# Algoritmo Mergesort

## Estrutura de Dados — QXD0010

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ


Prof. Atílio Gomes Luiz
gomes.atilio@ufc.br

Universidade Federal do Ceará


1º semestre/2021

# Introdução

# Introdução

Vimos três algoritmos de ordenação $O(n^2)$:

- `selectionsort`
- `bubblesort`
- `insertionsort`

# Introdução

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

Vimos três algoritmos de ordenação $O(n^2)$:

- `selectionsort`
- `bubblesort`
- `insertionsort`

Nessa aula veremos um algoritmo de ordenação $O(n \log n)$

# Introdução

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

Vimos três algoritmos de ordenação $O(n^2)$:

- `selectionsort`
- `bubblesort`
- `insertionsort`

Nessa aula veremos um algoritmo de ordenação $O(n \log n)$

Ele é baseado em uma técnica de projeto de algoritmos chamada Divisão e Conquista ou Dividir para Conquistar

# Um problema subjacente: Intercalação de dois vetores ordenados

# Intercalando dois vetores ordenados

Universidade Federal do Ceará
Campus Quixadá

Antes de tratar o problema da ordenação propriamente dito, é preciso resolver um problema auxiliar:

- Problema: Dados vetores crescentes $A[p..q]$ e $A[q+1..r]$, como rearranjar $A[p..r]$ em ordem crescente?

- Podemos dizer que o problema consiste em "intercalar" os dois vetores dados.

# Intercalando dois vetores ordenados

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

Antes de tratar o problema da ordenação propriamente dito, é preciso resolver um problema auxiliar:

- Problema: Dados vetores crescentes $A[p..q]$ e $A[q+1..r]$, como rearranjar $A[p..r]$ em ordem crescente?

- Podemos dizer que o problema consiste em "intercalar" os dois vetores dados.

É fácil resolver o problema em tempo proporcional ao quadrado do tamanho do vetor $A$:

# Intercalando dois vetores ordenados

Universidade Federal do Ceará
Campus Quixadá

Antes de tratar o problema da ordenação propriamente dito, é preciso resolver um problema auxiliar:

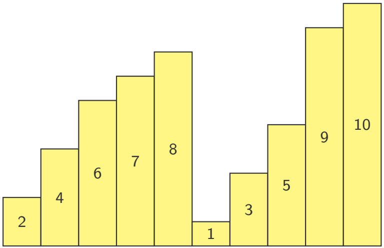
- Problema: Dados vetores crescentes $A[p..q]$ e $A[q+1..r]$, como rearranjar $A[p..r]$ em ordem crescente?

- Podemos dizer que o problema consiste em "intercalar" os dois vetores dados.

É fácil resolver o problema em tempo proporcional ao quadrado do tamanho do vetor $A$:

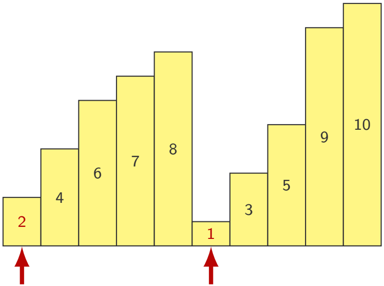
- Basta aplicar um dos algoritmos de ordenação da aula anterior.
- Mas com isso ignoramos o fato de que as duas "metades" do vetor original $A$ já estão ordenadas.
- Podemos ser mais eficientes?

# Intercalando dois vetores ordenados



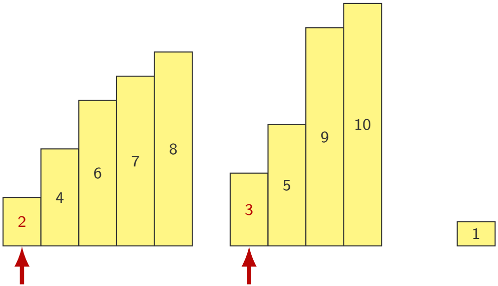

- Percorremos os dois subvetores

# Intercalando dois vetores ordenados



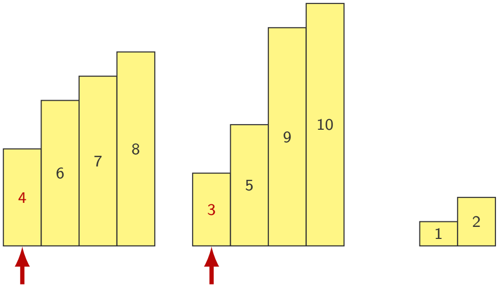

- Percorremos os dois subvetores
- Pegamos o mínimo e inserimos em um vetor auxiliar

# Intercalando dois vetores ordenados




- Percorremos os dois subvetores
- Pegamos o mínimo e inserimos em um vetor auxiliar

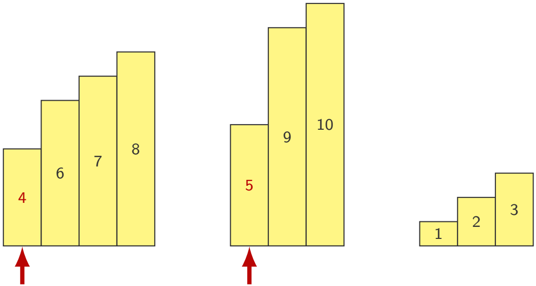
# Intercalando dois vetores ordenados




- Percorremos os dois subvetores
- Pegamos o mínimo e inserimos em um vetor auxiliar

# Intercalando dois vetores ordenados

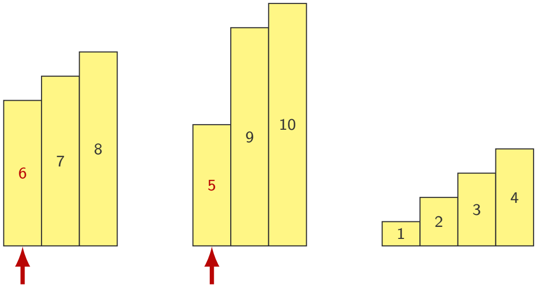





- Percorremos os dois subvetores
- Pegamos o mínimo e inserimos em um vetor auxiliar

# Intercalando dois vetores ordenados

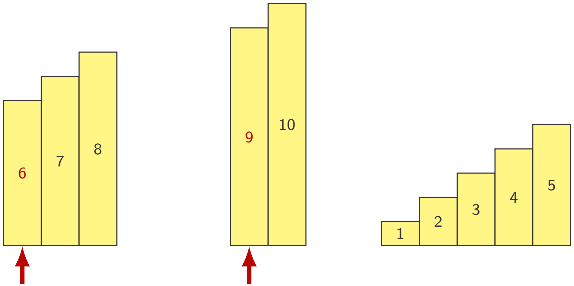





- Percorremos os dois subvetores
- Pegamos o mínimo e inserimos em um vetor auxiliar

# Intercalando dois vetores ordenados

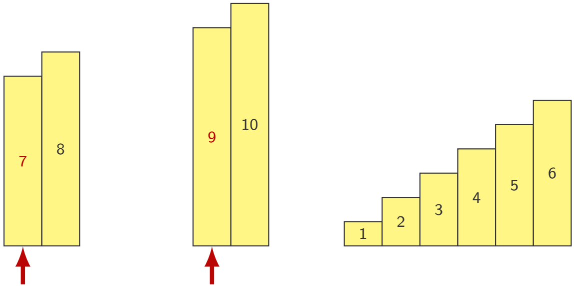





- Percorremos os dois subvetores
- Pegamos o mínimo e inserimos em um vetor auxiliar

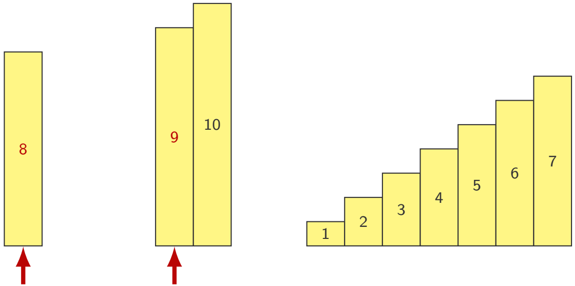
# Intercalando dois vetores ordenados






- Percorremos os dois subvetores
- Pegamos o mínimo e inserimos em um vetor auxiliar

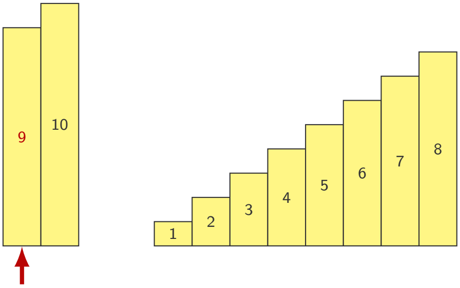
# Intercalando dois vetores ordenados




- Percorremos os dois subvetores
- Pegamos o mínimo e inserimos em um vetor auxiliar

# Intercalando dois vetores ordenados

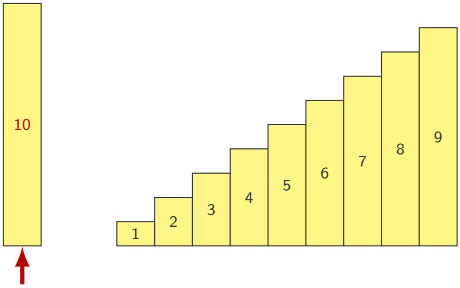





- Percorremos os dois subvetores
- Pegamos o mínimo e inserimos em um vetor auxiliar
- Depois copiamos o restante

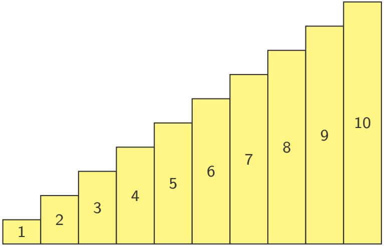
# Intercalando dois vetores ordenados

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ




- Percorremos os dois subvetores
- Pegamos o mínimo e inserimos em um vetor auxiliar
- Depois copiamos o restante

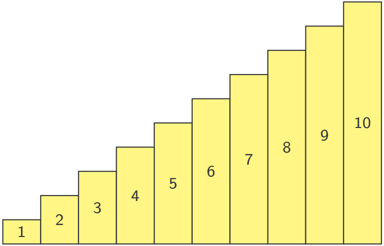
# Intercalando dois vetores ordenados




- Percorremos os dois subvetores
- Pegamos o mínimo e inserimos em um vetor auxiliar
- Depois copiamos o restante

# Intercalando dois vetores ordenados

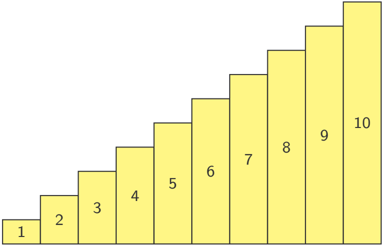


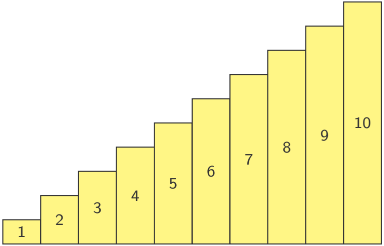



- Percorremos os dois subvetores
- Pegamos o mínimo e inserimos em um vetor auxiliar
- Depois copiamos o restante
- No final, copiamos do vetor auxiliar para o original

# Intercalando dois vetores ordenados






- Percorremos os dois subvetores
- Pegamos o mínimo e inserimos em um vetor auxiliar
- Depois copiamos o restante
- No final, copiamos do vetor auxiliar para o original

# Intercalando dois vetores ordenados

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

```
/* A funcao recebe vetores crescentes A[p..q] e A[q+1..r]
 * e rearranja A[p..r] em ordem crescente */
void Intercala (int A[], int p, int q, int r) {
  int *W = new int[r-p+1]; // Vetor auxiliar
  int i = p;
  int j = q+1;
  int k = 0;

  // Intercala A[p..q] e A[q+1..r]
  while (i <= q && j <= r) {
    if (A[i] <= A[j])
      W[k++] = A[i++];
    else
      W[k++] = A[j++];
  }
  while (i <= q) W[k++] = A[i++];
  while (j <= r) W[k++] = A[j++];

  // Copia vetor ordenado W para o vetor A
  for (i = p; i <= r; i++)
    A[i] = W[i-p];

  delete[] W; // libera memoria alocada
}
```

# Desempenho da Intercalação

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

- A função `Intercala` consome tempo proporcional ao número de comparações entre elementos do vetor.

Quantas comparações são feitas?

# Desempenho da Intercalação

- A função `Intercala` consome tempo proporcional ao número de comparações entre elementos do vetor.

Quantas comparações são feitas?

- a cada passo, aumentamos um em `i` ou em `j`

# Desempenho da Intercalação

- A função `Intercala` consome tempo proporcional ao número de comparações entre elementos do vetor.

Quantas comparações são feitas?

- a cada passo, aumentamos um em `i` ou em `j`
- no máximo $n = r - p + 1$

# Desempenho da Intercalação

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

- A função `Intercala` consome tempo proporcional ao número de comparações entre elementos do vetor.

Quantas comparações são feitas?

- a cada passo, aumentamos um em `i` ou em `j`
- no máximo $n = r - p + 1$
- Logo, o consumo de tempo no pior caso é proporcional ao número de elementos do vetor, ou seja, $O(n)$.

# Desempenho da Intercalação

Universidade Federal do Ceará
Campus Quixadá

- A função `Intercala` consome tempo proporcional ao número de comparações entre elementos do vetor.

Quantas comparações são feitas?

- a cada passo, aumentamos um em `i` ou em `j`
- no máximo $n = r - p + 1$
- Logo, o consumo de tempo no pior caso é proporcional ao número de elementos do vetor, ou seja, $O(n)$.
- O algoritmo de intercalação é, portanto, muito eficiente.

# Divisão e Conquista

# Divisão e conquista

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

Observação:

- A recursão parte do princípio que é mais fácil resolver problemas menores.
- Para certos problemas, podemos dividi-los em duas ou mais partes.

# Divisão e conquista



Observação:

- A recursão parte do princípio que é mais fácil resolver problemas menores.
- Para certos problemas, podemos dividi-los em duas ou mais partes.

  Etapas do paradigma de Divisão e Conquista:

  - Dividir: Quebramos o problema em vários subproblemas menores.
    - ex: quebramos um vetor a ser ordenado em dois.

# Divisão e conquista



Observação:

- A recursão parte do princípio que é mais fácil resolver problemas menores.
- Para certos problemas, podemos dividi-los em duas ou mais partes.

  Etapas do paradigma de Divisão e Conquista:

  - Dividir: Quebramos o problema em vários subproblemas menores.
    - ex: quebramos um vetor a ser ordenado em dois.
  - Conquistar: Os subproblemas são resolvidos recursivamente. Se eles forem pequenos o bastante, eles são resolvidos usando o próprio algoritmo que está sendo definido.
    - ex: um subvetor com um único elemento já está ordenado.

# Divisão e conquista

Observação:

- A recursão parte do princípio que é mais fácil resolver problemas menores.
- Para certos problemas, podemos dividi-los em duas ou mais partes.

  Etapas do paradigma de Divisão e Conquista:

  - Dividir: Quebramos o problema em vários subproblemas menores.
    - ex: quebramos um vetor a ser ordenado em dois.
  - Conquistar: Os subproblemas são resolvidos recursivamente. Se eles forem pequenos o bastante, eles são resolvidos usando o próprio algoritmo que está sendo definido.
    - ex: um subvetor com um único elemento já está ordenado.
  - Combinar: Combinamos a solução dos problemas menores a fim de obter a solução para o problema maior.
    - ex: intercalamos os dois vetores ordenados.

# Ordenação por intercalação (*MergeSort*)

- Algoritmo criado por John Von Neumann em 1945.

# Ordenação por intercalação (*MergeSort*)



O algoritmo MergeSort segue de perto o paradigma de Divisão e Conquista. Intuitivamente, ele opera da seguinte maneira:

# Ordenação por intercalação (*MergeSort*)

O algoritmo MergeSort segue de perto o paradigma de Divisão e Conquista. Intuitivamente, ele opera da seguinte maneira:

- Dividir: Dado um vetor com $n = r - p + 1$ inteiros $A[p \dots r]$, que se deseja ordenar, divida esse vetor em dois subvetores de elementos subsequentes $A[p \dots q]$ e $A[q + 1 \dots r]$, de modo que cada um dos subvetores tenha tamanho aproximadamente $n/2$.

# Ordenação por intercalação (*MergeSort*)

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

O algoritmo MergeSort segue de perto o paradigma de Divisão e Conquista. Intuitivamente, ele opera da seguinte maneira:

- Dividir: Dado um vetor com $n = r - p + 1$ inteiros $A[p \ldots r]$, que se deseja ordenar, divida esse vetor em dois subvetores de elementos subsequentes $A[p \ldots q]$ e $A[q + 1 \ldots r]$, de modo que cada um dos subvetores tenha tamanho aproximadamente $n/2$.

- Conquistar: Ordene os dois subvetores $A[p \ldots q]$ e $A[q + 1 \ldots r]$ recursivamente, usando o MergeSort.

# Ordenação por intercalação (*MergeSort*)

O algoritmo MergeSort segue de perto o paradigma de Divisão e Conquista. Intuitivamente, ele opera da seguinte maneira:

- Dividir: Dado um vetor com $n = r - p + 1$ inteiros $A[p \dots r]$, que se deseja ordenar, divida esse vetor em dois subvetores de elementos subsequentes $A[p \dots q]$ e $A[q + 1 \dots r]$, de modo que cada um dos subvetores tenha tamanho aproximadamente $n/2$.
- Conquistar: Ordene os dois subvetores $A[p \dots q]$ e $A[q + 1 \dots r]$ recursivamente, usando o MergeSort.
- Combinar: Intercale os dois subvetores ordenados a fim de produzir o vetor com $n$ inteiros ordenados.

# Ordenação por intercalação (*MergeSort*)

Ordenação:

- Recebemos um vetor `A` de tamanho $n$ com limites:
    - O vetor começa na posição `A[p]`
    - O vetor termina na posição `A[r]`

# Ordenação por intercalação (*MergeSort*)

Ordenação:

- Recebemos um vetor `A` de tamanho $n$ com limites:
  - O vetor começa na posição `A[p]`
  - O vetor termina na posição `A[r]`
- Dividimos o vetor em dois subvetores de tamanho $n/2$

# Ordenação por intercalação (*MergeSort*)

Ordenação:

- Recebemos um vetor `A` de tamanho $n$ com limites:
  - O vetor começa na posição `A[p]`
  - O vetor termina na posição `A[r]`
- Dividimos o vetor em dois subvetores de tamanho $n/2$
- O caso base é um vetor de tamanho $0$ ou $1$

# Ordenação por intercalação (*MergeSort*)

Ordenação:

- Recebemos um vetor `A` de tamanho $n$ com limites:
  - O vetor começa na posição `A[p]`
  - O vetor termina na posição `A[r]`
- Dividimos o vetor em dois subvetores de tamanho $n/2$
- O caso base é um vetor de tamanho $0$ ou $1$

```
void mergesort(int A[], int p, int r) {
    if (p < r) {
        int q = (p + r) / 2; // Dividir
        // Conquistar
        mergesort(A, p, q);
        mergesort(A, q + 1, r);
        // Combinar
        Intercala(A, p, q, r);
    }
}
```

# Merge Sort — Simulação 1



| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 5 | 7 | 3 | 11 | 31 | 13 | 17 | 8 |

# Merge Sort — Simulação 1

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 5 | 7 | 3 | 11 | 31 | 13 | 17 | 8 |

| | | | |
|---|---|---|---|
| 5 | 7 | 3 | 11 |

| | | | |
|---|---|---|---|
| 31 | 13 | 17 | 8 |

# Merge Sort — Simulação 1



| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 5 | 7 | 3 | 11 | 31 | 13 | 17 | 8 |

| | | | |
|---|---|---|---|
| 5 | 7 | 3 | 11 |

| | | | |
|---|---|---|---|
| 31 | 13 | 17 | 8 |

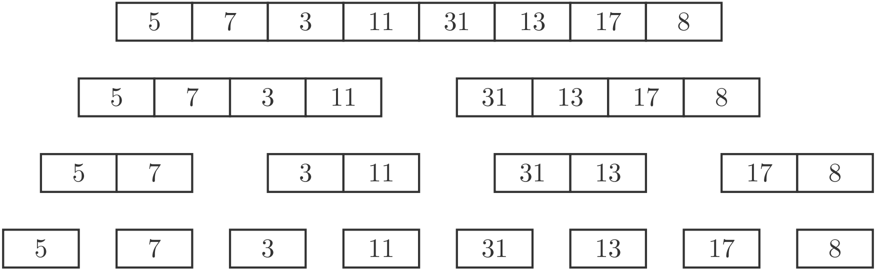
| | |
|---|---|
| 5 | 7 |

| | |
|---|---|
| 3 | 11 |

| | |
|---|---|
| 31 | 13 |

| | |
|---|---|
| 17 | 8 |

# Merge Sort — Simulação 1



# Merge Sort — Simulação 1



| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 5 | 7 | 3 | 11 | 31 | 13 | 17 | 8 |

[5 | 7 | 3 | 11]   [31 | 13 | 17 | 8]

[5 | 7]   [3 | 11]   [31 | 13]   [17 | 8]

[5]   [7]   [3]   [11]   [31]   [13]   [17]   [8]

[5 | 7]   [3 | 11]   [13 | 31]   [8 | 17]

# Merge Sort — Simulação 1



| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 5 | 7 | 3 | 11 | 31 | 13 | 17 | 8 |

| | | | |
|---|---|---|---|
| 5 | 7 | 3 | 11 |

| | | | |
|---|---|---|---|
| 31 | 13 | 17 | 8 |

| | |
|---|---|
| 5 | 7 |

| | |
|---|---|
| 3 | 11 |

| | |
|---|---|
| 31 | 13 |

| | |
|---|---|
| 17 | 8 |

| |
|---|
| 5 |

| |
|---|
| 7 |

| |
|---|
| 3 |

| |
|---|
| 11 |

| |
|---|
| 31 |

| |
|---|
| 13 |

| |
|---|
| 17 |

| |
|---|
| 8 |

| | |
|---|---|
| 5 | 7 |

| | |
|---|---|
| 3 | 11 |

| | |
|---|---|
| 13 | 31 |

| | |
|---|---|
| 8 | 17 |

| | | | |
|---|---|---|---|
| 3 | 5 | 7 | 11 |

| | | | |
|---|---|---|---|
| 8 | 13 | 17 | 31 |

# Merge Sort — Simulação 1

5 7 3 11 31 13 17 8

5 7 3 11     31 13 17 8

5 7     3 11     31 13     17 8

5     7     3     11     31     13     17     8

5 7     3 11     13 31     8 17

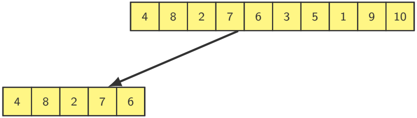
3 5 7 11     8 13 17 31

3 5 7 8 11 13 17 31

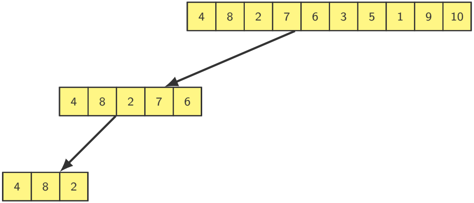
# Merge Sort — Simulação 2

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 4 | 8 | 2 | 7 | 6 | 3 | 5 | 1 | 9 | 10 |

# Merge Sort — Simulação 2

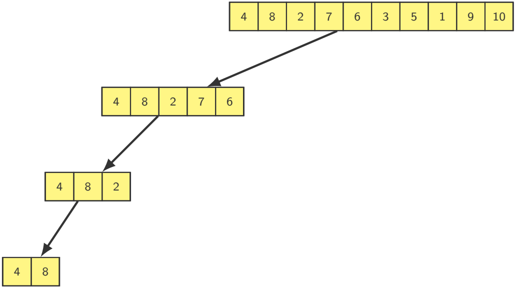


# Merge Sort — Simulação 2



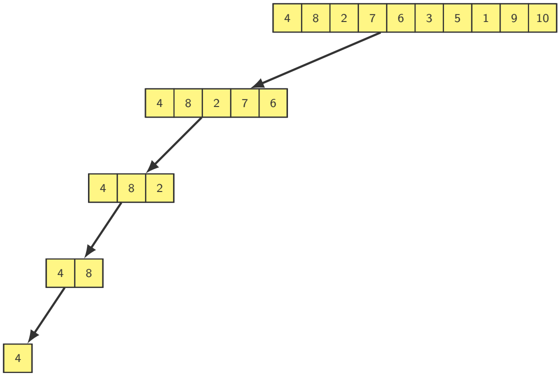
# Merge Sort — Simulação 2



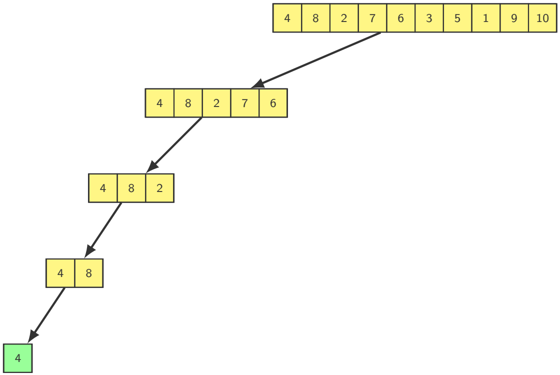


# Merge Sort — Simulação 2

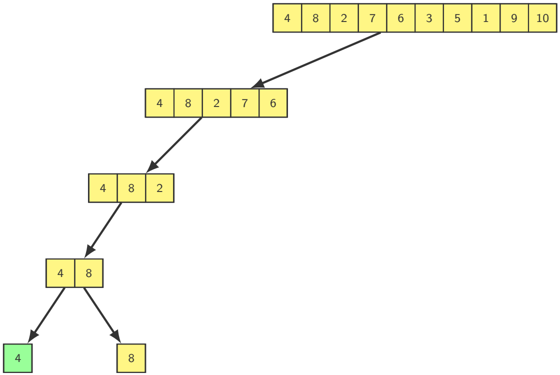
UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ



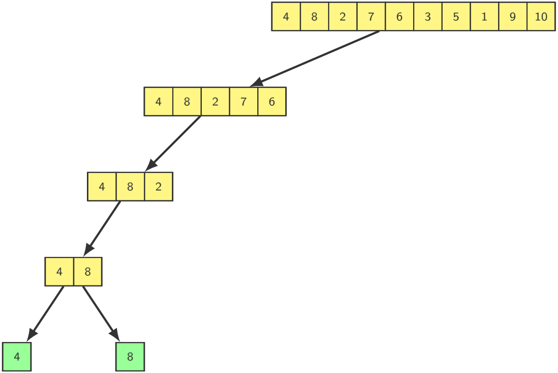
# Merge Sort — Simulação 2



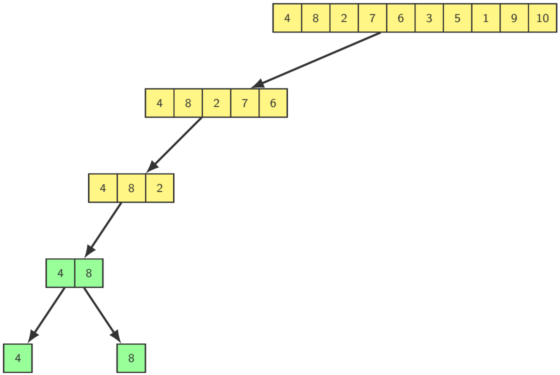
# Merge Sort — Simulação 2



# Merge Sort — Simulação 2

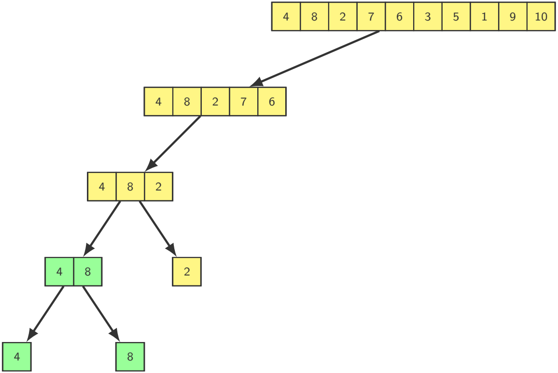


# Merge Sort — Simulação 2

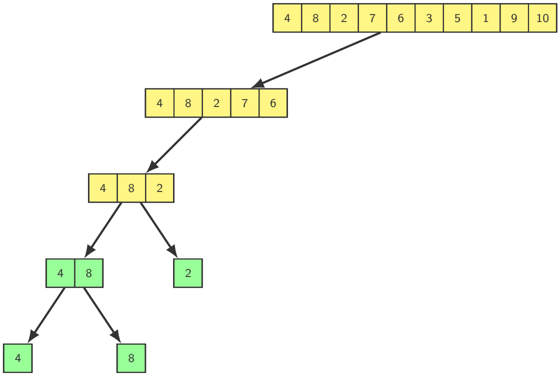


# Merge Sort — Simulação 2

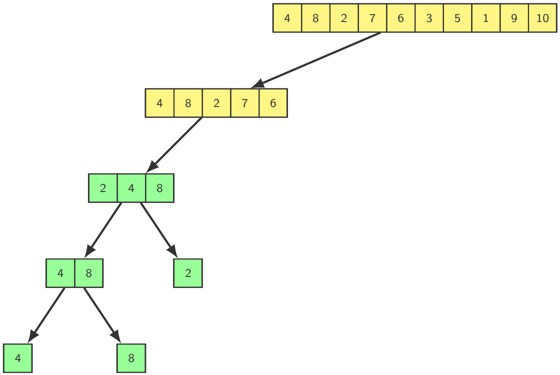
Universidade Federal do Ceará
Campus Quixadá



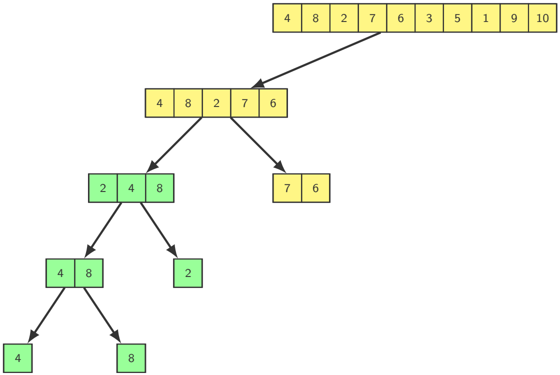
# Merge Sort — Simulação 2



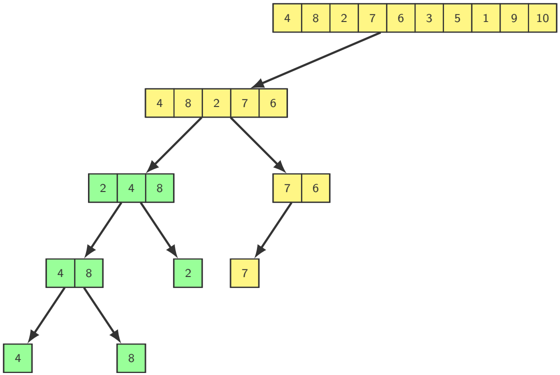
# Merge Sort — Simulação 2



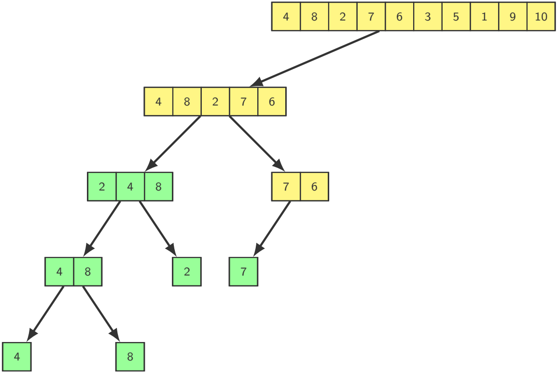
# Merge Sort — Simulação 2



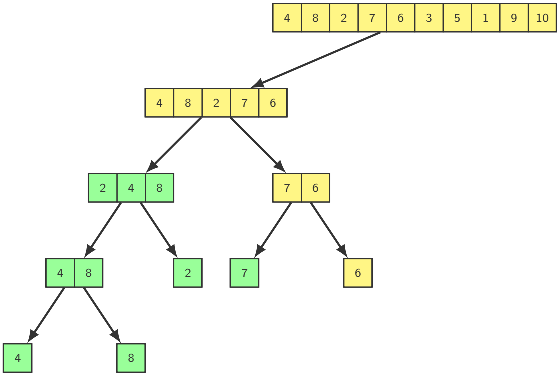
# Merge Sort — Simulação 2



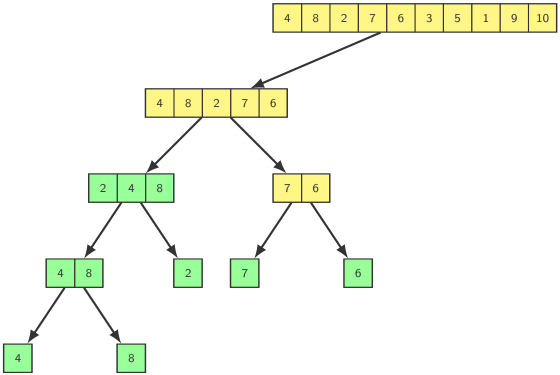
# Merge Sort — Simulação 2



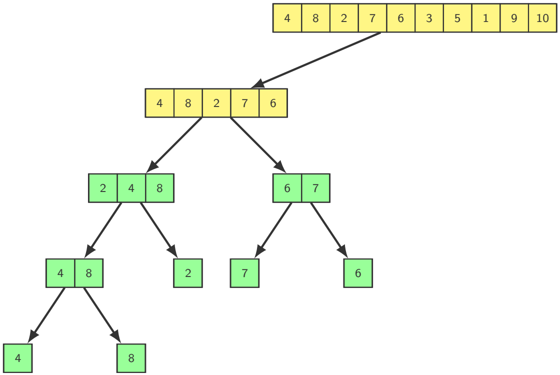
# Merge Sort — Simulação 2



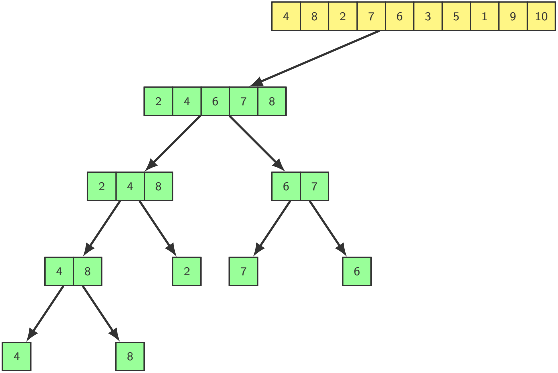
# Merge Sort — Simulação 2



# Merge Sort — Simulação 2

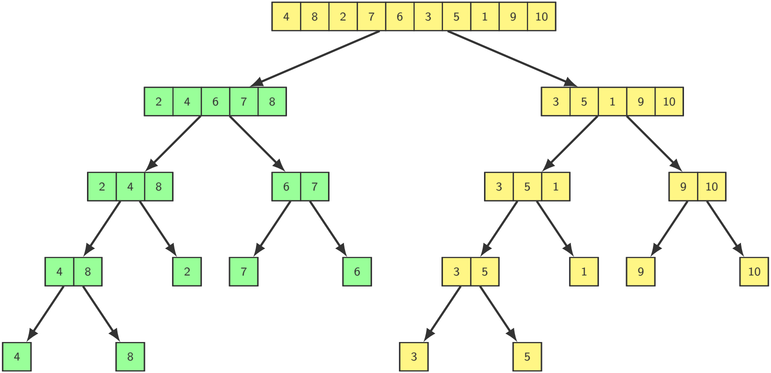


# Merge Sort — Simulação 2

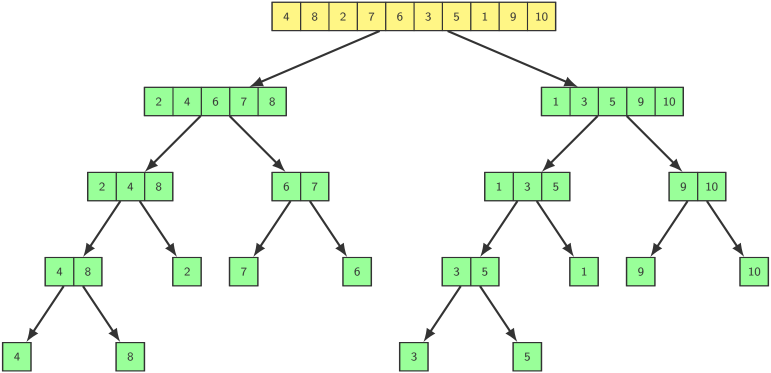


# Merge Sort — Simulação 2

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

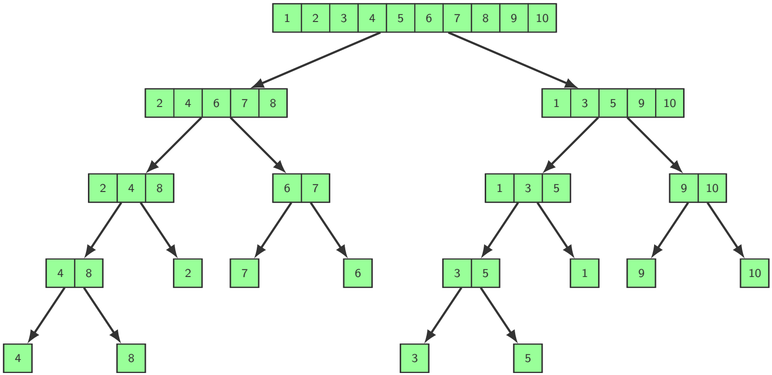


# Merge Sort — Simulação 2

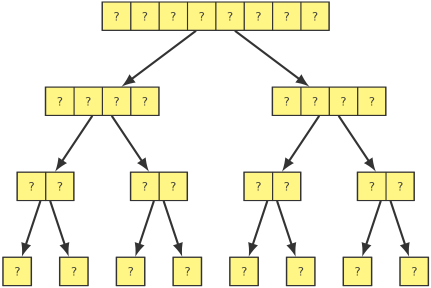


# Merge Sort — Simulação 2

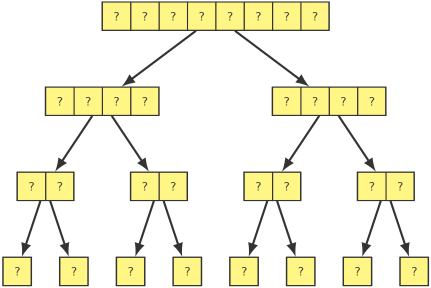




# Tempo de execução para $n = 2^l$



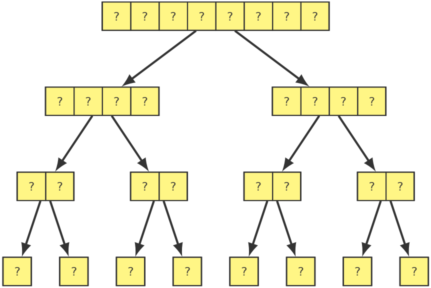
# Tempo de execução para $n = 2^l$




$\leq c \cdot n$

- No primeiro nível fazemos um `merge` com $n$ elementos

# Tempo de execução para $n = 2^l$



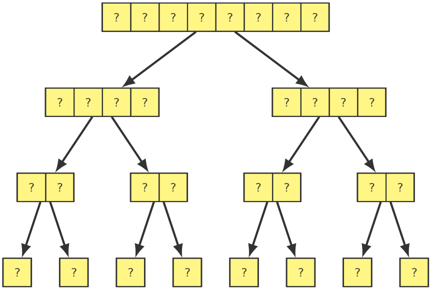

$\leq c \cdot n$

$\leq c \cdot 2(n/2)$

- No primeiro nível fazemos um `merge` com $n$ elementos
- No segundo fazemos dois `merge` com $n/2$ elementos

# Tempo de execução para $n = 2^l$

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ



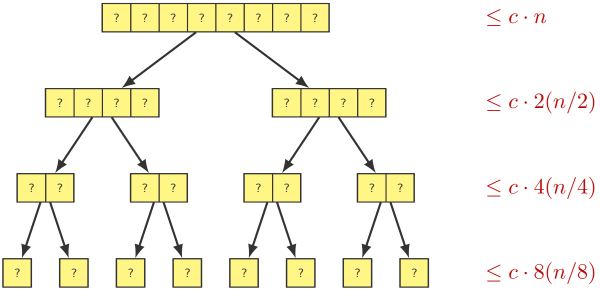
$\leq c \cdot n$

$\leq c \cdot 2(n/2)$

$\leq c \cdot 4(n/4)$

- No primeiro nível fazemos um `merge` com $n$ elementos
- No segundo fazemos dois `merge` com $n/2$ elementos
- No $(k+1)$-ésimo fazemos $2^k$ `merge` com $n/2^k$ elementos

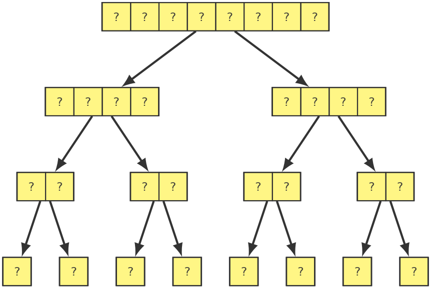
# Tempo de execução para $n = 2^l$




- No primeiro nível fazemos um `merge` com $n$ elementos
- No segundo fazemos dois `merge` com $n/2$ elementos
- No $(k+1)$-ésimo fazemos $2^k$ `merge` com $n/2^k$ elementos
- No último gastamos tempo constante $n$ vezes

# Tempo de execução para $n = 2^l$





$\leq c \cdot n$

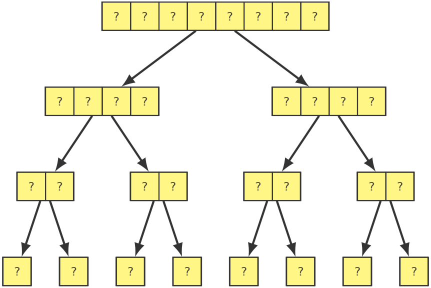
$\leq c \cdot 2(n/2)$

$\leq c \cdot 4(n/4)$

$\leq c \cdot 8(n/8)$

- No nível $k$ gastamos tempo $\leq c \cdot n$

# Tempo de execução para $n = 2^l$




$\leq c \cdot n$

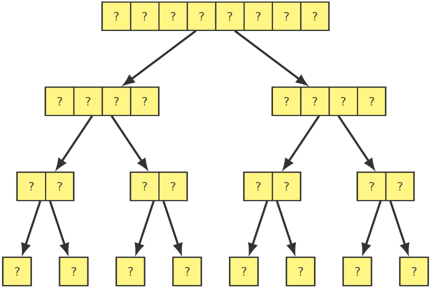
$\leq c \cdot 2(n/2)$

$\leq c \cdot 4(n/4)$

$\leq c \cdot 8(n/8)$

- No nível $k$ gastamos tempo $\leq c \cdot n$
- Quantos níveis temos?

# Tempo de execução para $n = 2^l$




$\leq c \cdot n$

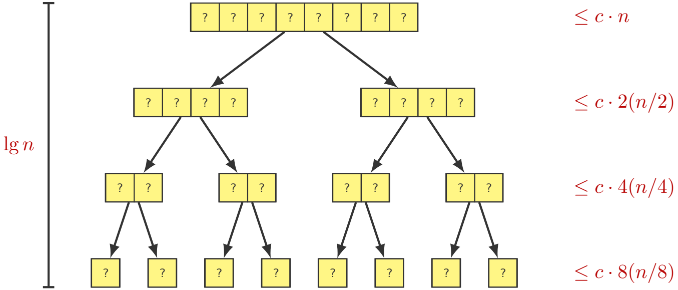
$\leq c \cdot 2(n/2)$

$\leq c \cdot 4(n/4)$

$\leq c \cdot 8(n/8)$

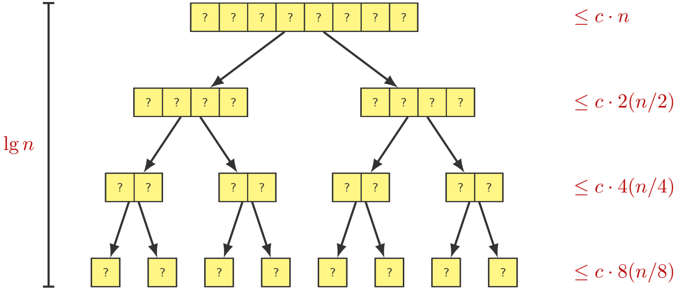
- No nível $k$ gastamos tempo $\leq c \cdot n$
- Quantos níveis temos?
  - Dividimos $n$ por $2$ até que fique menor ou igual a $1$

# Tempo de execução para $n = 2^l$



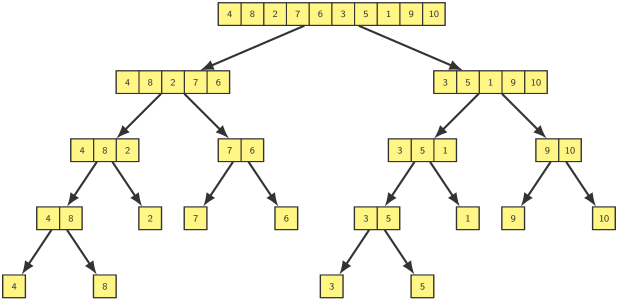

- No nível $k$ gastamos tempo $\leq c \cdot n$
- Quantos níveis temos?
  - Dividimos $n$ por $2$ até que fique menor ou igual a $1$
  - Ou seja, $l = \lg n$

# Tempo de execução para $n = 2^l$

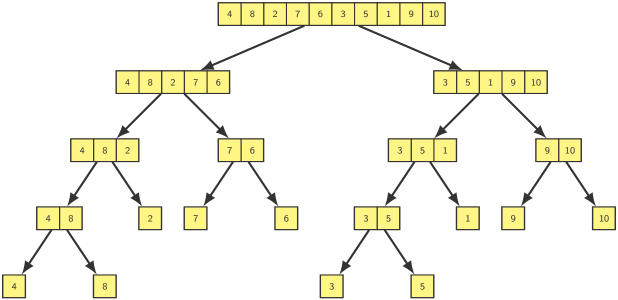



- No nível $k$ gastamos tempo $\leq c \cdot n$
- Quantos níveis temos?
  - Dividimos $n$ por $2$ até que fique menor ou igual a $1$
  - Ou seja, $l = \lg n$
- Tempo total: $c\, n \lg n = O(n \lg n)$

# Tempo de execução para $n$ qualquer

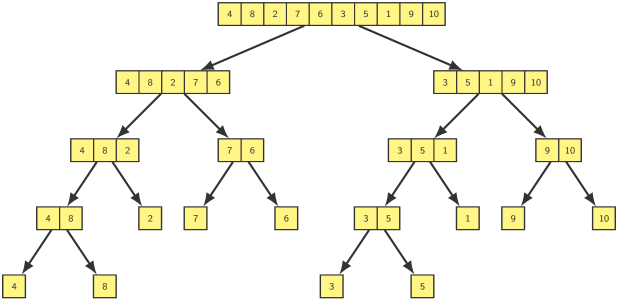


# Tempo de execução para $n$ qualquer




Qual o tempo de execução para $n$ que não é potência de $2$?

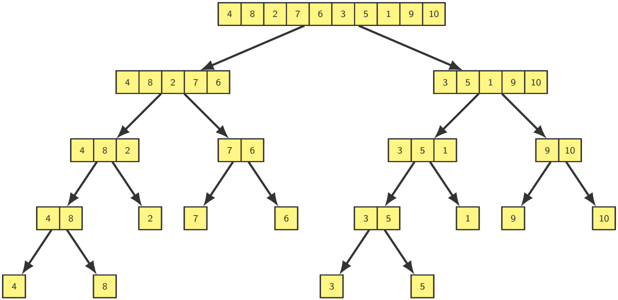
# Tempo de execução para $n$ qualquer




Qual o tempo de execução para $n$ que não é potência de $2$?

- Seja $2^k$ a próxima potência de $2$ depois de $n$

# Tempo de execução para $n$ qualquer

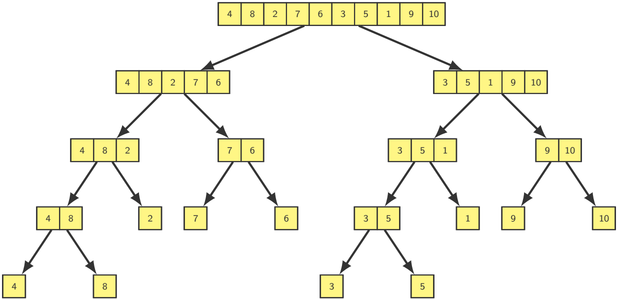



Qual o tempo de execução para $n$ que não é potência de $2$?

- Seja $2^k$ a próxima potência de $2$ depois de $n$
  - Exemplo: Se $n = 3000$, a próxima potência é $4096 = 2^{12}$

# Tempo de execução para $n$ qualquer

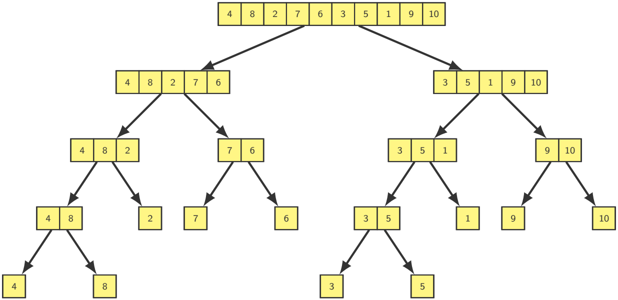





Qual o tempo de execução para $n$ que não é potência de $2$?

- Seja $2^k$ a próxima potência de $2$ depois de $n$
  - Exemplo: Se $n = 3000$, a próxima potência é $4096 = 2^{12}$
- Temos que $2^{k-1} < n < 2^k$. Ou seja, $2^k < 2n$.

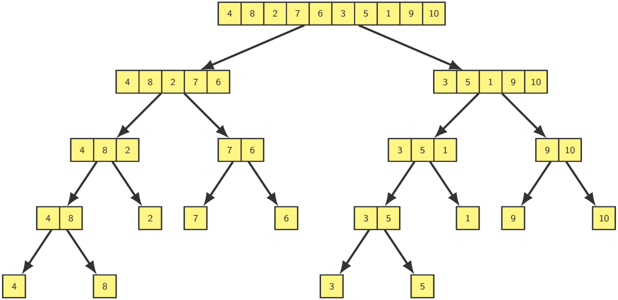
# Tempo de execução para $n$ qualquer




Qual o tempo de execução para $n$ que não é potência de $2$?

- Seja $2^k$ a próxima potência de $2$ depois de $n$
  - Exemplo: Se $n = 3000$, a próxima potência é $4096 = 2^{12}$
- Temos que $2^{k-1} < n < 2^k$. Ou seja, $2^k < 2n$.
- O tempo de execução para $n$ é menor do que

# Tempo de execução para $n$ qualquer

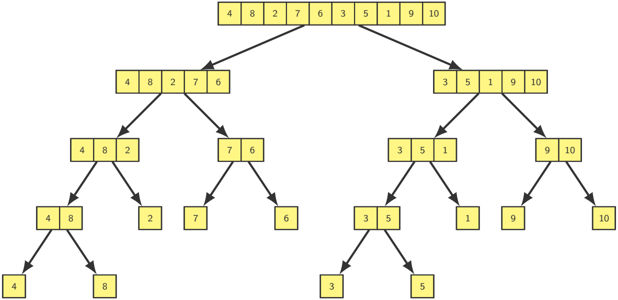



Qual o tempo de execução para $n$ que não é potência de $2$?

- Seja $2^k$ a próxima potência de $2$ depois de $n$
  - Exemplo: Se $n = 3000$, a próxima potência é $4096 = 2^{12}$
- Temos que $2^{k-1} < n < 2^k$. Ou seja, $2^k < 2n$.
- O tempo de execução para $n$ é menor do que

$$c\,2^k\,\lg 2^k$$

# Tempo de execução para $n$ qualquer



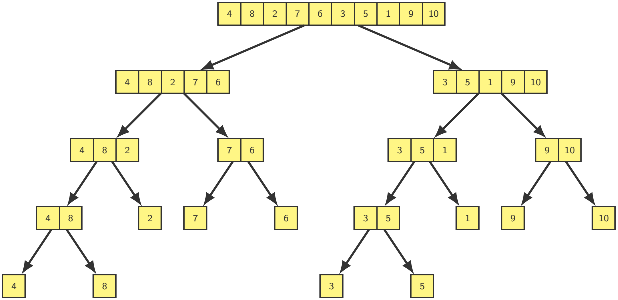

Qual o tempo de execução para $n$ que não é potência de $2$?

- Seja $2^k$ a próxima potência de $2$ depois de $n$
  - Exemplo: Se $n = 3000$, a próxima potência é $4096 = 2^{12}$
- Temos que $2^{k-1} < n < 2^k$. Ou seja, $2^k < 2n$.
- O tempo de execução para $n$ é menor do que

$$c\,2^k \lg 2^k$$

# Tempo de execução para $n$ qualquer





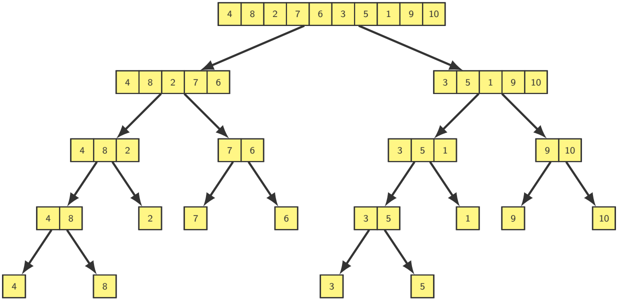

Qual o tempo de execução para $n$ que não é potência de $2$?

- Seja $2^k$ a próxima potência de $2$ depois de $n$
    - Exemplo: Se $n = 3000$, a próxima potência é $4096 = 2^{12}$
- Temos que $2^{k-1} < n < 2^k$. Ou seja, $2^k < 2n$.
- O tempo de execução para $n$ é menor do que

$$c\,2^k \lg 2^k \leq 2cn \lg(2n)$$

# Tempo de execução para $n$ qualquer

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

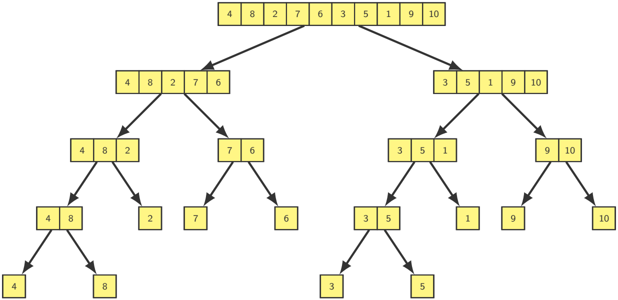



Qual o tempo de execução para $n$ que não é potência de $2$?

- Seja $2^k$ a próxima potência de $2$ depois de $n$
  - Exemplo: Se $n = 3000$, a próxima potência é $4096 = 2^{12}$
- Temos que $2^{k-1} < n < 2^k$. Ou seja, $2^k < 2n$.
- O tempo de execução para $n$ é menor do que

$$c\,2^k \lg 2^k \leq 2cn \lg(2n) = 2cn(\lg 2 + \lg n)$$

# Tempo de execução para $n$ qualquer



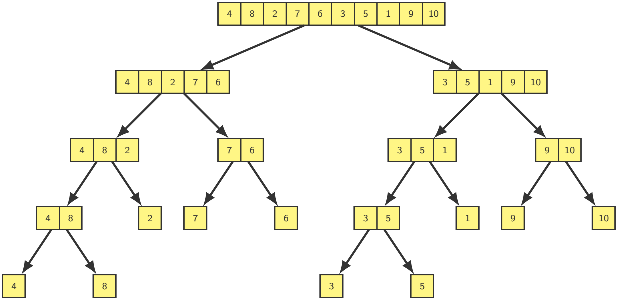

Qual o tempo de execução para $n$ que não é potência de $2$?

- Seja $2^k$ a próxima potência de $2$ depois de $n$
  - Exemplo: Se $n = 3000$, a próxima potência é $4096 = 2^{12}$
- Temos que $2^{k-1} < n < 2^k$. Ou seja, $2^k < 2n$.
- O tempo de execução para $n$ é menor do que

$$c\,2^k\,\lg 2^k \leq 2cn\,\lg(2n) = 2cn(\lg 2 + \lg n) = 2cn + 2cn\lg n$$

# Tempo de execução para $n$ qualquer






Qual o tempo de execução para $n$ que não é potência de $2$?

- Seja $2^k$ a próxima potência de $2$ depois de $n$
  - Exemplo: Se $n = 3000$, a próxima potência é $4096 = 2^{12}$
- Temos que $2^{k-1} < n < 2^k$. Ou seja, $2^k < 2n$.
- O tempo de execução para $n$ é menor do que

$$c\,2^k \lg 2^k \leq 2cn \lg(2n) = 2cn(\lg 2 + \lg n) = 2cn + 2cn \lg n = O(n \lg n)$$

# Exercícios

# Exercício

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

Faça uma versão do MergeSort para listas duplamente encadeadas.

# Exercício

Implemente a função
`void mergeAB(int *v, int *a, int n, int *b, int m)`
que dados vetores `a` e `b` de tamanho `n` e `m` faz a intercalação de `a` e `b` e armazena no vetor `v`. Suponha que `v` já está alocado e que tem tamanho maior ou igual a `n+m`.

# FIM