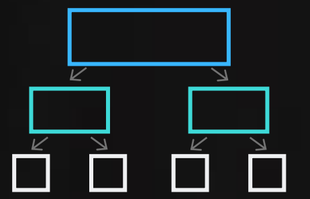


ALGORITMOS DE ORDENAÇÃO

Merge Sort

Sumário

Contexto Histórico

1940

John Von Neumann propõe o algoritmo Merge Sort

1945

EDVAC (Electronic Discrete Variable Automatic Computer)

1950

O Merge Sort começa a ganhar popularidade

1960

O paradigma "divisão e conquista" é formalizado

1970

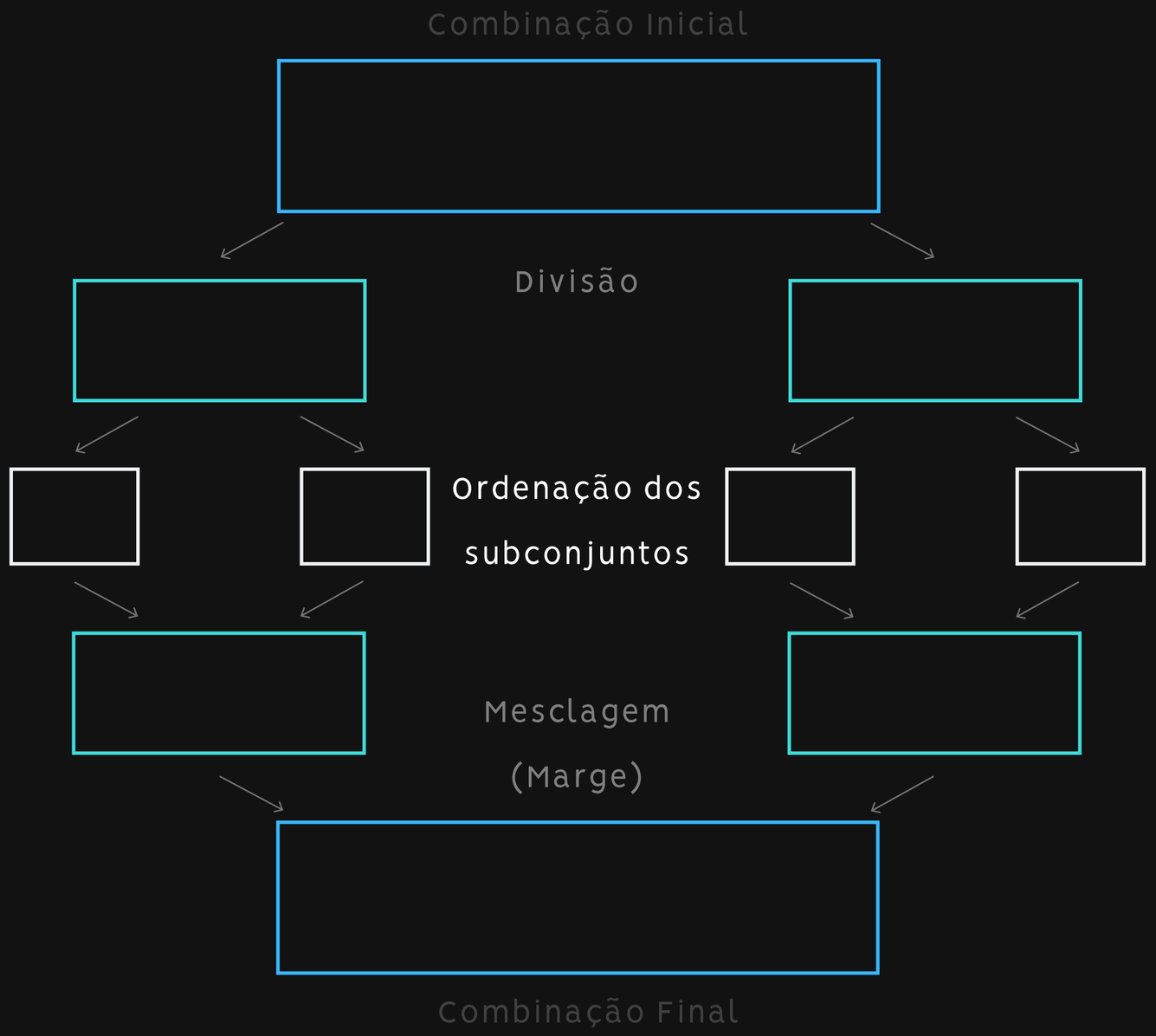
Pesquisadores exploram a otimização do Merge Sort.

1980-1990

Avanço dos computadores e a necessidade crescente de ordenação eficiente

ATUAL

O Merge Sort continua sendo um algoritmo amplamente estudado e aplicado na ciência da computação.



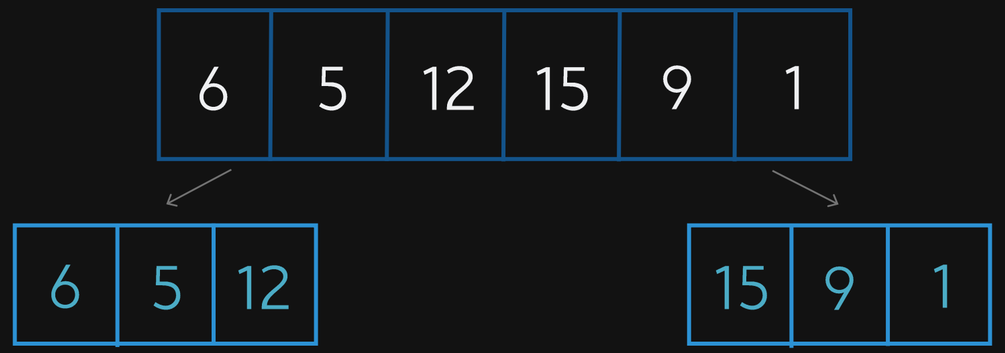
ORDENAÇÃO ADOTADA

Divisão e conquista

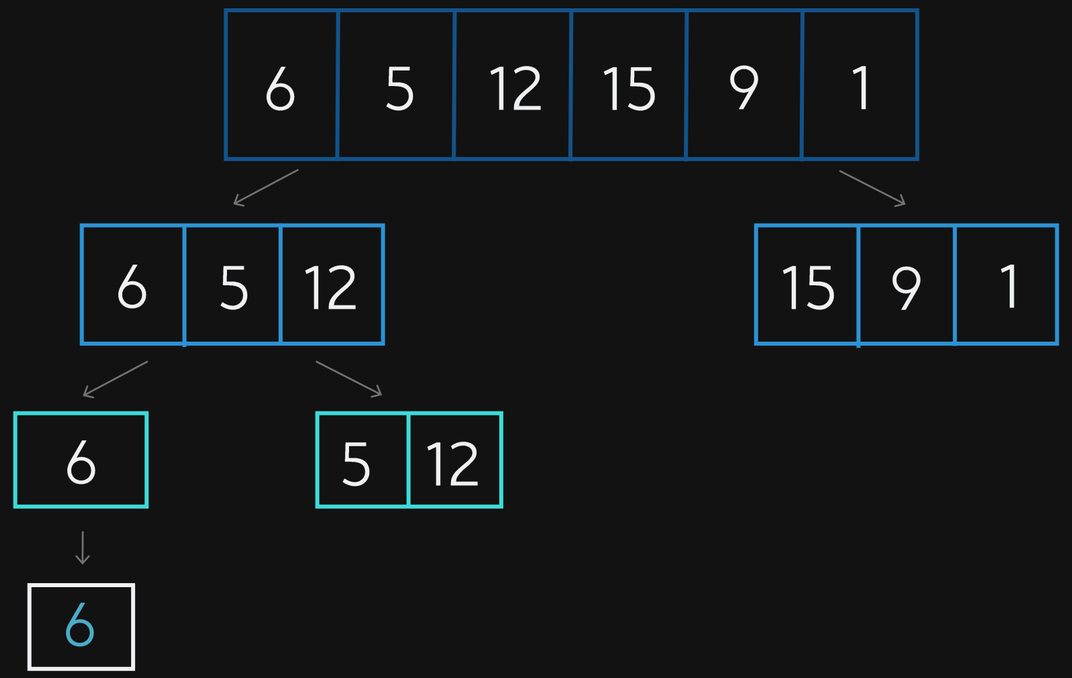
FUNÇÃO MERGE()

FUNÇÃO MERGE()

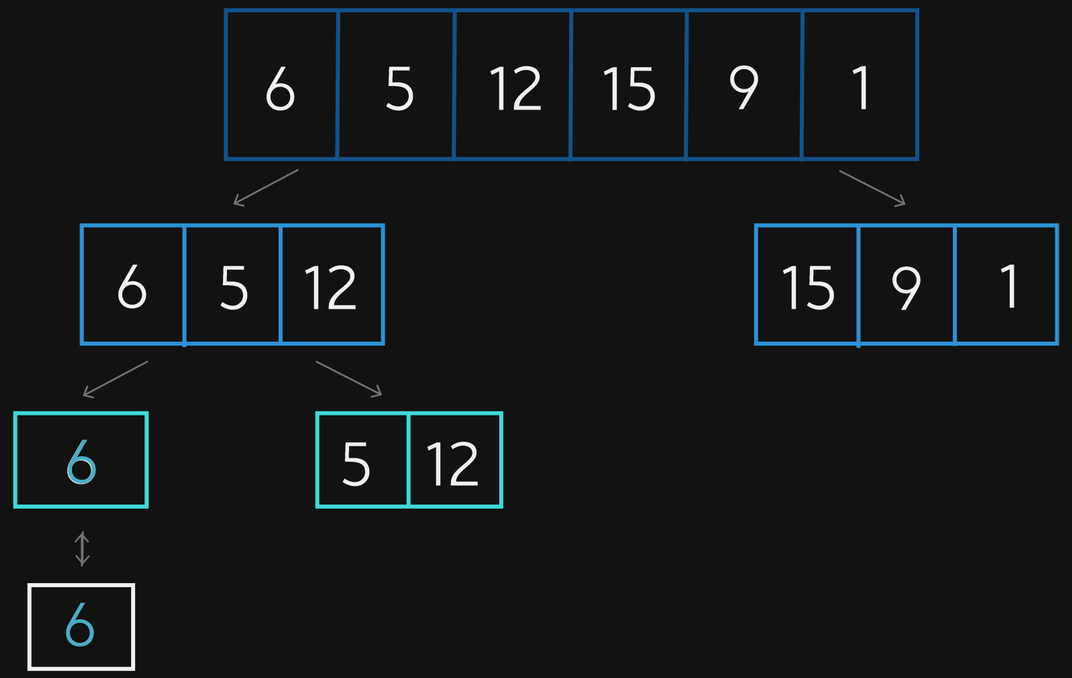
FUNÇÃO MERGE()



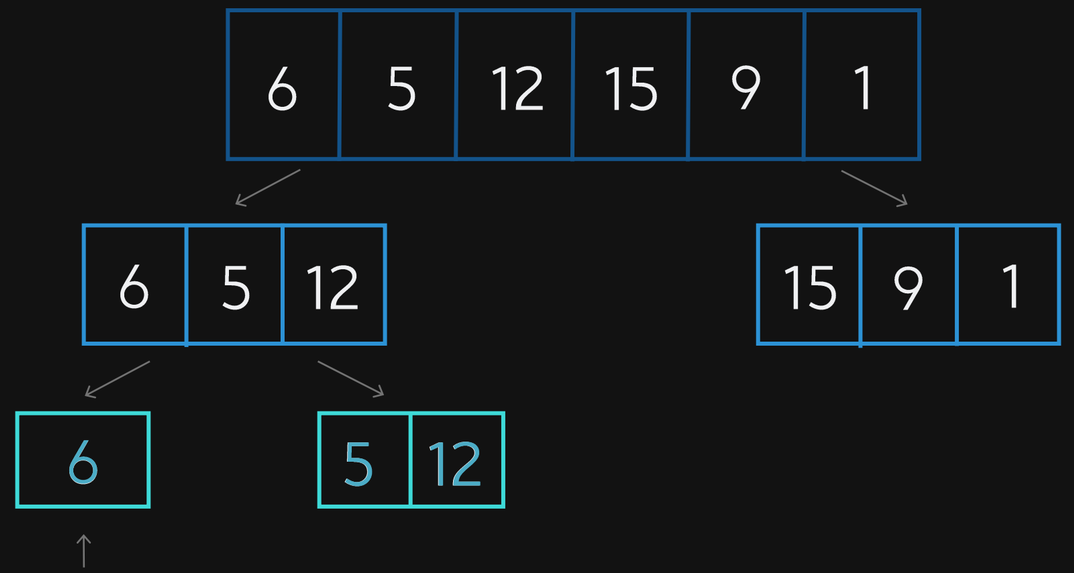
FUNÇÃO MERGE()



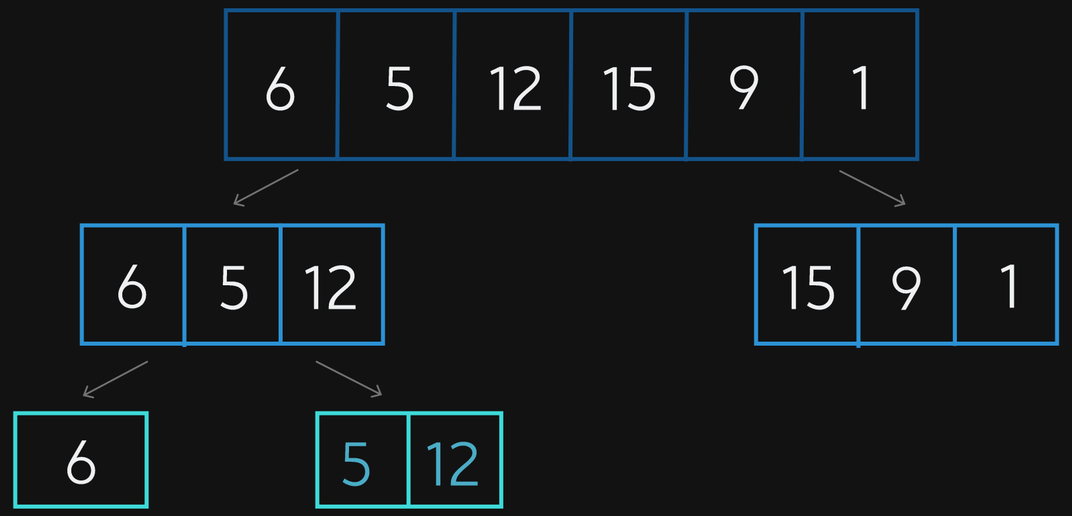
FUNÇÃO MERGE()



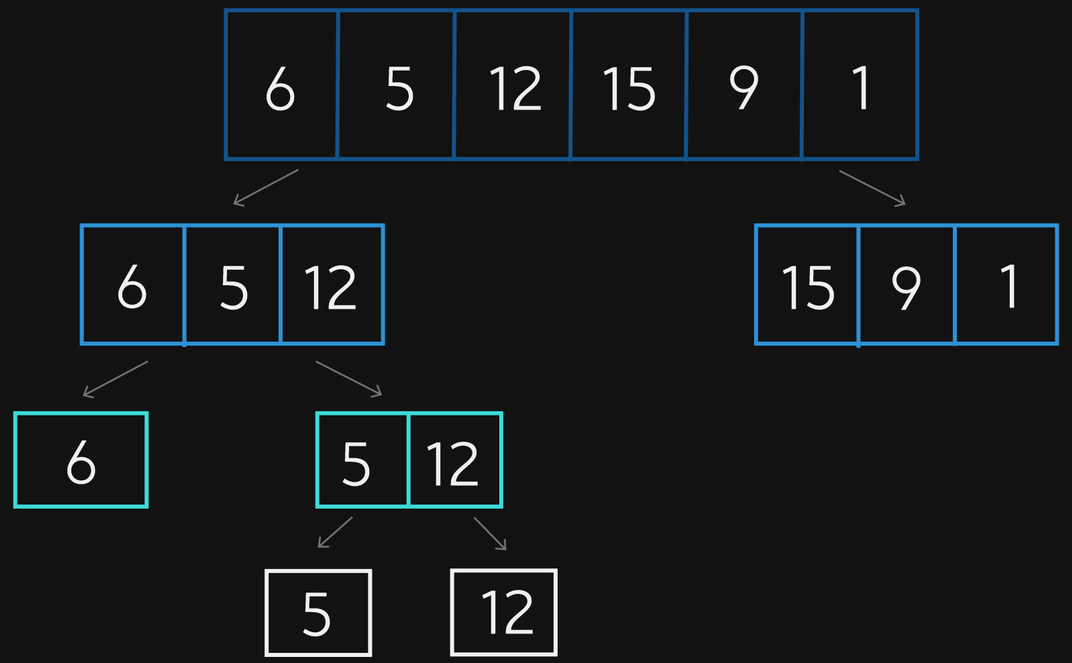
FUNÇÃO MERGE()



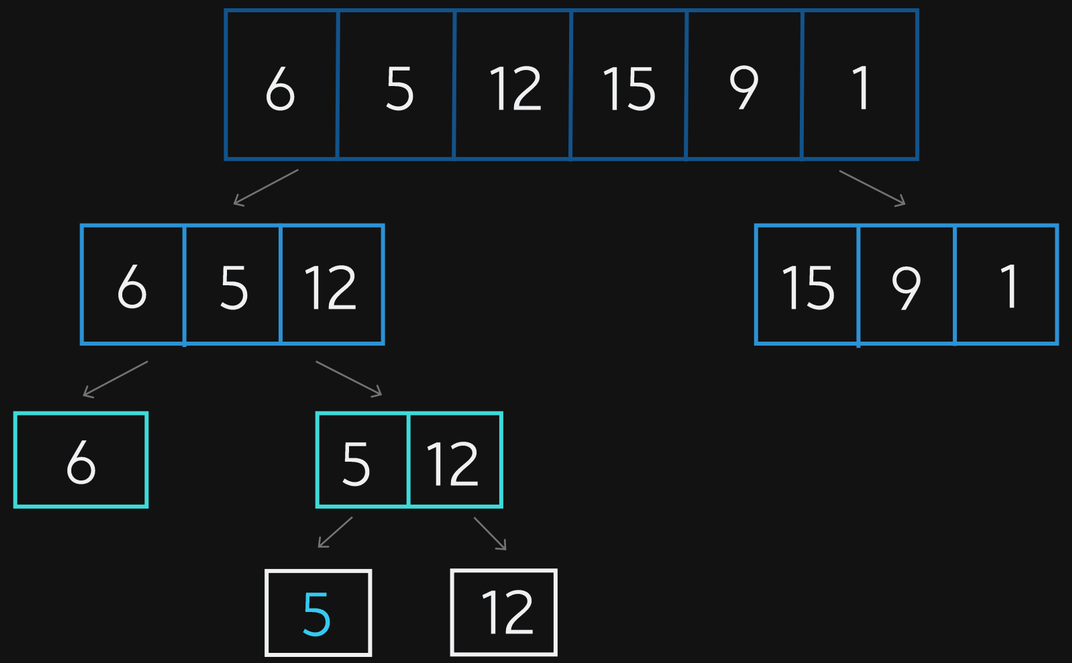
FUNÇÃO MERGE()



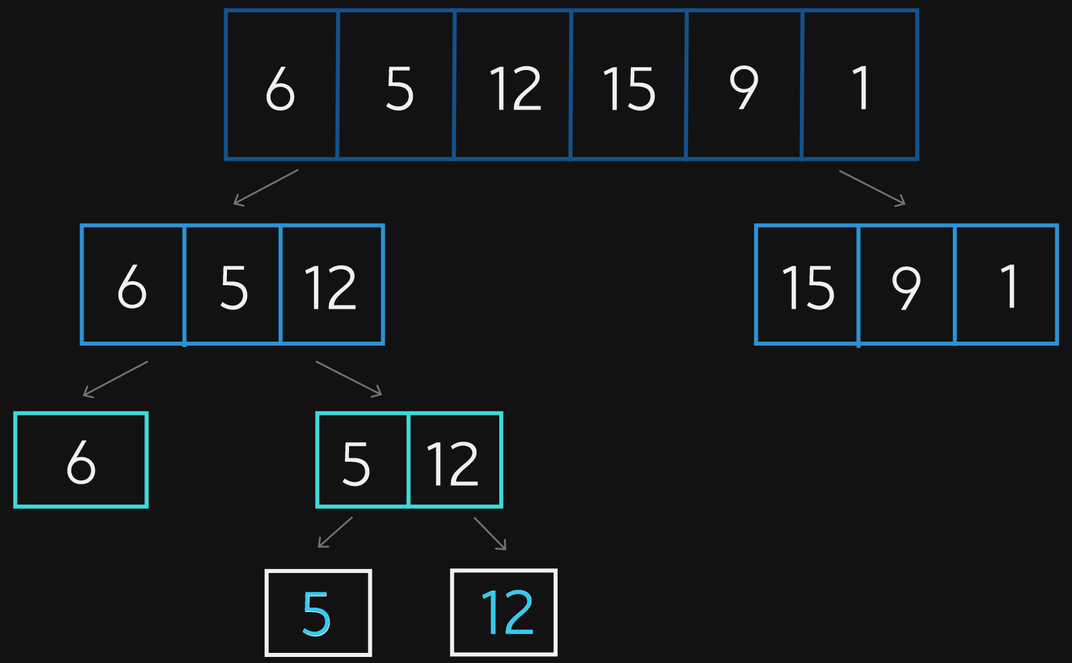
FUNÇÃO MERGE()



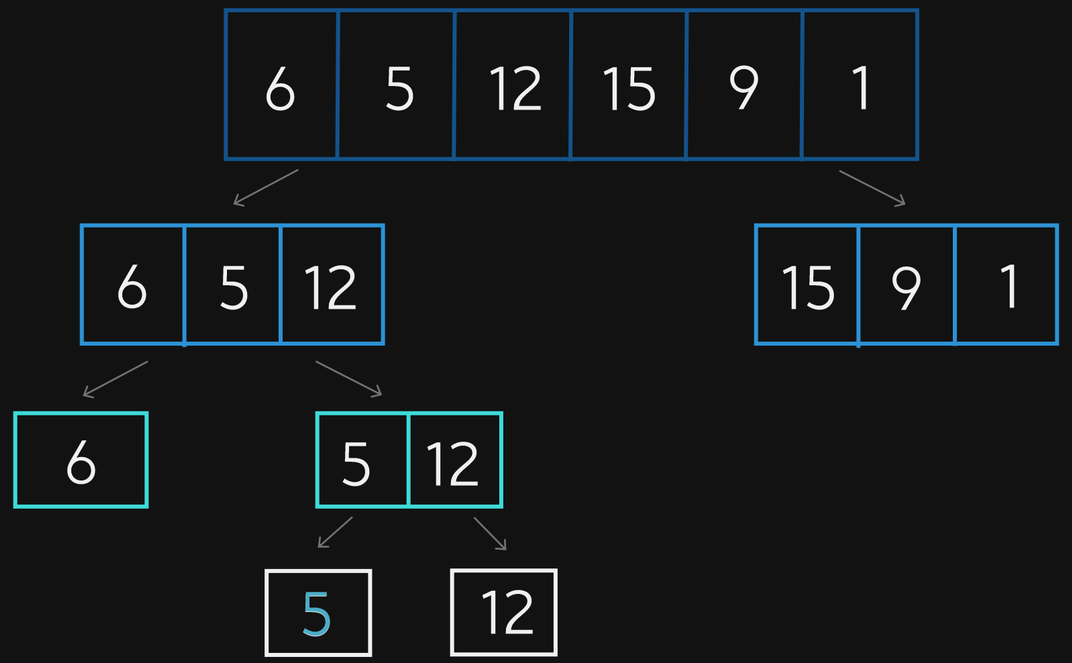
FUNÇÃO MERGE()



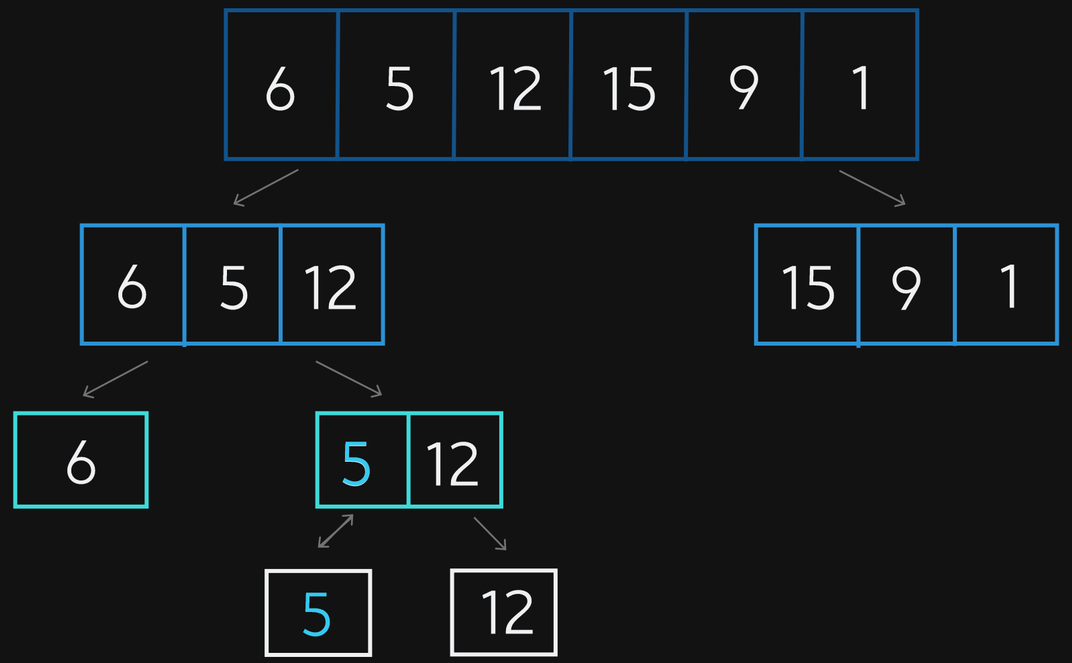
FUNÇÃO MERGE()



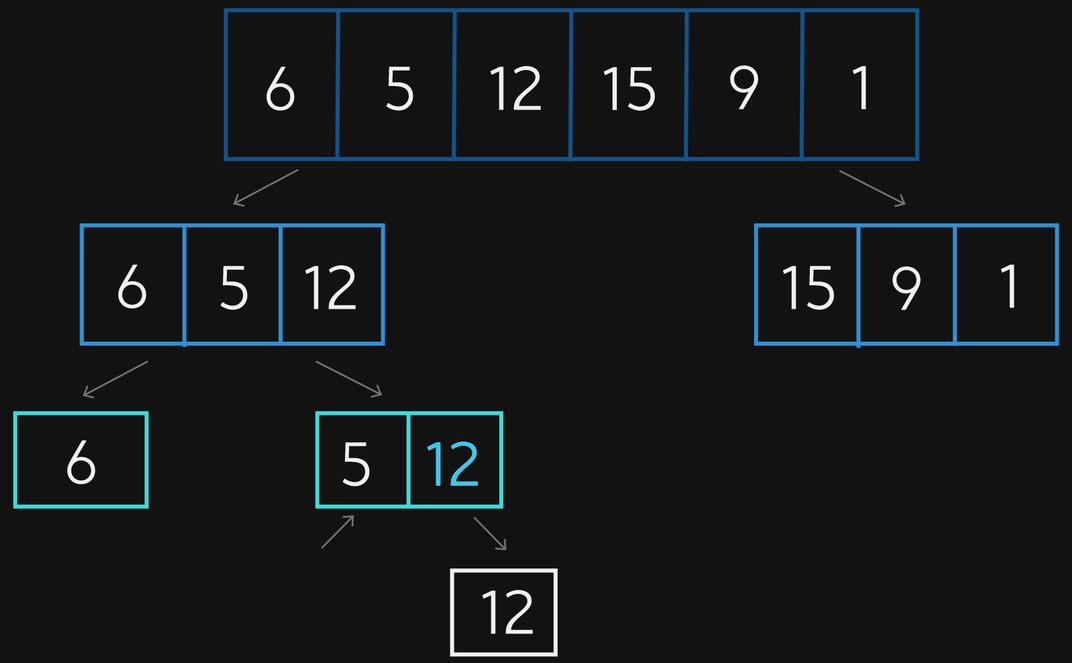
FUNÇÃO MERGE()



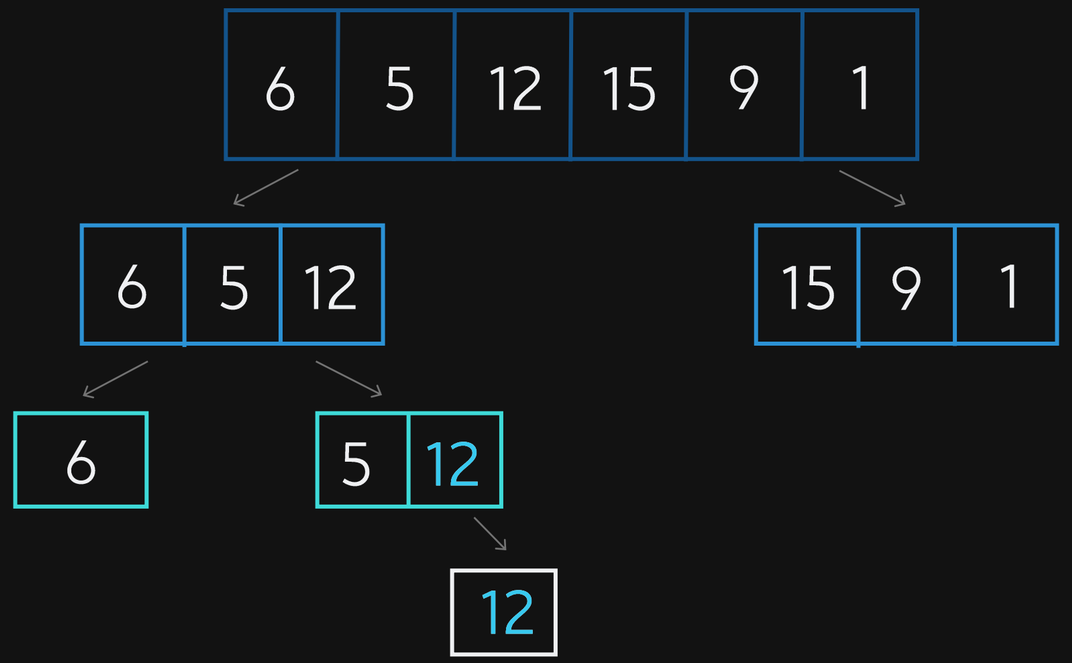
FUNÇÃO MERGE()



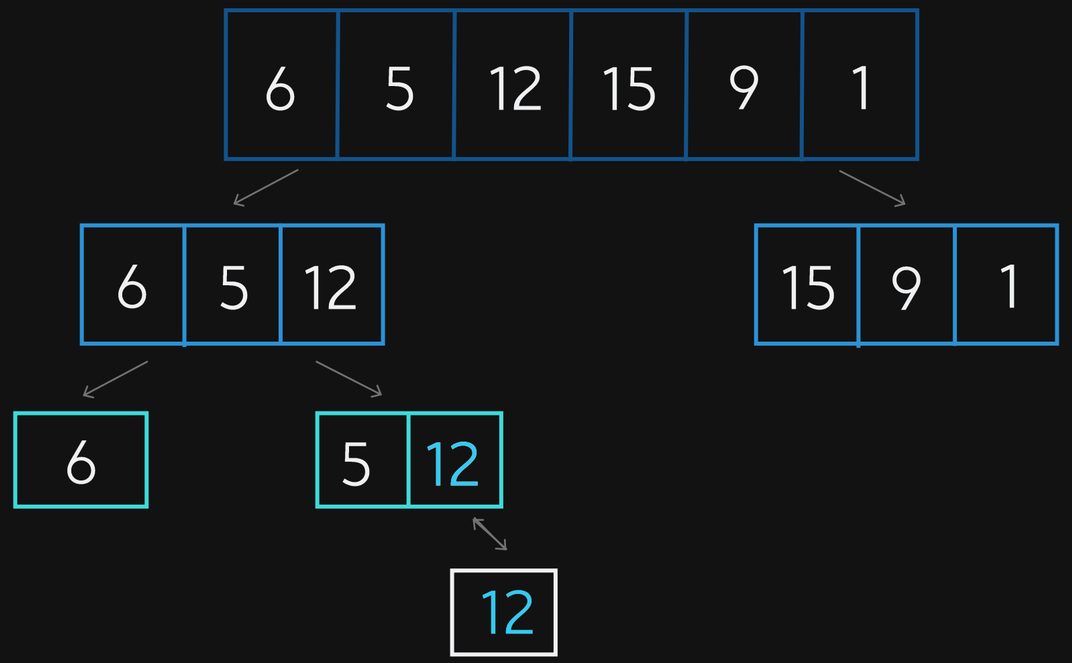
FUNÇÃO MERGE()



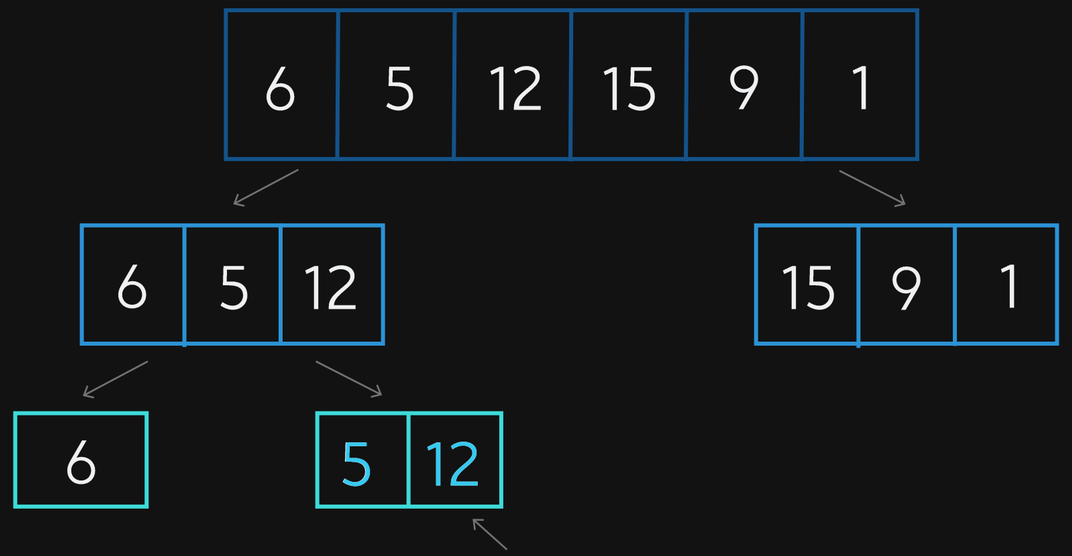
FUNÇÃO MERGE()



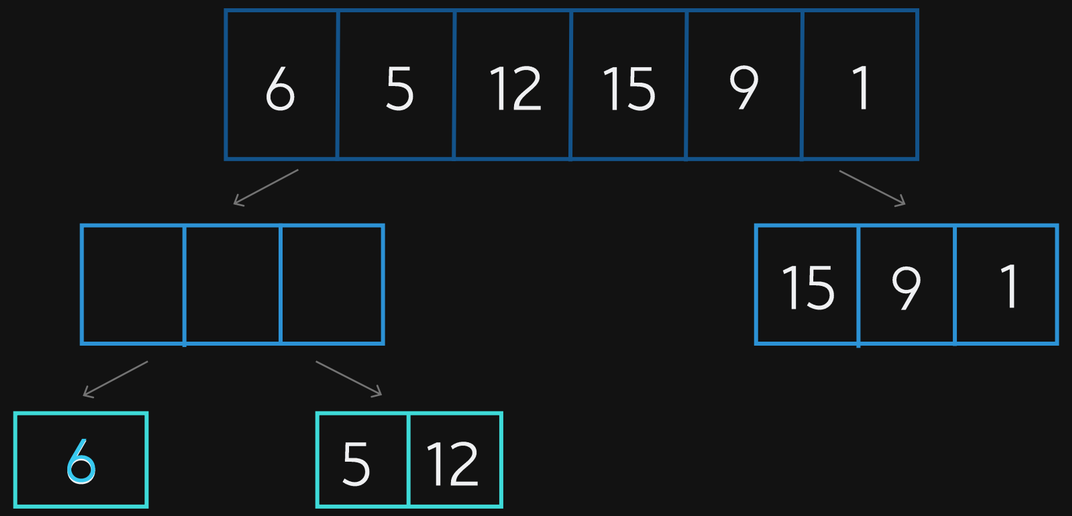
FUNÇÃO MERGE()



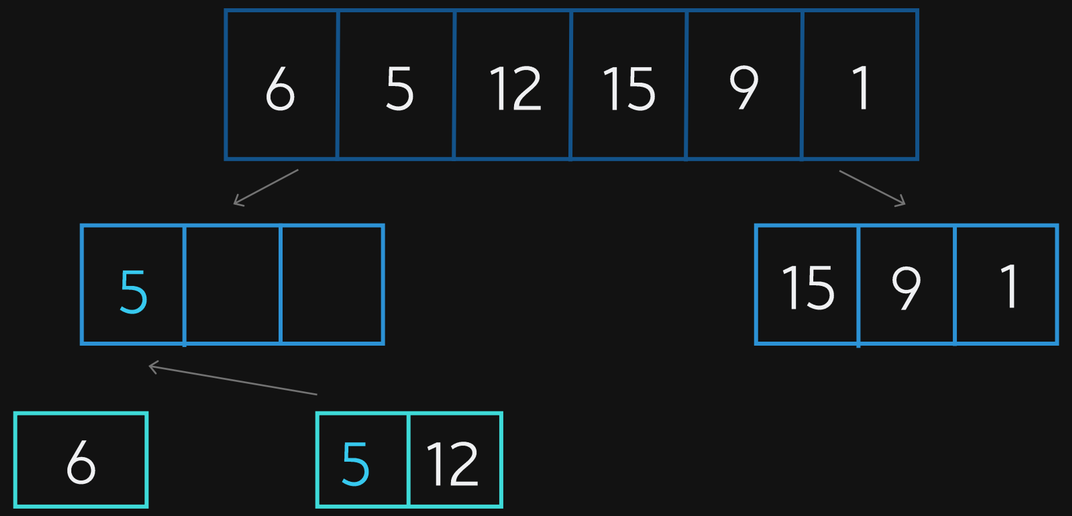
FUNÇÃO MERGE()



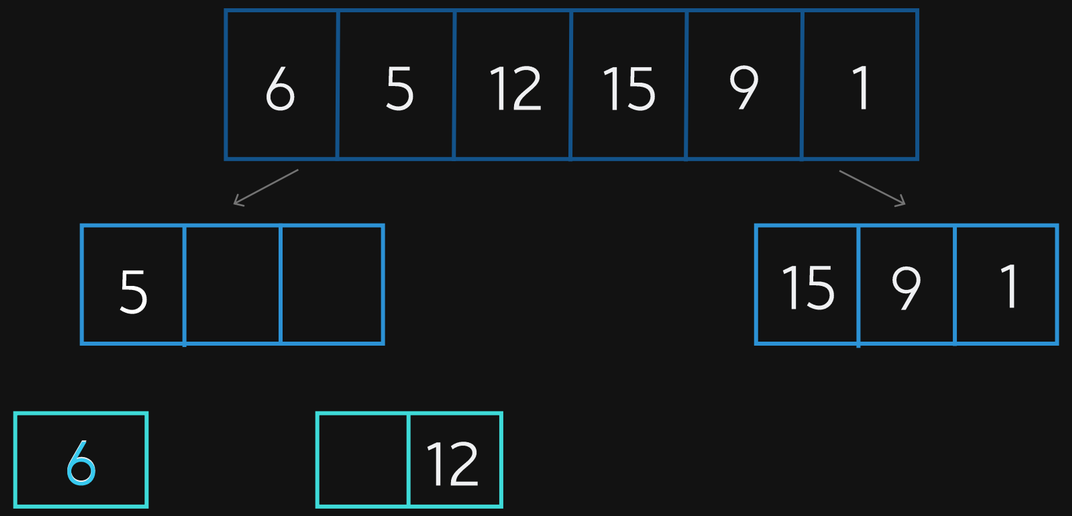
FUNÇÃO MERGE()



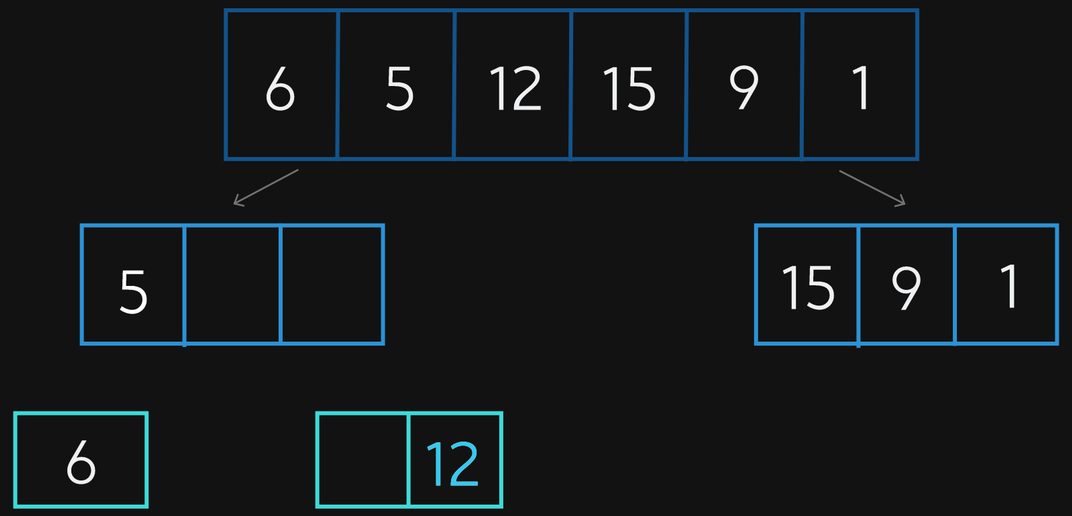
FUNÇÃO MERGE()



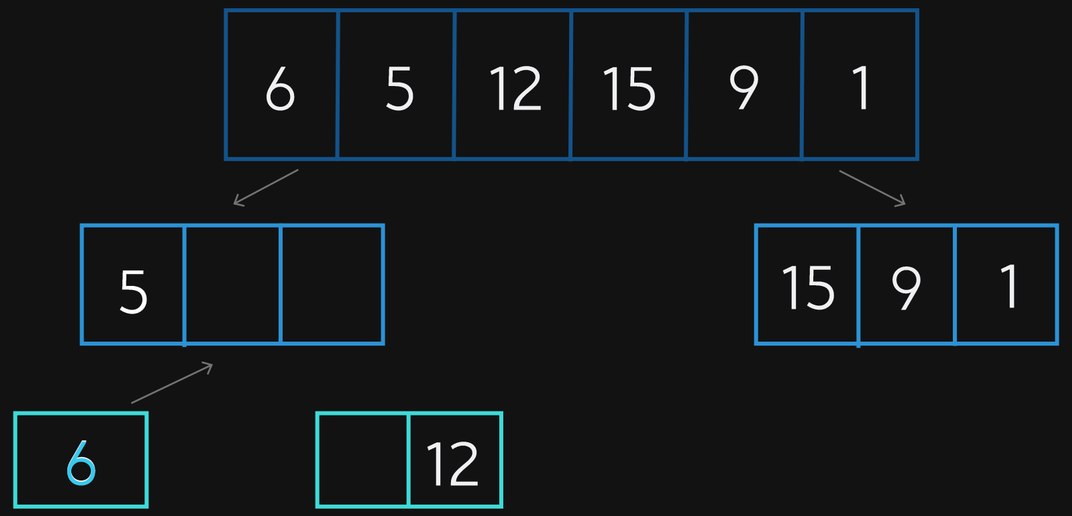
FUNÇÃO MERGE()



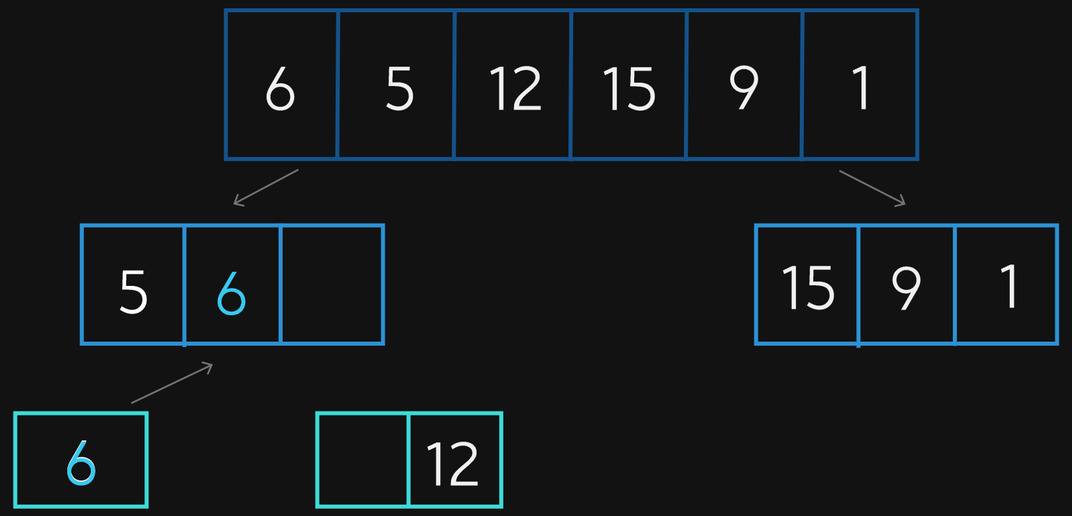
FUNÇÃO MERGE()



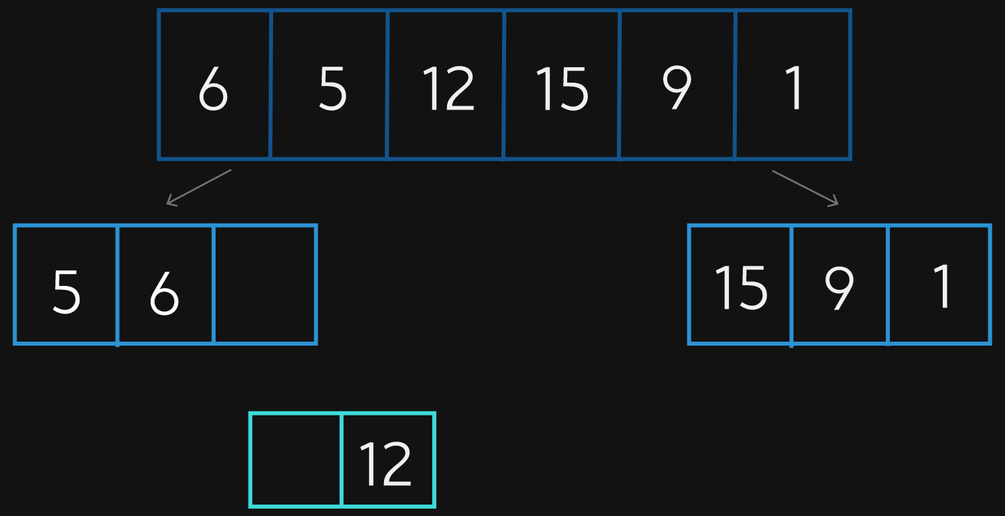
FUNÇÃO MERGE()



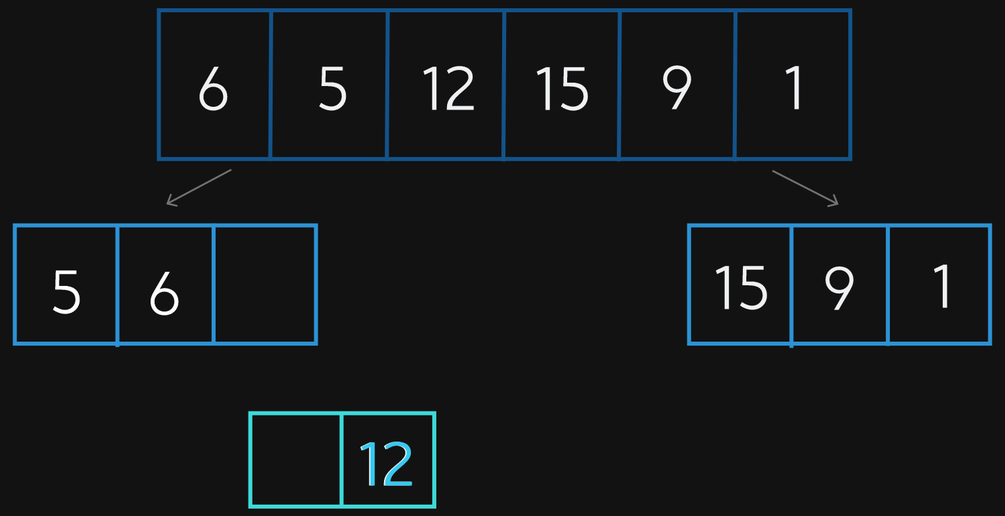
FUNÇÃO MERGE()



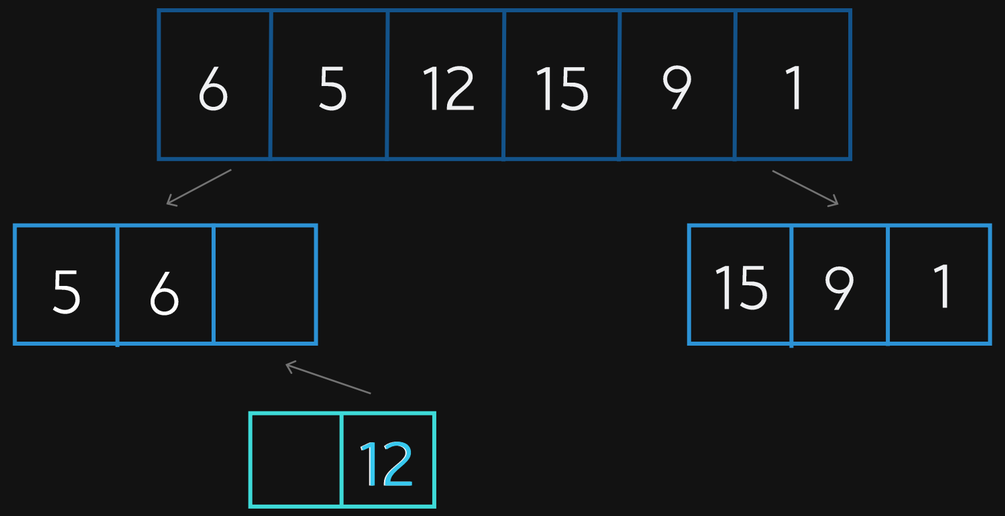
FUNÇÃO MERGE()



FUNÇÃO MERGE()

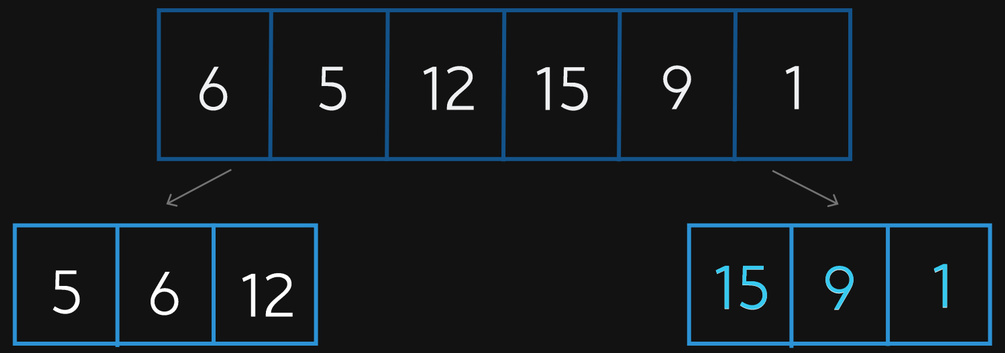


FUNÇÃO MERGE()

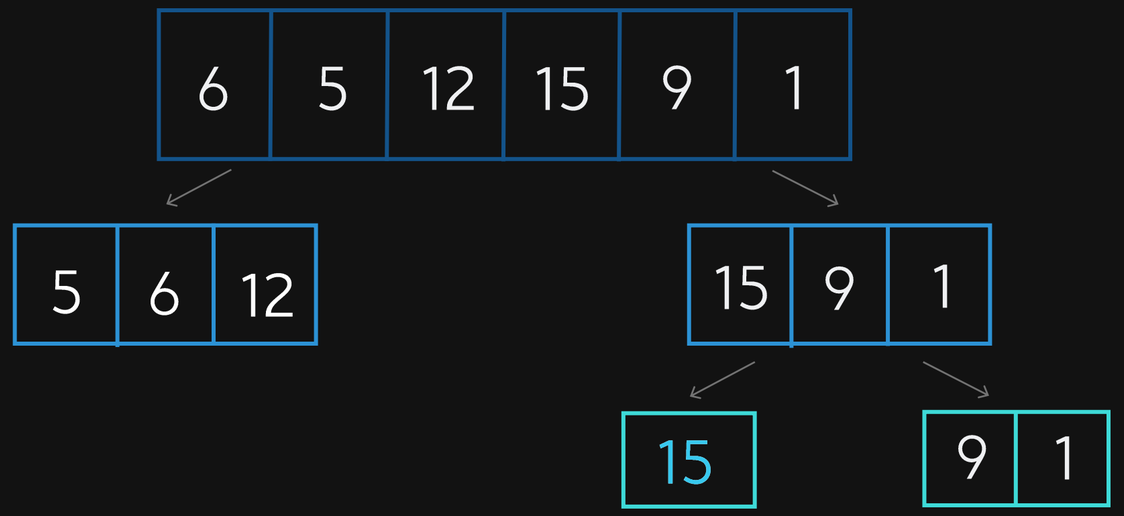


FUNÇÃO MERGE()

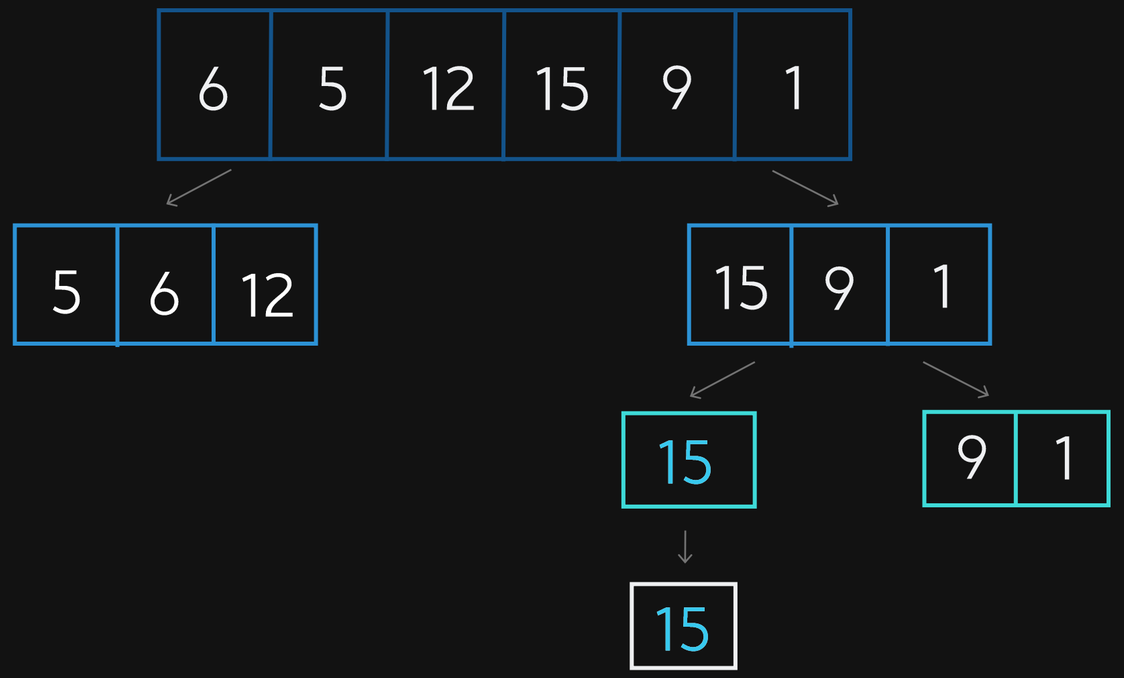
FUNÇÃO MERGE()



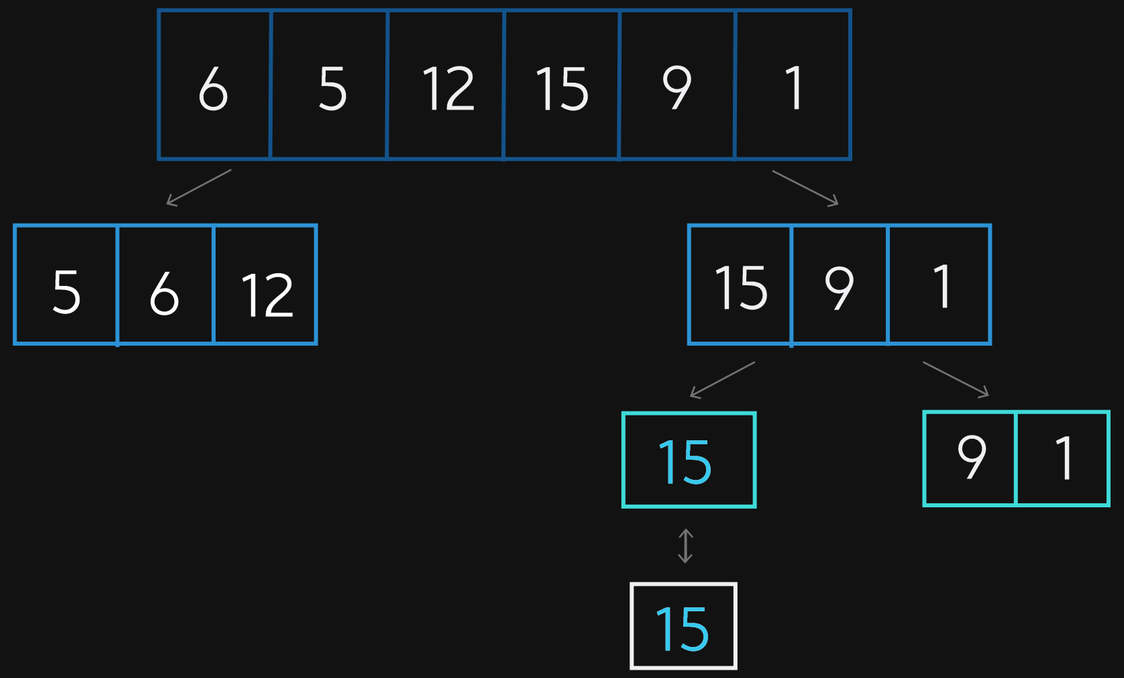
FUNÇÃO MERGE()



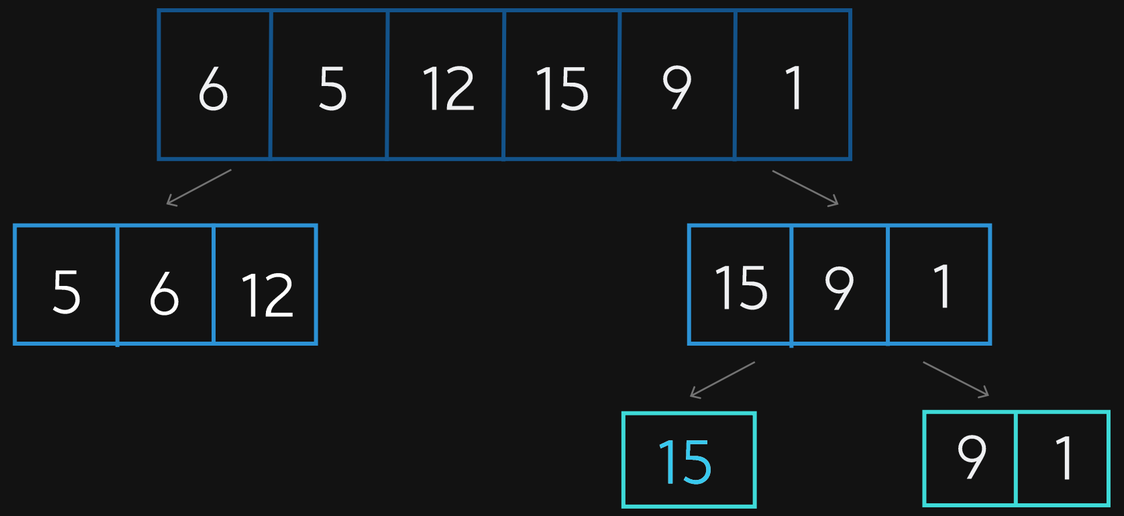
FUNÇÃO MERGE()



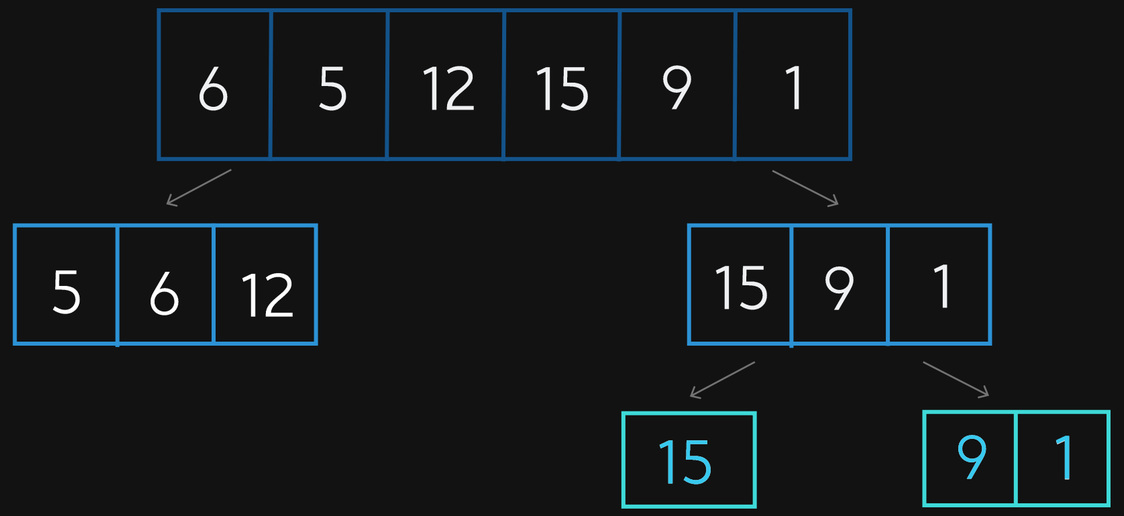
FUNÇÃO MERGE()



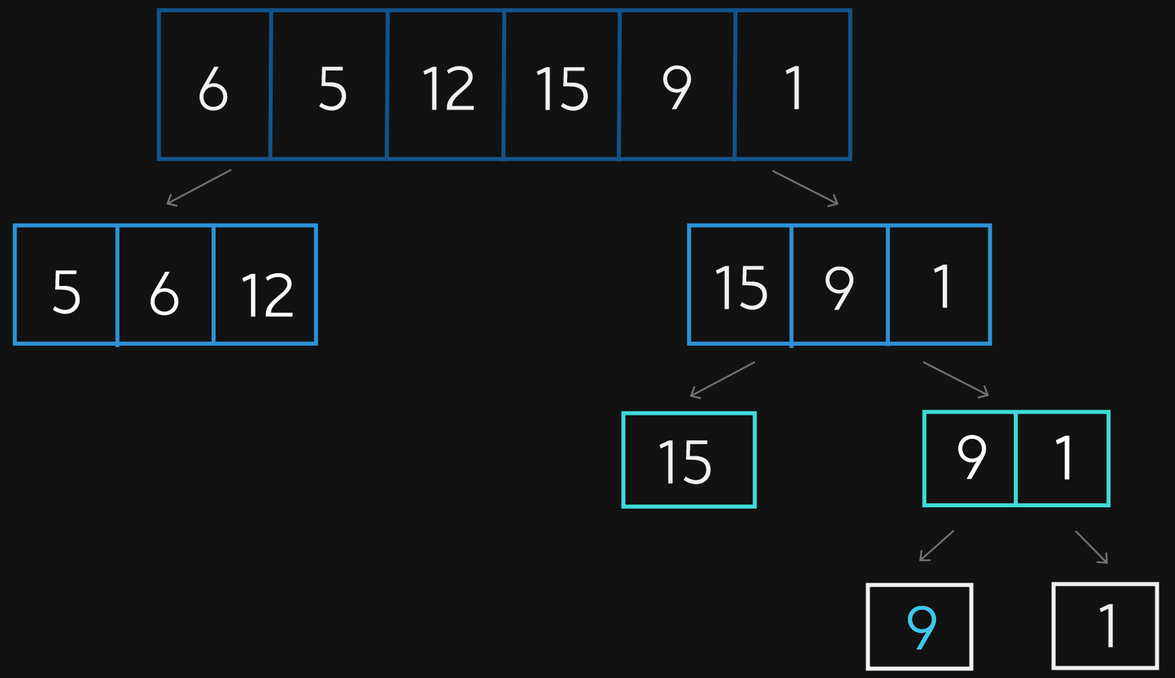
FUNÇÃO MERGE()



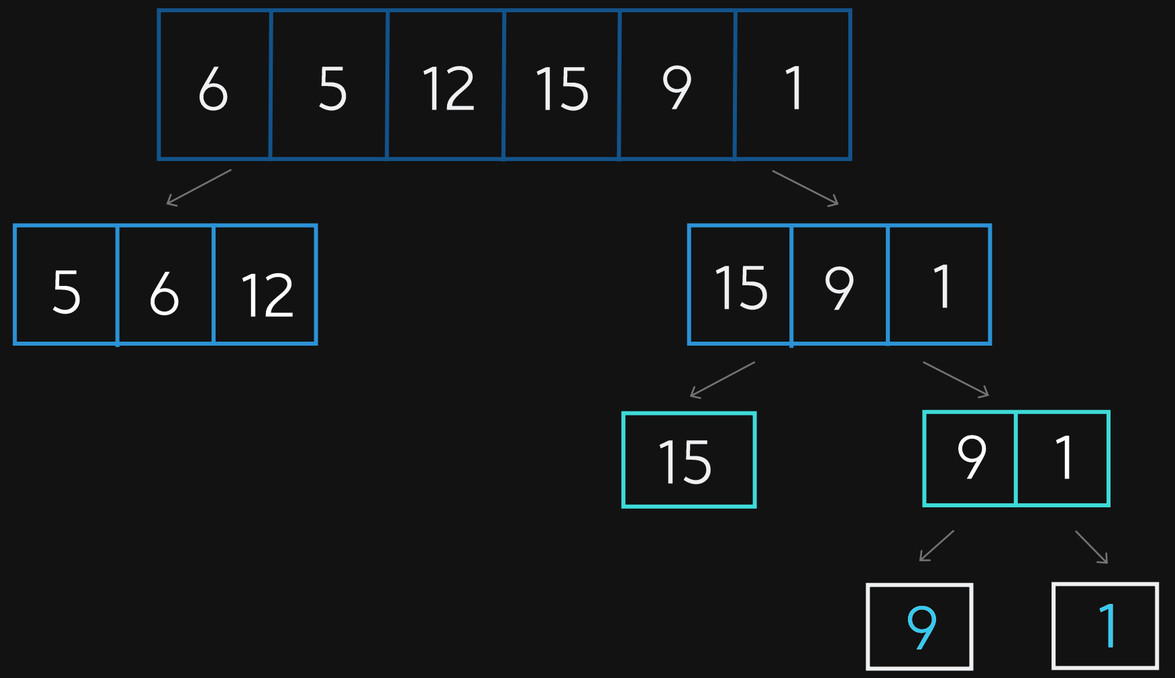
FUNÇÃO MERGE()



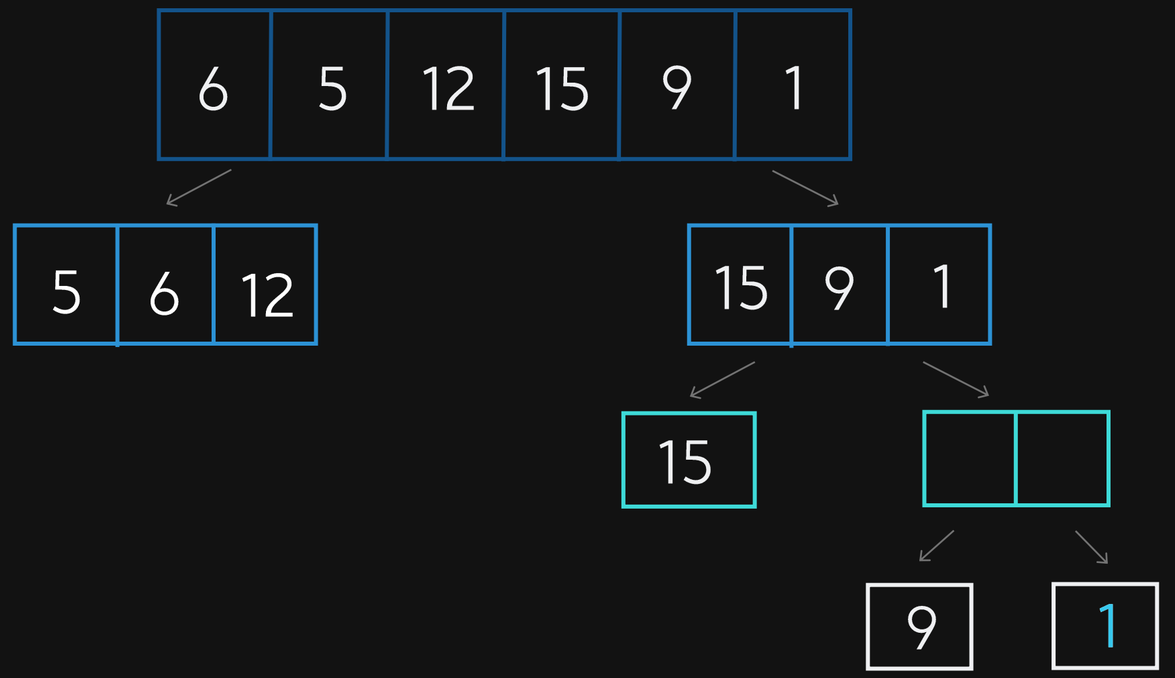
FUNÇÃO MERGE()



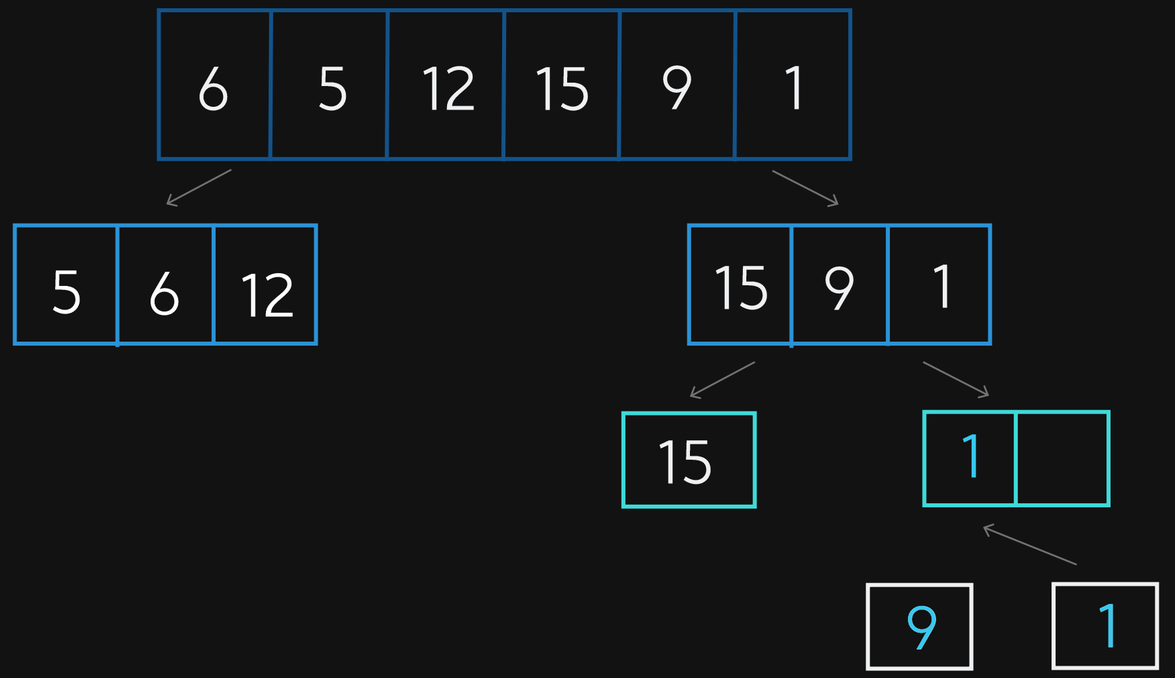
FUNÇÃO MERGE()



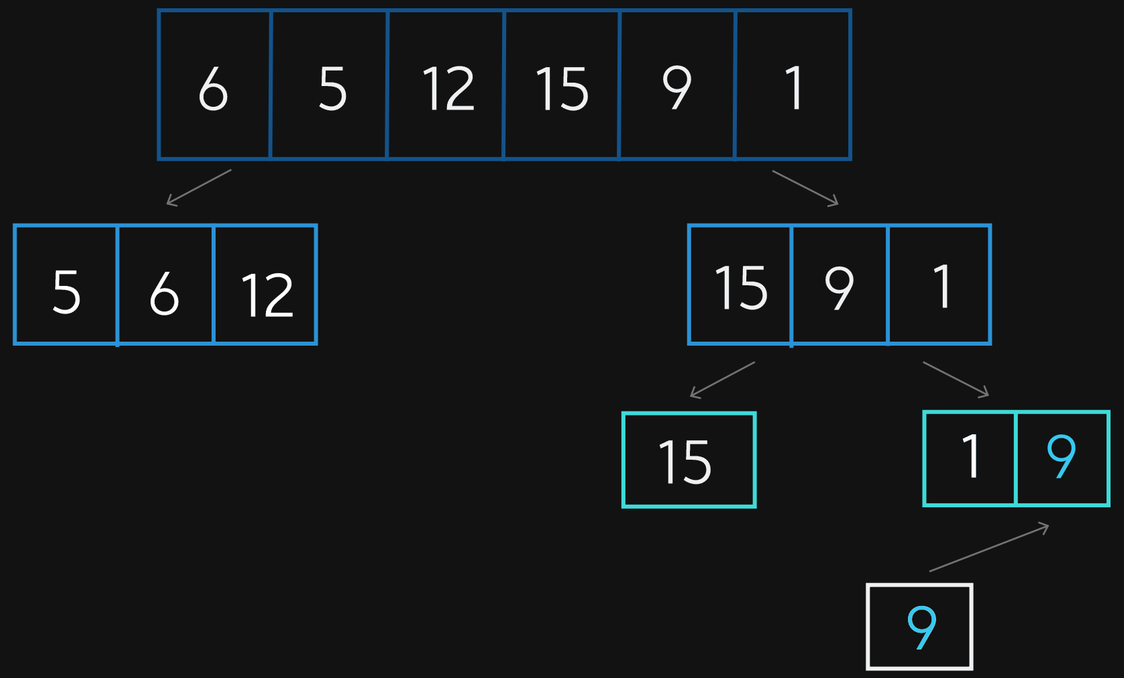
FUNÇÃO MERGE()



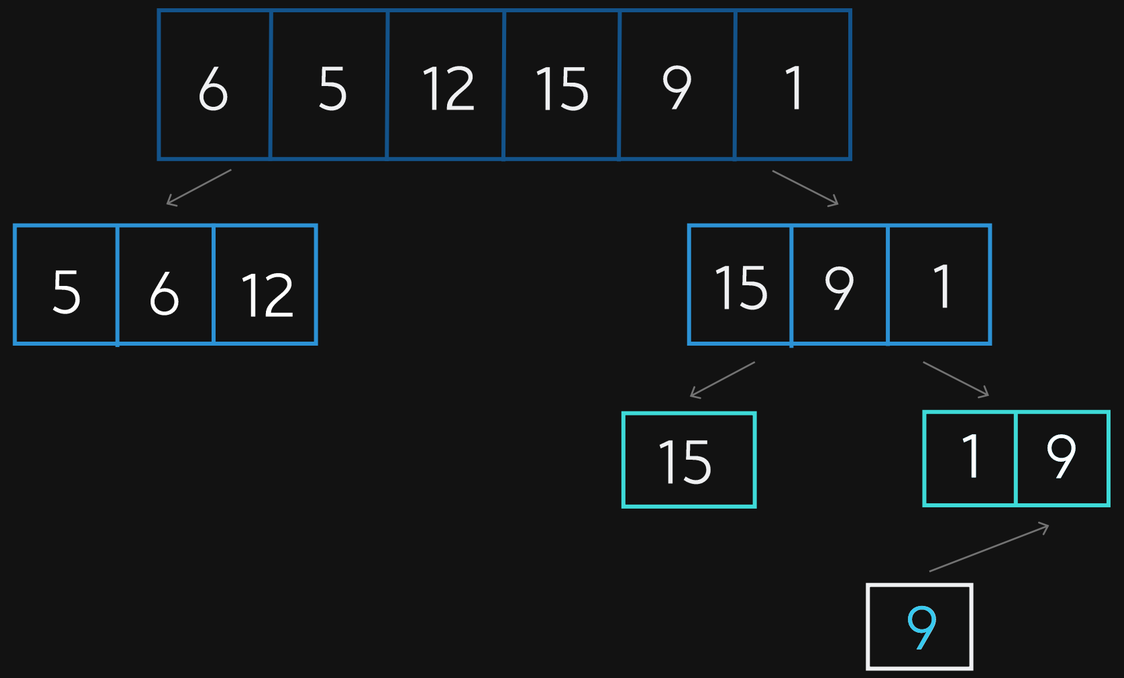
FUNÇÃO MERGE()



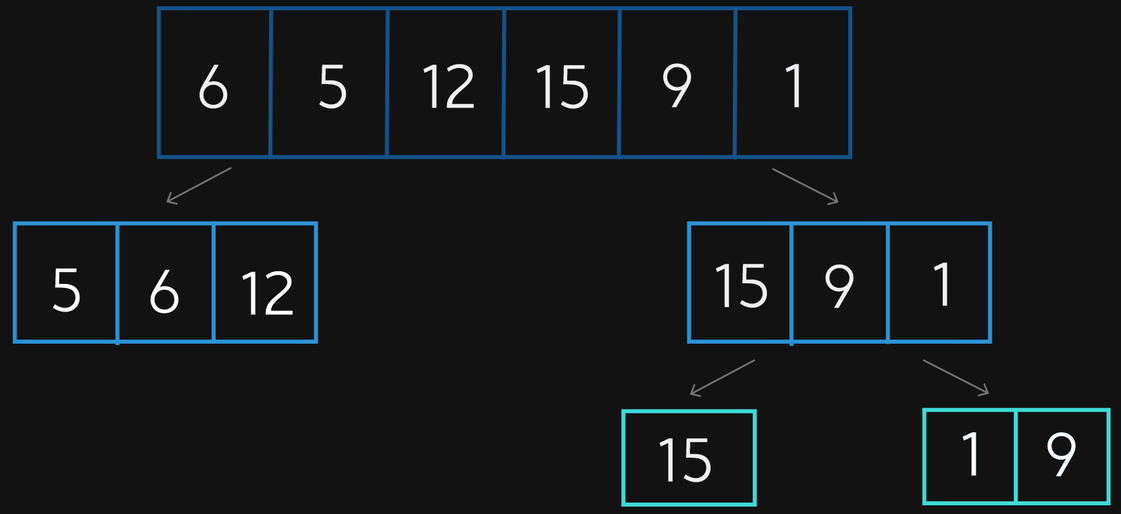
FUNÇÃO MERGE()



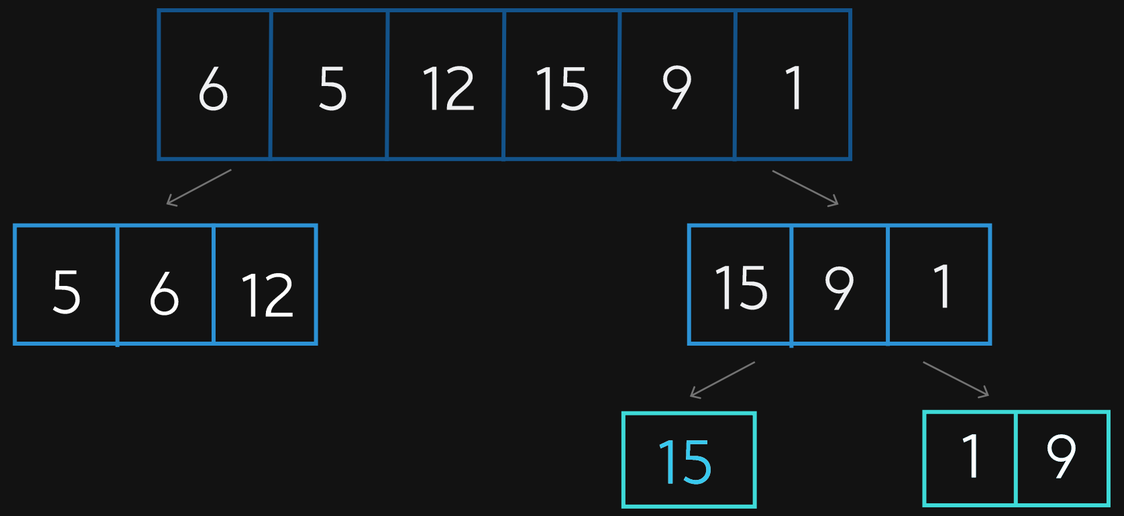
FUNÇÃO MERGE()



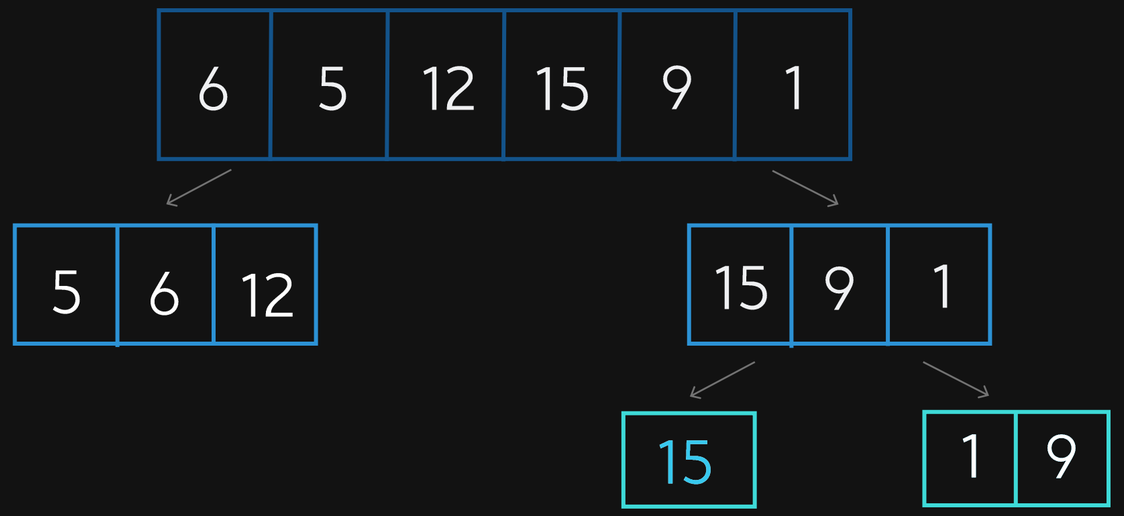
FUNÇÃO MERGE()



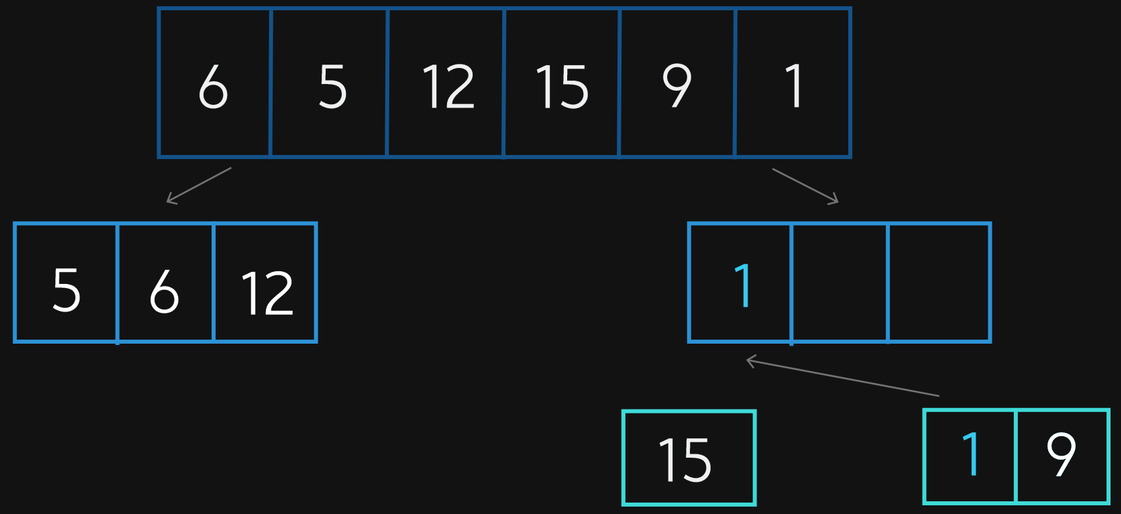
FUNÇÃO MERGE()



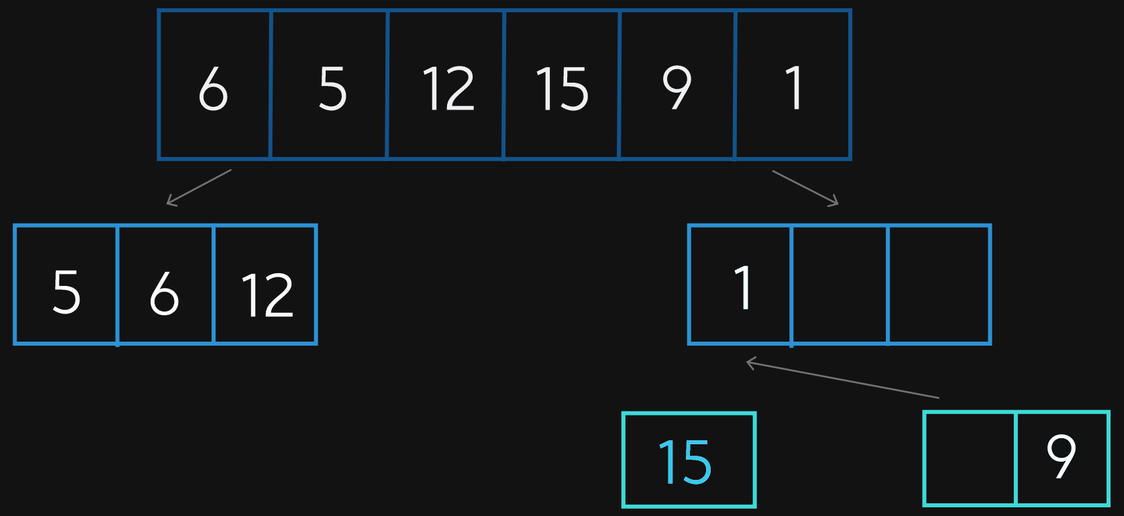
FUNÇÃO MERGE()



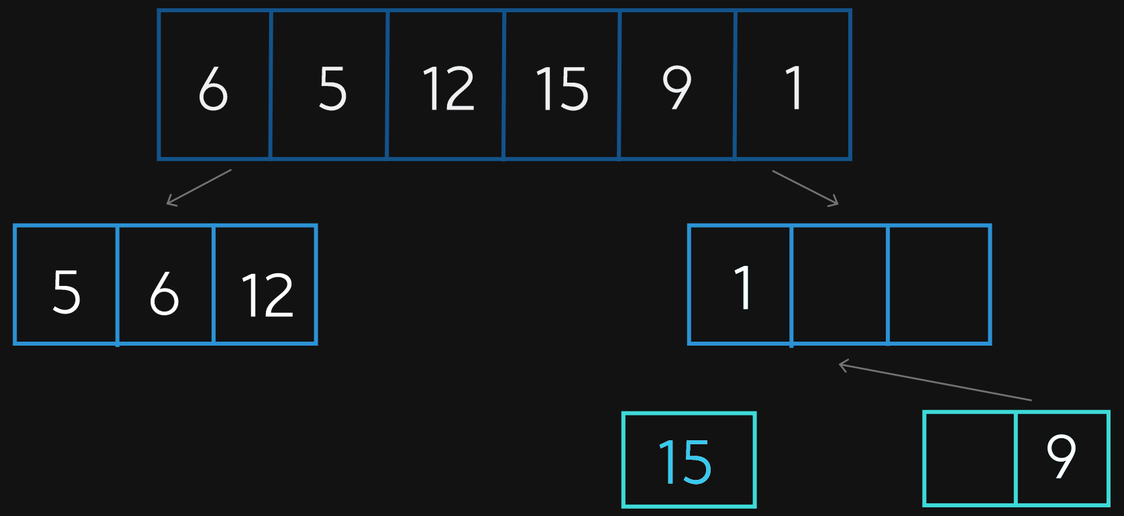
FUNÇÃO MERGE()



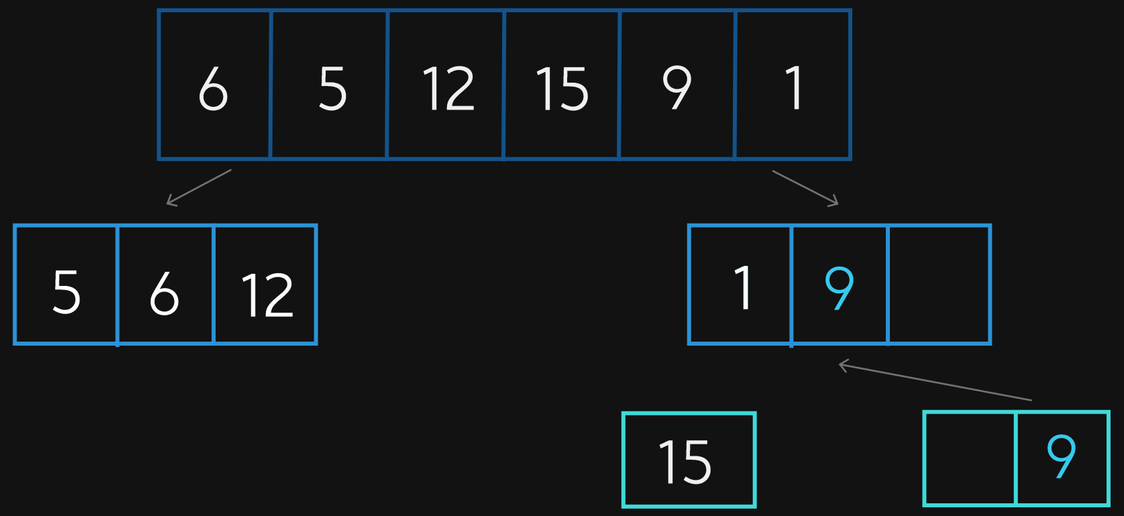
FUNÇÃO MERGE()



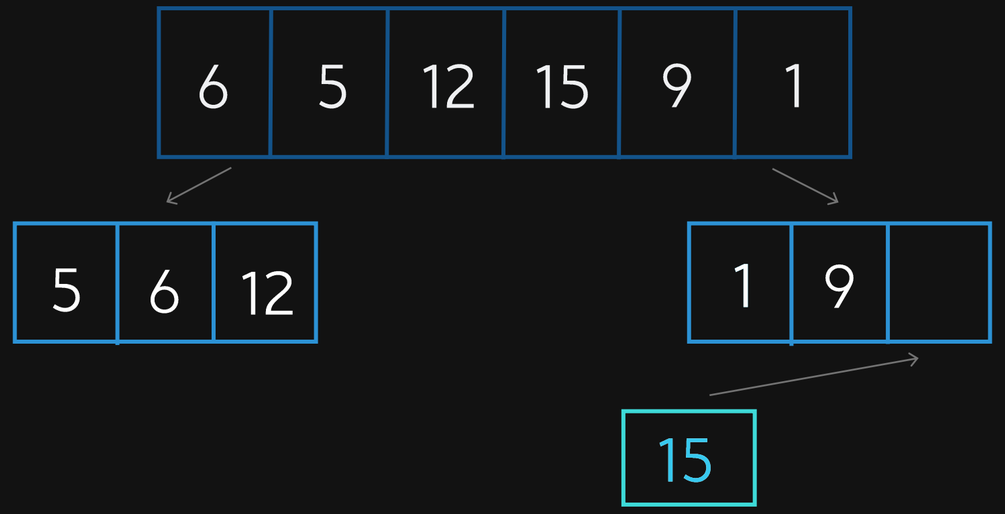
FUNÇÃO MERGE()



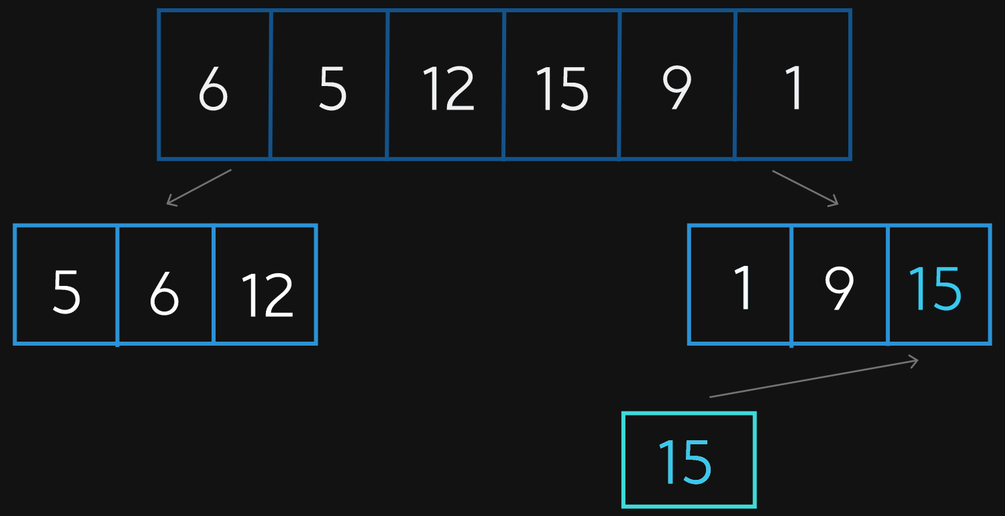
FUNÇÃO MERGE()



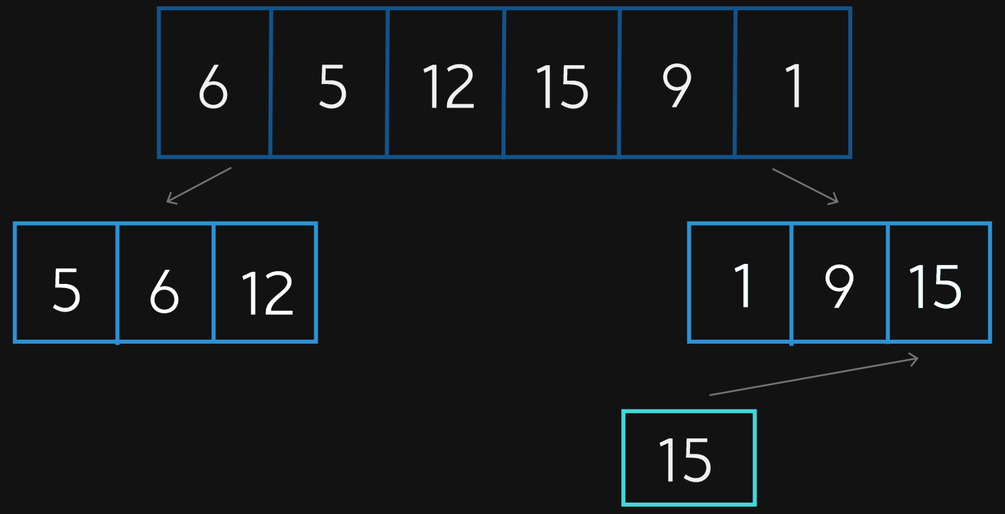
FUNÇÃO MERGE()



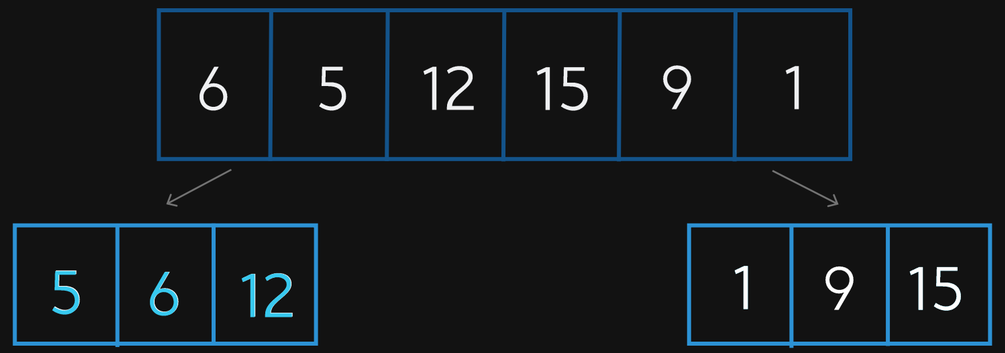
FUNÇÃO MERGE()



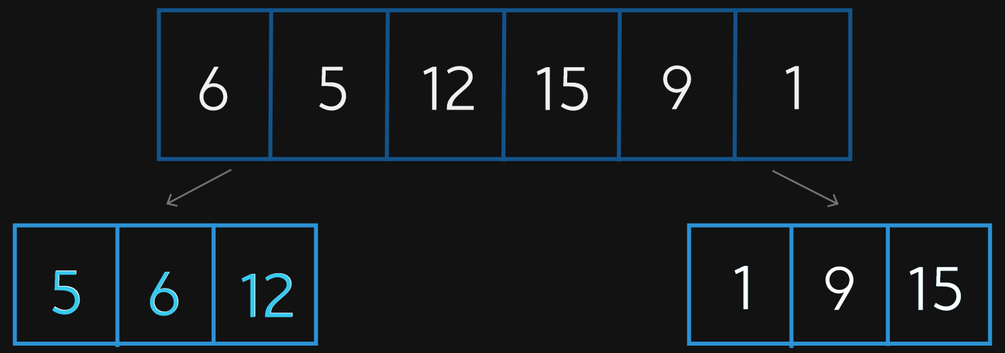
FUNÇÃO MERGE()



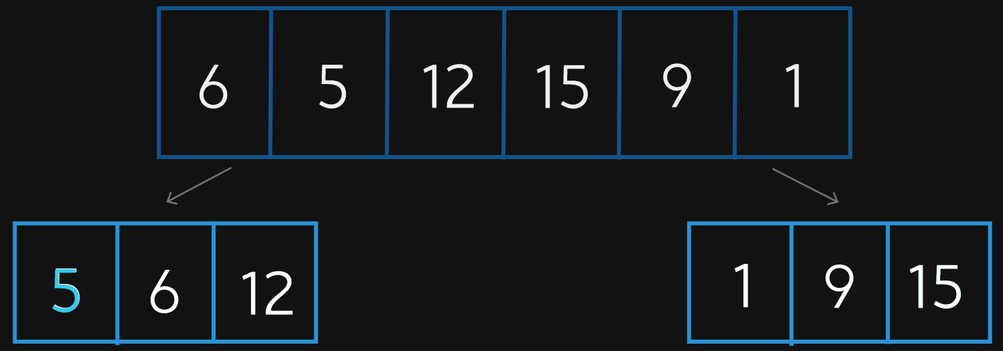
FUNÇÃO MERGE()



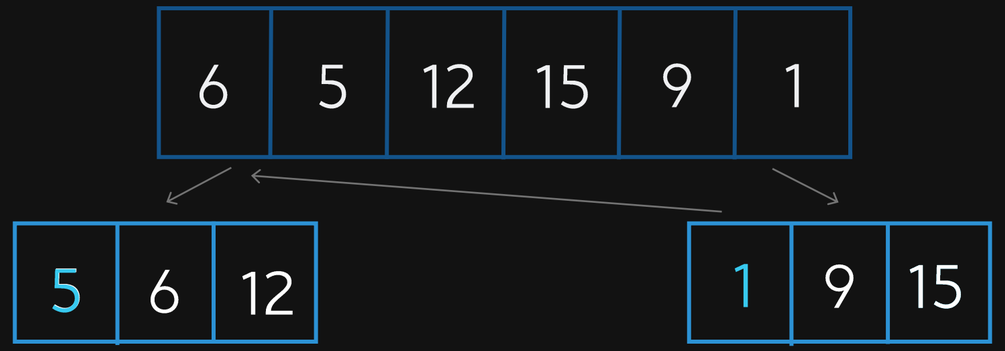
FUNÇÃO MERGE()



FUNÇÃO MERGE()



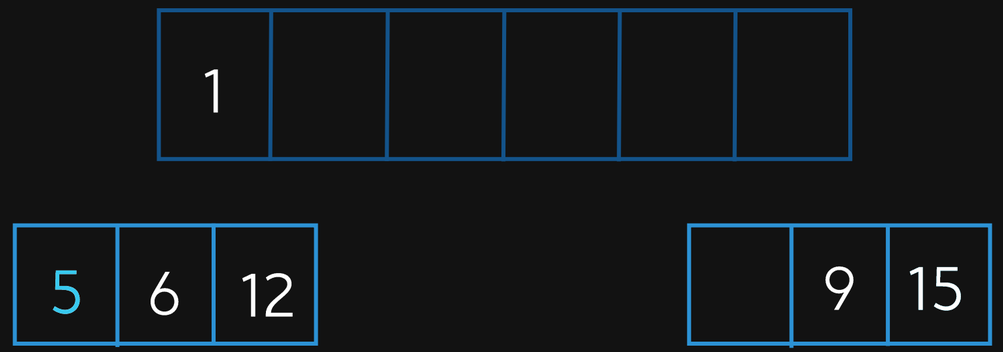
FUNÇÃO MERGE()



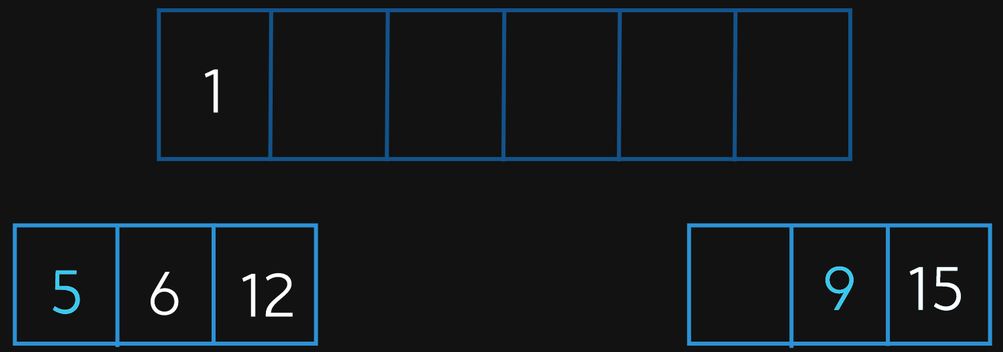
FUNÇÃO MERGE()

FUNÇÃO MERGE()

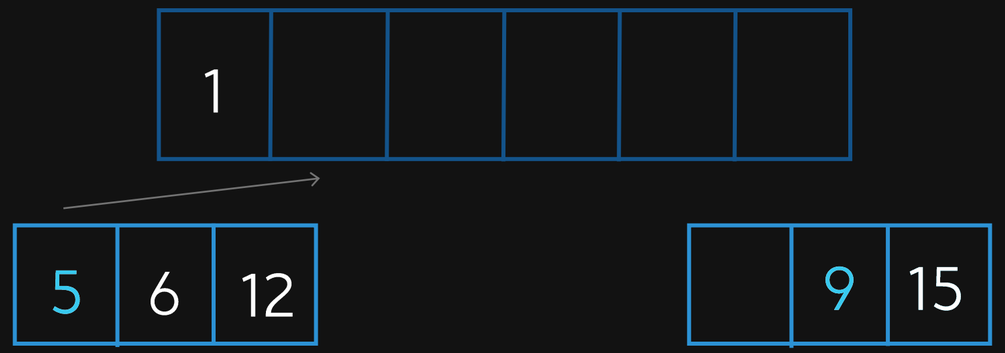
FUNÇÃO MERGE()



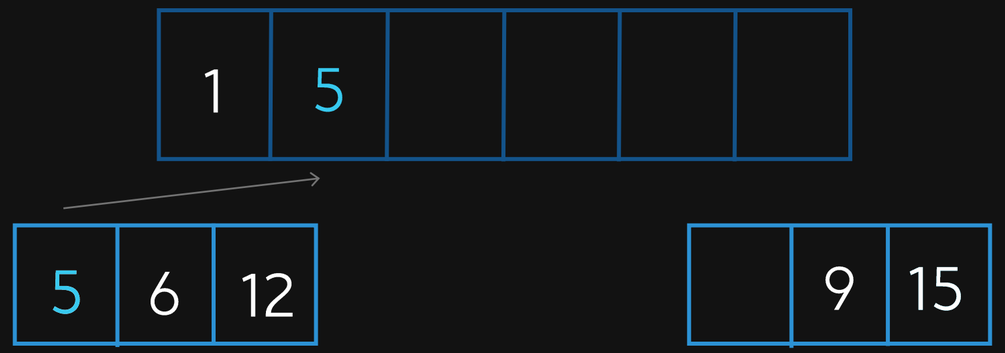
FUNÇÃO MERGE()



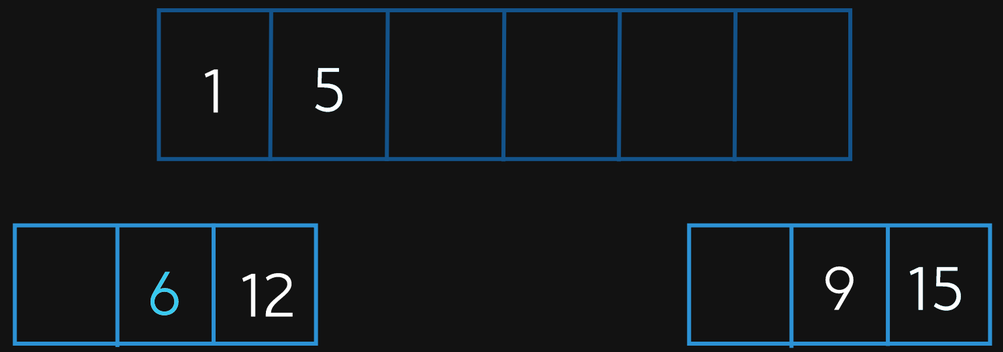
FUNÇÃO MERGE()



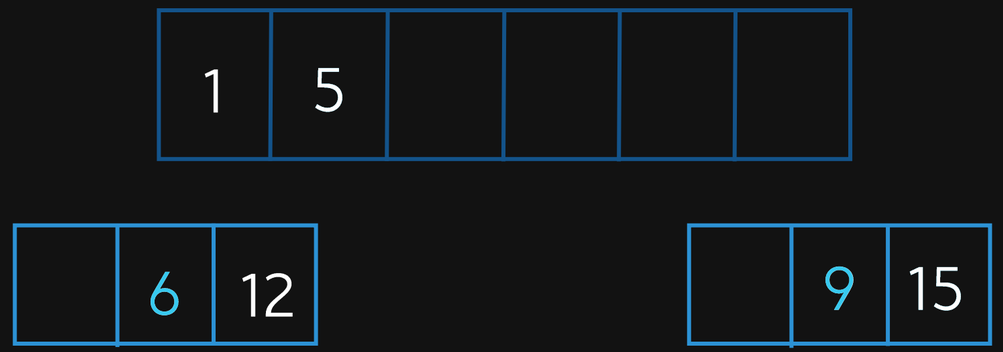
FUNÇÃO MERGE()



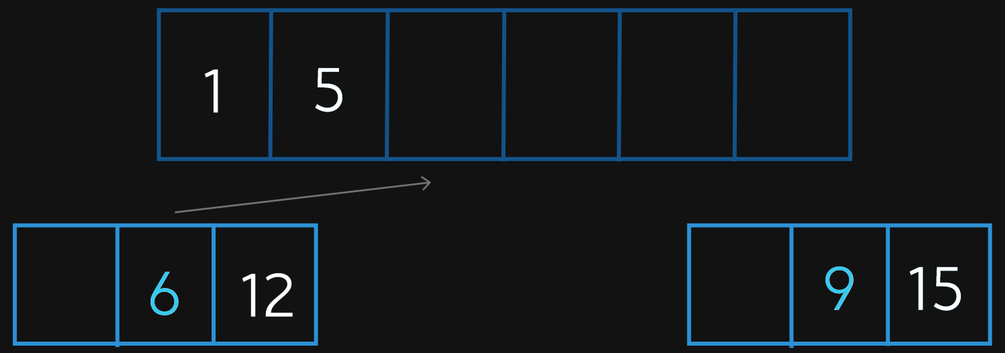
FUNÇÃO MERGE()



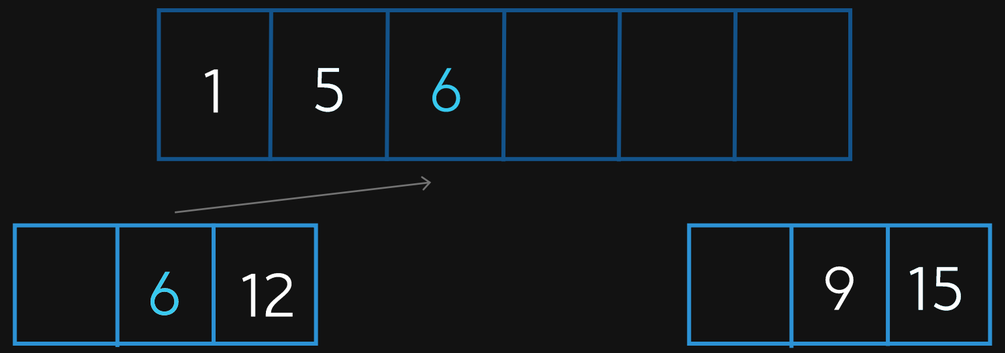
FUNÇÃO MERGE()



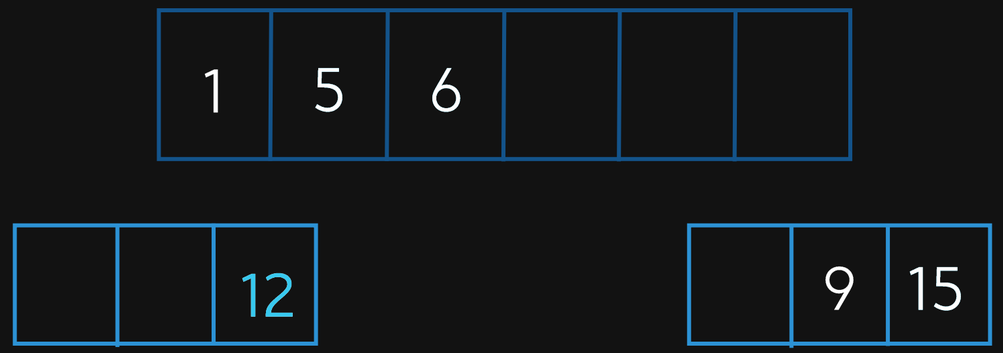
FUNÇÃO MERGE()



FUNÇÃO MERGE()

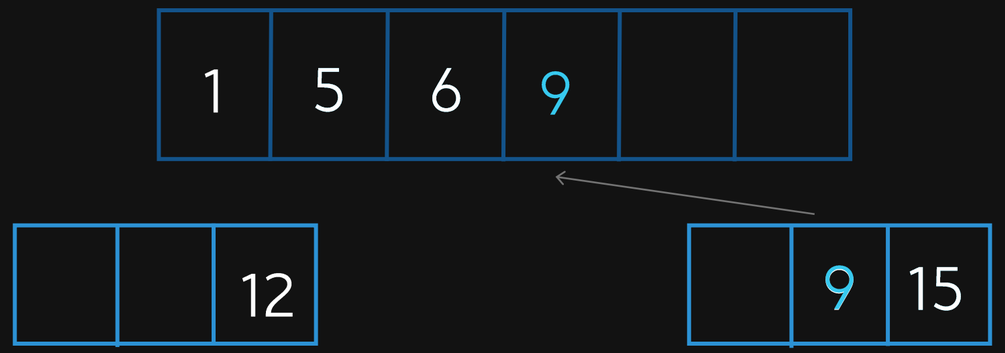


FUNÇÃO MERGE()

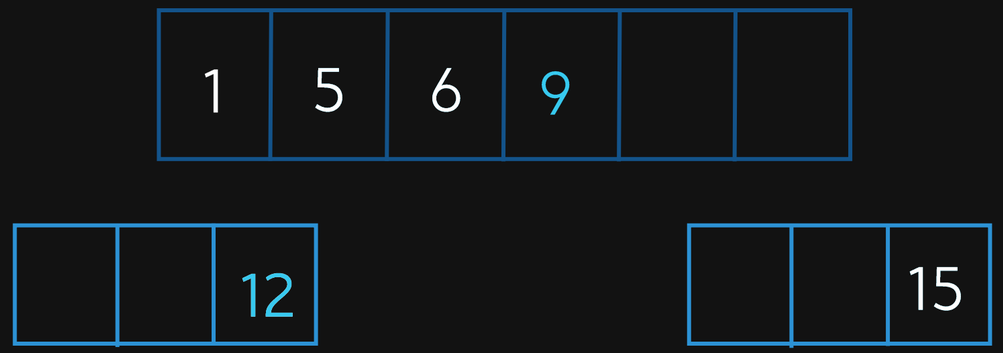


FUNÇÃO MERGE()

FUNÇÃO MERGE()

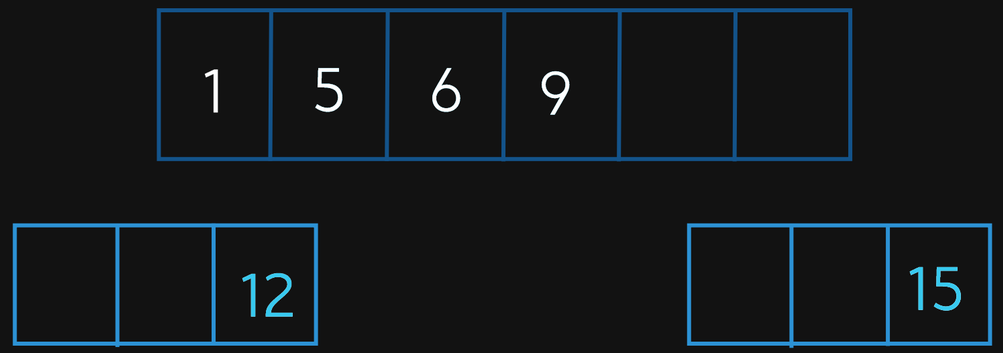


FUNÇÃO MERGE()



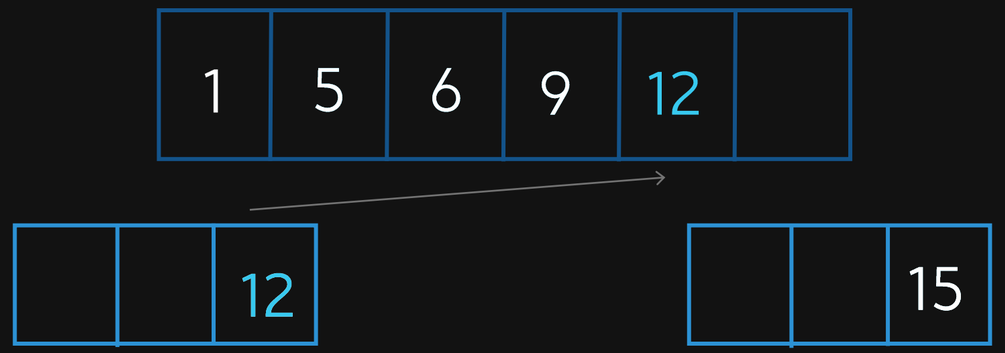
FUNÇÃO MERGE()

FUNÇÃO MERGE()

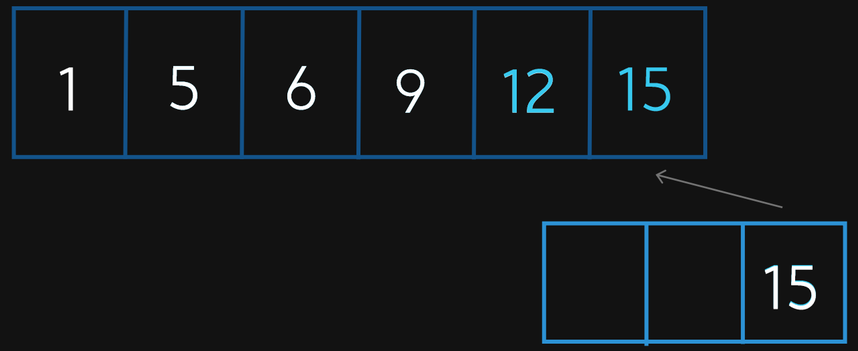


FUNÇÃO MERGE()

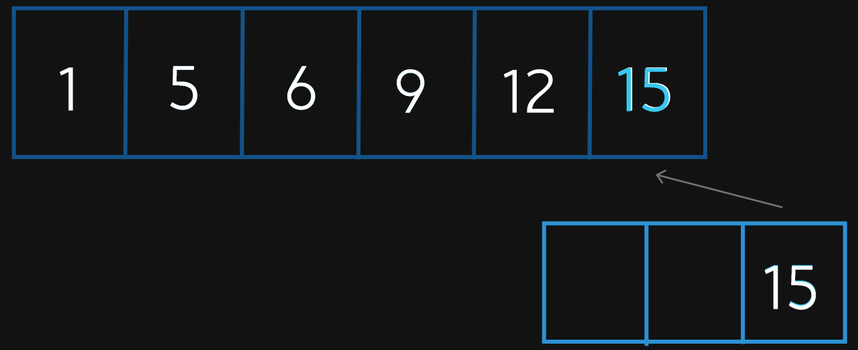
FUNÇÃO MERGE()



FUNÇÃO MERGE()



FUNÇÃO MERGE()

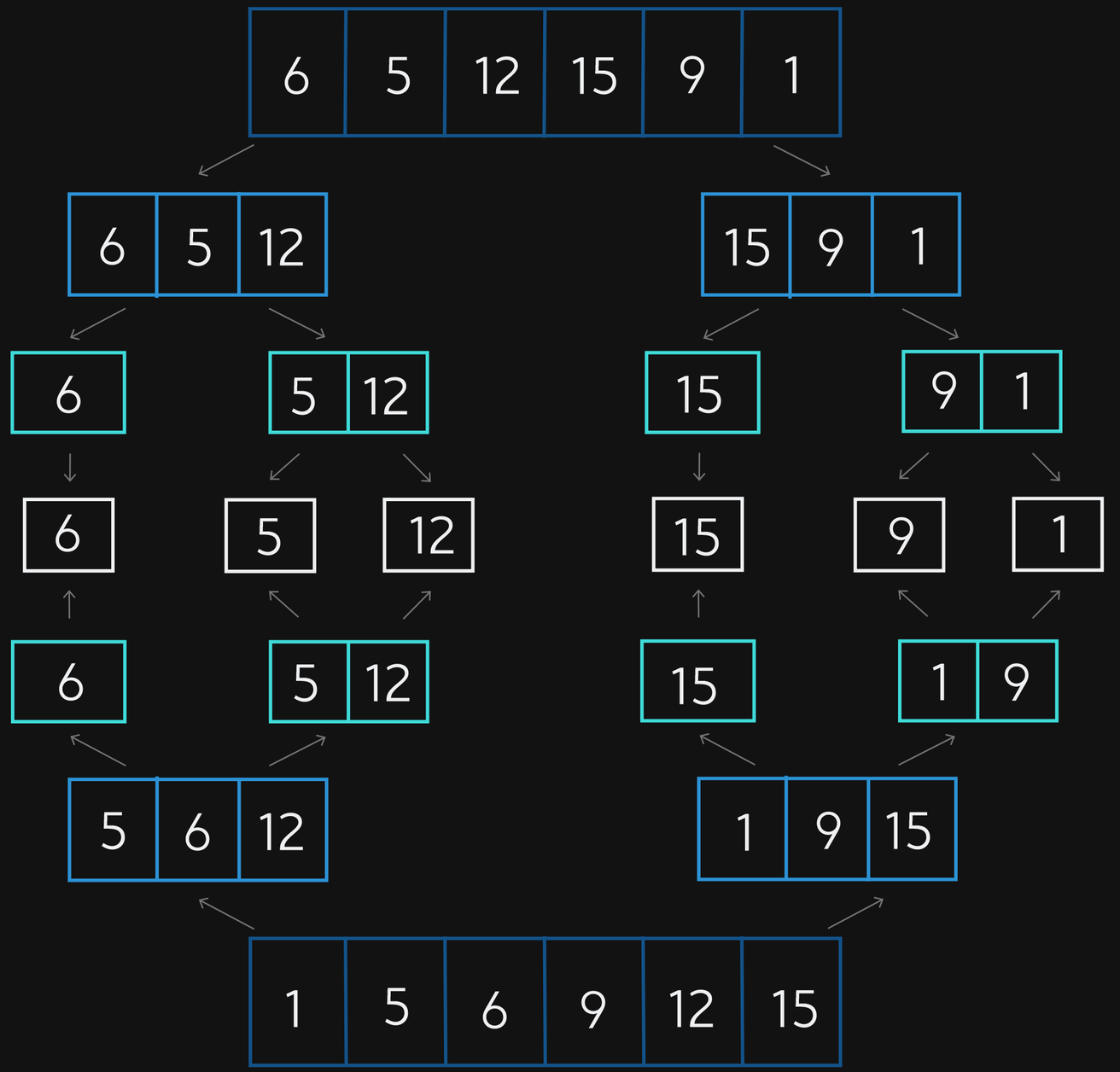


1	5	6	9	12	15

FUNÇÃO MERGE()

FUNÇÃO MERGE()

FUNÇÃO MERGE()



```
Merge(A, p, q, r)
  n1 = q − p + 1
  n2 = r − q
  sejam L[1..n1 + 1] e R[1..n2 + 1] novos arranjos
  for i = 1 to n1 do
      L[i] = A[p + i − 1]
  end for
  for j = 1 to n2 do
      R[j] = A[q + j]
  end for
  L[n1 + 1] = ∞
  R[n2 + 1] = ∞
  i = 1
  j = 1
  for k = p to r do
      if L[i] ≤ R[j] then
        A[k] = L[i]
        i = i + 1
      else
        A[k] = R[j]
        j = j + 1
      end if
  end for
```

```
MERGESORT(A, p, r):
    if p < r then
        q ← ⌊(p + r)/2⌋
        MERGESORT(A, p, q)
        MERGESORT(A, q + 1, r)
        MERGE(A, p, q, r)
    end if
end
```

PSEUDOCÓDIGO

ANÁLISE DE COMPLEXIDADE

Complexidade de Tempo

- O(n log n) onde n é o número total de elementos.
- Usa recursão e a técnica de "dividir e conquistar"
- O Teorema Mestre é utilizado para resolver a recorrência

$$T(n) = 2T\left(\frac{n}{2}\right) + \theta(n)$$

- A complexidade de tempo do Merge Sort é $\theta(n \log n)$

Complexidade de Espaço

- Espaço Adicional
- Fase de divisão
- O(n)

ANÁLISE DE COMPLEXIDADE

CEFET-MG

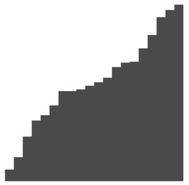
Visualização Merge Sort

Disciplina: Algoritmos e Estruturas de Dados I
Professor: Michel Pires
Curso: Engenharia de Computação
2024/1

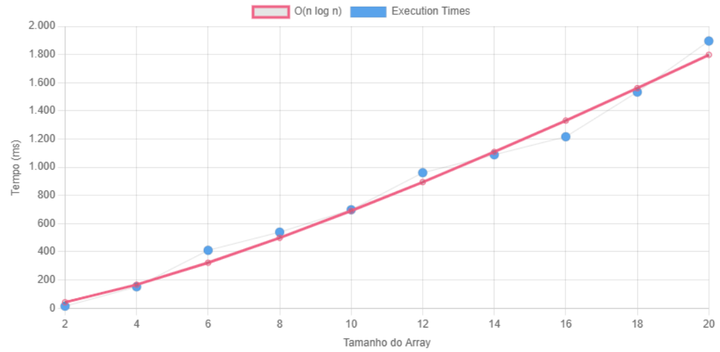
Array size: 20

Number of executions: 10

New Values



Execução	Tamanho do Array	Tempo de Execução (ms)
1	2	14.00
2	4	151.50
3	6	410.40
4	8	539.40
5	10	698.30
6	12	960.90
7	14	1088.30
8	16	1216.30
9	18	1533.10
10	20	1895.50



Desenvolvido por: Maira Lacerda, Maria Eduarda Teixeira e Sergio Henrique
Acesse o trabalho completo

AMBIENTE DE EXECUÇÃO

- Processador: Intel Core i7-1360P (18MB Cache, up to 5.00 GHz)
- Memória: 16GB 4800MHz LPDDR5 Memory Onboard
- Sistema Operacional: Ununtu 24.04 LTS - 64 bits

DADOS DE ENTRADA

- Aleatórios
- Crescentes
- Decrescentes
- Quase crescentes
- Quase decrescentes

MODELOS DE APLICAÇÃO

MODELOS DE APLICAÇÃO

Vantagens do Merge Sort

- Eficiência Assintótica
- Estabilidade
- Adaptação a Diferentes Estruturas de Dados
- Facilidade de Implementação

MODELOS DE APLICAÇÃO

Uso de Memória

Desvantagens do Merge Sort

Desempenho em certos casos

MODELOS DE APLICAÇÃO

EXEMPLOS PRÁTICOS

- Ordenação de Grandes Arquivos de Dados
- Rede de Telecomunicações
- Integração de Dados
- Aplicações em Bioinformática
- Aplicações Web

GENERALIZAÇÃO

1

Custom Sequence

2

Field Based

3

K-Way Merge Sort

4

Parallel Merge Sort

Linguagens Compiladas vs. Interpretadas

Linguagens Compiladas

Simplicidade e eficiência - Gestão precisa da memória através de ponteiros

Baixo nível, orientada objetos e recursos de alto nível - Como utilização do vector, em que o usuário não possui manipulação total

Desenvolvida pela Microsoft, poderosa e com recursos de alto nível - Pode perder desempenho rodando em outros S.O.

Baixo nível, permite manuseamento de memória (qntd. utilizada) - Busca trazer o melhor da linguagem interpretada e compilada. Levando performance e produtividade.

Linguagens Compiladas vs. Interpretadas

Linguagens Interpretadas

Alto nível, JVM que converte os bytecodes em tempo de execução. Acesso a memória restrito pelo usuário.

Roda em qualquer navegador, alta taxa de leitura de dados com a tecnologia assíncrona do Node.js - Utilizando uma thread somente para o processamento de dados da execução, enquanto outras realizam a execução de outras tarefas - Alta complexidade.

Alto nível. Interpretado em just-in-time(JIT).

Alto nível - Apesar da facilidade de implementação como grande ecossistema de bibliotecas, sua execução perde desempenho devido a interpretação em tempo de execução e acesso a memória e manipulação de vetores sem controle do usuário.

Linguagens Compiladas vs. Interpretadas

Porcentagem - RANDOM 1

47%

<

94%

>

50%

>

51,26%

>

89%

>

245%

<

79,4%

>

72%

>

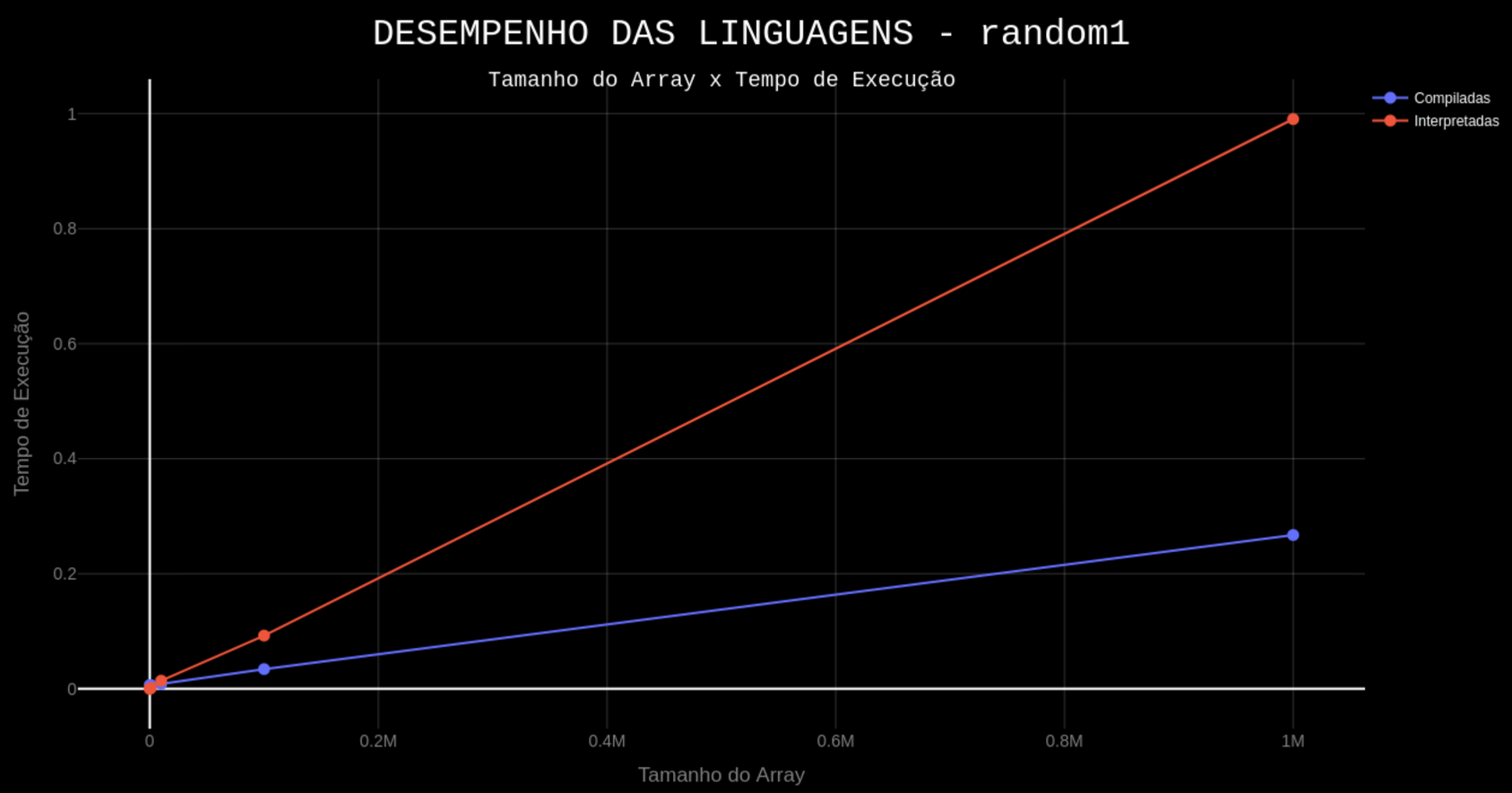
Linguagens Compiladas vs. Interpretadas

Linguagens Compiladas

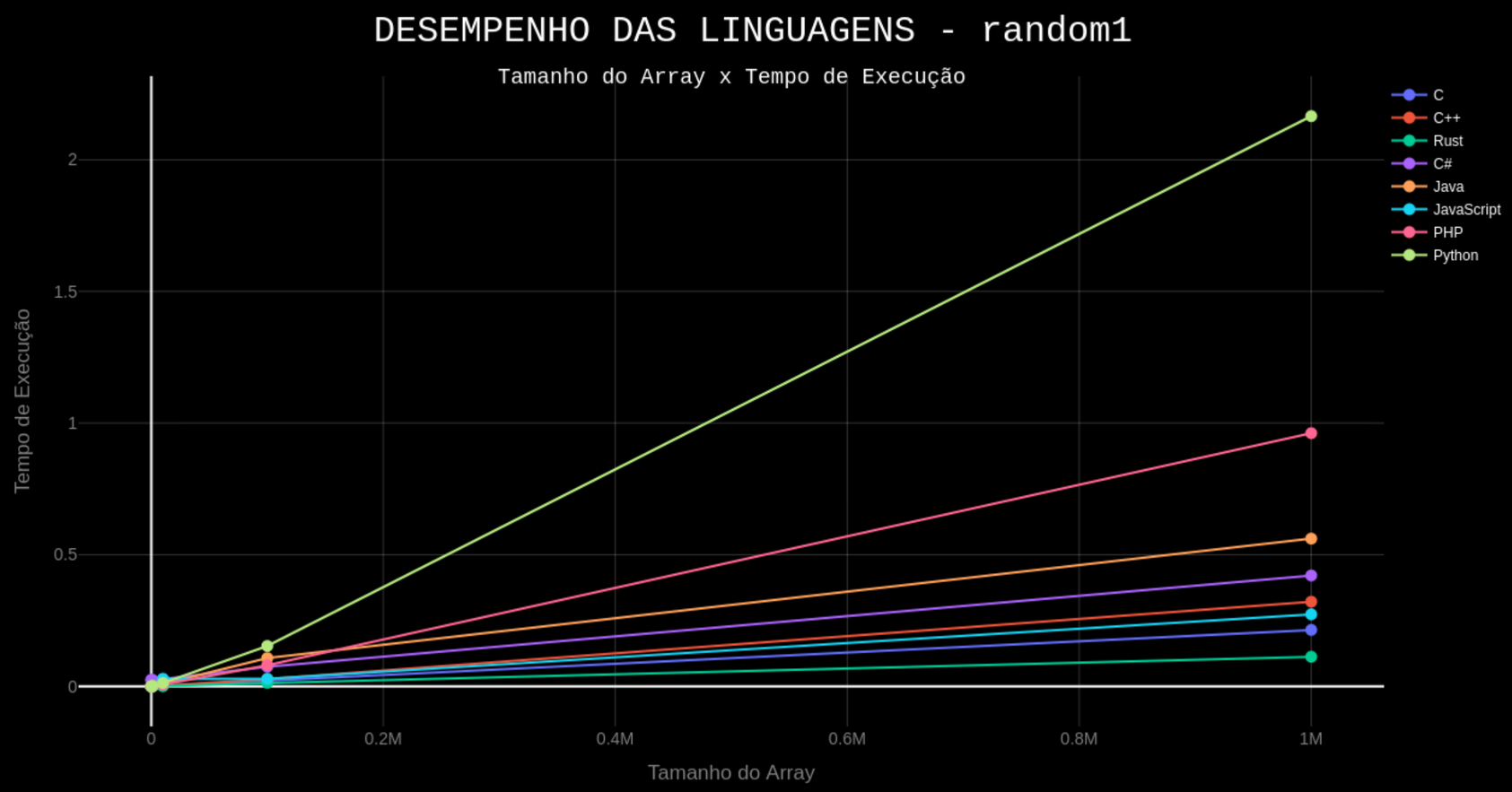
- C
- C++
- C#
- Rust

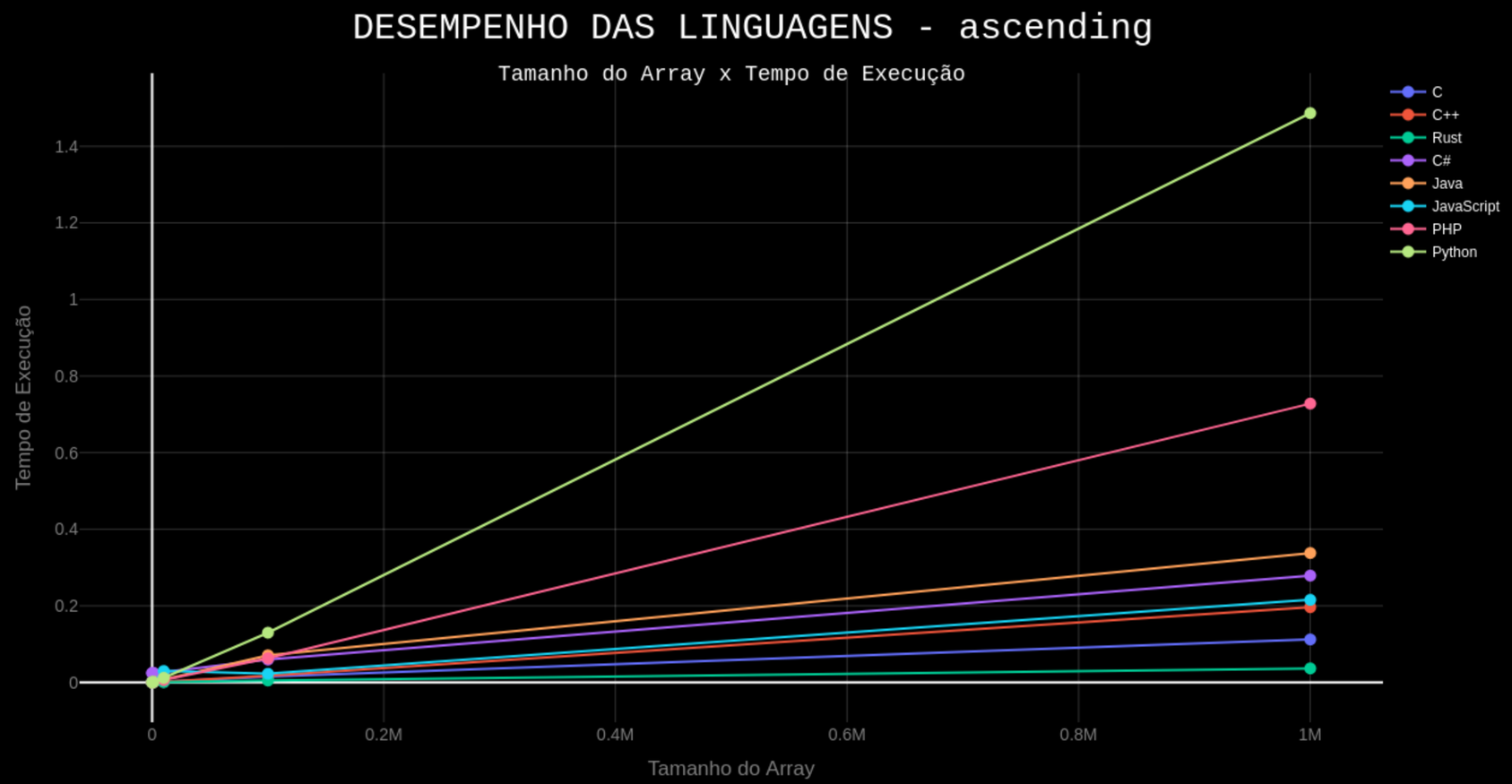
Linguagens Interpretadas

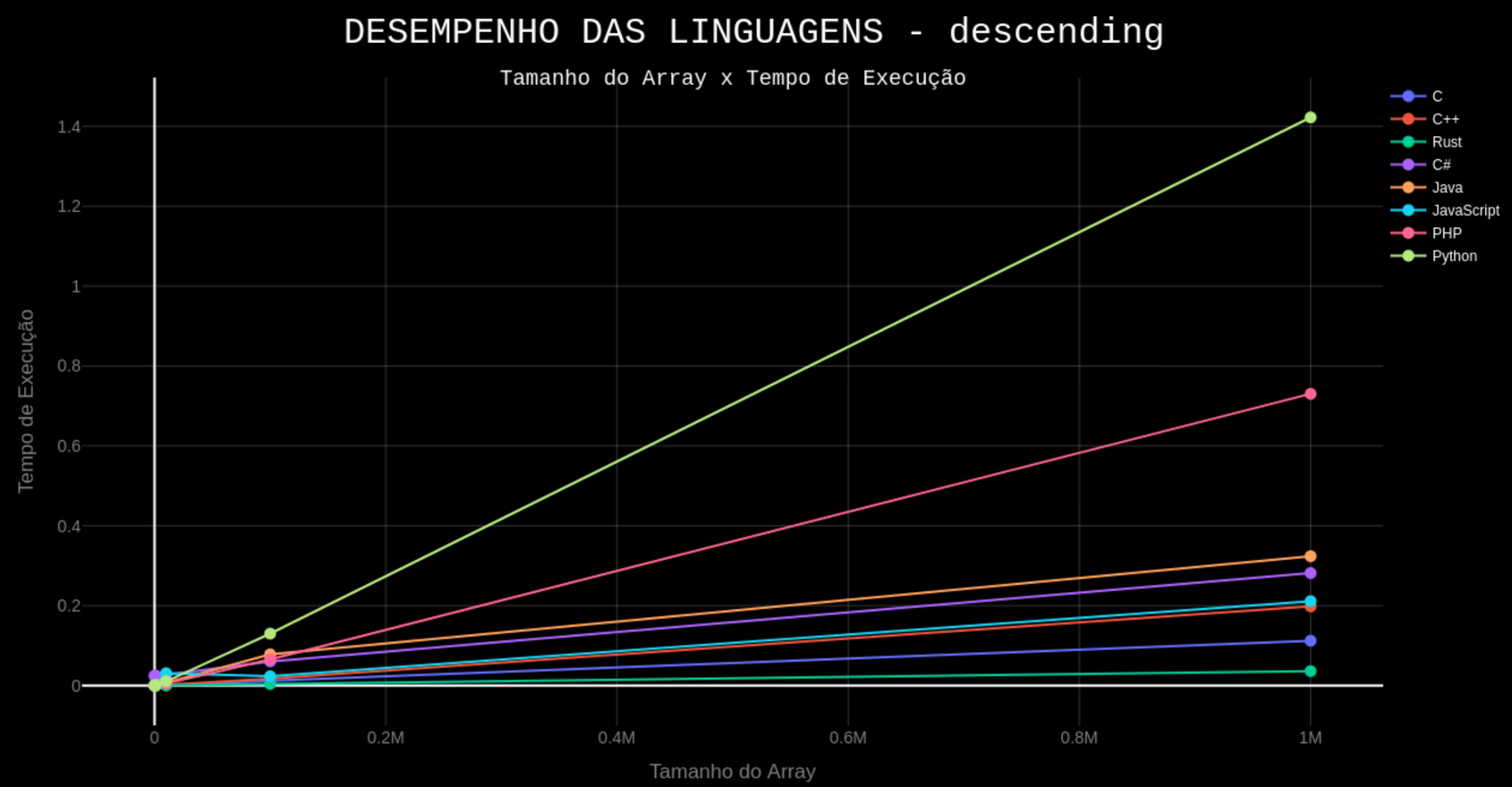
- Java
- Java Script
- PHP
- Python

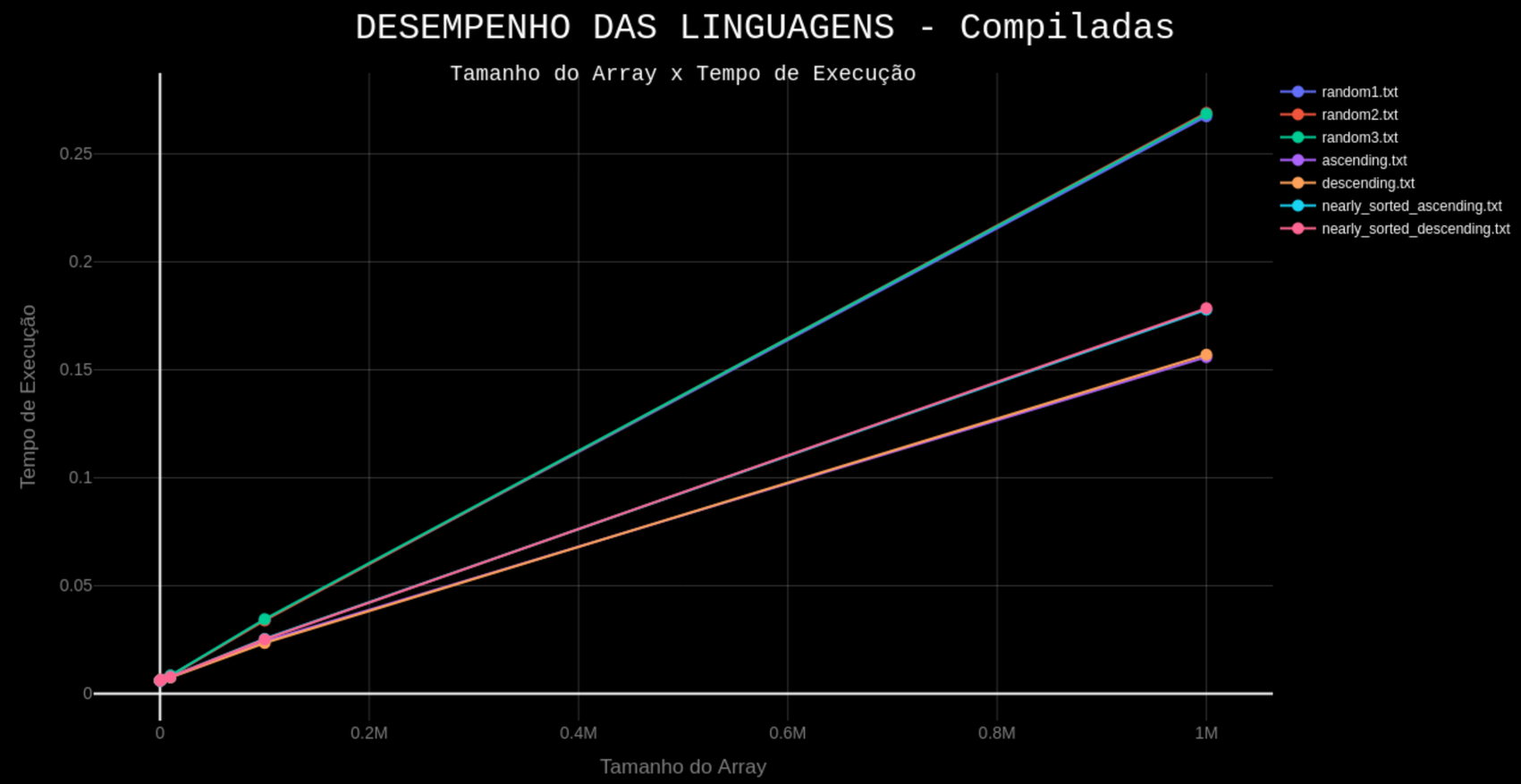


RESULTADOS

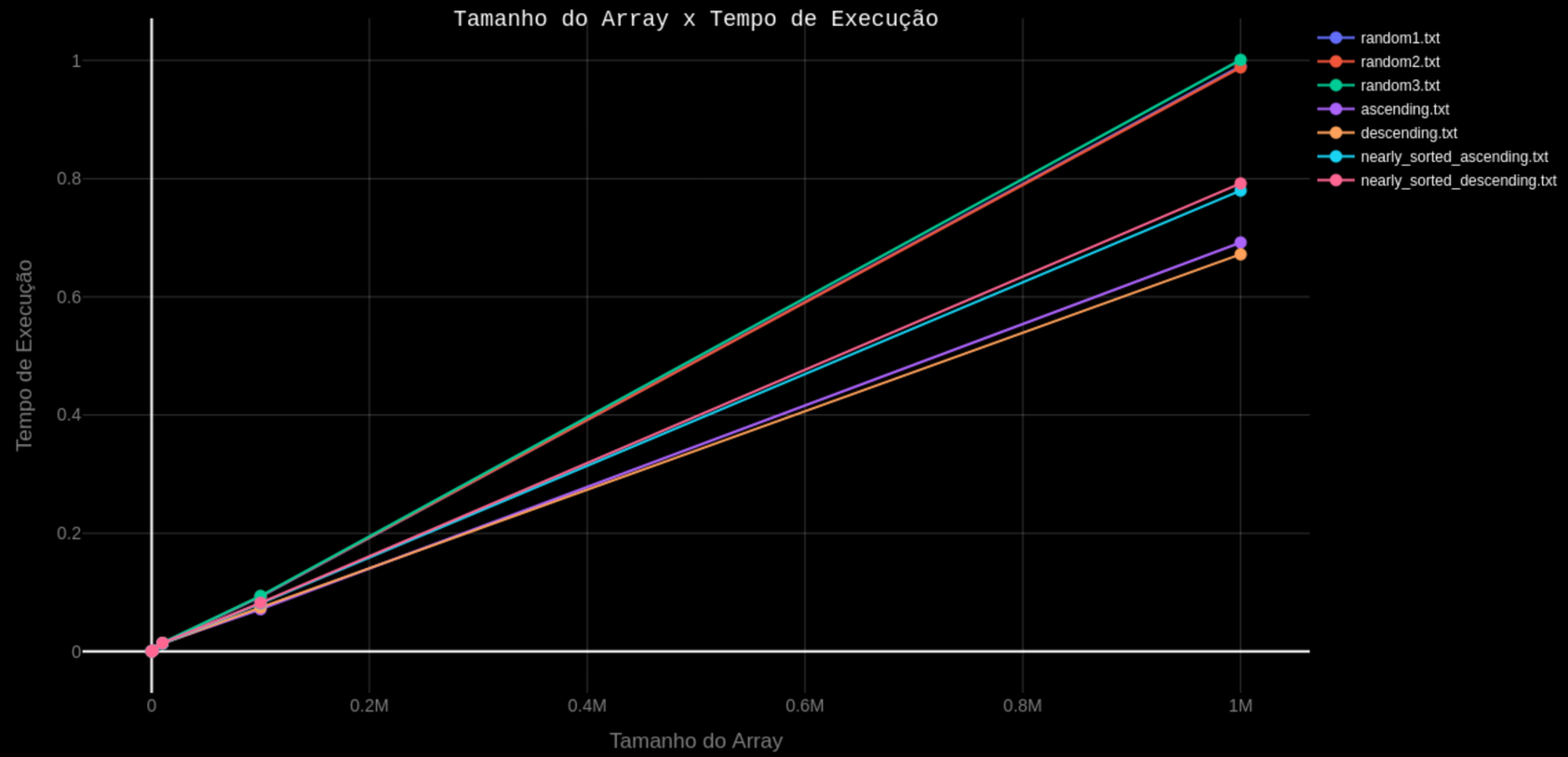

DESEMPENHO DAS LINGUAGENS - random1
Tamanho do Array x Tempo de Execução
Tempo de Execução
Tamanho do Array
0
0.5
1
1.5
2
0
0.2M
0.4M
0.6M
0.8M
1M
C
C++
Rust
C#
Java
JavaScript
PHP
Python


DESEMPENHO DAS LINGUAGENS - ascending
Tamanho do Array x Tempo de Execução
Tempo de Execução
1.4
1.2
1
0.8
0.6
0.4
0.2
0
0
0.2M
0.4M
0.6M
0.8M
1M
Tamanho do Array
C
C++
Rust
C#
Java
JavaScript
PHP
Python


DESEMPENHO DAS LINGUAGENS - descending
Tamanho do Array x Tempo de Execução
Tempo de Execução
1.4
1.2
1
0.8
0.6
0.4
0.2
0
0
0.2M
0.4M
0.6M
0.8M
1M
Tamanho do Array
C
C++
Rust
C#
Java
JavaScript
PHP
Python


DESEMPENHO DAS LINGUAGENS - Compiladas
Tamanho do Array x Tempo de Execução
Tempo de Execução
0.25
0.2
0.15
0.1
0.05
0
0
0.2M
0.4M
0.6M
0.8M
1M
Tamanho do Array
random1.txt
random2.txt
random3.txt
ascending.txt
descending.txt
nearly_sorted_ascending.txt
nearly_sorted_descending.txt

DESEMPENHO DAS LINGUAGENS - Interpretadas



CONCLUSÃO

- Visão mais aprofundada sobre algoritmos de ordenação e como podem ser implementados em diferentes linguagens
- Diferenças de desempenho entre linguagens e as diferenças que afetam o desempenho
- As generalizações e como elas são usadas para otimizar o desempenho em diferentes cenários

DÚVIDAS

Porque mesmo que o Merge Sort tenha complexidade de tempo O(n log n) no pior, melhor e caso médio, ele apresentou uma pequeníssima diferença de tempo de execução entre os diferentes tipos de arquivos de entrada, como os aleatórios e os já ordenados?

Como as generalizações do Merge Sort podem ser usadas para otimizar o desempenho do algoritmo em diferentes cenários?

REFERÊNCIAS

[1] D. E. Knuth, The Art of Computer Programming, Vol. 3: Sorting and Searching, 2nd ed.
[2] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, Algoritmos, 3rd ed.
[3] M. Jeon and D. Kim, ”Parallel Merge Sort with Load Balancing,” International Journal of Parallel Programming, vol. 31, no. 1, pp. 22-45, Feb. 2003
[4] J. Doe and A. Smith, ”Comparative of Advanced Sorting Algorithms (Quick Sort, Heap Sort, Merge Sort, Intro Sort, Radix Sort) Based on Time and Memory Usage,” 2021. [Online].
[5] A. B. Author, C. D. Contributor, and E. F. Researcher, ”Performance analysis of merge sort algorithms,” in Proceedings of the International Conference on Electronics and Sustainable Communication Systems (ICESC 2020), IEEE Xplore, 2020, pp. 123-130
[6] Z. Marszałek, ”Parallelization of Modified Merge Sort Algorithm,” Symmetry, vol. 9, no. 9, pp. 1-12, 2017. [Online].
[7] G. H. Researcher and I. J. Developer, ”Efficient Parallel Merge Sort for Fixed and Variable Length Keys,” 2012. [Online]
[8] L. M. Researcher and N. O. Analyst, ”Dynamic Memory Adjustment for External Mergesort,” 2010. [Online]
[9] P. Q. Engineer, ”Speeding Up External Mergesort,” 2009. [Online]
[10] A. Inkeri Verkamo, ”Performance comparison of distributive and mergesort as external sorting algorithms,” Journal of Systems and Software, vol. 12, no. 4, pp. 315-320, 1989. [Online]
[11] Prof. Tulio Toffolo, ”Ordenação: Merge Sort”- UFOP.
[12] Robert Sedgewick and Kevin Wayne, ”Algorithms,” 2010.[Online]

INTEGRANTES

- Maíra Beatriz de Almeida Lacerda
- Maria Eduarda Teixeira Souza
- Sergio Henrique Quedas Ramos

OBRIGADO PELA ATENÇÃO!