Ordenação com Mergesort

Projeto e Análise de Algoritmos

Bruno Prado

Departamento de Computação / UFS

Introdução

- O que é o Mergesort?
  - É um algoritmo de ordenação estável criado pelo cientista John von Neumann no ano de 1945

Introdução

- O que é o Mergesort?
  - É um algoritmo de ordenação estável criado pelo cientista John von Neumann no ano de 1945
  - Utiliza a estratégia de Divisão e Conquista que processa a entrada em partes menores

Introdução

- Estratégia de Divisão e Conquista
  1. Etapa de divisão do problema
     - Subproblemas menores

Introdução

- Estratégia de Divisão e Conquista
  1. Etapa de divisão do problema
     - Subproblemas menores
  2. Resolver os subproblemas
     - As soluções são mais simples

Introdução

- Estratégia de Divisão e Conquista
  1. Etapa de divisão do problema
     - Subproblemas menores
  2. Resolver os subproblemas
     - As soluções são mais simples
  3. Etapa de conquista da solução completa
     - Os resultados parciais são combinados

Mergesort

- Etapa de divisão

```
 1  // Padrão de tipos por tamanho
 2  #include <stdint.h>
...
18  // Mergesort recursivo
19  void mergesort(int32_t* S, int32_t* E, int32_t i,
        int32_t j) {
20      // Caso base
21      if(i < j) {
22          // Índice do meio do vetor
23          int32_t m = i + (j - i) / 2;
24          // Divisão em subvetores
25          mergesort(S, E, i, m);
26          mergesort(S, E, m + 1, j);
27          // Combinação dos resultados
28          intercalar(S, E, i, m, j);
29      }
30  }
```

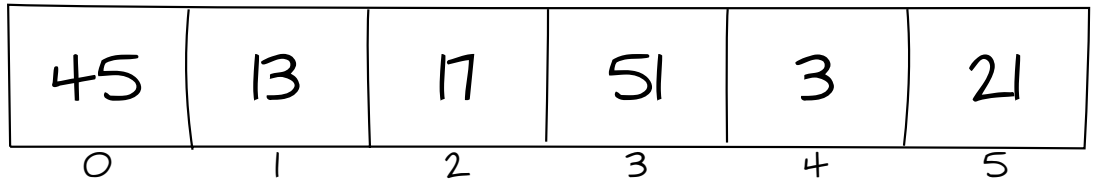
Mergesort

- Etapa de divisão

```
 1  // Padrão de tipos por tamanho
 2  #include <stdint.h>
...
18  // Mergesort recursivo
19  void mergesort(int32_t* S, int32_t* E, int32_t i,
        int32_t j) {
20      // Caso base
21      if(i < j) {
22          // Índice do meio do vetor
23          int32_t m = i + (j - i) / 2;
24          // Divisão em subvetores
25          mergesort(S, E, i, m);
26          mergesort(S, E, m + 1, j);
27          // Combinação dos resultados
28          intercalar(S, E, i, m, j);
29      }
30  }
```

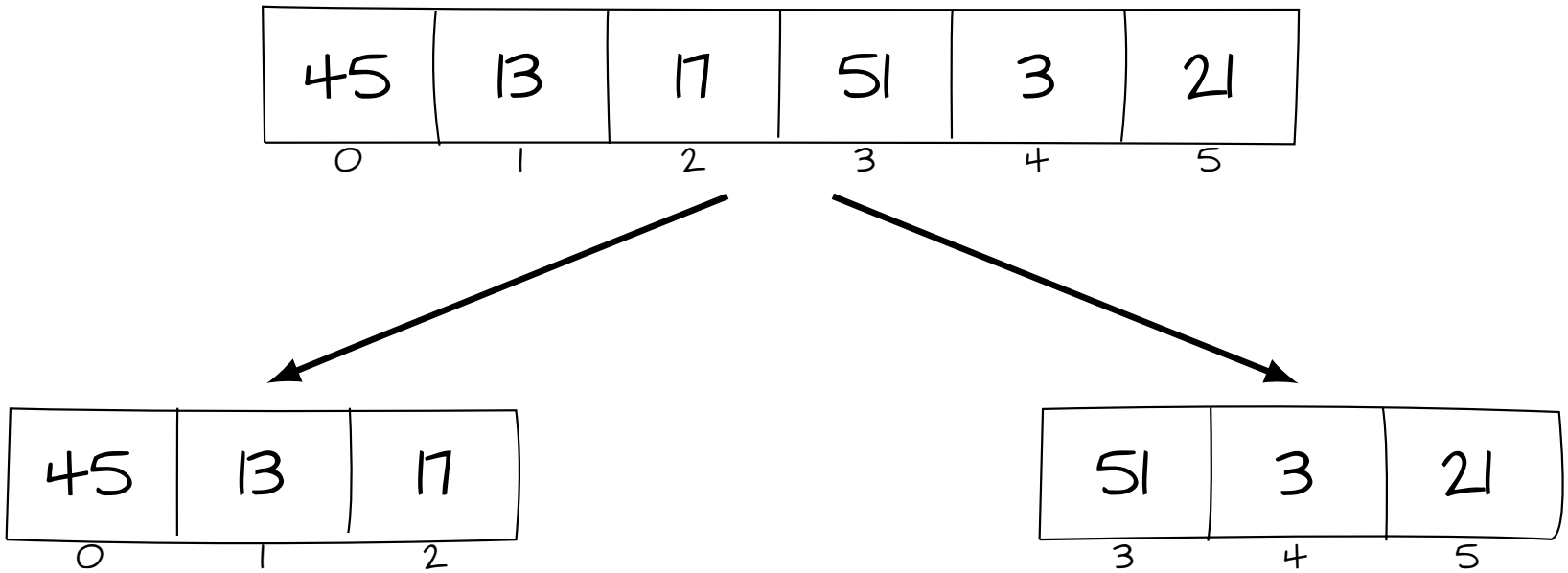
Mergesort

- Etapa de divisão



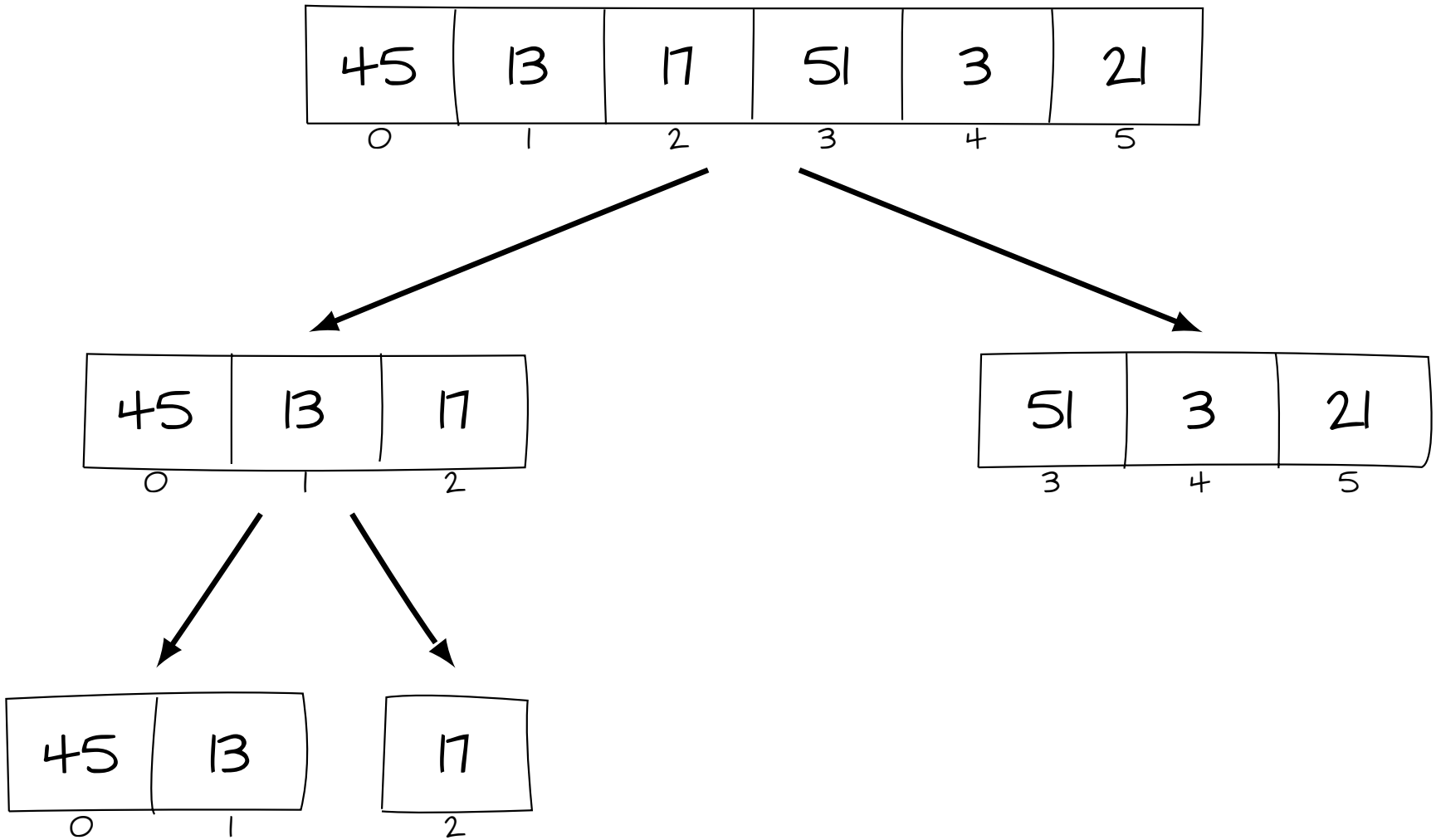
Mergesort

- Etapa de divisão



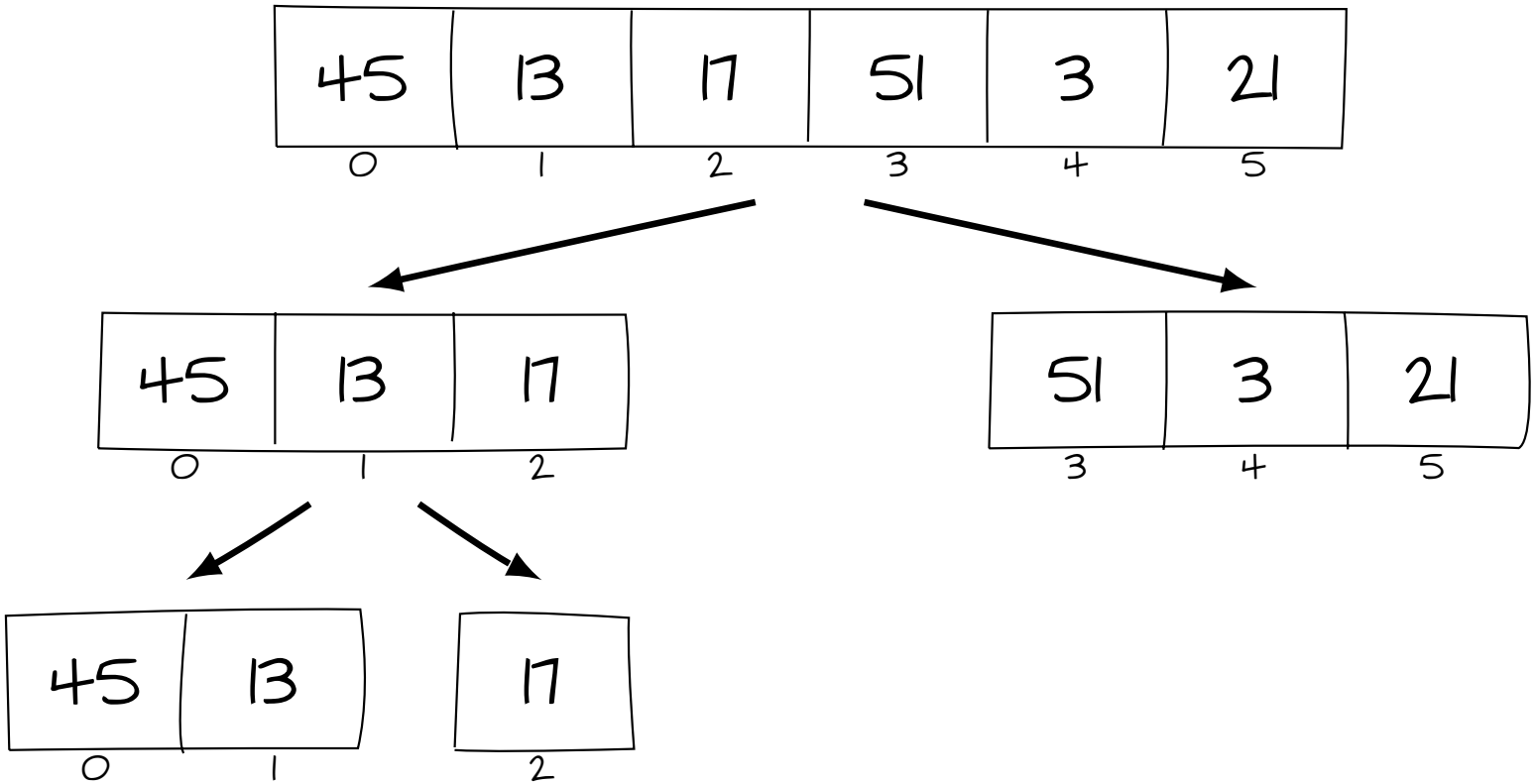
Mergesort

- Etapa de divisão



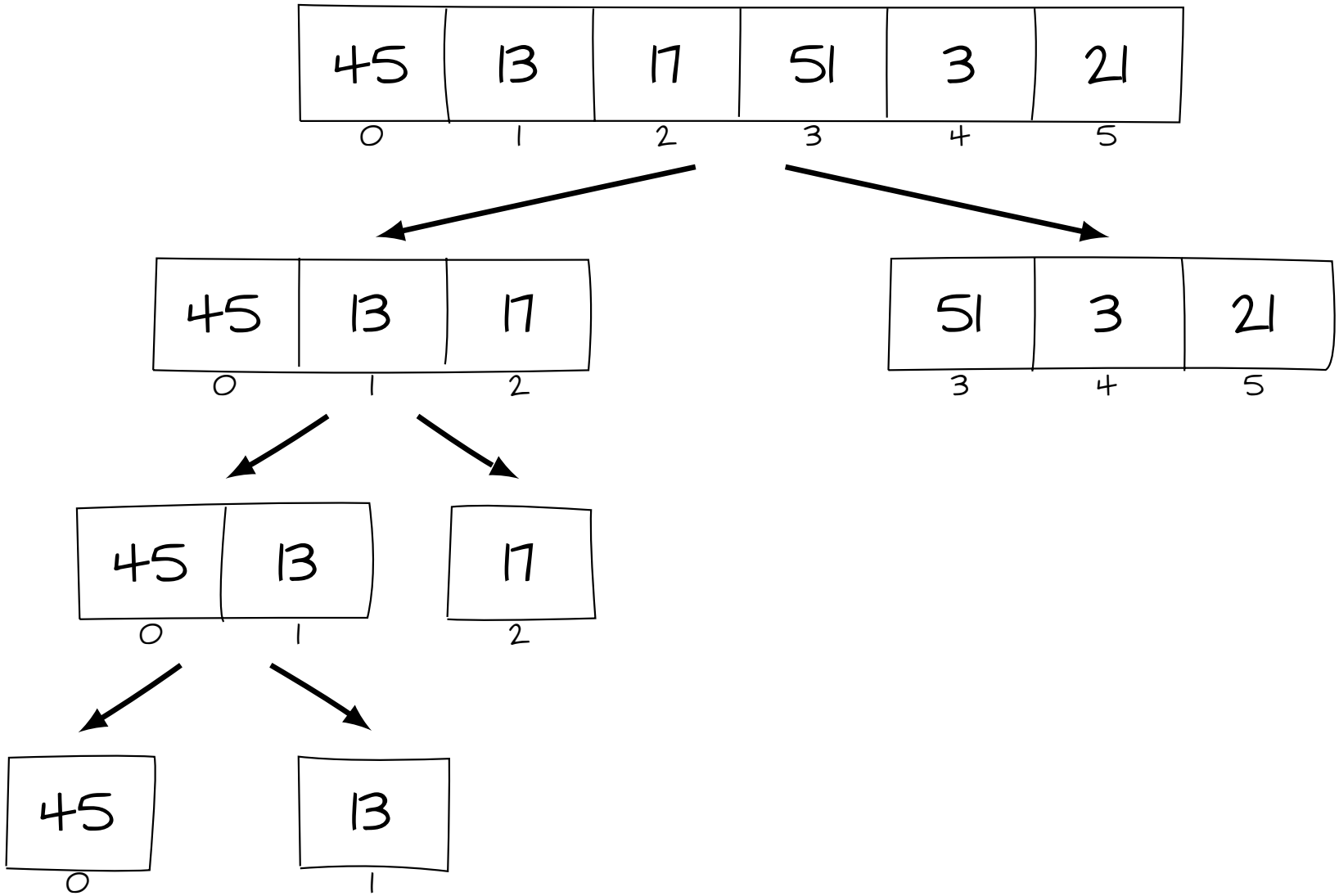
Mergesort

- Etapa de divisão



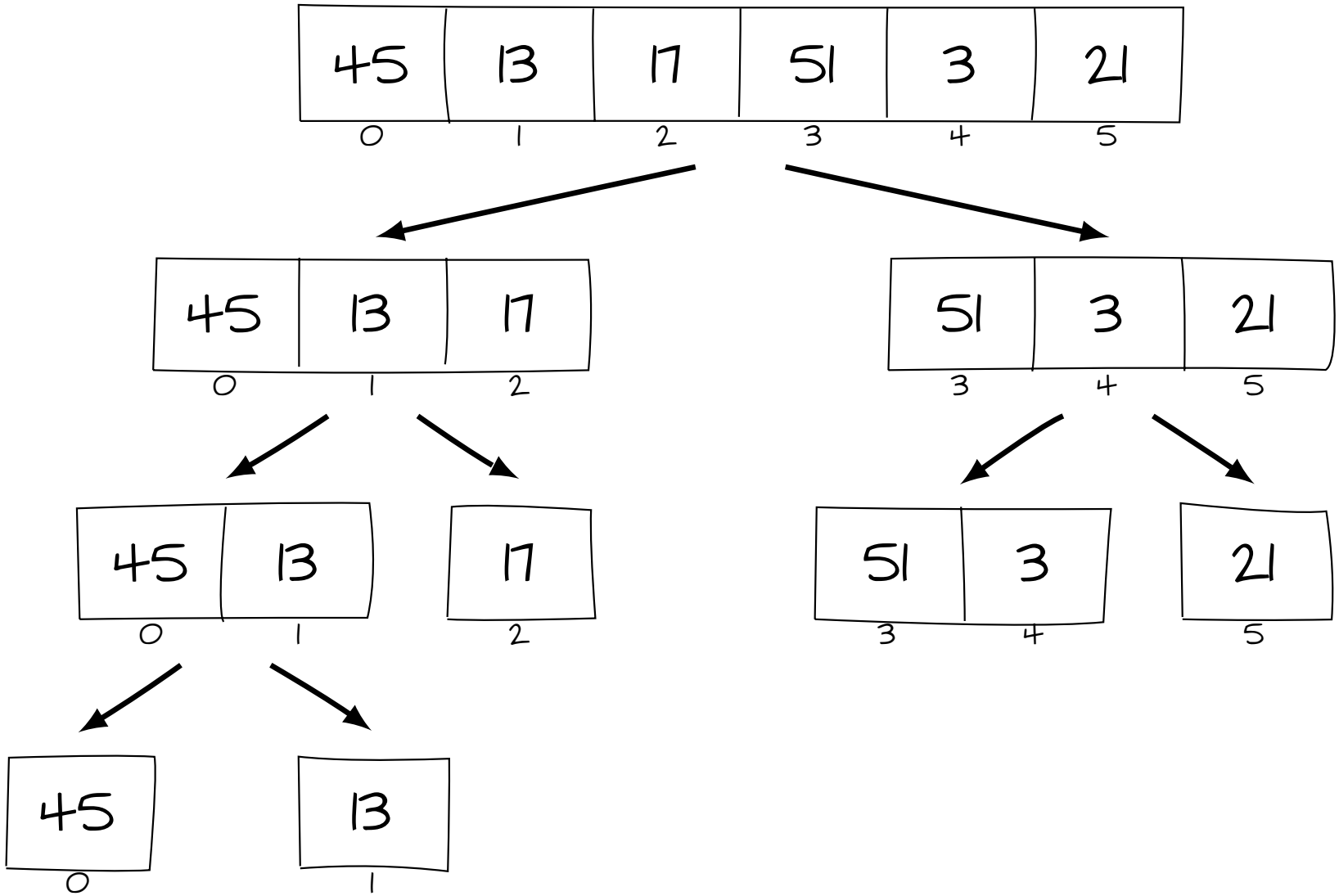
Mergesort

- Etapa de divisão



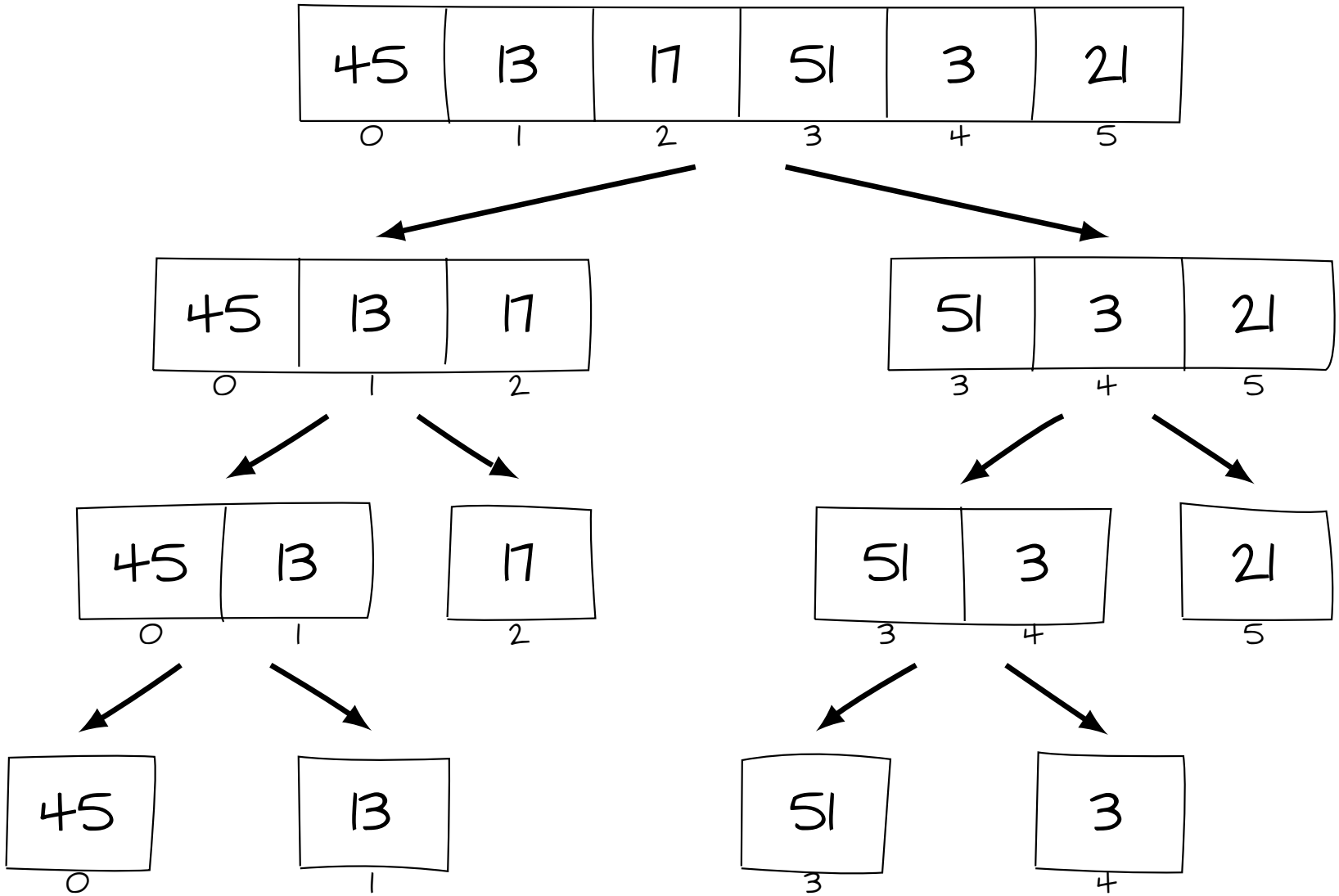
Mergesort

- Etapa de divisão



Mergesort

- Etapa de divisão



Mergesort

- Etapa de conquista

```
 1  // Padrão de tipos por tamanho
 2  #include <stdint.h>
...
18  // Mergesort recursivo
19  void mergesort(int32_t* S, int32_t* E, int32_t i,
        int32_t j) {
20      // Caso base
21      if(i < j) {
22          // Índice do meio do vetor
23          int32_t m = i + (j - i) / 2;
24          // Divisão em subvetores
25          mergesort(S, E, i, m);
26          mergesort(S, E, m + 1, j);
27          // Combinação dos resultados
28          intercalar(S, E, i, m, j);
29      }
30  }
```

Mergesort

- Etapa de conquista

```
 1  // Padrão de tipos por tamanho
 2  #include <stdint.h>
...
18  // Mergesort recursivo
19  void mergesort(int32_t* S, int32_t* E, int32_t i,
        int32_t j) {
20      // Caso base
21      if(i < j) {
22          // Índice do meio do vetor
23          int32_t m = i + (j - i) / 2;
24          // Divisão em subvetores
25          mergesort(S, E, i, m);
26          mergesort(S, E, m + 1, j);
27          // Combinação dos resultados
28          intercalar(S, E, i, m, j);
29      }
30  }
```

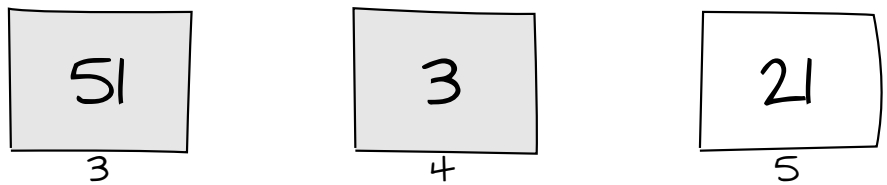
Mergesort

- Etapa de conquista

```
1   // Padrão de tipos por tamanho
2   #include <stdint.h>
3   // Intercalação de vetores
4   void intercalar(int32_t* S, int32_t* E, int32_t i,
        int32_t m, int32_t j) {
5       // Índices dos vetores
6       int32_t i1 = i, i2 = m + 1, k = i;
7       // Iteração enquanto houver elementos
8       while(i1 <= m && i2 <= j) {
9           // Comparação dos elementos
10          if(E[i1] <= E[i2]) S[k++] = E[i1++];
11          else S[k++] = E[i2++];
12      }
13      // Cópia dos subvetores
14      if(i1 > m) copiar(&S[k], &E[i2], j - i2 + 1);
15      else copiar(&S[k], &E[i1], m - i1 + 1);
16      copiar(&E[i], &S[i], j - i + 1);
17  }
...
...
```

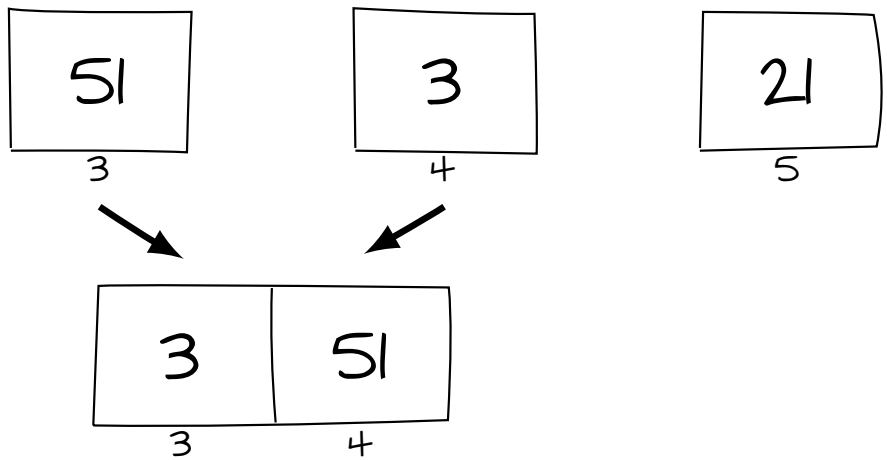
Mergesort

- Etapa de conquista



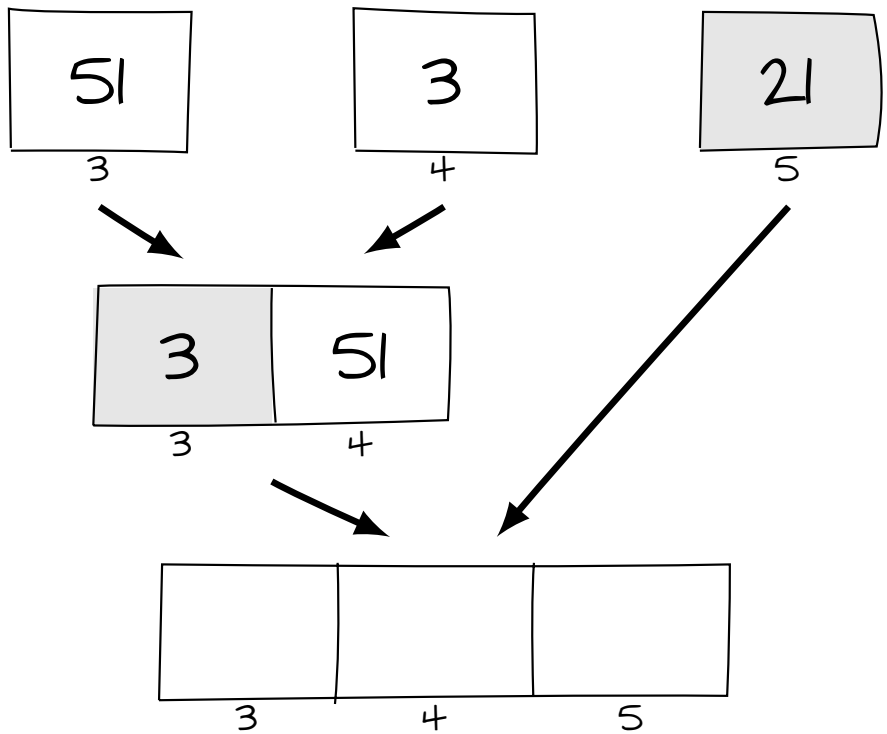
Mergesort

- Etapa de conquista



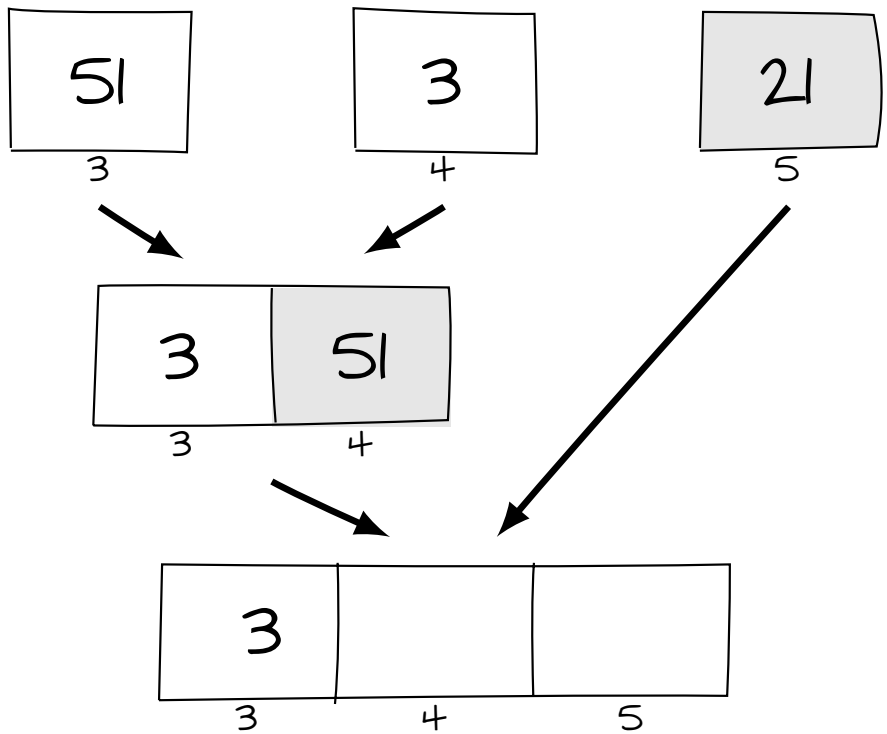
Mergesort

- Etapa de conquista



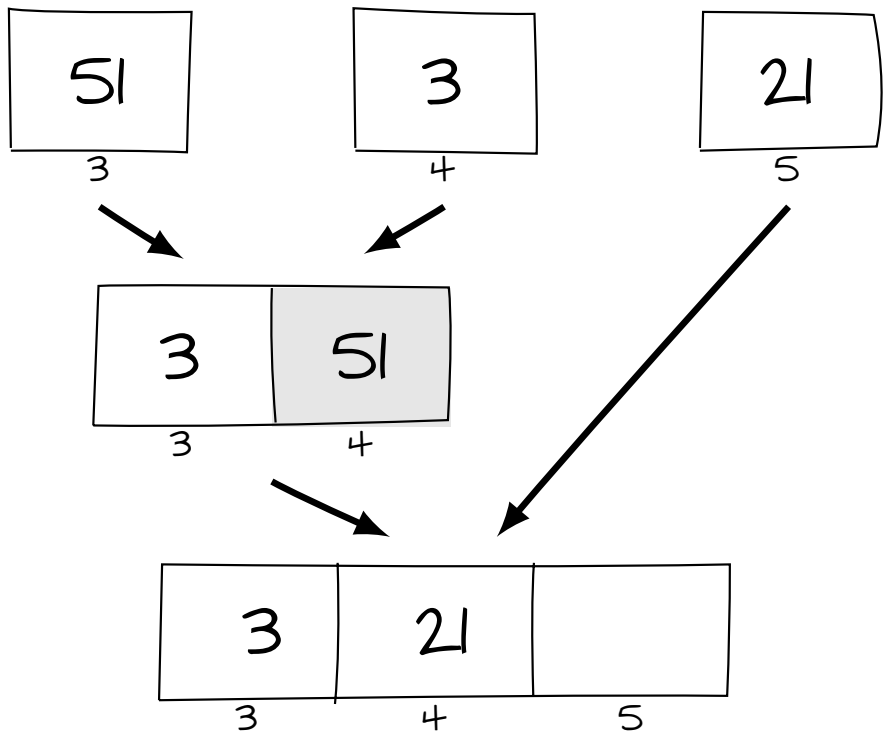
Mergesort

- Etapa de conquista



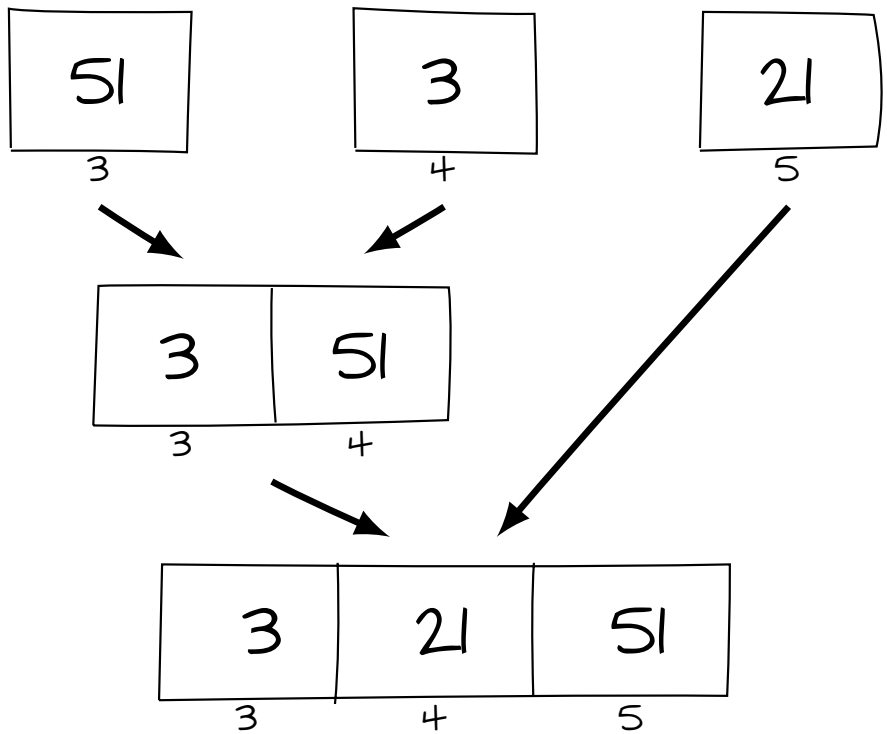
Mergesort

- Etapa de conquista



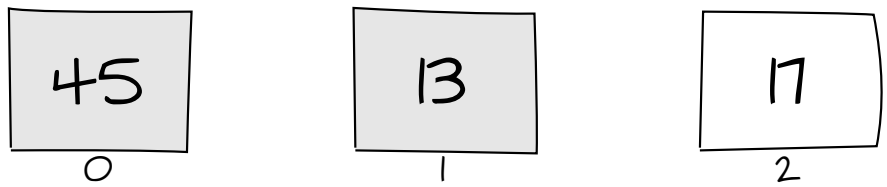
Mergesort

- Etapa de conquista



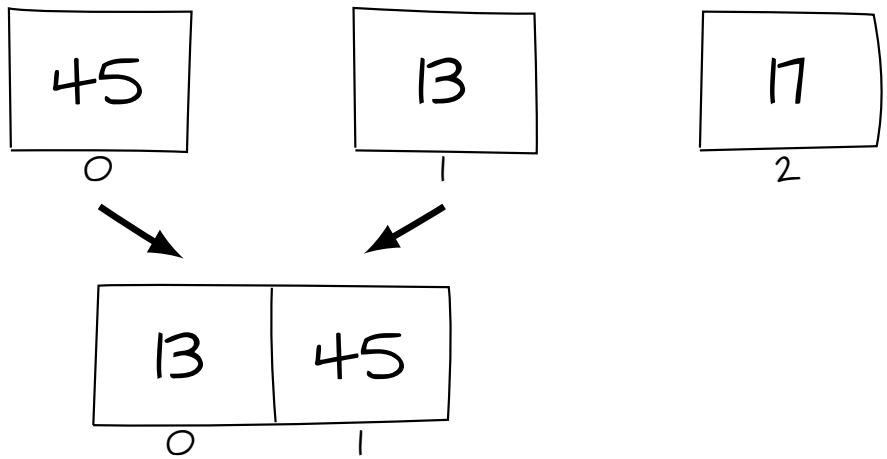
Mergesort

- Etapa de conquista



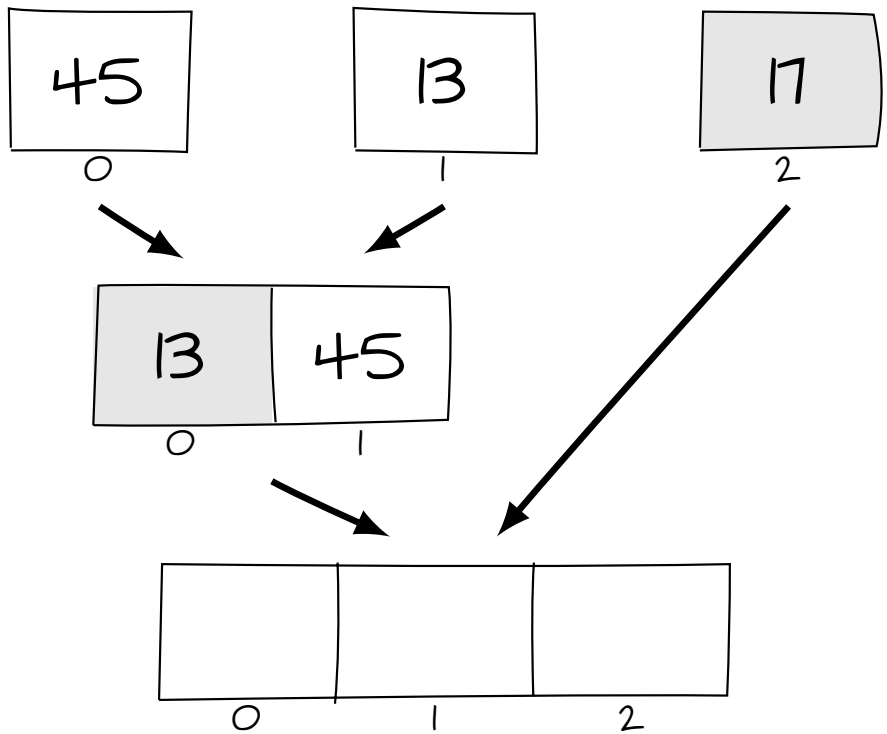
Mergesort

- Etapa de conquista



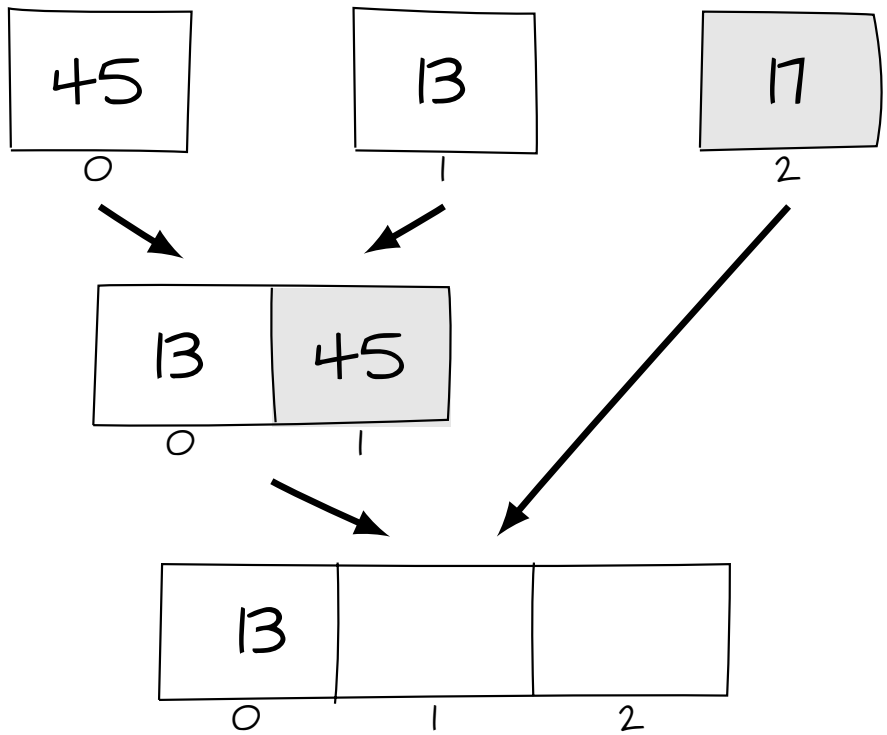
Mergesort

- Etapa de conquista



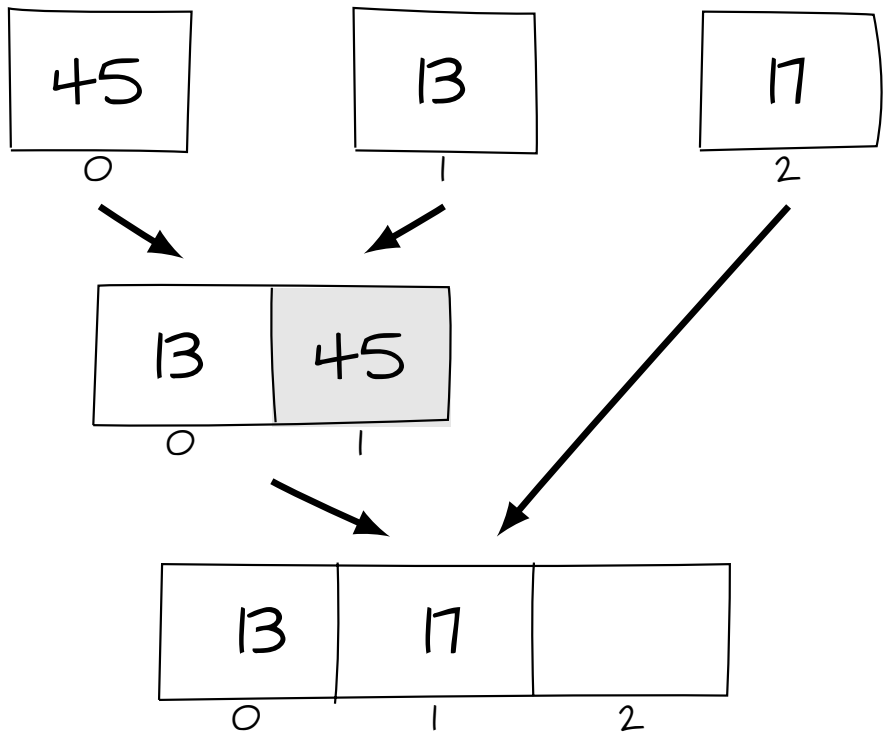
Mergesort

- Etapa de conquista



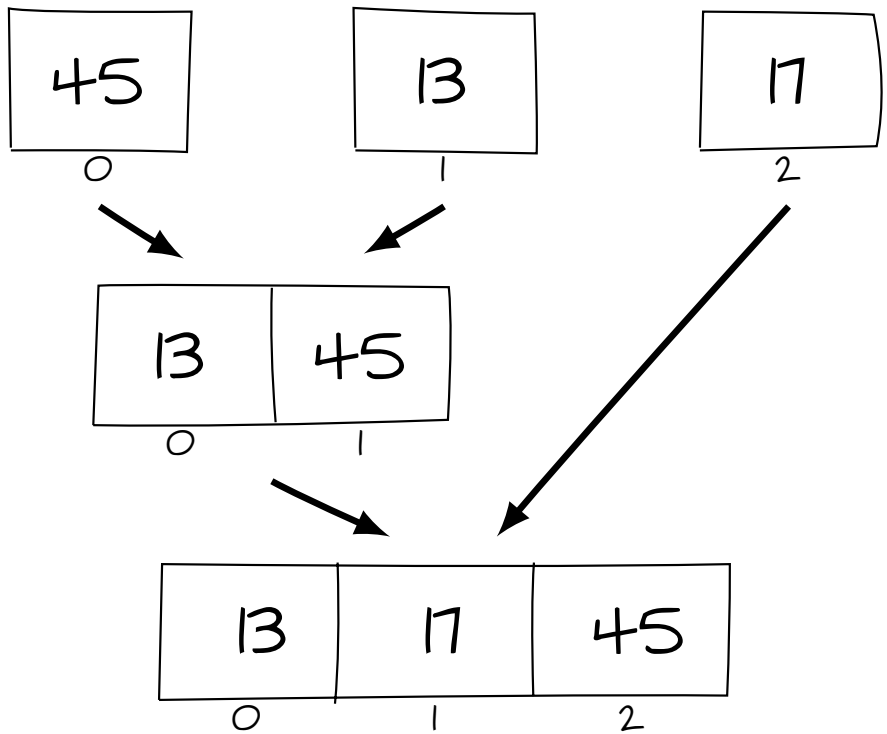
Mergesort

- Etapa de conquista



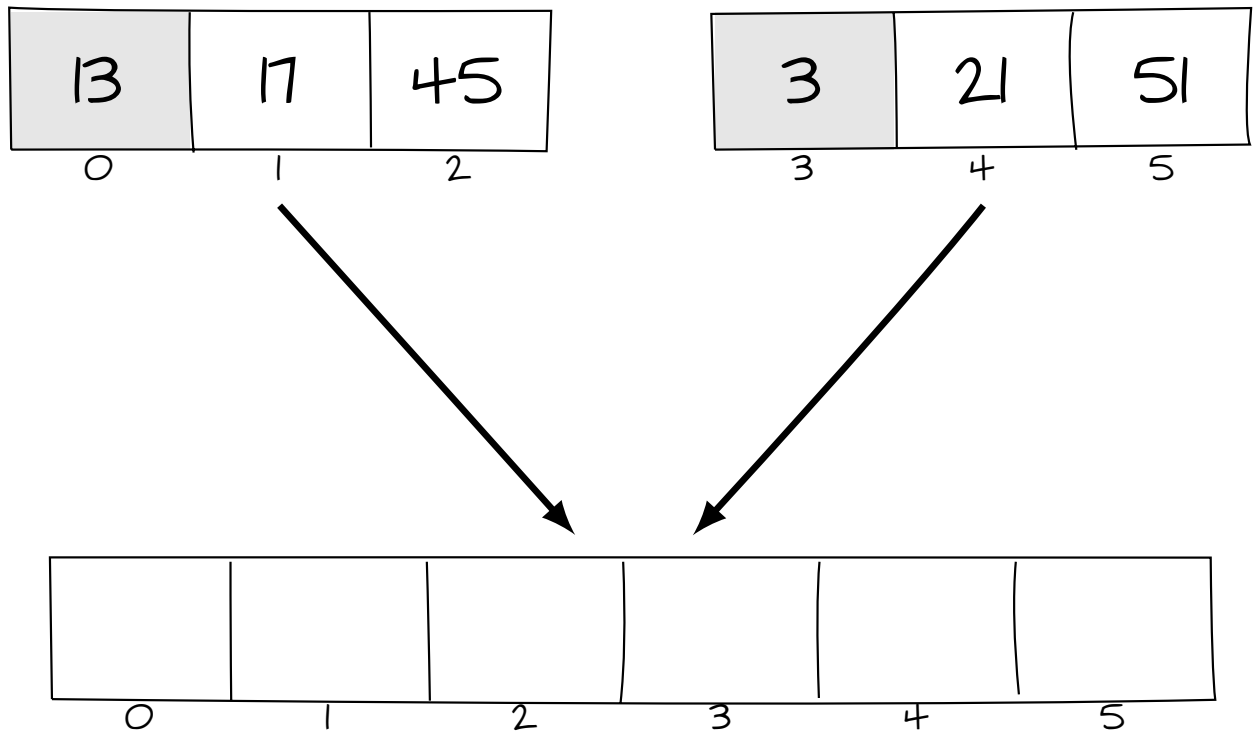
Mergesort

- Etapa de conquista



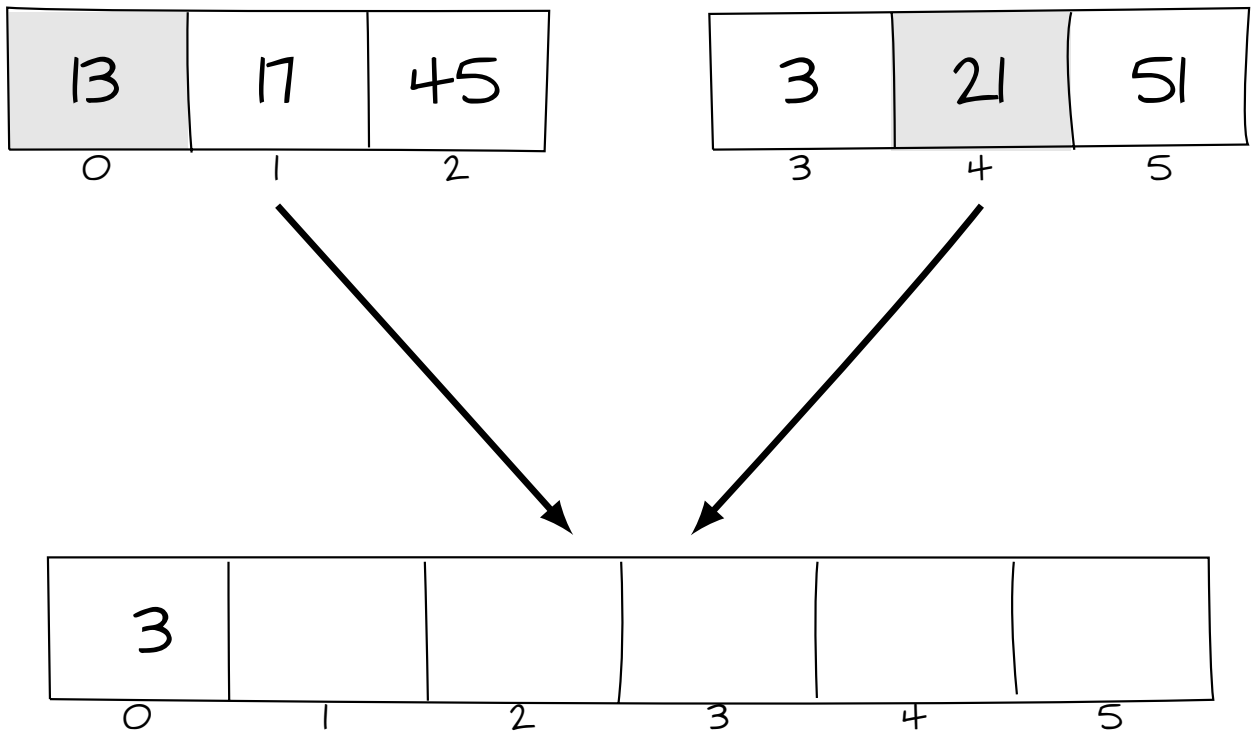
Mergesort

- Etapa de conquista



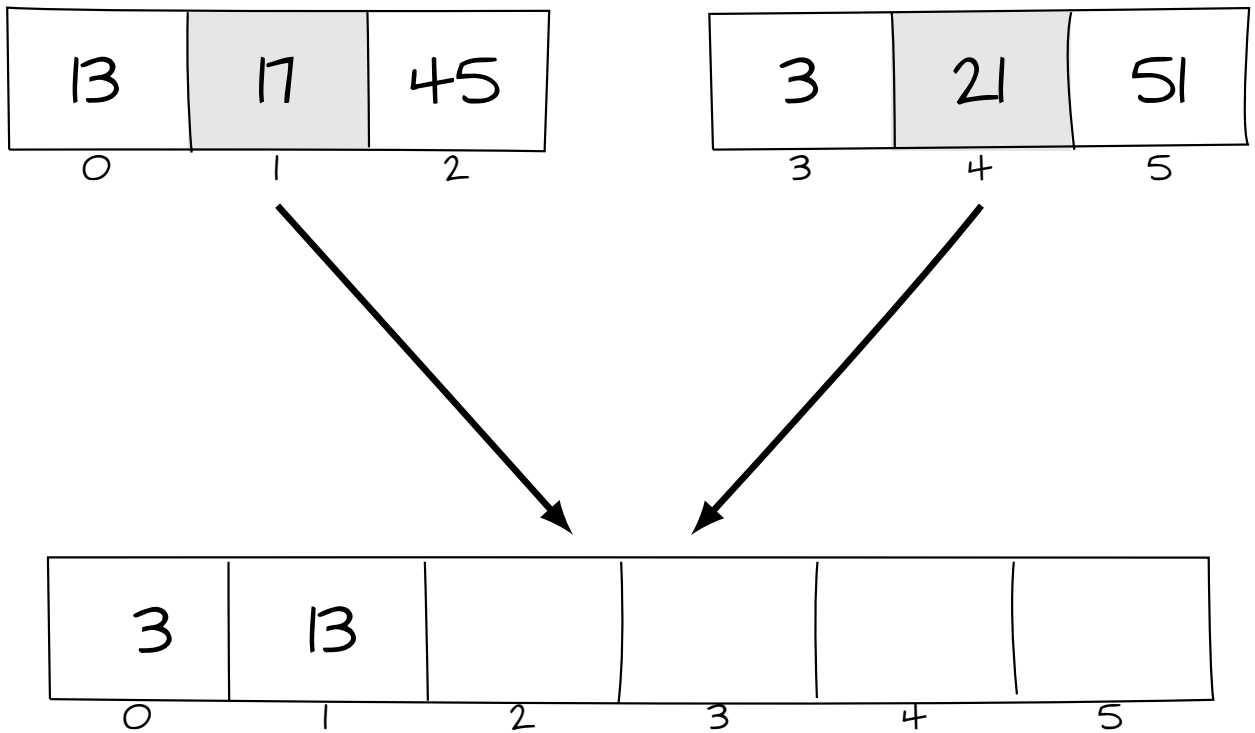
Mergesort

- Etapa de conquista



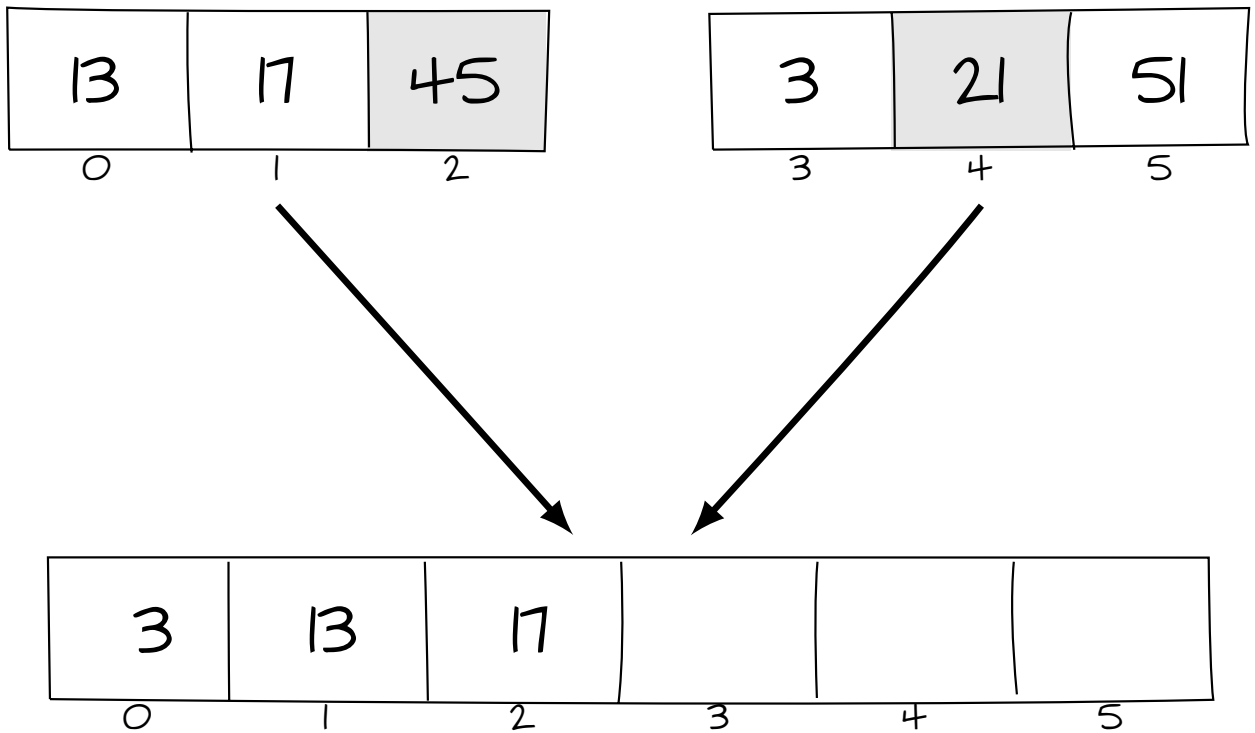
Mergesort

- Etapa de conquista



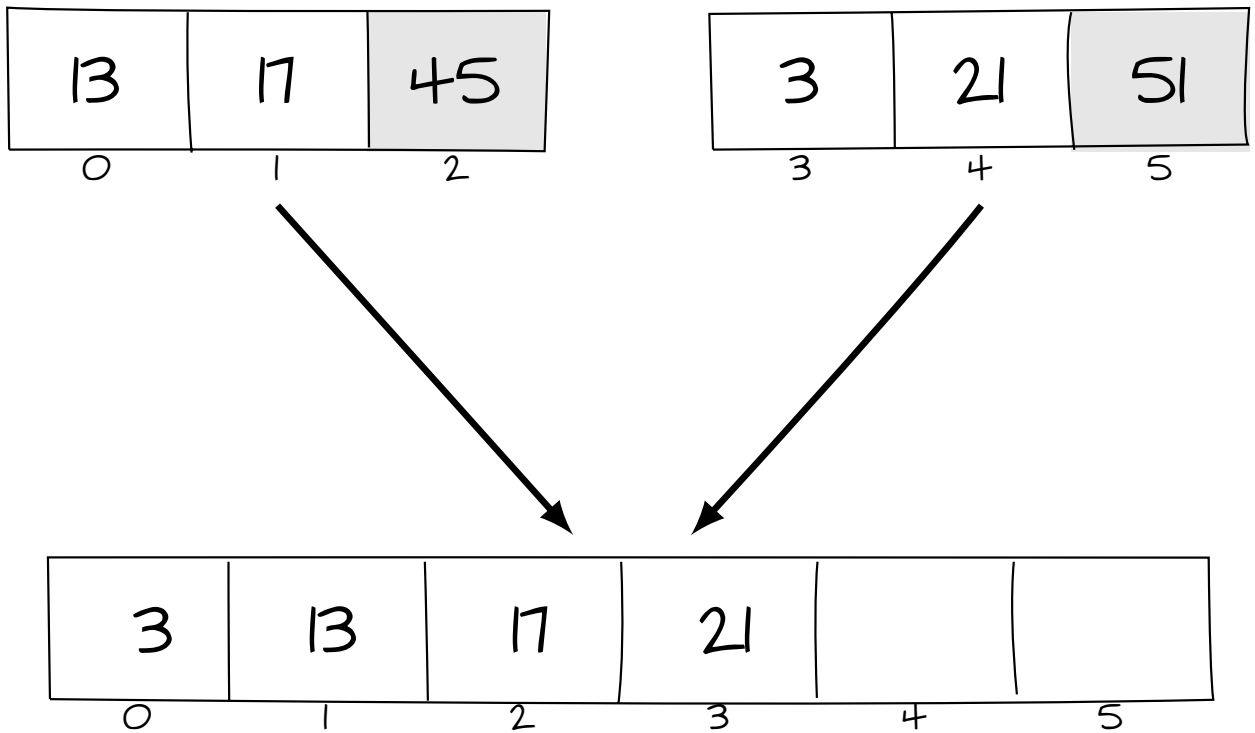
Mergesort

- Etapa de conquista



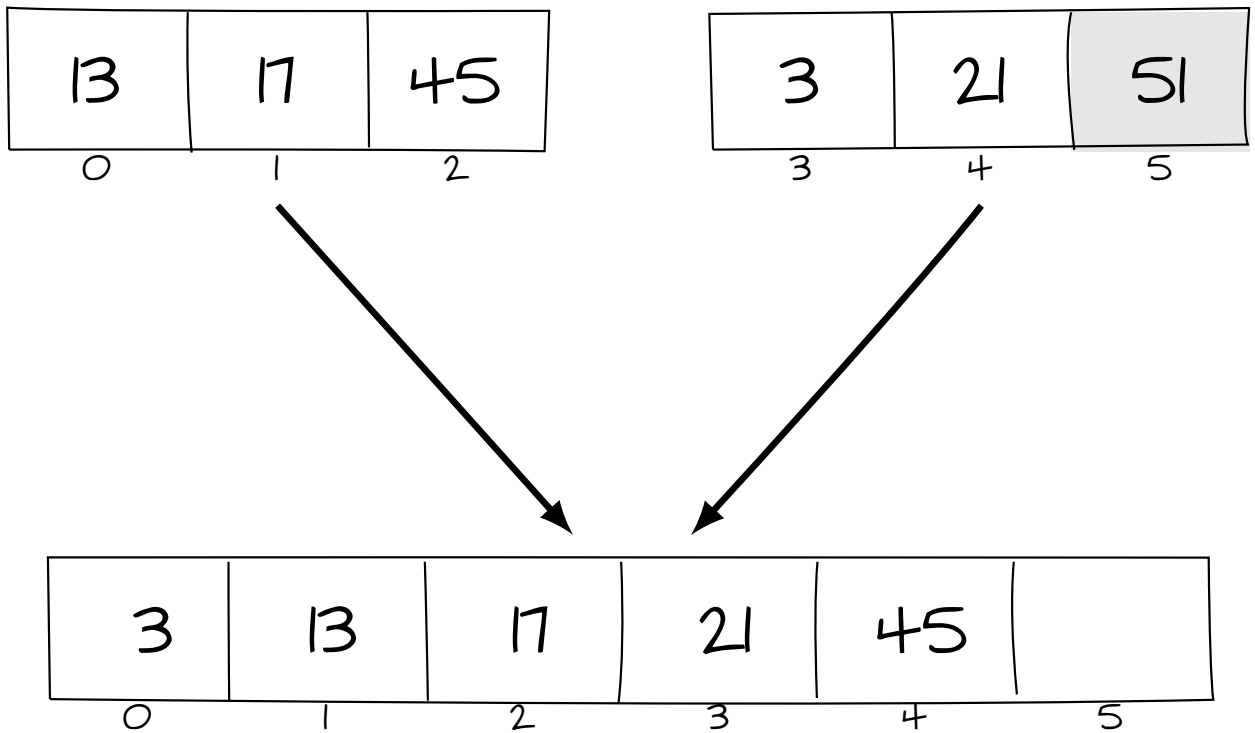
Mergesort

- Etapa de conquista



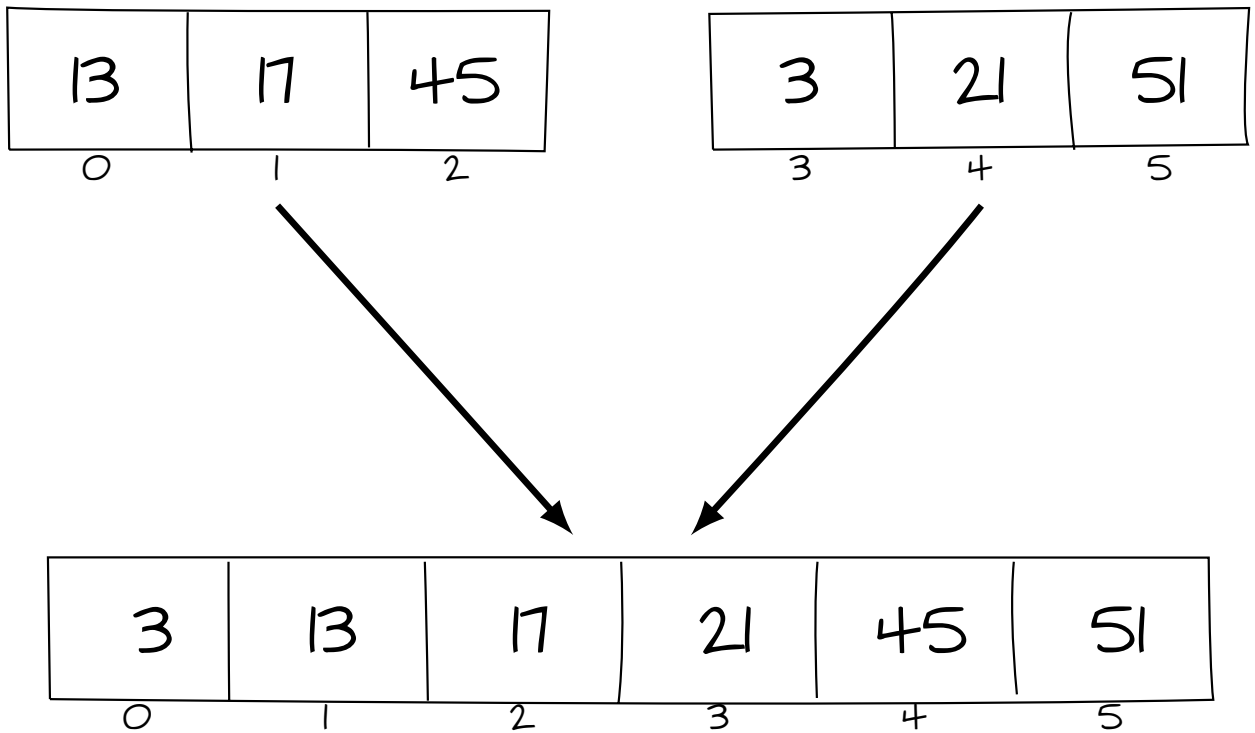
Mergesort

- Etapa de conquista



Mergesort

- Etapa de conquista



Mergesort

- Análise de complexidade
  - Caso base $T(1) = 1$
    - O vetor só possui um único elemento

Mergesort

- Análise de complexidade
  - Caso base $T(1) = 1$
    - O vetor só possui um único elemento
  - Recorrência $T(n) = 2T(n/2) + n$
    - Dividindo vetor em dois com metade do tamanho
    - Realizando a intercalação com custo linear

Mergesort

- Análise de complexidade
  - Resolvendo recorrência

$$T(n) = 2\left[2T\left(\frac{n}{4}\right) + \frac{n}{2}\right] + n = 4T\left(\frac{n}{4}\right) + 2n$$

Mergesort

- Análise de complexidade
  - Resolvendo recorrência

$$T(n) = 2\left[2T\left(\frac{n}{4}\right) + \frac{n}{2}\right] + n = 4T\left(\frac{n}{4}\right) + 2n$$

$$T(n) = 4\left[2T\left(\frac{n}{8}\right) + \frac{n}{4}\right] + 2n = 8T\left(\frac{n}{8}\right) + 3n$$

Mergesort

- Análise de complexidade
  - Resolvendo recorrência

$$T(n) = 2\left[2T\left(\frac{n}{4}\right) + \frac{n}{2}\right] + n = 4T\left(\frac{n}{4}\right) + 2n$$

$$T(n) = 4\left[2T\left(\frac{n}{8}\right) + \frac{n}{4}\right] + 2n = 8T\left(\frac{n}{8}\right) + 3n$$

$$T(n) = 8\left[2T\left(\frac{n}{16}\right) + \frac{n}{8}\right] + 3n = 16T\left(\frac{n}{16}\right) + 4n$$

Mergesort

- Análise de complexidade
  - Resolvendo recorrência

$$T(n) = 2\left[2T\left(\frac{n}{4}\right) + \frac{n}{2}\right] + n = 4T\left(\frac{n}{4}\right) + 2n$$

$$T(n) = 4\left[2T\left(\frac{n}{8}\right) + \frac{n}{4}\right] + 2n = 8T\left(\frac{n}{8}\right) + 3n$$

$$T(n) = 8\left[2T\left(\frac{n}{16}\right) + \frac{n}{8}\right] + 3n = 16T\left(\frac{n}{16}\right) + 4n$$

$$T(n) = 2^k T\left(\frac{n}{2^k}\right) + kn$$

Mergesort

- Análise de complexidade
  - Resolvendo recorrência

$$T(n) = 2^k T\left(\frac{n}{2^k}\right) + kn$$

Mergesort

- Análise de complexidade
  - Resolvendo recorrência

$$T(n) = 2^k T\left(\frac{n}{2^k}\right) + kn$$

$$T(1) = 1 \longrightarrow \frac{n}{2^k} = 1 \longrightarrow k = \log_2 n$$

Mergesort

- Análise de complexidade
  - Resolvendo recorrência

$$T(n) = 2^k T\left(\frac{n}{2^k}\right) + kn$$

$$T(1) = 1 \longrightarrow \frac{n}{2^k} = 1 \longrightarrow k = \log_2 n$$

$$T(n) = 2^{\log_2 n} T(1) + n \log_2 n \longrightarrow T(n) = n + n \log_2 n$$

Mergesort

- Análise de complexidade
  - Resolvendo recorrência

$$T(n) = 2^k T\left(\frac{n}{2^k}\right) + kn$$

$$T(1) = 1 \longrightarrow \frac{n}{2^k} = 1 \longrightarrow k = \log_2 n$$

$$T(n) = 2^{\log_2 n} T(1) + n \log_2 n \longrightarrow T(n) = n + n \log_2 n$$

$$T(n) = O(n \log_2 n)$$

Mergesort

- Análise de complexidade
  - Espaço $\Theta(n + \log n) = \Theta(n)$
  - Tempo $\Theta(n \log_2 n)$

Mergesort

- Características do Mergesort
  ✓ Paralelismo: a entrada é dividida em partes que podem ser resolvidas de forma paralela

Mergesort

- Características do Mergesort
  ✓ Paralelismo: a entrada é dividida em partes que podem ser resolvidas de forma paralela
  ✓ Eficiência de espaço $\Theta(n)$ e tempo $\Theta(n \log n)$

Mergesort

- Características do Mergesort
  ✓ Paralelismo: a entrada é dividida em partes que podem ser resolvidas de forma paralela
  ✓ Eficiência de espaço $\Theta(n)$ e tempo $\Theta(n \log n)$
  ✓ Acesso a memória mais eficiente: conjuntos de dados menores e sequenciais cabem na cache

Mergesort

- Características do Mergesort
  - ✓ Paralelismo: a entrada é dividida em partes que podem ser resolvidas de forma paralela
  - ✓ Eficiência de espaço $\Theta(n)$ e tempo $\Theta(n \log n)$
  - ✓ Acesso a memória mais eficiente: conjuntos de dados menores e sequenciais cabem na cache
  - ✓ Estabilidade: preserva a ordem relativa dos elementos

Mergesort

- Características do Mergesort
  - ✗ Recursão: a utilização de pilha que é limitada

Mergesort

- Características do Mergesort
  - ✗ Recursão: a utilização de pilha que é limitada
  - ✗ Menor desempenho pelo acesso intensivo a memória

Mergesort

- Características do Mergesort
  - ✗ Recursão: a utilização de pilha que é limitada
  - ✗ Menor desempenho pelo acesso intensivo a memória
  - ✗ Escolha dos casos base: evitar processamento desnecessário de entradas pequenas e triviais

Mergesort

- Características do Mergesort
  - ✗ Recursão: a utilização de pilha que é limitada
  - ✗ Menor desempenho pelo acesso intensivo a memória
  - ✗ Escolha dos casos base: evitar processamento desnecessário de entradas pequenas e triviais
  - ✗ Subproblemas repetidos: subvetores idênticos

Mergesort

- Aplicações
  - Execução paralela

Mergesort

- Aplicações
  - Execução paralela
  - Dispositivos de acesso sequencial

Mergesort

- Aplicações
  - Execução paralela
  - Dispositivos de acesso sequencial
  - Grande volume de dados
  - ...

Exemplo

- Considerando o algoritmo de ordenação Mergesort, ordene o vetor 23, 32, 54, 92, 74, 23, 1, 43, 63 e 12
  - Utilize o critério crescente de ordenação
  - Execute passo a passo cada etapa do algoritmo

Exercício

- A empresa de automação portuária Poxim Tech está desenvolvendo um sistema para movimentação automatizada dos contêineres de carga de origem internacional no Porto de Sergipe para maximizar a eficiência da fiscalização aduaneira
  - Todos os contêineres possuem um cadastro eletrônico contendo informações sobre o código do contêiner, o CNPJ da empresa importadora e o peso líquido em quilos da carga
  - A inspeção dos contêineres é realizada sempre que existe alguma divergência entre as informações cadastradas, como o CNPJ informado ou a diferença percentual maior do que 10% no peso líquido
  - Na triagem dos contêineres são fiscalizados os contêineres com a seguinte ordem de prioridade:
    1. Divergência de CNPJ
    2. Maior diferença percentual de peso líquido

Exercício

- Formato de arquivo de entrada
  - [#*n contêineres cadastrados*]
  - [*Código* 1] [*CNPJ* 1] [*Peso* 1]
  - ...
  - [*Código n*] [*CNPJ n*] [*Peso n*]
  - [#*m contêineres selecionados*]
  - [*Código* 1] [*CNPJ* 1] [*Peso* 1]
  - ...
  - [*Código m*] [*CNPJ m*] [*Peso m*]

```
1  6
2  QOZJ7913219␣34.699.211/9365-11␣13822
3  FCCU4584578␣50.503.434/5731-28␣16022
4  KTAJ0603546␣20.500.522/6013-58␣25279
5  ZYHU3978783␣43.172.263/4442-14␣24543
6  IKQZ7582839␣51.743.446/1183-18␣12160
7  HAAZ0273059␣25.699.428/4746-79␣16644
8  5
9  ZYHU3978783␣43.172.263/4442-14␣29765
10 IKQZ7582839␣51.743.446/1113-18␣18501
11 KTAJ0603546␣20.500.522/6113-58␣17842
12 QOZJ7913219␣34.699.211/9365-11␣16722
13 FCCU4584578␣50.503.434/5731-28␣16398
```

Exercício

- Formato de arquivo de saída
  - A sequência de fiscalização dos contêineres do navio, com a causa da triagem e seguindo a ordem de cadastramento dos contêineres

```
1 KTAJ0603546: 20.500.522/6013-58<->20.500.522/6113-58
2 IKQZ7582839: 51.743.446/1183-18<->51.743.446/1113-18
3 QOZJ7913219: 2900kg (21%)
4 ZYHU3978783: 5222kg (21%)
```