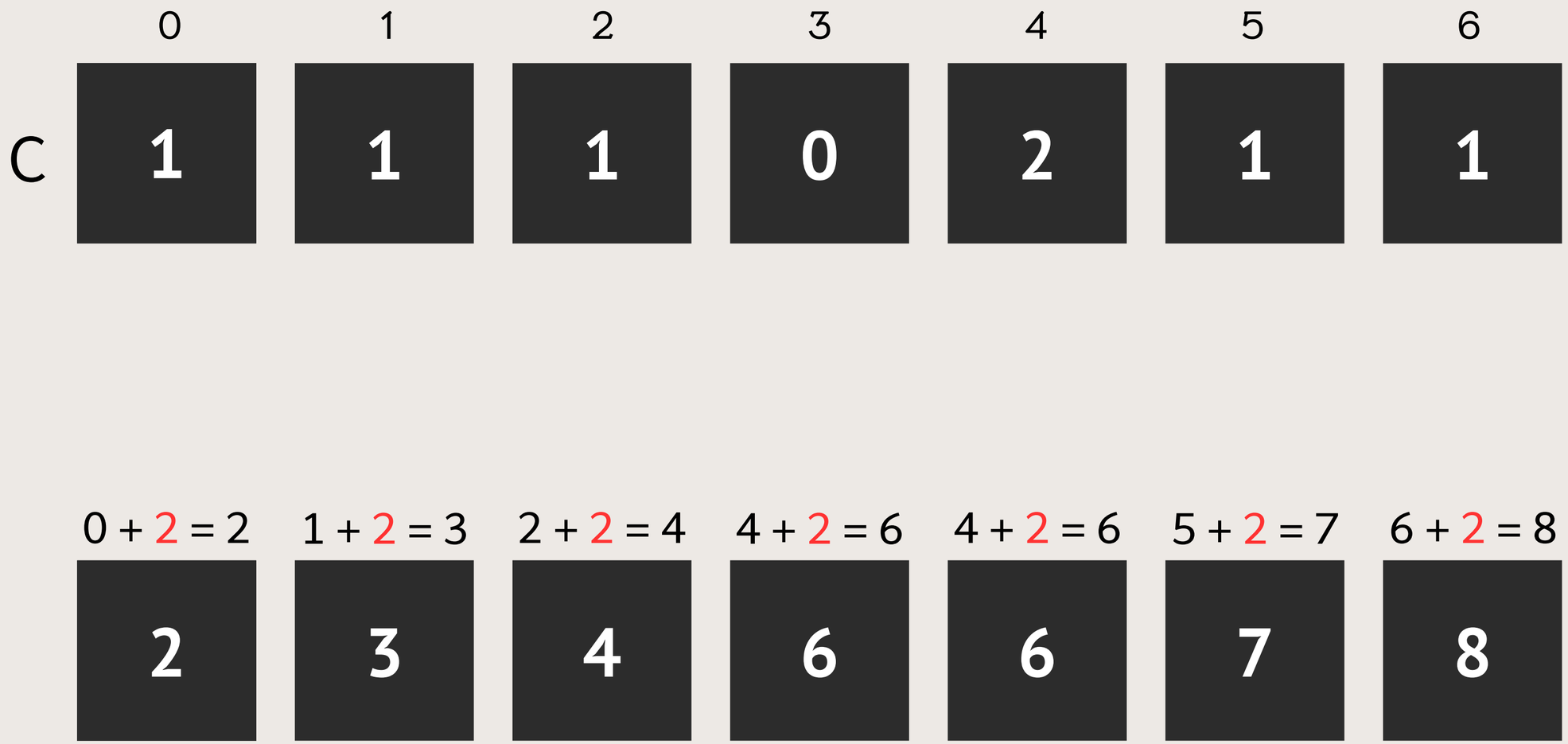
Índice + Mínimo = Elemento

Coletar e Converter



$$0 + 2 = 0$$

Coletar e Converter



Coletar e Converter

Implementação

Implementação em diferentes linguagens de programação

Implementação nas Linguagens

- 
- 
- 
- 
- 
- 

Implementação no código

Counting Sort

```
[ ]  a = [2, 5, 3, 0, 2, 3, 0, 3]
     c = [0] * (max(a) + 1)

     # Passo 1: Contagem das ocorrências dos elementos
     for i in a:
         c[i] += 1

     # Passo 2: Determinação das posições finais
     for i in range(1,len(c)):
         c[i] += c[i - 1]

     # Passo 3: Construção do array ordenado
     b = [0] * len(a)
     for i in reversed(a):
         b[c[i] - 1] = i
         c[i] -= 1

     print(b)
```

```
[0, 0, 2, 2, 3, 3, 3, 5]
```

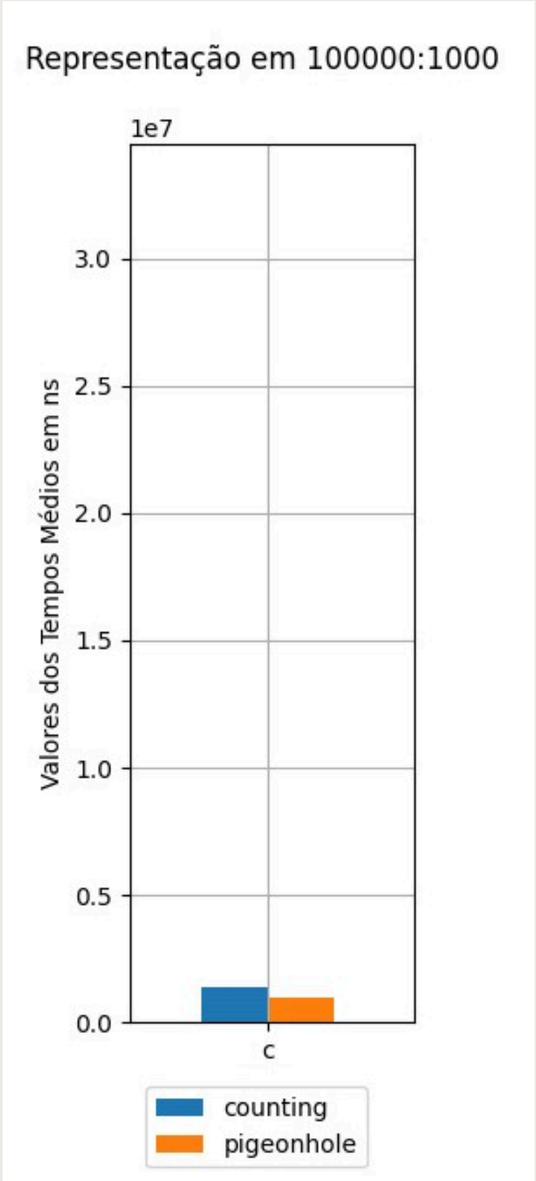
Pigeonhole Sort

```
a = [6, 3, 2, 7, 4, 8, 6]

# Encontrar o mínimo e o máximo no vetor
minimum = min(a)
maximum = max(a)

# Determinar o tamanho do vetor de Pigeonholes
size = maximum - minimum + 1

# Criar o vetor de Pigeonholes inicialmente com zeros
c = [0] * size

# Distribuir os elementos do vetor original nos Pigeonholes
for number in a:
    c[number - minimum] += 1

# Reconstruir o vetor ordenado a partir dos Pigeonholes
index = 0
for j in range(size):
    while c[j] > 0:
        a[index] = j + minimum
        index += 1
        c[j] -= 1

print(a)
```

```
[2, 3, 4, 6, 6, 7, 8]
```

Implementação nas Linguagens

- Baixo nível: Oferece controle detalhado sobre o hardware.
- Compilada: O código fonte em C é convertido diretamente para código de máquina pelo compilador antes de ser executado.
- Desempenho: Extremamente rápido devido à sua proximidade com o hardware.
- Uso: Amplamente usada em sistemas operacionais, drivers, e sistemas embarcados.
- A compilação direta para código de máquina permite uma execução muito eficiente de algoritmos de ordenação, com sobrecarga mínima em termos de tempo de execução e uso de memória.



Implementação nas Linguagens

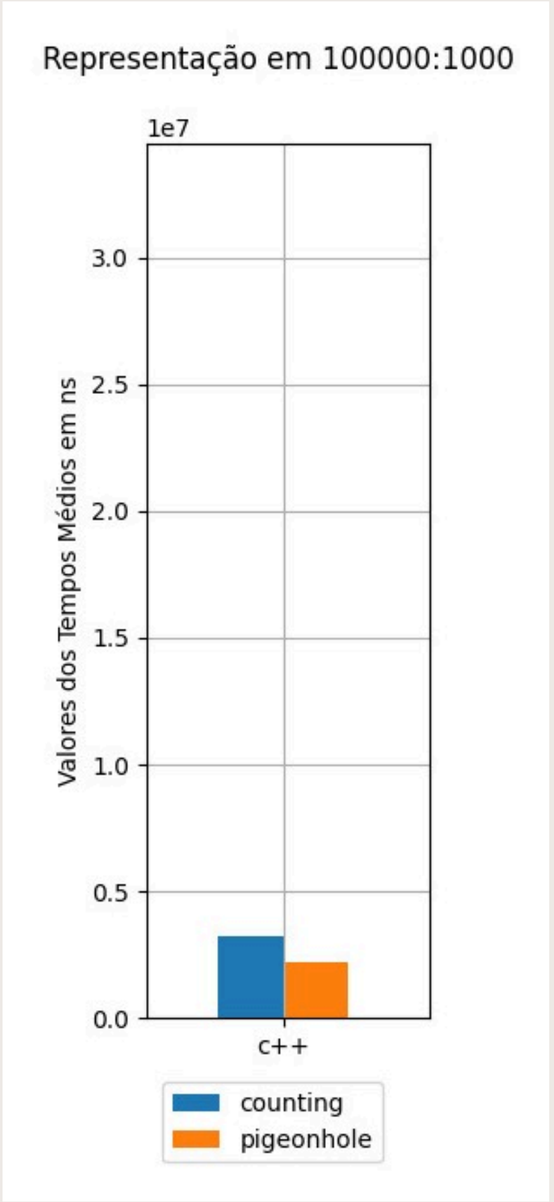
- Orientada a objetos: Suporta programação orientada a objetos, oferecendo classes e objetos.

- Compilada: Semelhante ao C, o código C++ é compilado para código de máquina.

- Desempenho: Mantém a alta performance do C, com adição de recursos avançados.

- Uso: Utilizada em aplicações de alto desempenho, jogos, sistemas em tempo real, e aplicações gráficas.

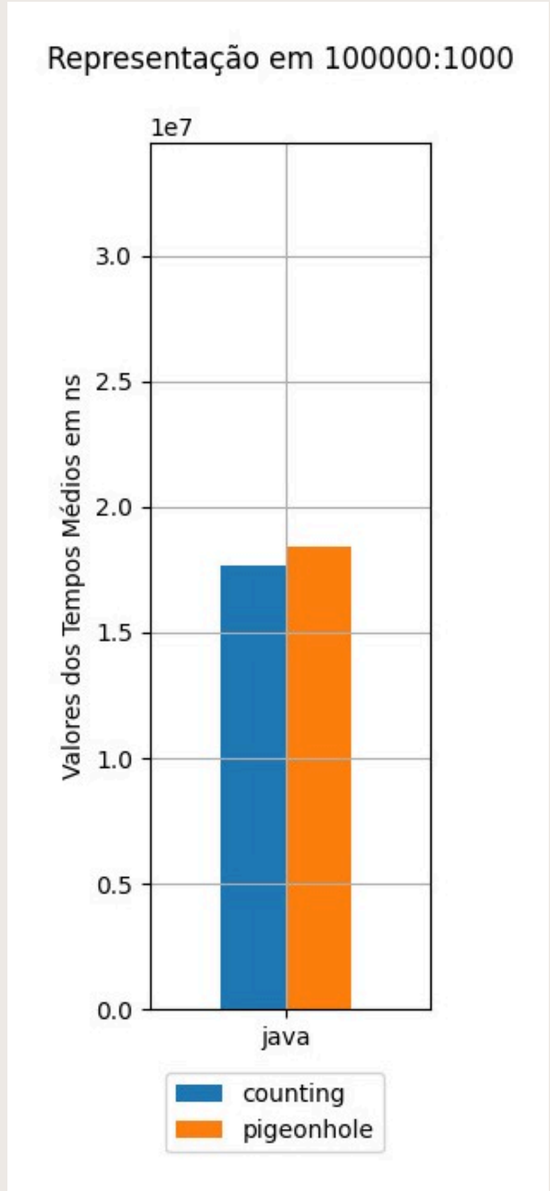
- Assim como C, a compilação resulta em alta eficiência de execução. No entanto, características adicionais de orientação a objetos podem introduzir uma pequena sobrecarga, mas geralmente não é significativa para algoritmos de ordenação bem otimizados.



Implementação nas Linguagens

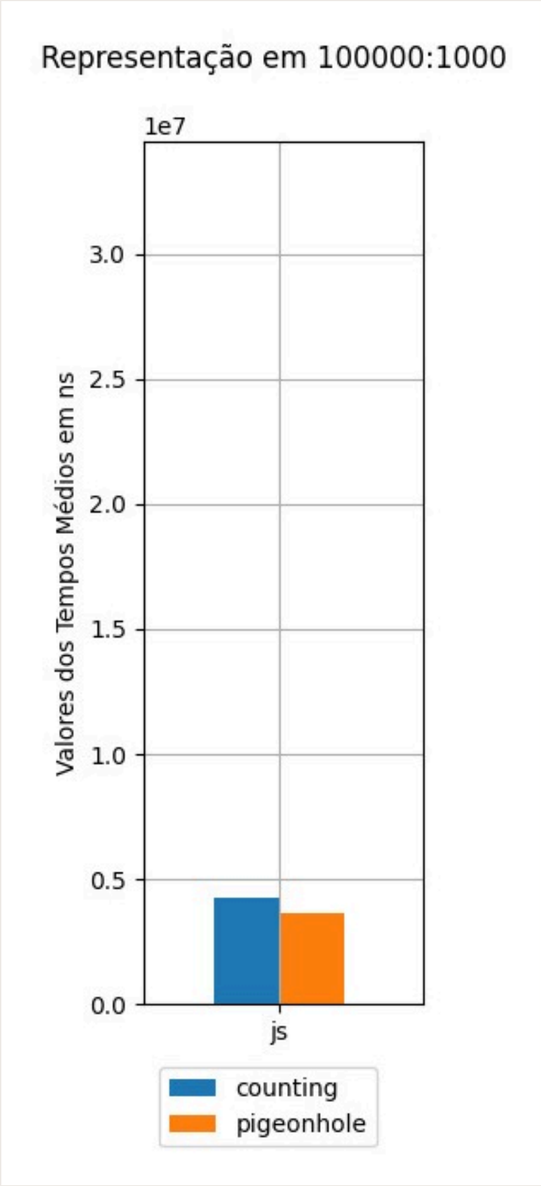
- Orientada a objetos: Fortemente baseada em objetos e classes.

- Portabilidade: "Escreva uma vez, execute em qualquer lugar", devido à JVM.
- Compilada e Interpretada (Bytecode): O código Java é compilado para bytecode, que é executado pela JVM (Java Virtual Machine).
- Uso: Usada em aplicações corporativas, desenvolvimento Android, e sistemas distribuídos
- A JVM otimiza a execução do bytecode, tornando a performance boa, mas geralmente não tão rápida quanto linguagens compiladas diretamente para código de máquina. Algoritmos de ordenação podem ser mais lentos em comparação com C/C++, mas ainda são eficientes.



Implementação nas Linguagens

- Interpretada: Executada diretamente pelo navegador web.
- Orientada a scripts: Utilizada principalmente para interatividade e dinamismo em páginas web.
- Desempenho: Melhorado através de engines modernas como V8, mas historicamente mais lenta.
- Uso: Essencial para desenvolvimento web front-end, e também usada no back-end com Node.js.: Amplamente usada em sistemas operacionais, drivers, e sistemas embarcados.
- A execução no navegador e a interpretação linha por linha podem introduzir sobrecarga. Engines modernas melhoraram a performance, mas algoritmos de ordenação podem ainda ser mais lentos em comparação com linguagens compiladas.



Implementação nas Linguagens

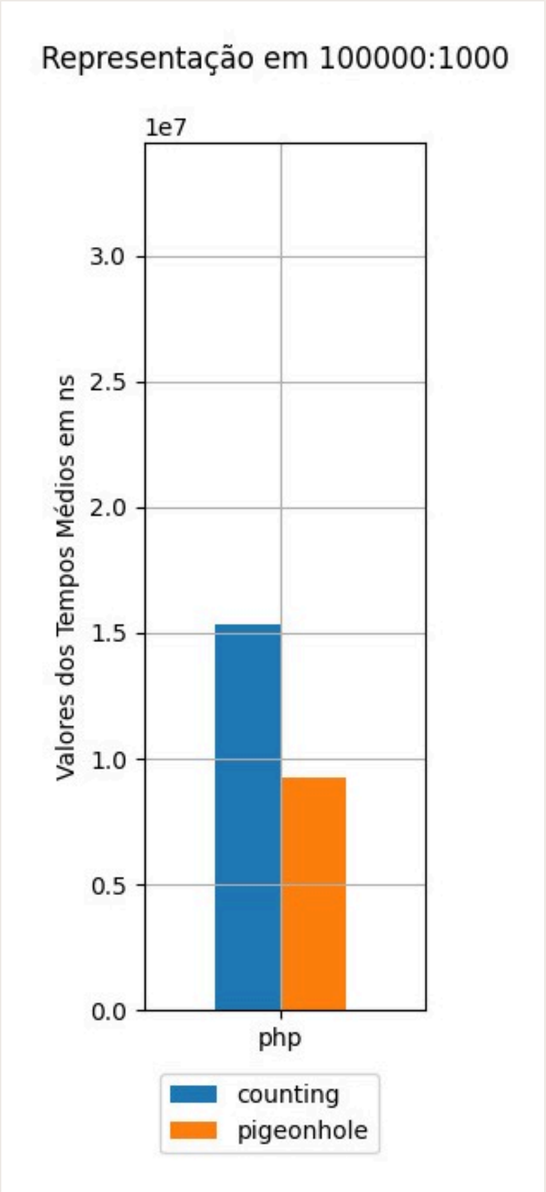
- Interpretada: O código PHP é executado pelo interpretador no servidor, linha por linha.

- Orientada a scripts: Principalmente usada para desenvolvimento web e scripts do lado do servidor.

- Desempenho: Menos eficiente em termos de execução em comparação com linguagens compiladas.

- Uso: Amplamente utilizada para desenvolvimento web, gerenciamento de conteúdo e construção de sites dinâmicos.

- A interpretação linha a linha pode introduzir sobrecarga significativa, tornando algoritmos de ordenação mais lentos em comparação com linguagens compiladas. A execução em ambiente de servidor também pode adicionar latência.



Implementação nas Linguagens

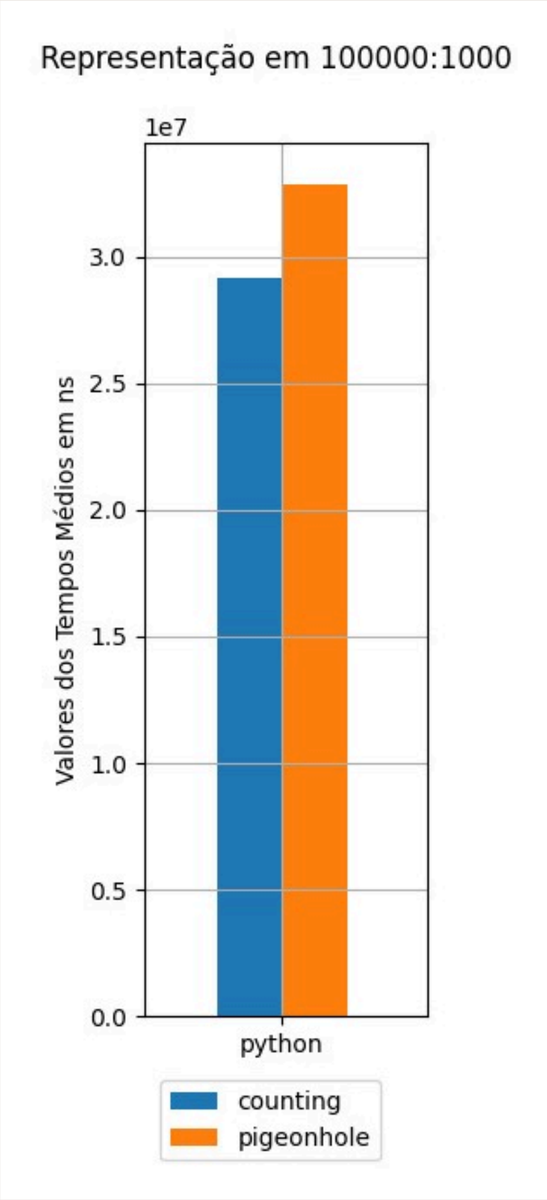
- Interpretada: O código é executado pelo interpretador Python linha por linha.

- Alto nível: Oferece sintaxe simples e clara, ideal para desenvolvimento rápido.

- Desempenho: Menos eficiente em termos de tempo de execução comparado com linguagens compiladas.

- Uso: Utilizada em desenvolvimento web, ciência de dados, automação, aprendizado de máquina, entre outros.

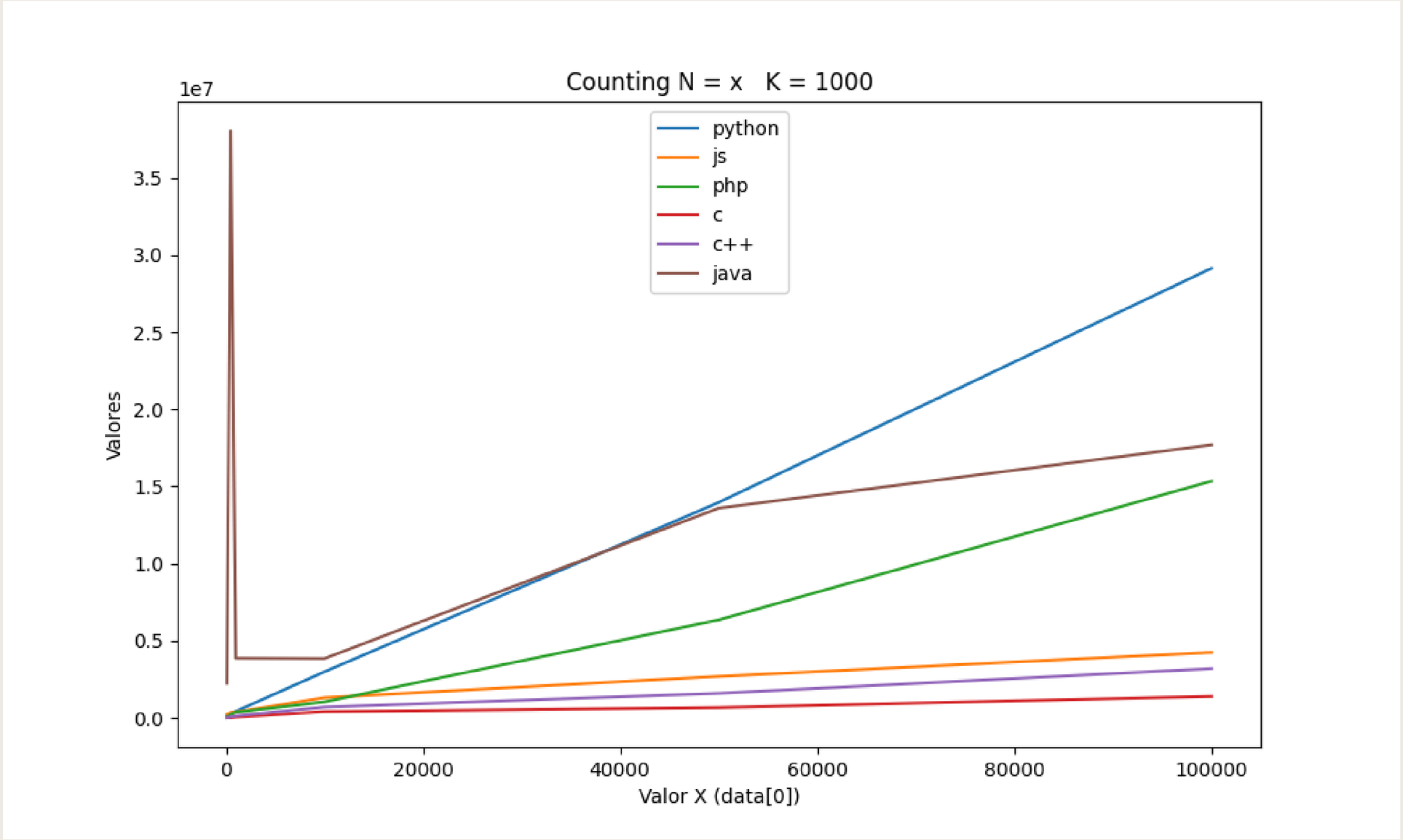
- A interpretação linha a linha introduz sobrecarga significativa. Algoritmos de ordenação em Python geralmente são mais lentos em comparação com linguagens compiladas. No entanto, bibliotecas otimizadas como NumPy podem mitigar alguns desses impactos.



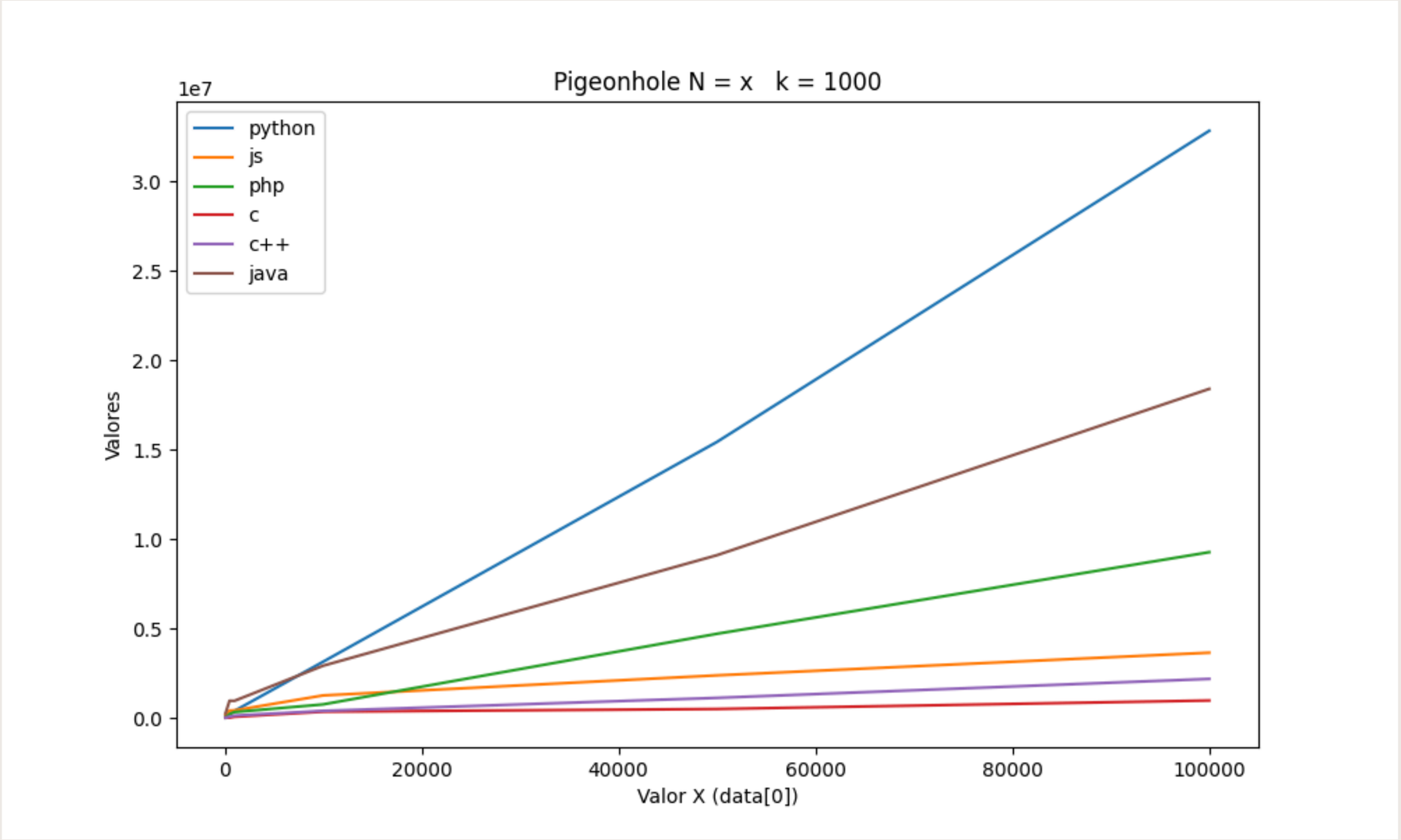
Resultados

Análise do Gráfico Geral

Resultados Counting Sort



Resultados Pigeonhole Sort



Vantagens

- O Counting Sort é um algoritmo estável.
- O Counting Sort é fácil de codificar.
- O Counting Sort geralmente é mais rápido do que os algoritmos de ordenação baseados em comparação, como merge sort e quicksort, se o número de elementos (n) é grande, mas o valor máximo (k) é relativamente pequeno.

Desvantagens

- Funciona melhor com inteiros não negativos.
- O Counting Sort é ineficiente se o intervalo de valores (k) a serem classificados for muito grande.
- O Counting Sort não é um algoritmo de classificação in-place, ele usa espaço extra para classificar os elementos da matriz.

Pigeonhole Sort

Vantagens

- O Pigeonhole Sort é um algoritmo estável.
- O Pigeonhole Sort é fácil de entender e implementar.
- O Pigeonhole Sort é muito eficiente quando o intervalo de valores (k) no vetor de entrada é pequeno, pois evita muitas das comparações que outros algoritmos de classificação fazem.

Desvantagens

- Pode ser lento se o intervalo for muito maior que o número de elementos no vetor de entrada.
- O Pigeonhole Sort requer uma grande quantidade de memória para armazenar os pigeonholes, o que pode ser um problema se o intervalo for muito grande.
- O Pigeonhole Sort não é um algoritmo de classificação in-place, ele usa espaço extra para classificar os elementos do vetor.

• Complexidade

Melhor Caso	**O(n+k)**
Pior Caso	**O(n+k)**
Média	**O(n+k)**
Estabilidade	**Sim**

Em todos os casos a complexidade é a mesma porque, independentemente da disposição dos elementos no array, o algoritmo percorre o array o mesmo número de vezes.

• Complexidade

Melhor Caso	**O(n+k)**
Pior Caso	**O(n+k)**
Média	**O(n+k)**
Estabilidade	**Sim**

Em todos os casos a complexidade é a mesma, porque a criação dos buracos e a distribuição dos elementos são feitas independentemente da ordem inicial dos elementos de entrada.

Aplicações

Situações de uso dos Algoritmos

Aplicações do Counting Sort

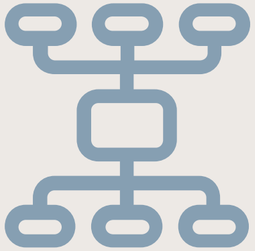
- **Classificação de Notas de Exames:** Em sistemas educacionais, o Counting Sort pode ser usado para classificar as notas de uma prova quando o intervalo de notas é restrito.
- **Classificação de IDs:** Em bancos de dados ou sistemas que gerenciam IDs únicos dentro de um intervalo pequeno, o Counting Sort pode acelerar a ordenação desses IDs.
- **Análise de Frequências de Caracteres:** Em processamento de texto, o Counting Sort pode ser utilizado para ordenar caracteres em um texto para análise de frequência, como em algoritmos de compressão de dados.
- **Ordenação de Contagens em Análise Estatística:** Em estatísticas e análise de dados, o Counting Sort pode ser usado para ordenar contagens de eventos ou ocorrências quando os dados são inteiros e o intervalo é limitado.
- **Processamento de Imagens Digitais:** Em alguns algoritmos de processamento de imagens, especialmente em operações que envolvem histograma, o Counting Sort pode ajudar a ordenar pixels por intensidade de cor.





Aplicações do Pigeonhole Sort

- **Ordenação de Valores de Posição em Computação Gráfica:** Na computação gráfica, os valores de posição dos pixels ou vértices podem estar dentro de um intervalo restrito, tornando o Pigeonhole Sort útil para ordenar rapidamente esses valores para operações de renderização ou transformações geométricas.



- **Ordenação de Dados em Aplicações de Redes:** Em algumas aplicações de redes, como roteamento e classificação de pacotes, os identificadores de pacotes ou endereços IP podem ser mapeados para intervalos restritos, permitindo a aplicação do Pigeonhole Sort para ordenar esses identificadores de maneira eficiente.

- **Processamento de Dados em Tempo Real:** Adequado para sistemas em tempo real que exigem ordenação rápida e eficiente de fluxos de dados com valores numéricos restritos. Exemplos incluem sistemas de monitoramento em tempo real e processamento de eventos.

Conclusão

Considerações finais

Conclusão

Counting Sort e Pigeonhole Sort são algoritmos de ordenação eficientes para conjuntos de dados onde a faixa de valores é limitada e relativamente pequena em comparação com o número de elementos.

Ambos os algoritmos são altamente eficientes em contextos específicos onde as condições de entrada são favoráveis. No entanto, para conjuntos de dados com uma ampla gama de valores ou em situações onde o uso de memória é uma preocupação, algoritmos de ordenação comparativos tradicionais, como Quick Sort ou Merge Sort, podem ser mais apropriados. A escolha entre Counting Sort e Pigeonhole Sort deve ser baseada nas características específicas dos dados a serem ordenados e nas restrições de recursos do sistema.

Em linguagens compiladas, como C e C++, os algoritmos tiveram um melhor custo computacional, enquanto nas interpretadas, o desempenho decaiu, principalmente em Python.

Referências:

- http://desenvolvendosoftware.com.br/algoritmos/ordenacao/counting-sort.html
- https://www.programiz.com/dsa/counting-sort#:~:text=Counting%20sort%20is%20a%20sorting,index%20of%20the%20auxiliary%20array.

Obrigado!

Você tem alguma pergunta?

github.com/AndersonR-S/Counting-e-pigeonhole-sort

Kauã Lucas, Humberto Henrique, Anderson

:D