Mergesort — Ordenação por intercalação

Estrutura de Dados — QXD0010

Prof. Atílio Gomes Luiz

gomes.atilio@ufc.br

Universidade Federal do Ceará

2º semestre/2025

Introdução

- Dizemos que um algoritmo de ordenação é **baseado em comparações** se, a fim de decidir a ordem de dois elementos, ele só pode compará-los. Ou seja, o algoritmo não usa propriedades específicas dos dados, apenas perguntas do tipo:

  "*O elemento A é menor que o elemento B?*"

Introdução

- Dizemos que um algoritmo de ordenação é **baseado em comparações** se, a fim de decidir a ordem de dois elementos, ele só pode compará-los. Ou seja, o algoritmo não usa propriedades específicas dos dados, apenas perguntas do tipo:

  "*O elemento A é menor que o elemento B?*"

- Existe uma prova formal de que todo algoritmo de ordenação baseado em comparações requer pelo menos $\Omega(n \log_2 n)$ no pior caso.

Introdução

- Dizemos que um algoritmo de ordenação é **baseado em comparações** se, a fim de decidir a ordem de dois elementos, ele só pode compará-los. Ou seja, o algoritmo não usa propriedades específicas dos dados, apenas perguntas do tipo:

  "*O elemento A é menor que o elemento B?*"

- Existe uma prova formal de que todo algoritmo de ordenação baseado em comparações requer pelo menos $\Omega(n \log_2 n)$ no pior caso.

- Nesta aula veremos um algoritmo de ordenação $O(n \log n)$ comparações no pior caso.

Introdução

- Dizemos que um algoritmo de ordenação é **baseado em comparações** se, a fim de decidir a ordem de dois elementos, ele só pode compará-los. Ou seja, o algoritmo não usa propriedades específicas dos dados, apenas perguntas do tipo:

  "*O elemento A é menor que o elemento B?*"

- Existe uma prova formal de que todo algoritmo de ordenação baseado em comparações requer pelo menos $\Omega(n \log_2 n)$ no pior caso.

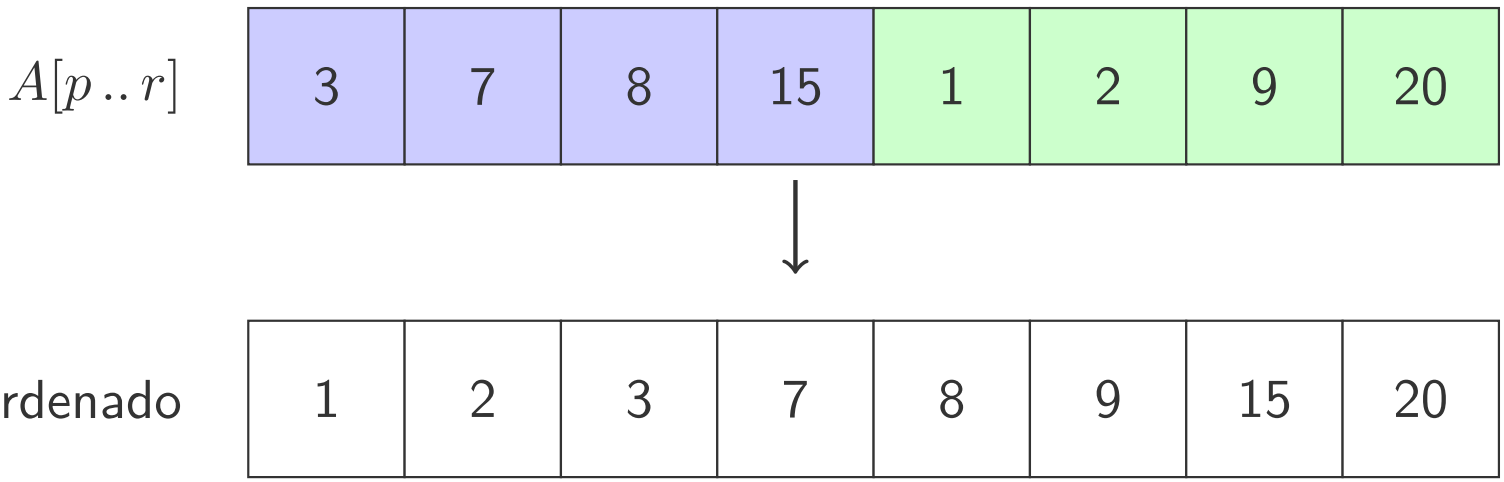
- Nesta aula veremos um algoritmo de ordenação $O(n \log n)$ comparações no pior caso.
  - Ele é baseado na técnica **Dividir para Conquistar**

Um problema subjacente: Intercalação de dois subvetores ordenados

Intercalação de Subvetores Ordenados

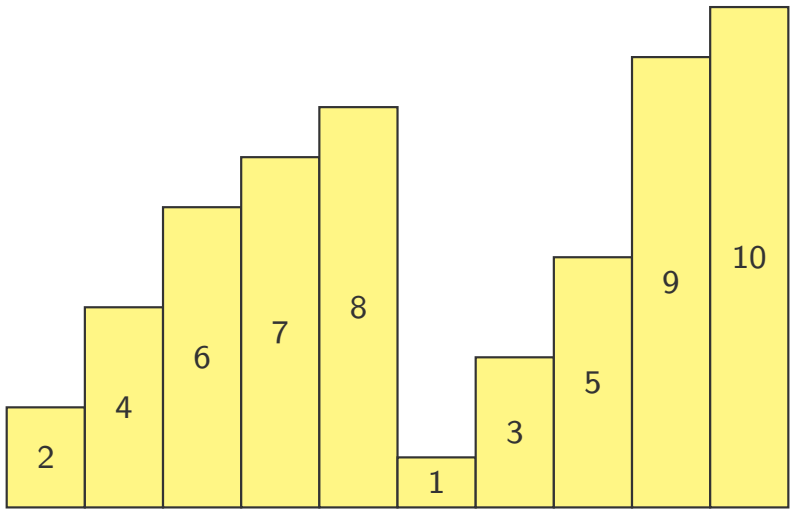
UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

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



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

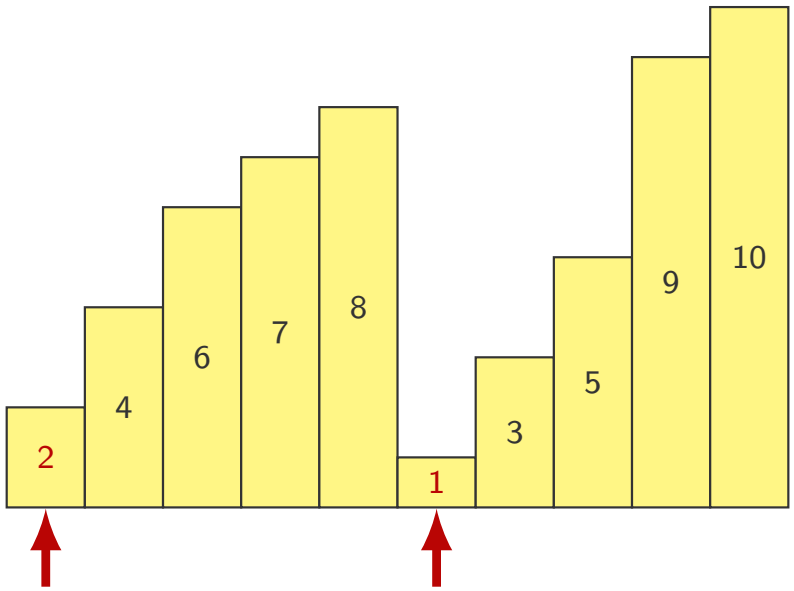
Intercalando dois subvetores ordenados



- Percorremos os dois subvetores

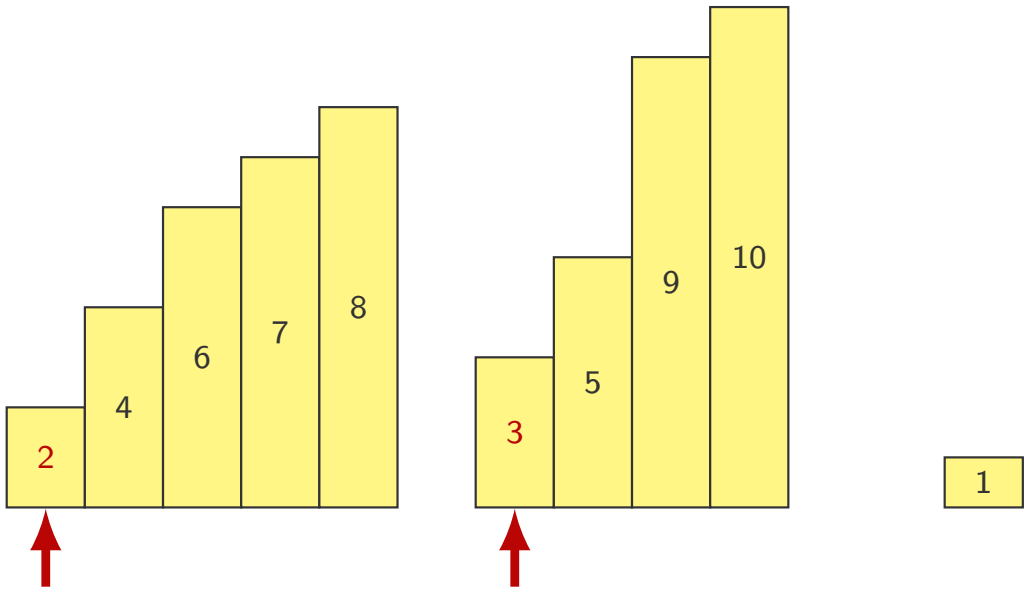
Intercalando dois subvetores ordenados

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ



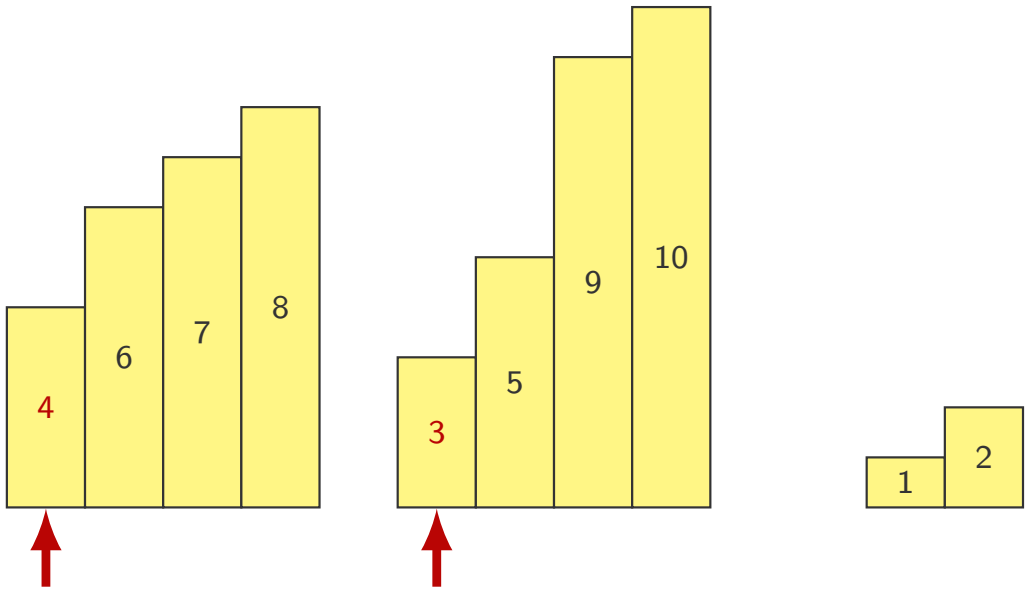
- Percorremos os dois subvetores
- Pegamos o mínimo e inserimos em um vetor auxiliar

Intercalando dois subvetores ordenados



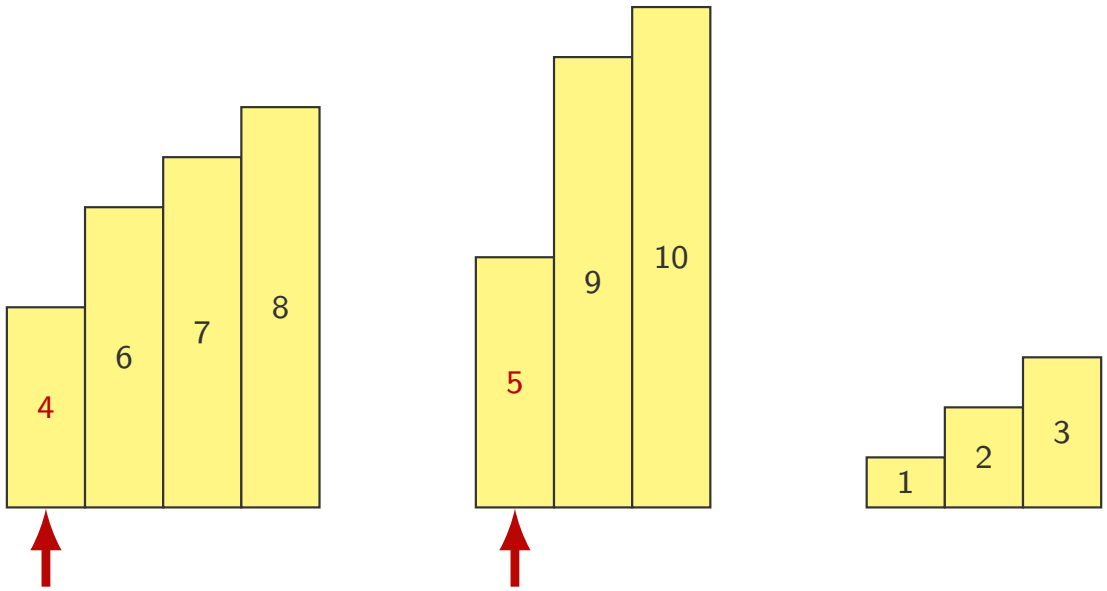
- Percorremos os dois subvetores
- Pegamos o mínimo e inserimos em um vetor auxiliar

Intercalando dois subvetores ordenados



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

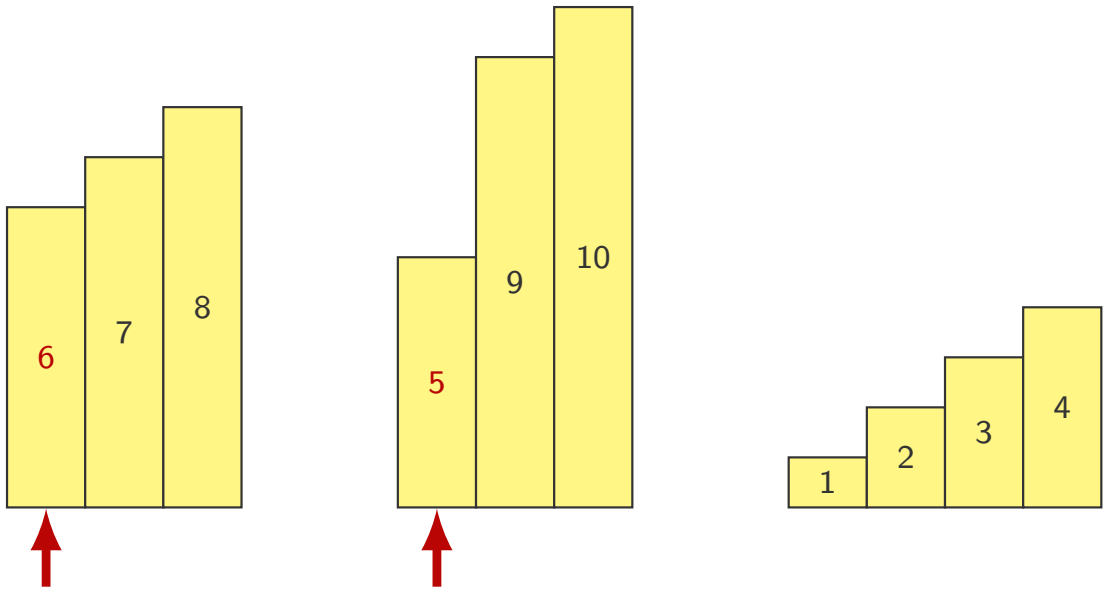
Intercalando dois subvetores ordenados



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

Intercalando dois subvetores ordenados

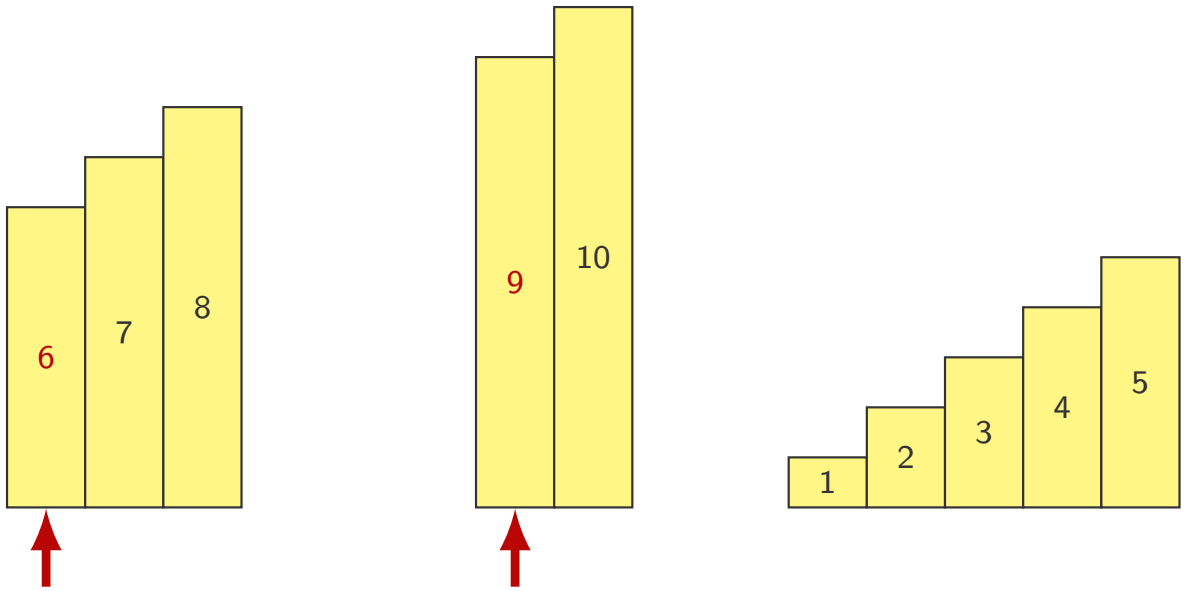
UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ



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

Intercalando dois subvetores ordenados

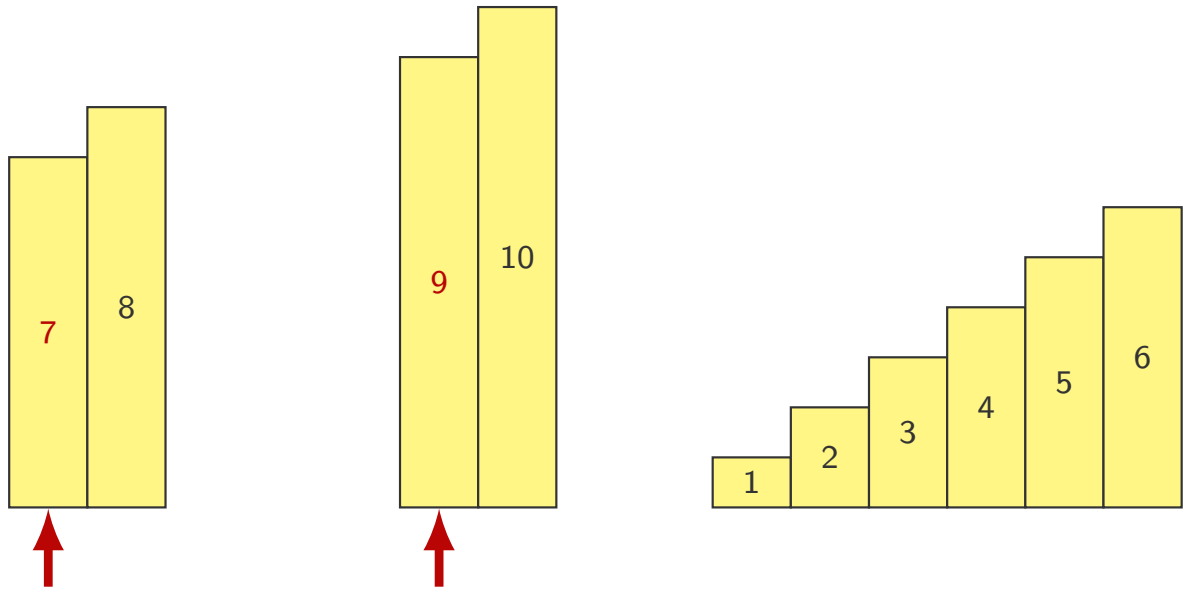
UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ



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

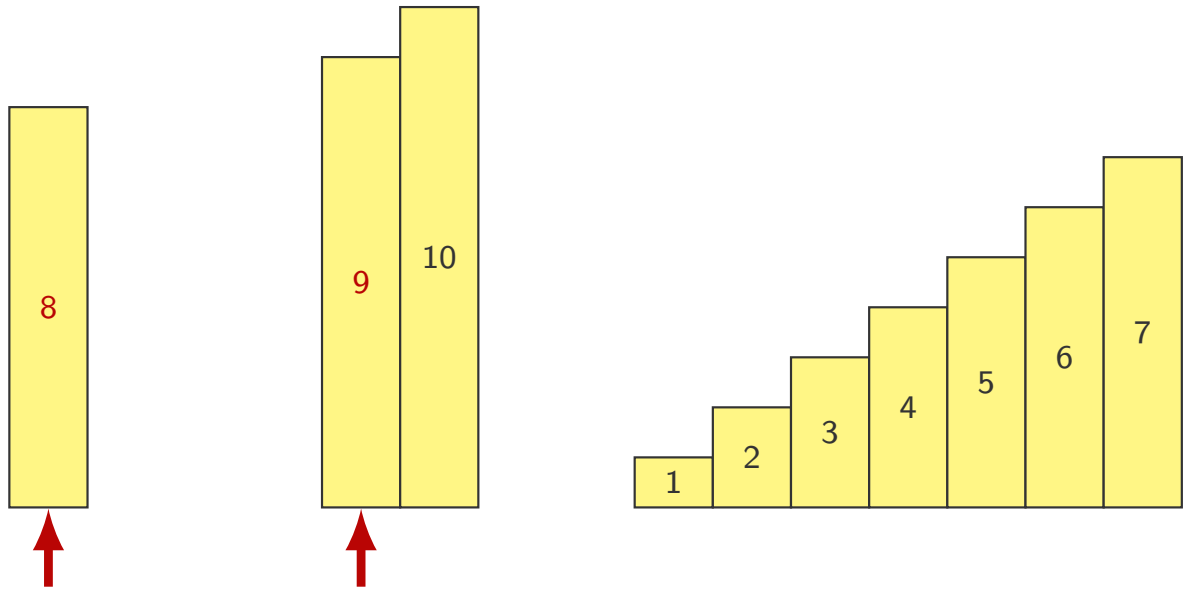
Intercalando dois subvetores ordenados

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ



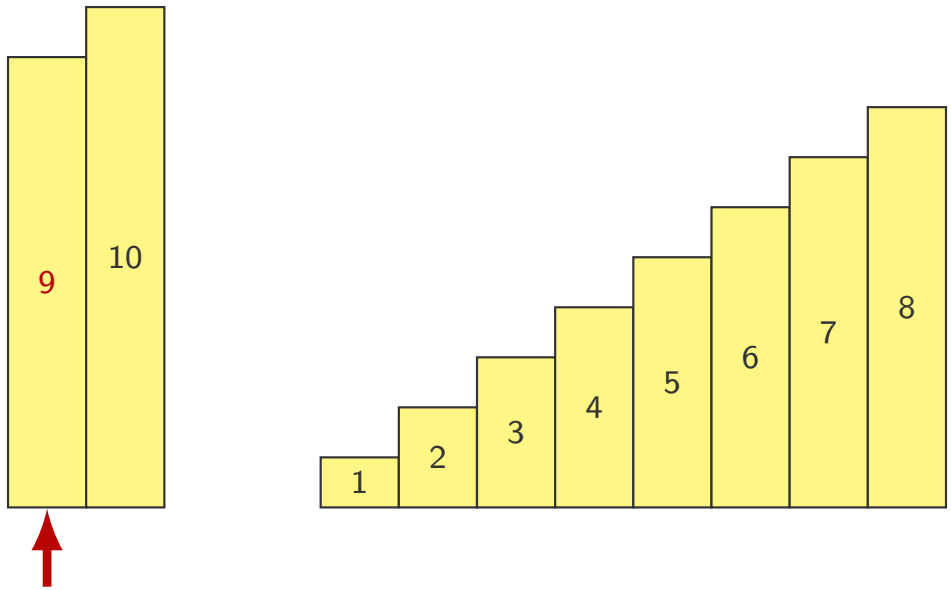
- Percorremos os dois subvetores
- Pegamos o mínimo e inserimos em um vetor auxiliar

Intercalando dois subvetores ordenados



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

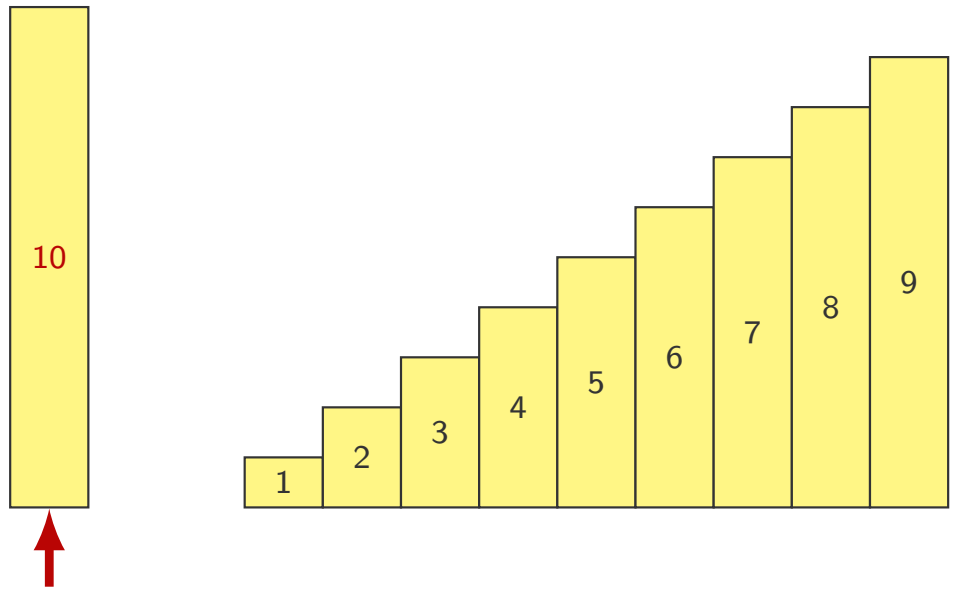
Intercalando dois subvetores ordenados



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

Intercalando dois subvetores ordenados

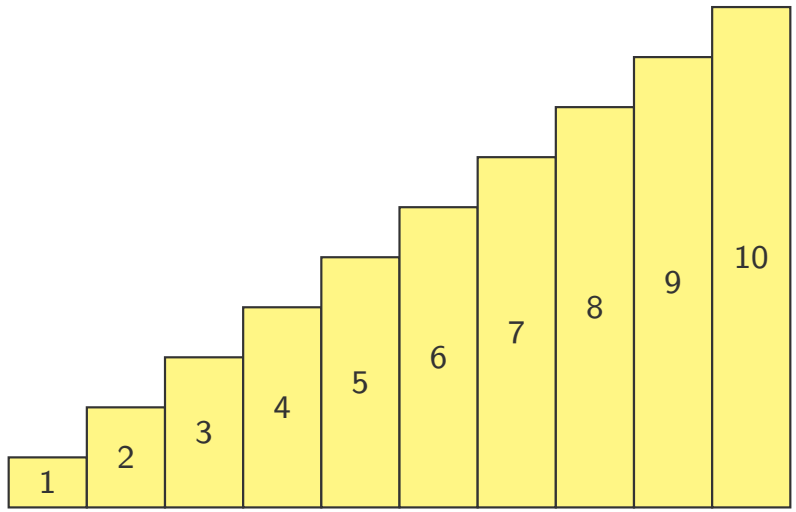
Universidade Federal do Ceará
Campus Quixadá



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

Intercalando dois subvetores ordenados

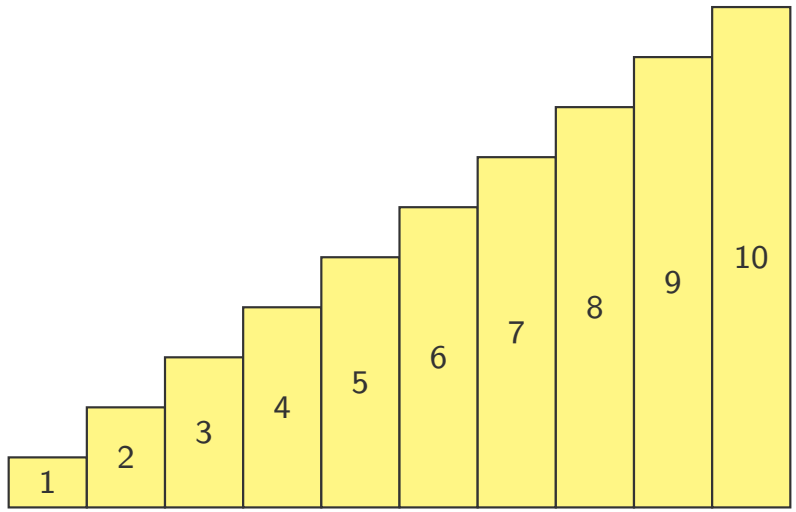
UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ



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

Intercalando dois subvetores ordenados

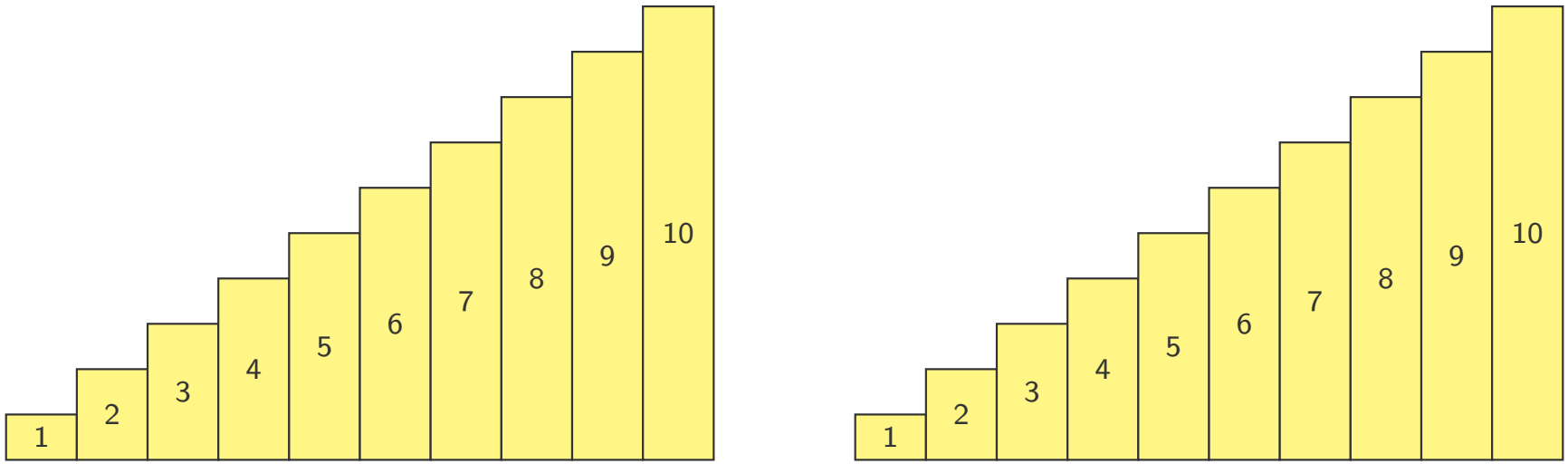
UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ



- 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 subvetores ordenados

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ



- 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 subvetores ordenados

Universidade Federal do Ceará
Campus Quixadá

```
1  /* A funcao recebe vetores crescentes A[p..q] e A[q+1..r]
2   * e rearranja A[p..r] em ordem crescente */
3  void Intercala (int A[], int p, int q, int r) {
4    int W[r-p+1]; // Vetor auxiliar
5    int i = p, j = q+1, k = 0;
6
7    // Intercala A[p..q] e A[q+1..r]
8    while (i <= q && j <= r) {
9      if (A[i] <= A[j])
10       W[k++] = A[i++];
11     else
12       W[k++] = A[j++];
13   }
14   while (i <= q) W[k++] = A[i++];
15   while (j <= r) W[k++] = A[j++];
16
17   // Copia vetor ordenado W para o vetor A
18   for (i = p; i <= r; i++)
19     A[i] = W[i-p];
20 }
```

Complexidade da Intercalação

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

Quantas comparações são feitas?

Complexidade 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`

Complexidade 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 $n = r - p + 1$

Complexidade 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 $n = r - p + 1$
- Logo, o consumo de tempo no pior caso é proporcional ao número de elementos do vetor, ou seja, $O(n)$.

Complexidade 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 $n = r - p + 1$
- 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

Divisão e Conquista é um paradigma de projeto de algoritmos em que um problema é resolvido dividindo-o em subproblemas menores, resolvendo recursivamente cada um deles e, em seguida, combinando essas soluções para formar a solução final.

Divisão e conquista

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

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

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*)

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

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 $A[p \ldots r]$, 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 $n/2$.

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 $A[p \ldots r]$, 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 $A[p \ldots r]$, 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.

- Combinar: Intercale os dois subvetores ordenados a fim de produzir o vetor com n inteiros ordenados.

Ordenação por intercalação (*MergeSort*)

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

Ordenação:

- Recebemos um vetor `A` de tamanho $n = r - p + 1$ 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 $n = r - p + 1$ 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 $n = r - p + 1$ 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*)

Universidade Federal do Ceará
Campus Quixadá

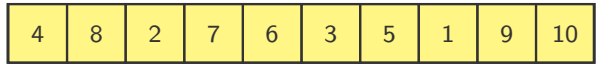
Ordenação:

- Recebemos um vetor `A` de tamanho $n = r - p + 1$ 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

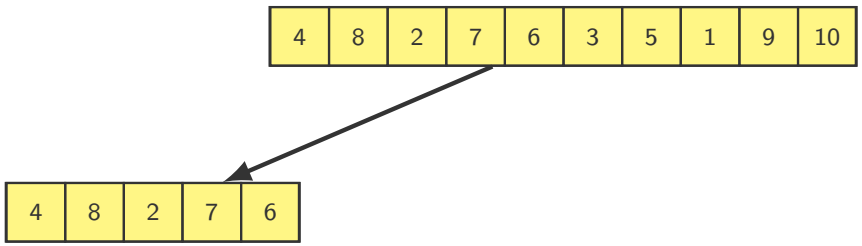
```
1  void mergesort(int A[], int p, int r) {
2      if (p < r) {
3          int q = (p + r) / 2; // Dividir
4          // Conquistar
5          mergesort(A, p, q);
6          mergesort(A, q + 1, r);
7          // Combinar
8          Intercala(A, p, q, r);
9      }
10 }
```

Merge Sort — Simulação

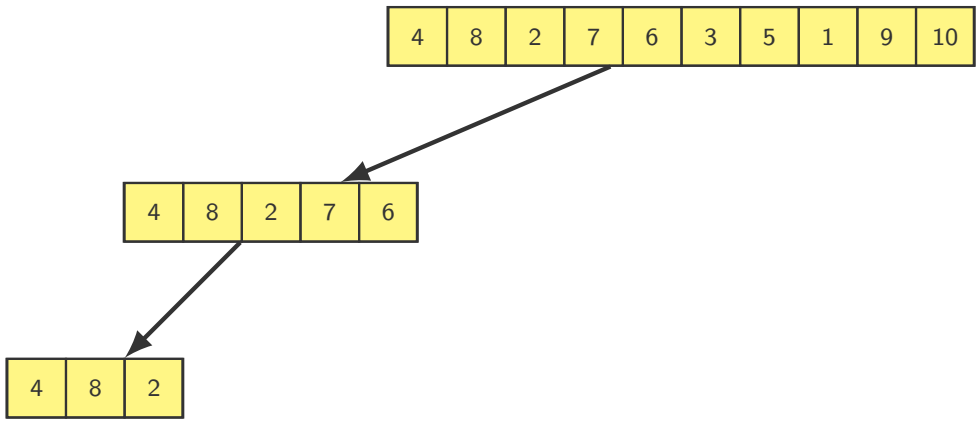
UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ



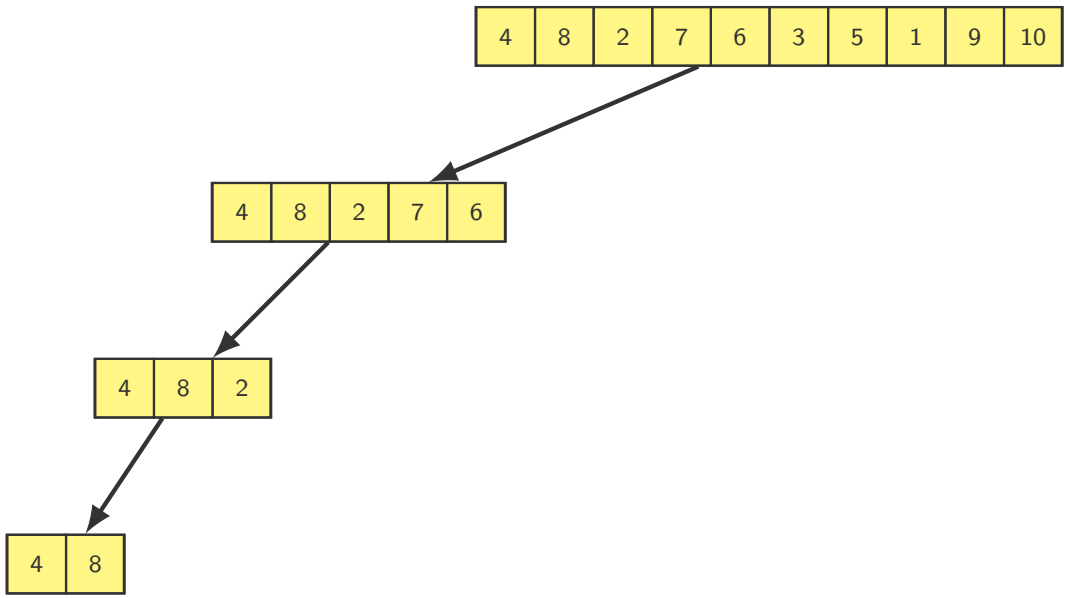
Merge Sort — Simulação



Merge Sort — Simulação

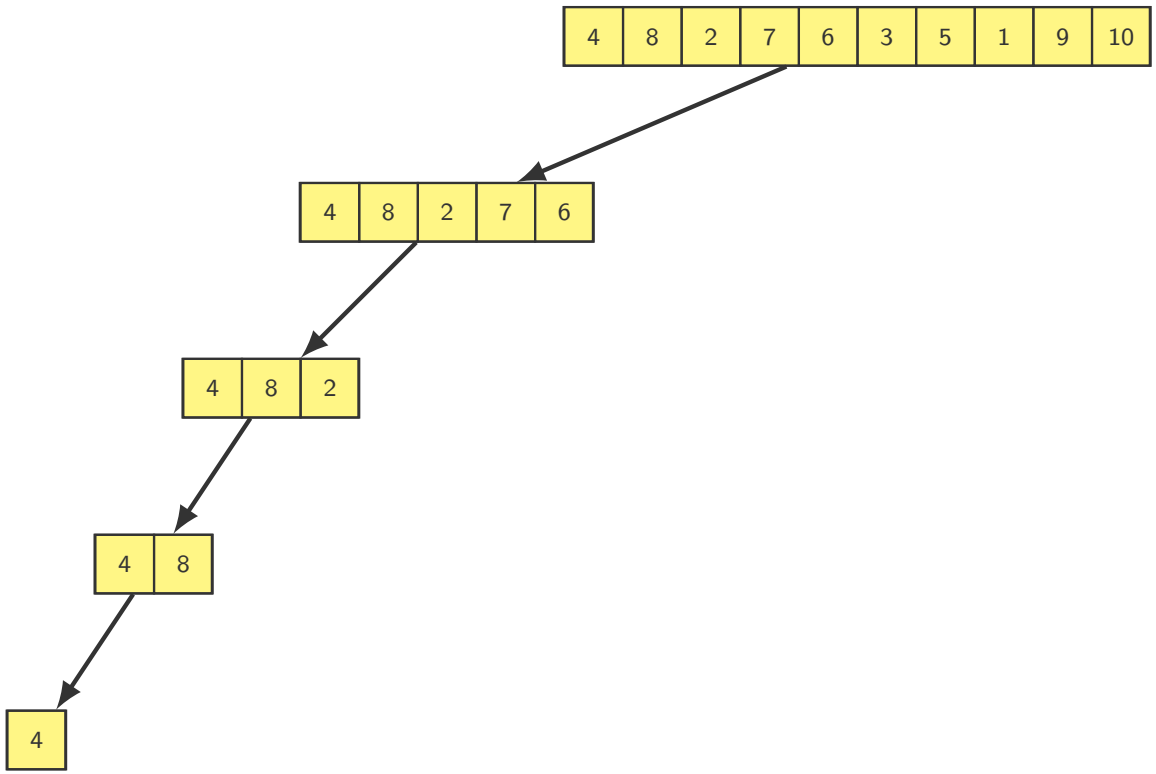


Merge Sort — Simulação



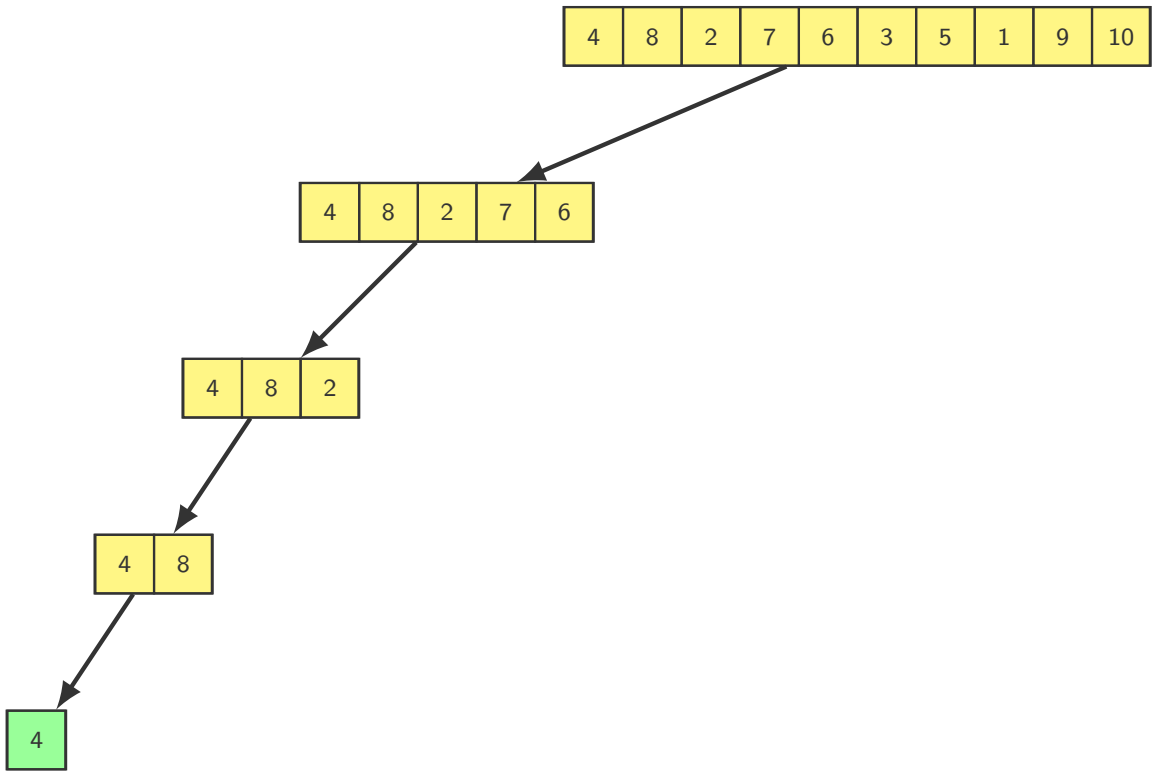
Merge Sort — Simulação

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ



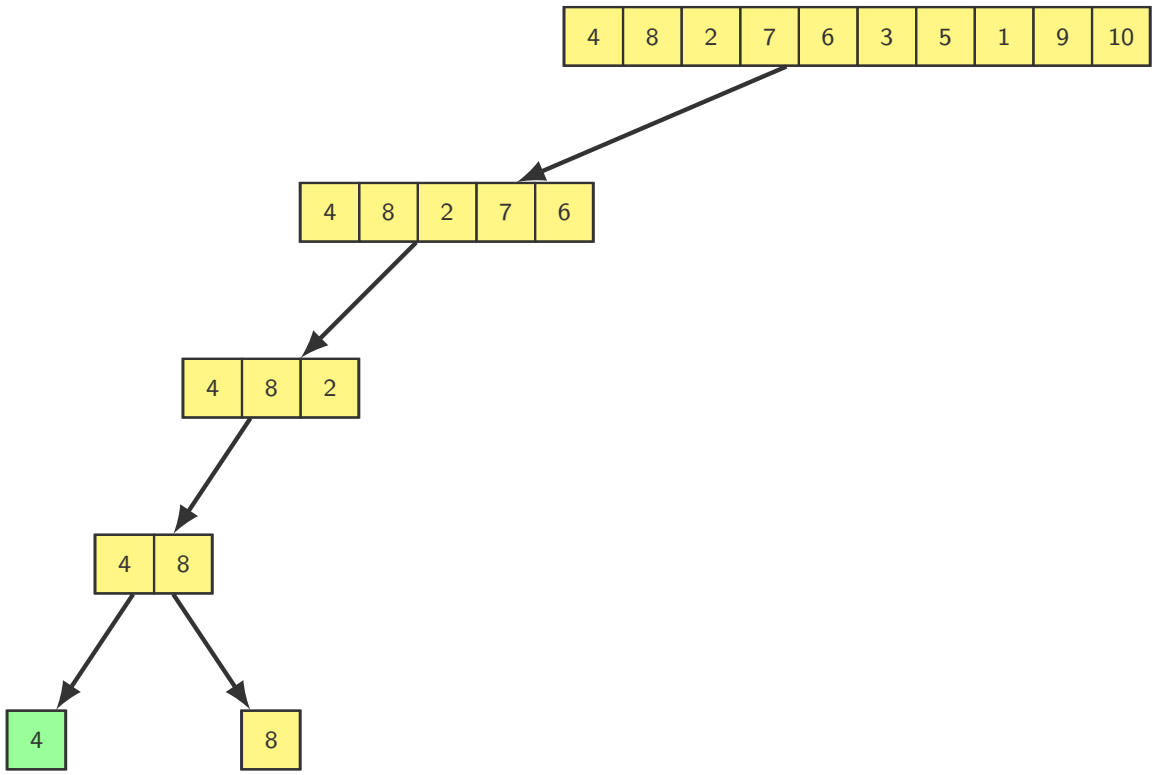
Merge Sort — Simulação

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ



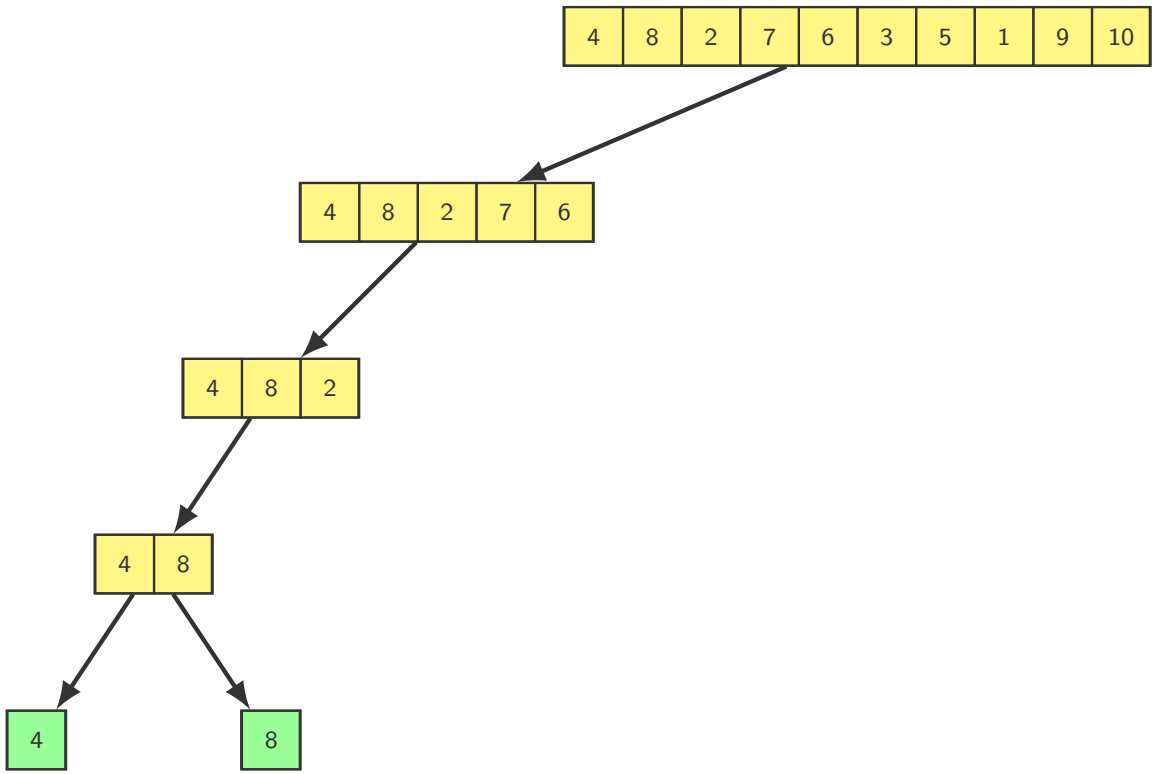
Merge Sort — Simulação

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

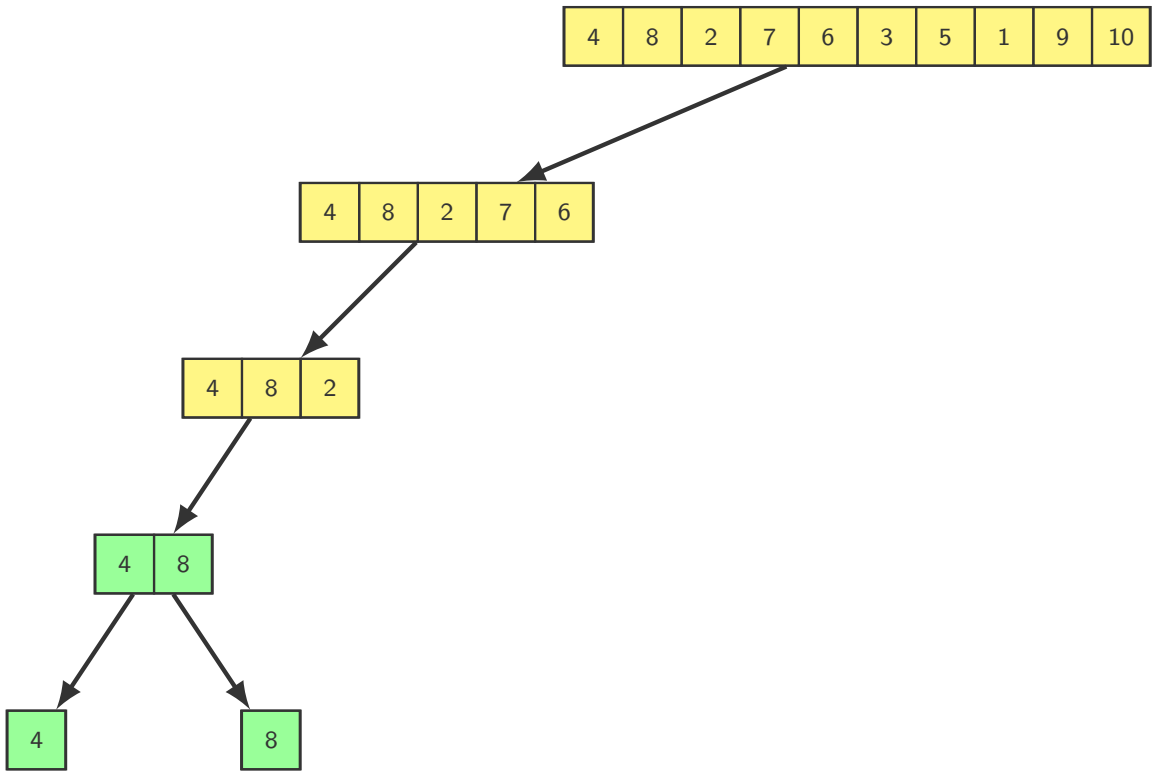


Merge Sort — Simulação

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

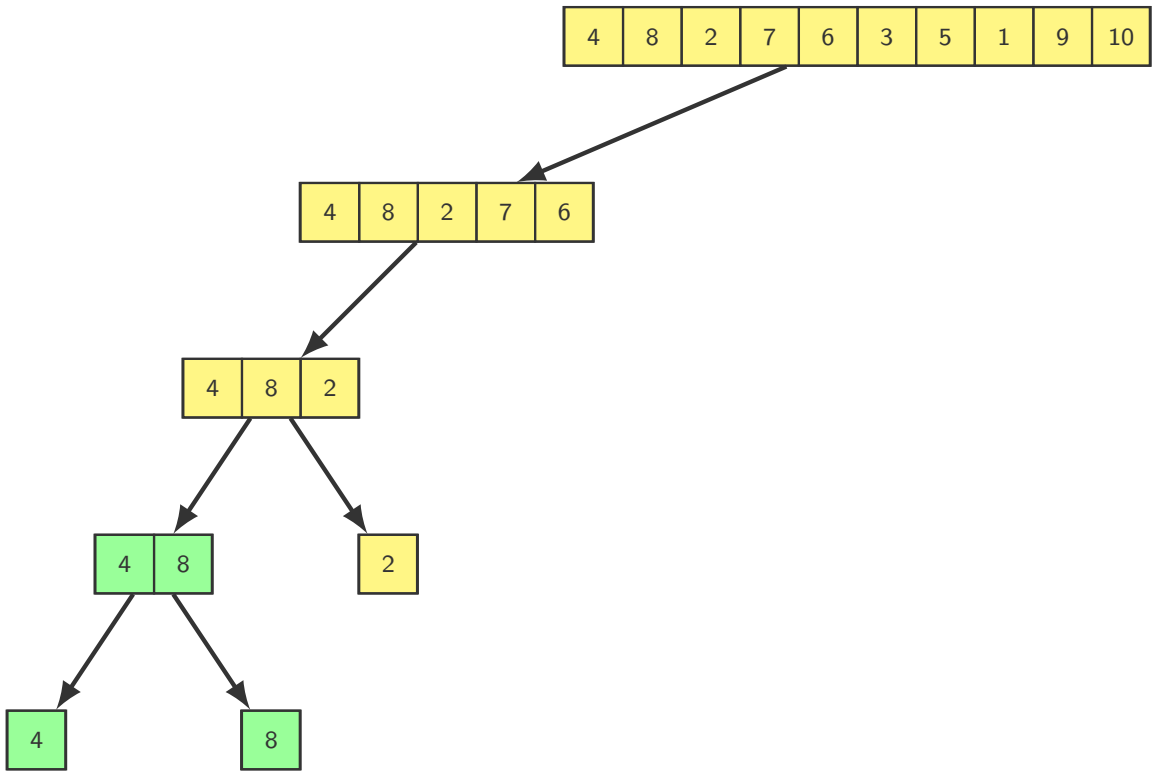


Merge Sort — Simulação



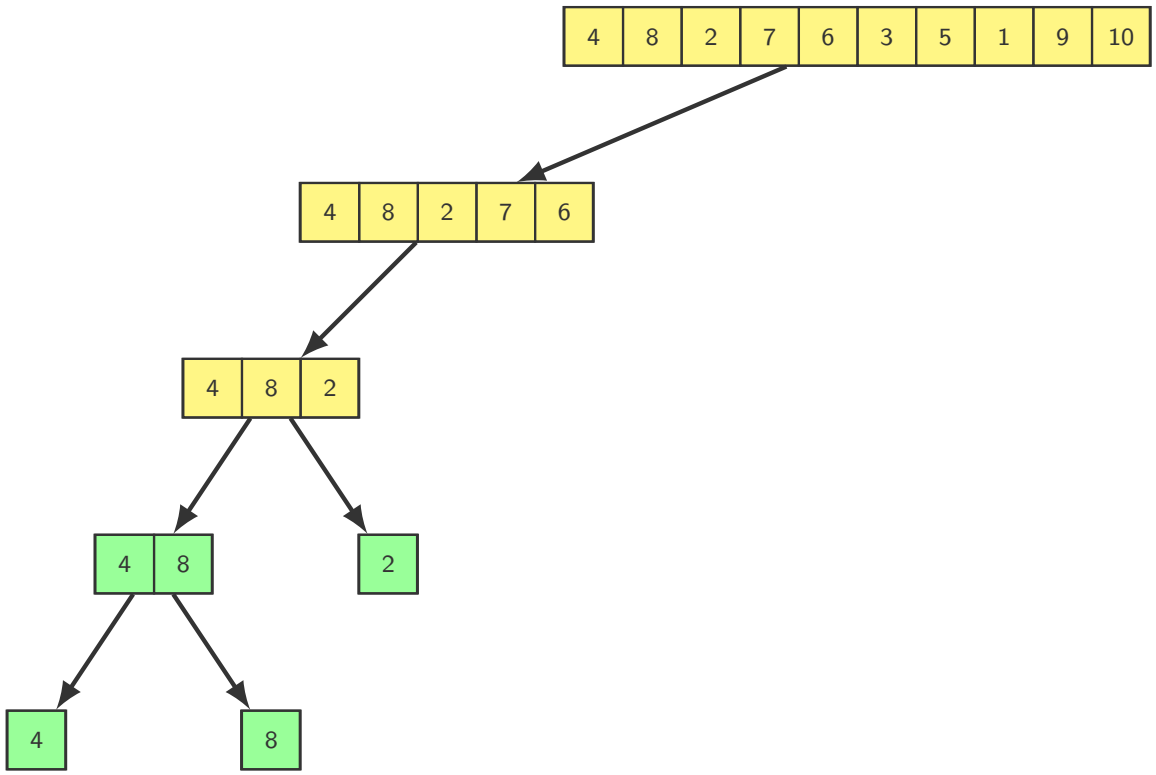
Merge Sort — Simulação

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ



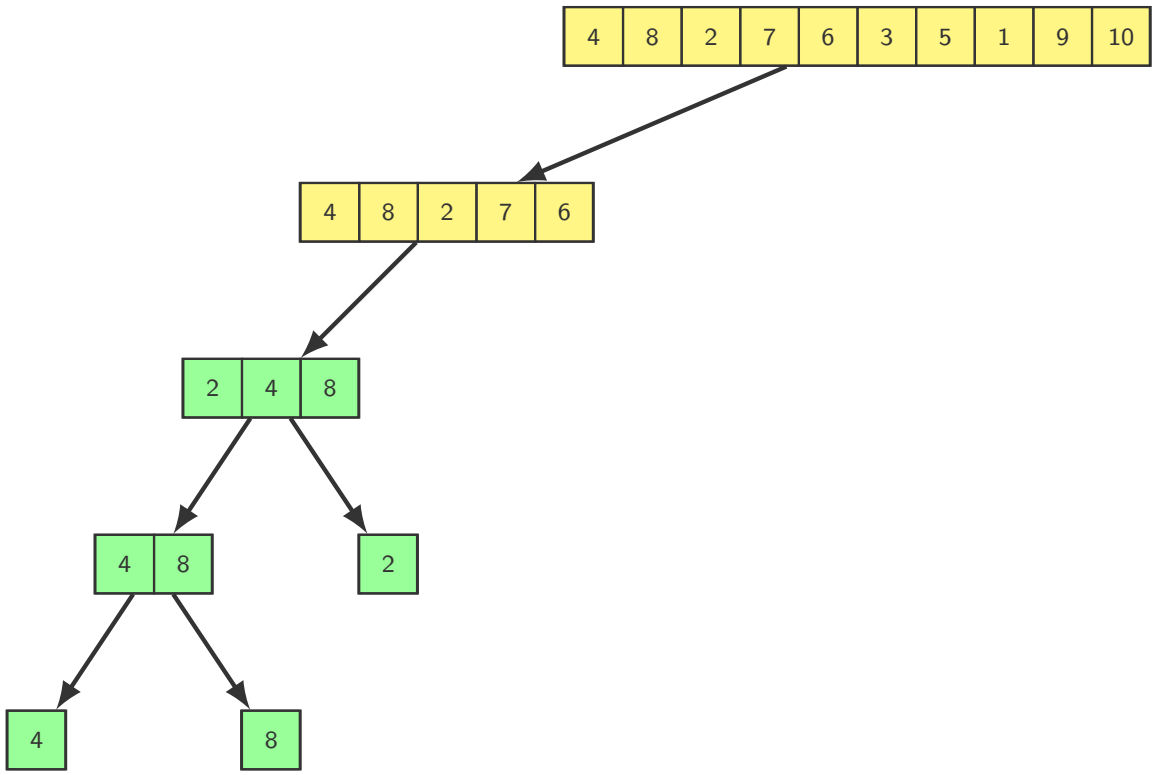
Merge Sort — Simulação

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

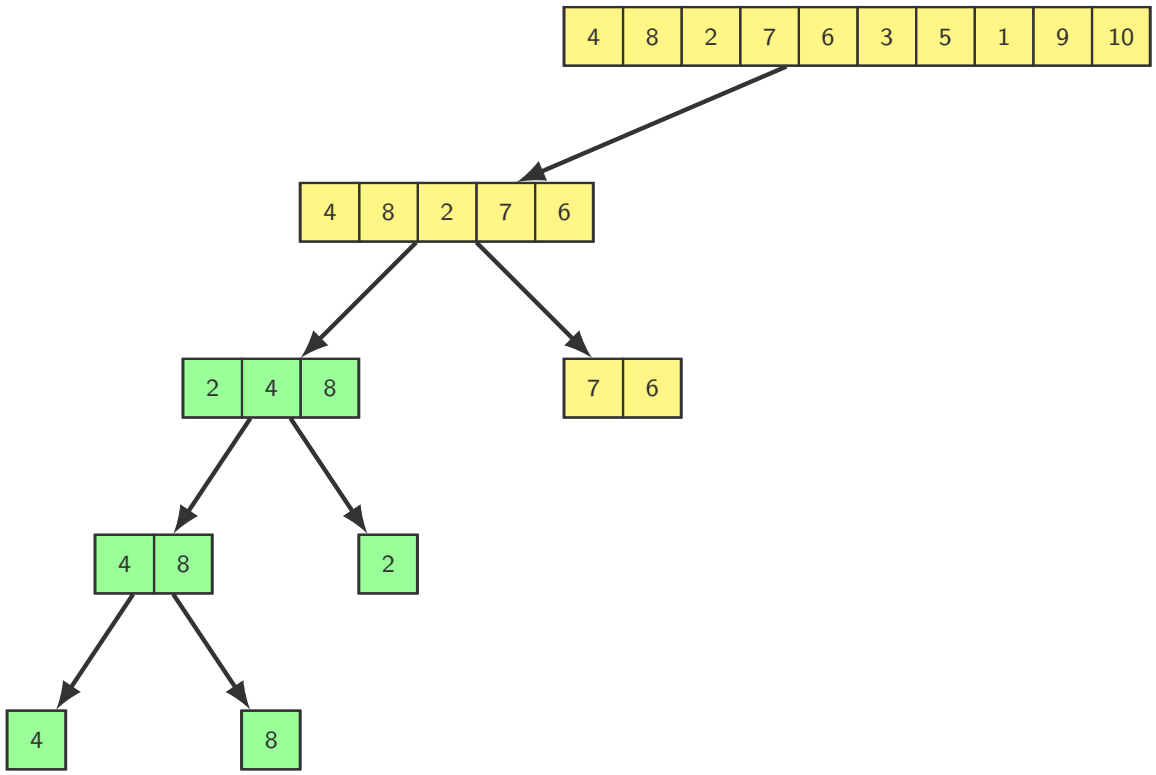


Merge Sort — Simulação

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

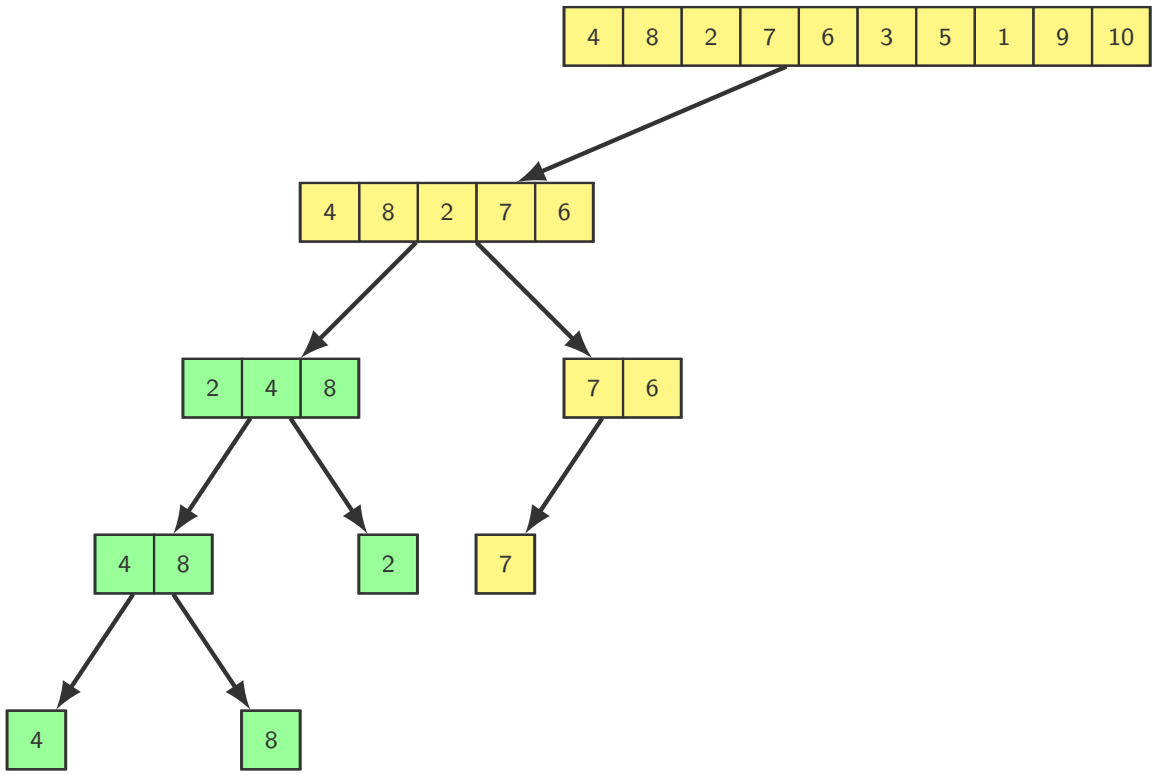


Merge Sort — Simulação



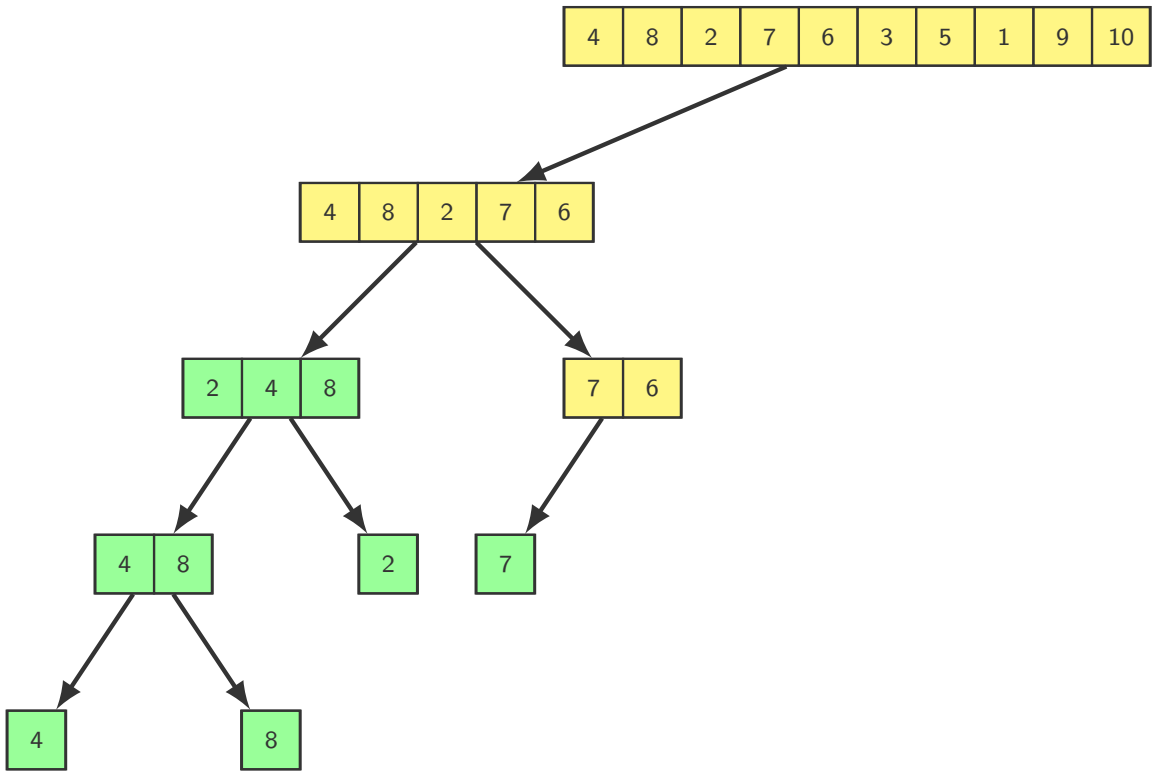
Merge Sort — Simulação

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

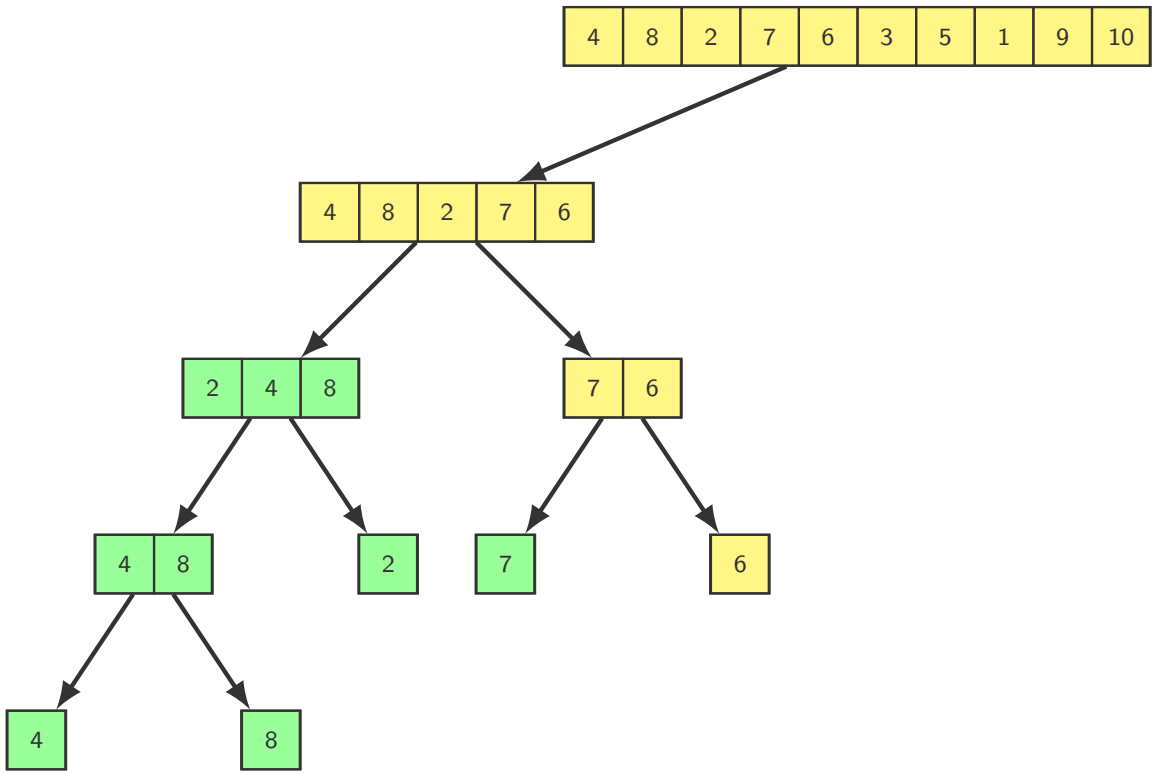


Merge Sort — Simulação

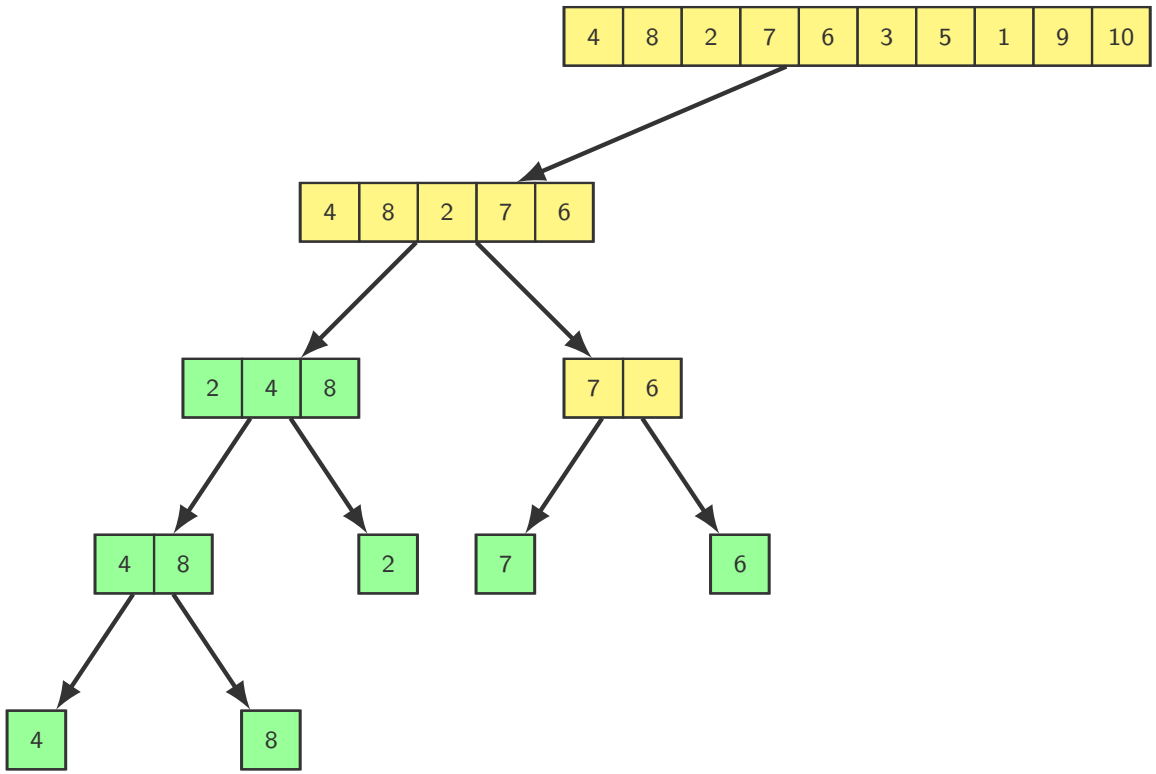
UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ



Merge Sort — Simulação

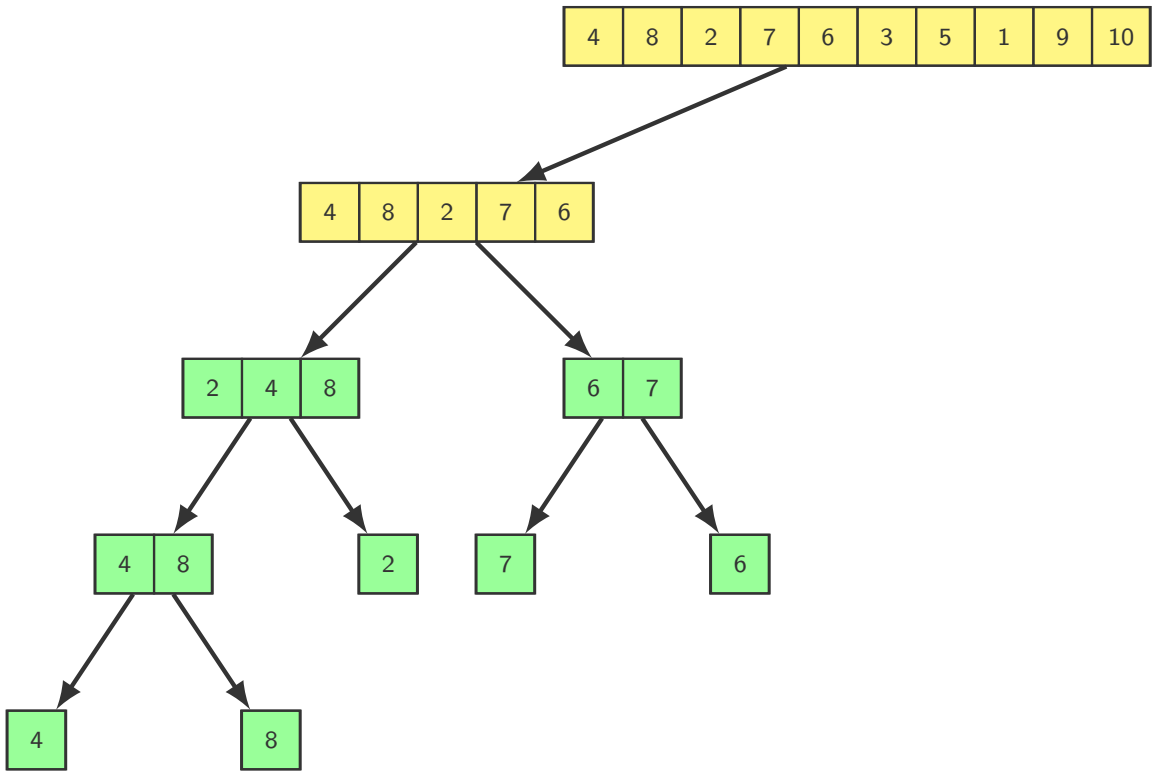


Merge Sort — Simulação

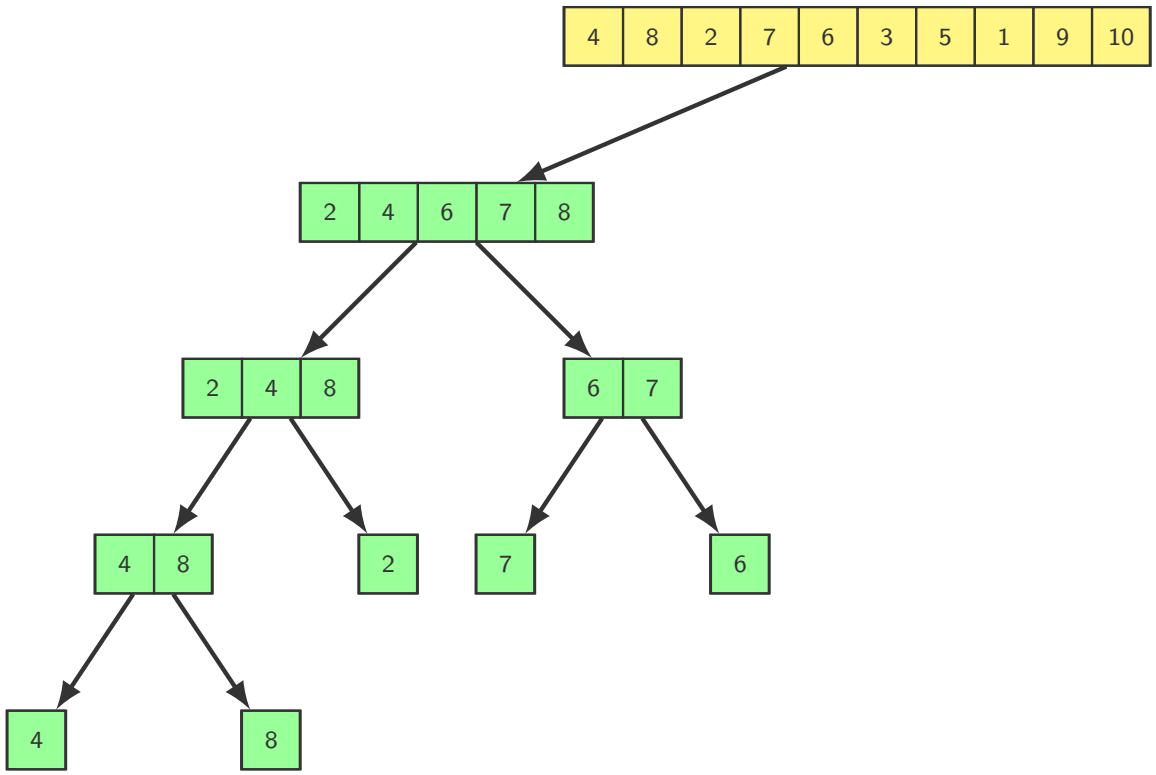


Merge Sort — Simulação

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

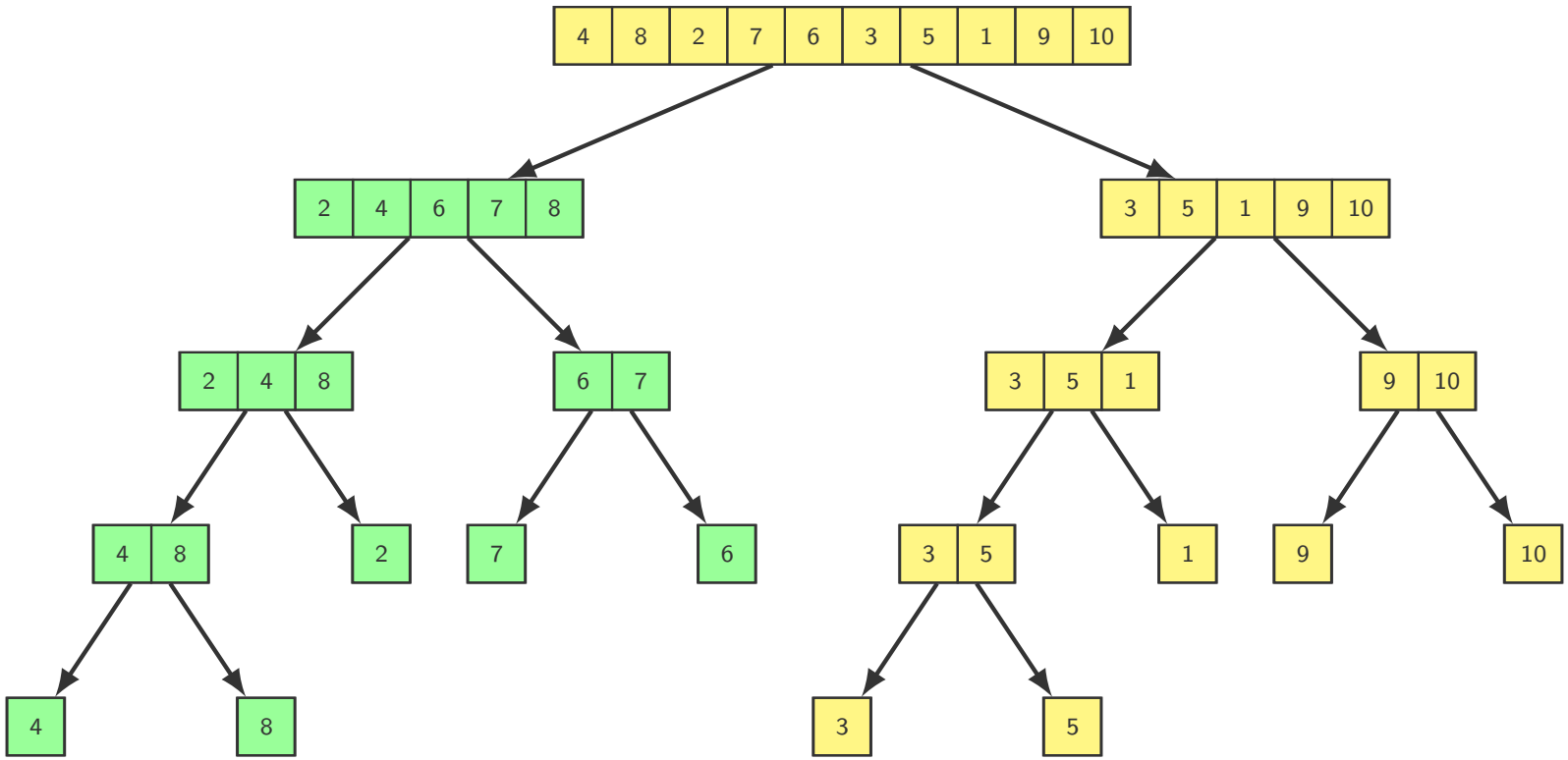


Merge Sort — Simulação



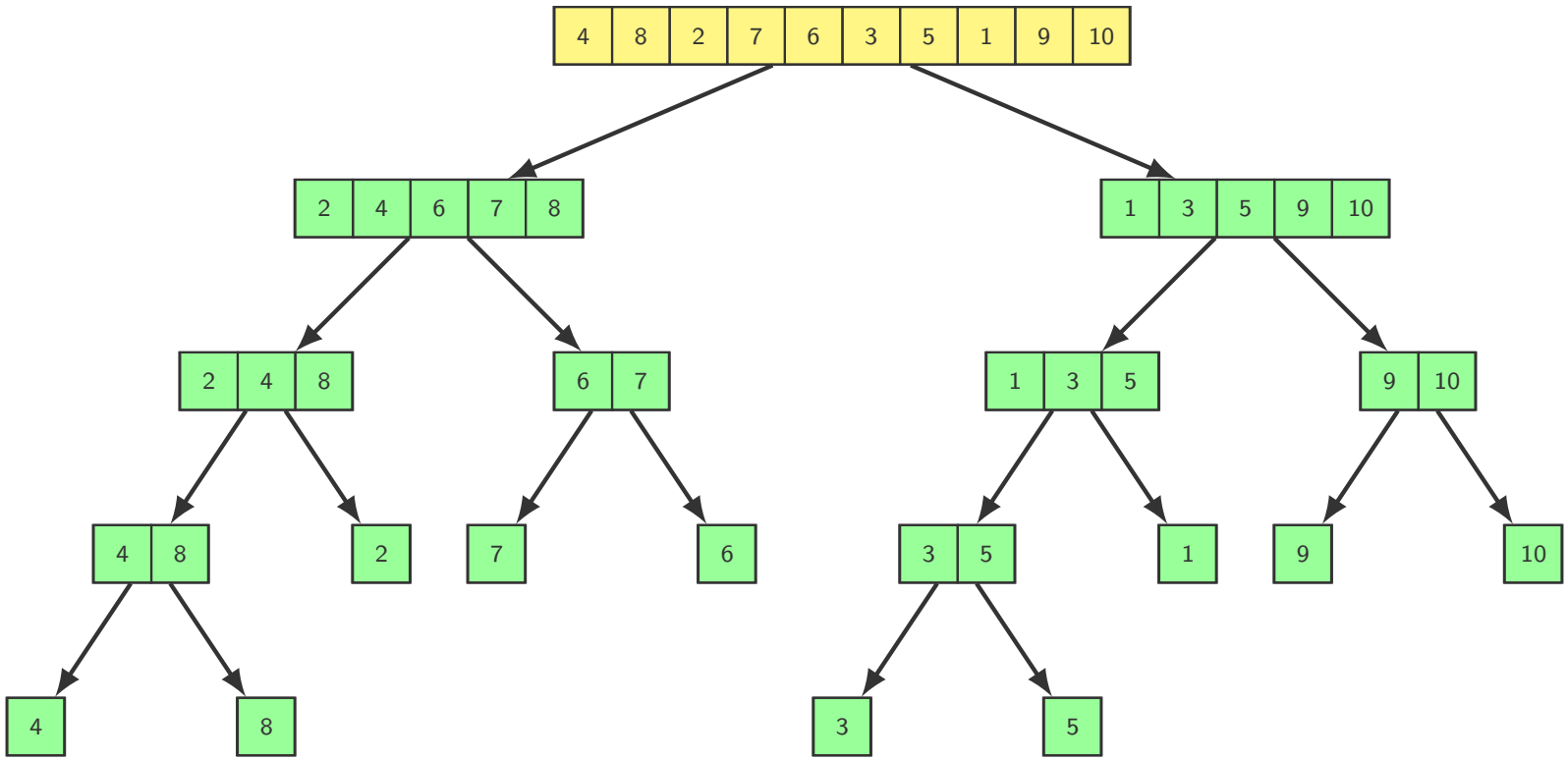
Merge Sort — Simulação

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ



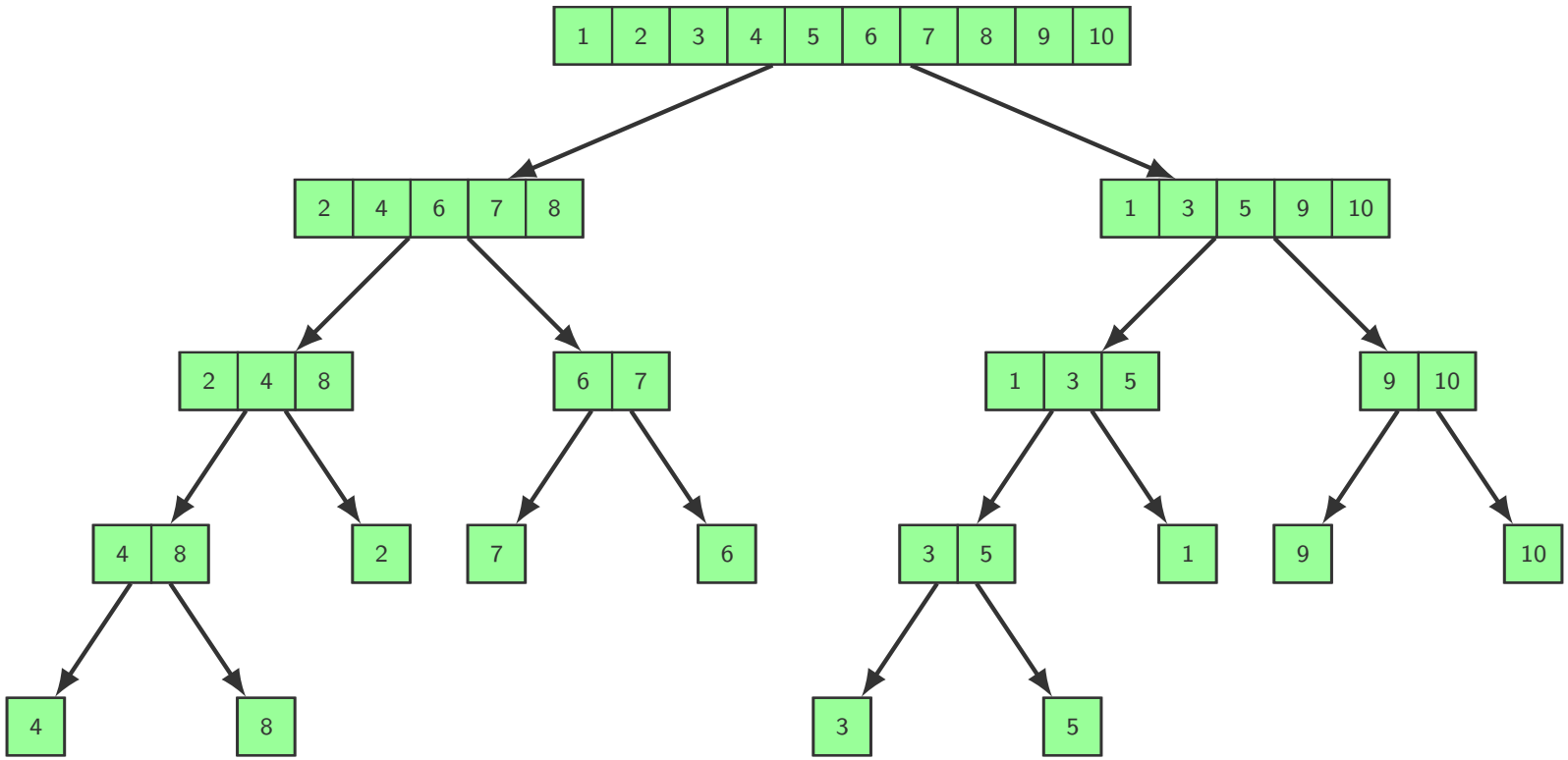
Merge Sort — Simulação

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ



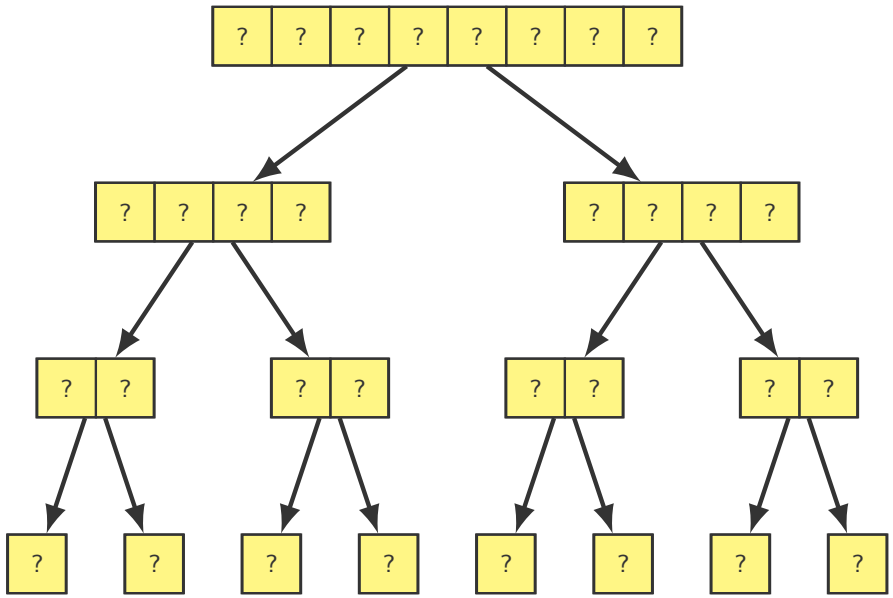
Merge Sort — Simulação

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ



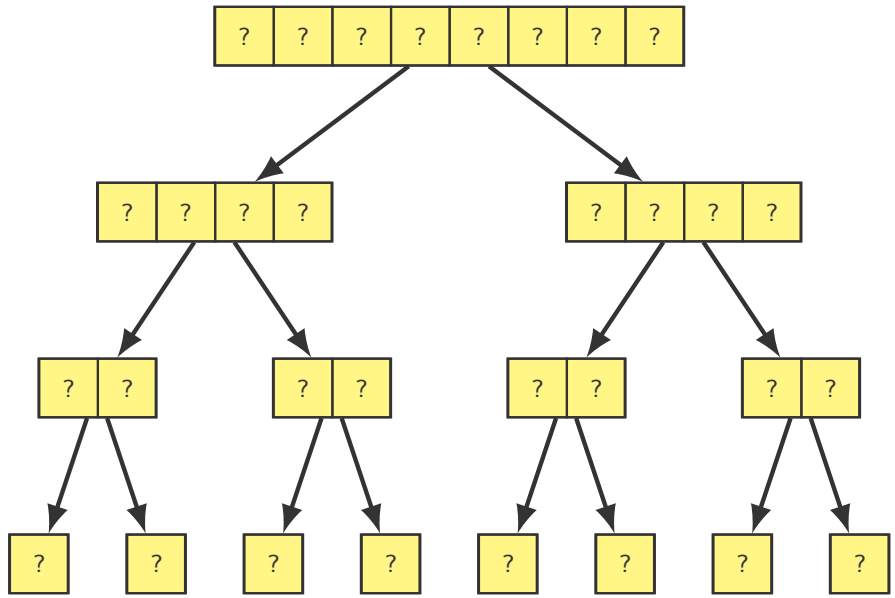
Tempo de execução para $n = 2^l$

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ



Tempo de execução para $n = 2^l$

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

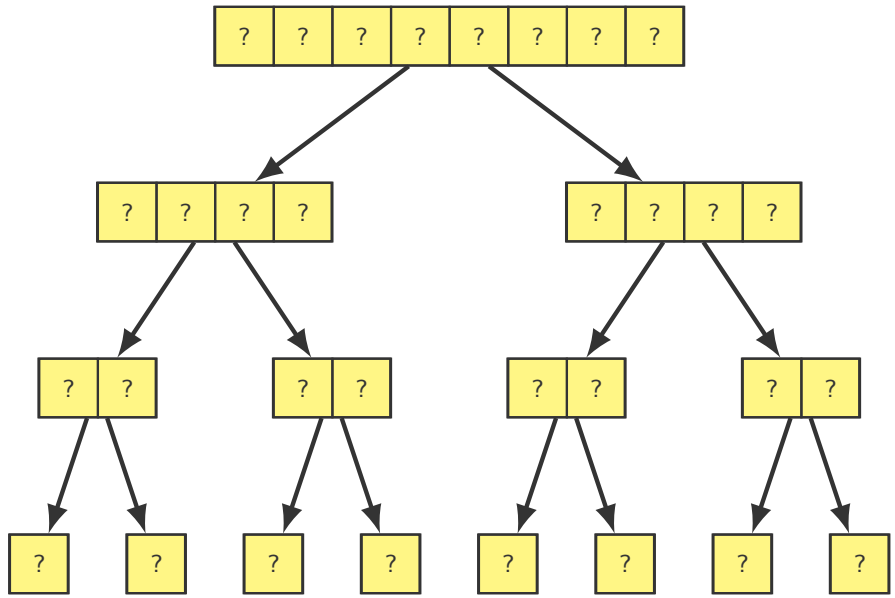


$\leq c \cdot n$

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

Tempo de execução para $n = 2^l$

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ



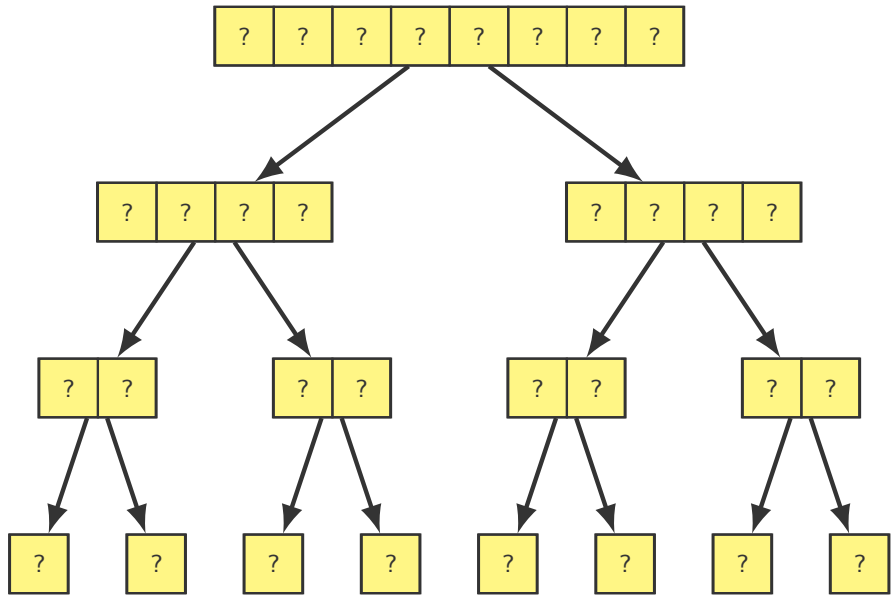
$\leq c \cdot n$

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

- 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



$\leq c \cdot n$

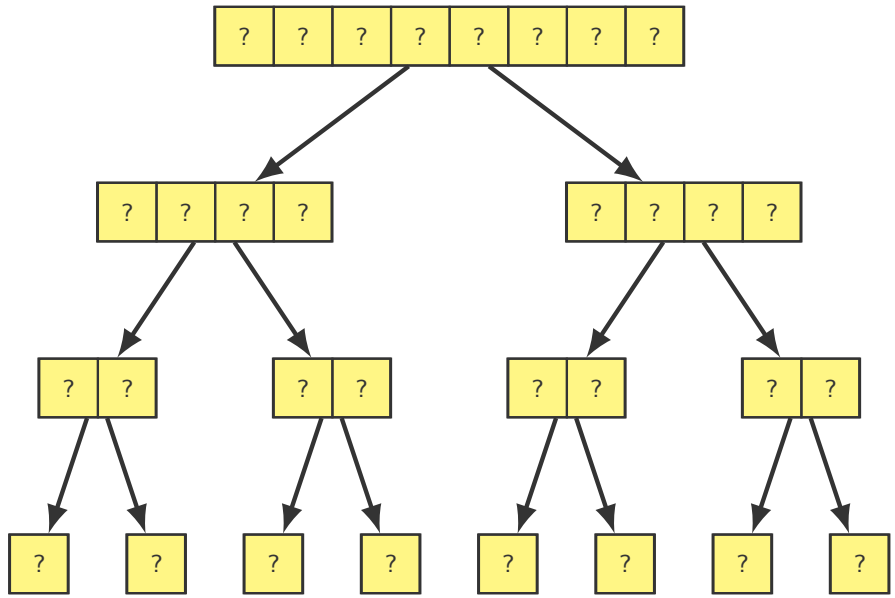
$\leq c \cdot 2(n/2)$

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

- 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 $n = 2^l$

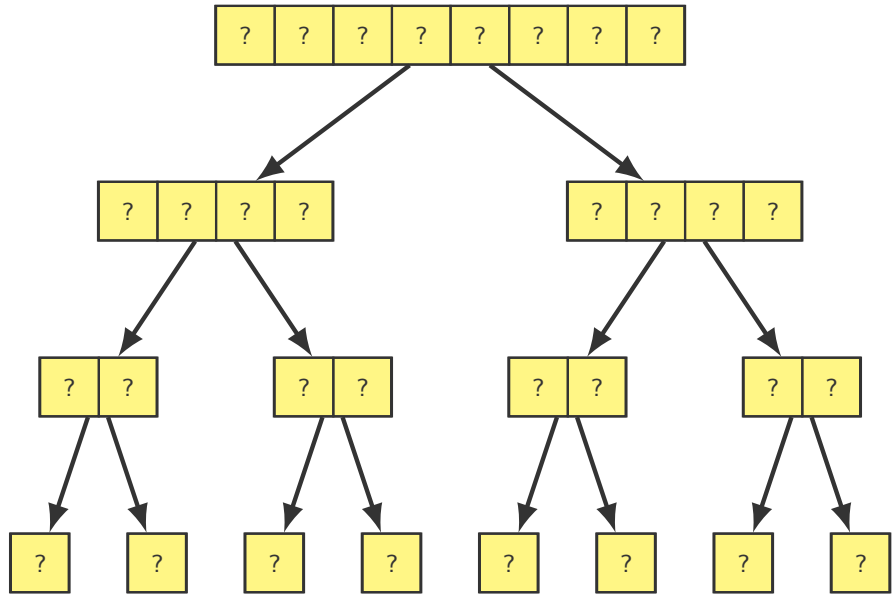
UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ



- 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 $n = 2^l$

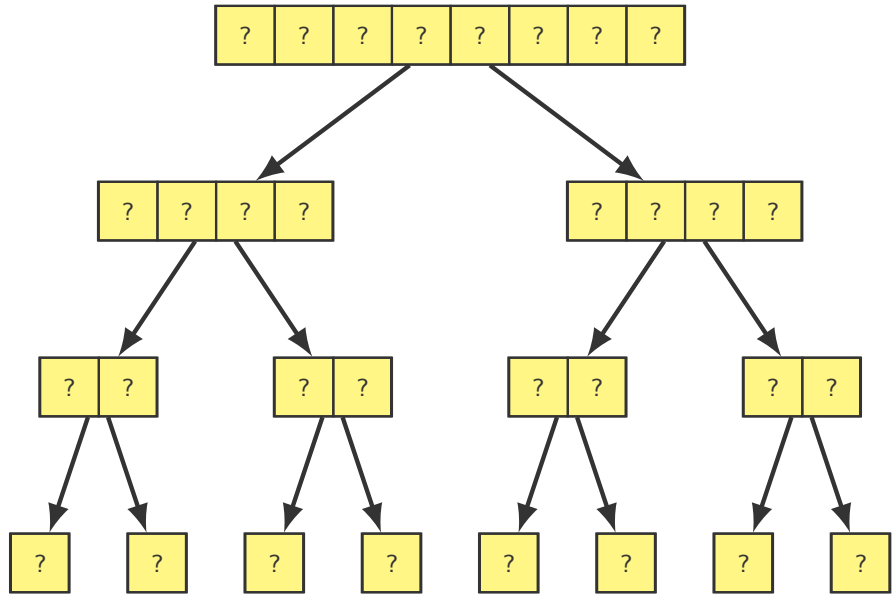
UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ



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

Tempo de execução para $n = 2^l$

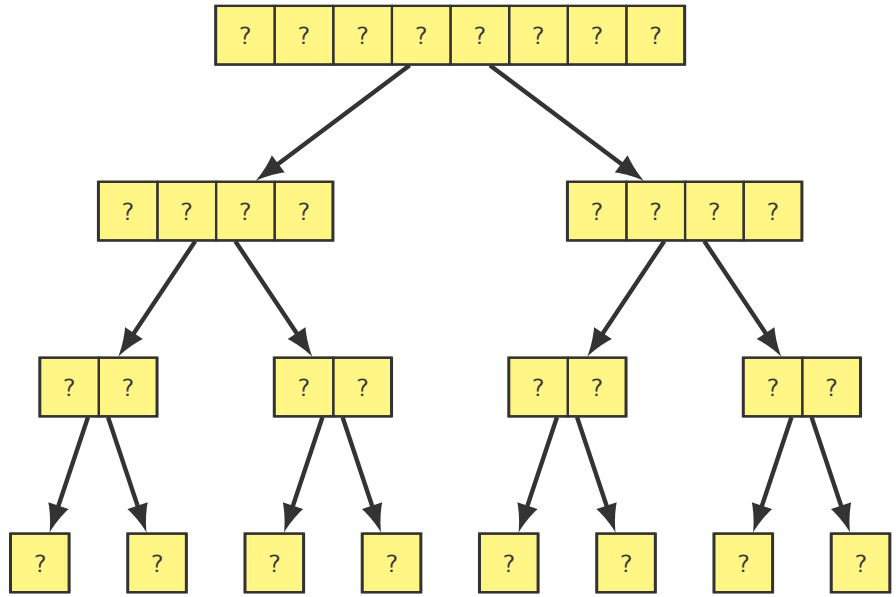
UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ



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

Tempo de execução para $n = 2^l$

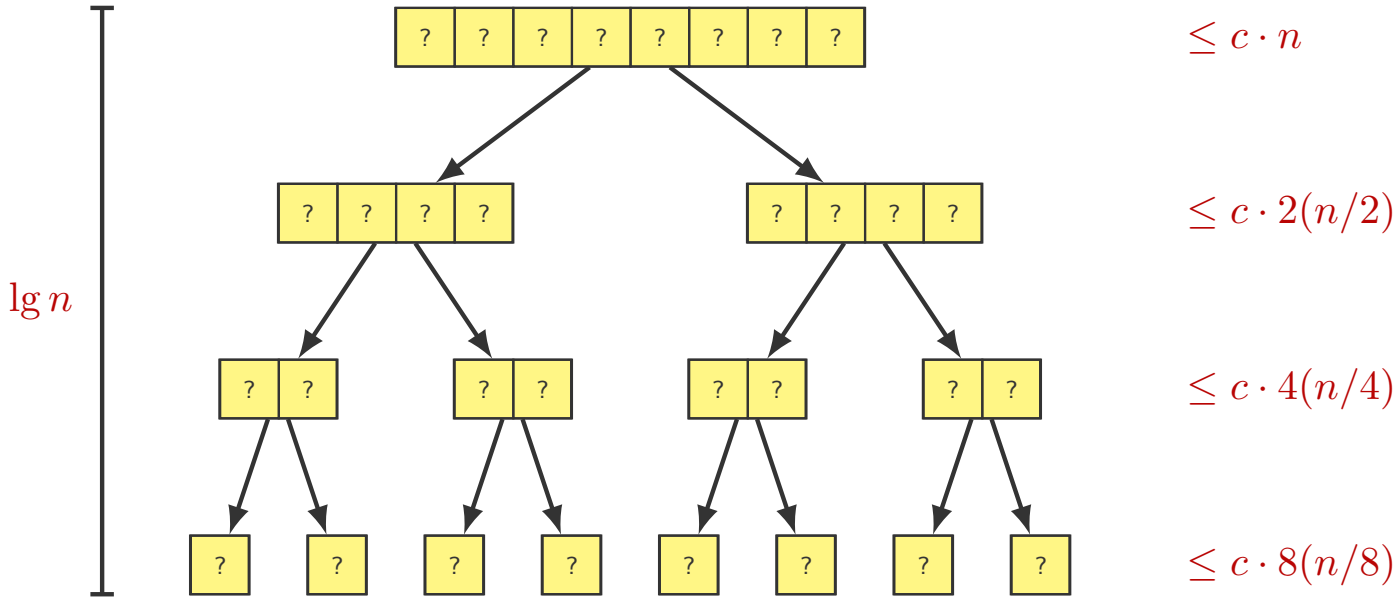
UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ



- 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 $n = 2^l$

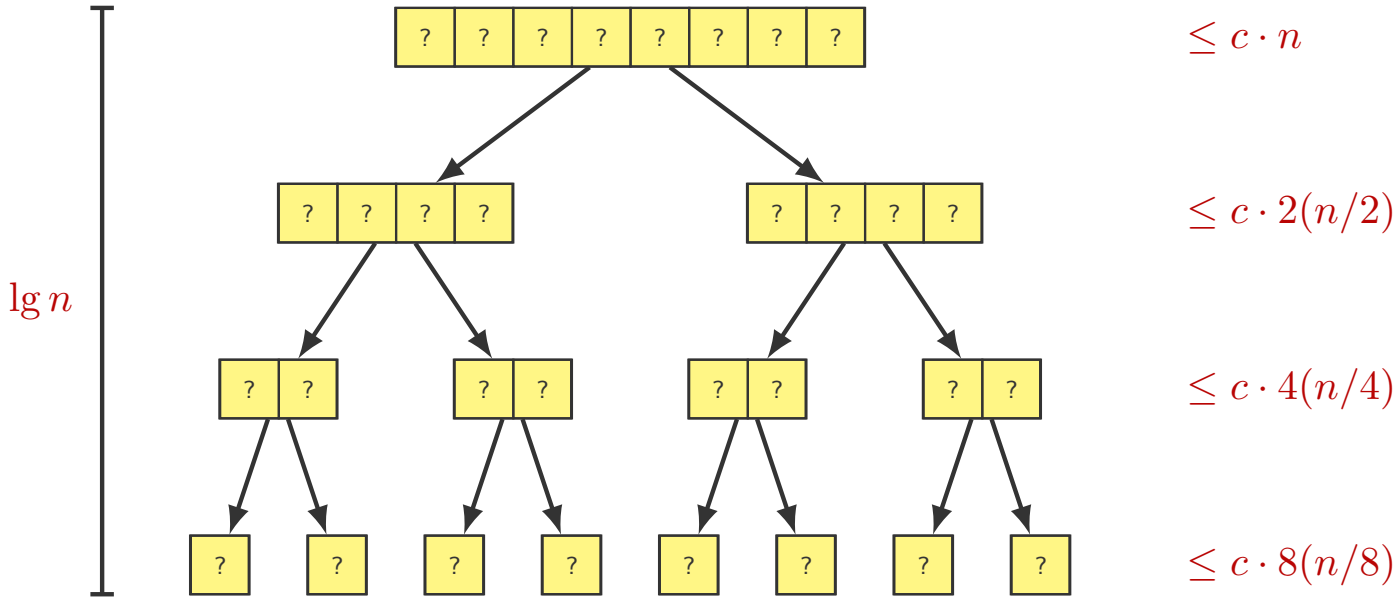
UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ



- 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 $n = 2^l$

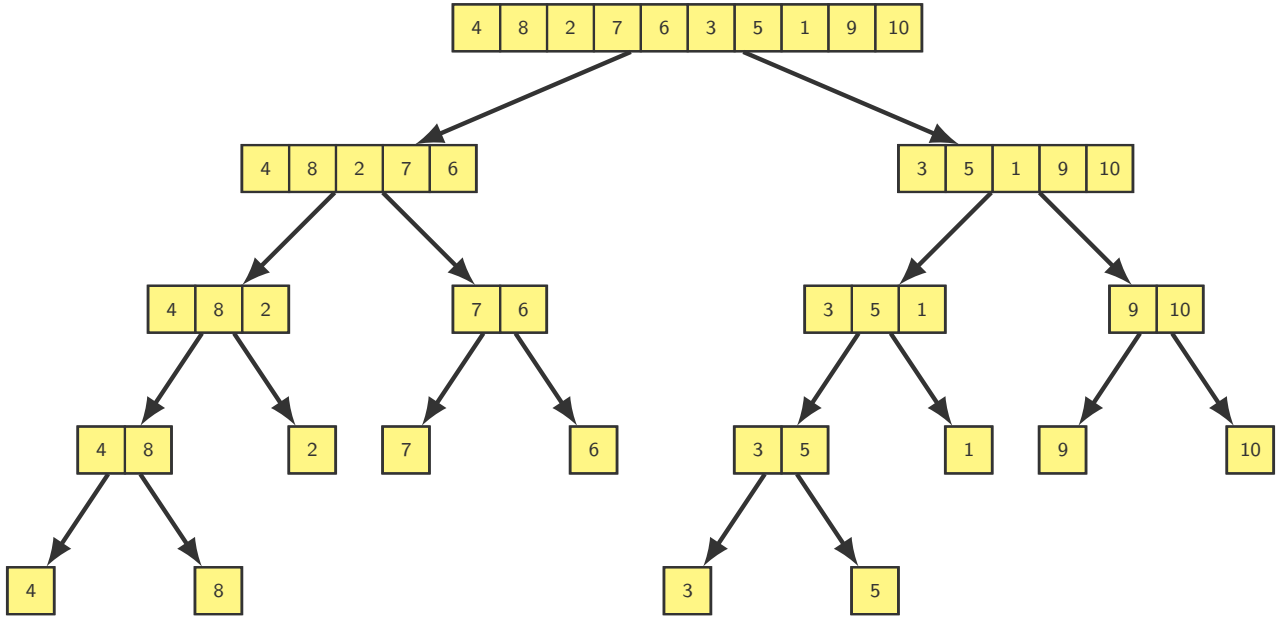
UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ



- 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: $c\, n \lg n = O(n \lg n)$

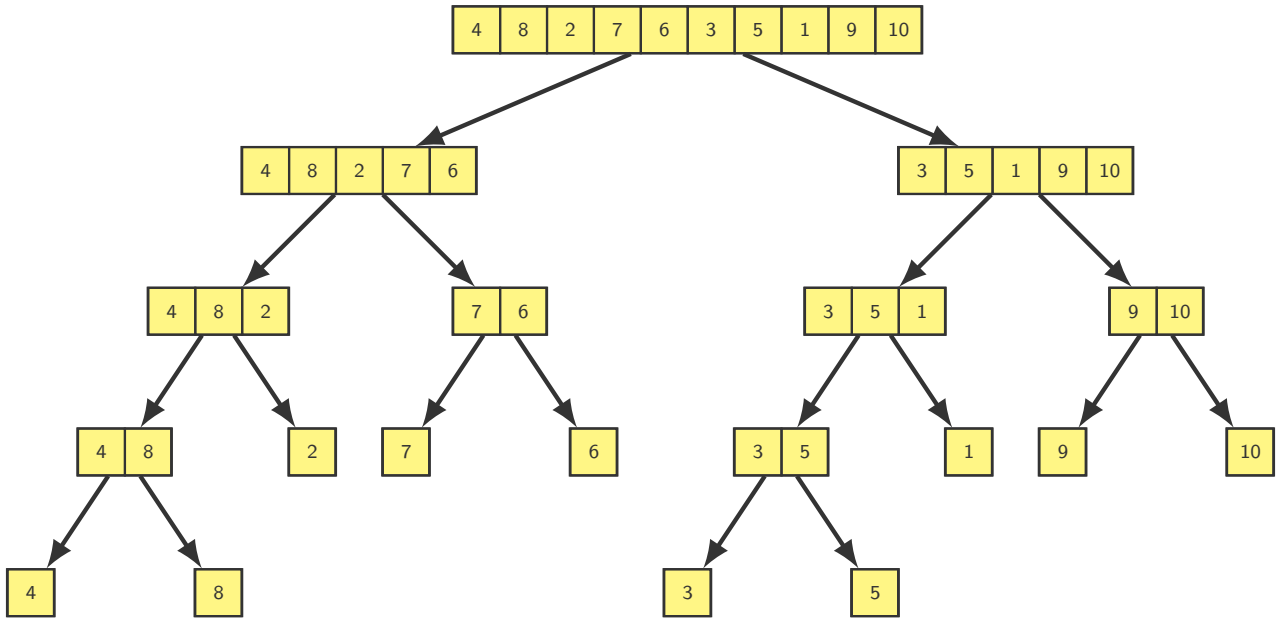
Tempo de execução para n qualquer

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ



Tempo de execução para n qualquer

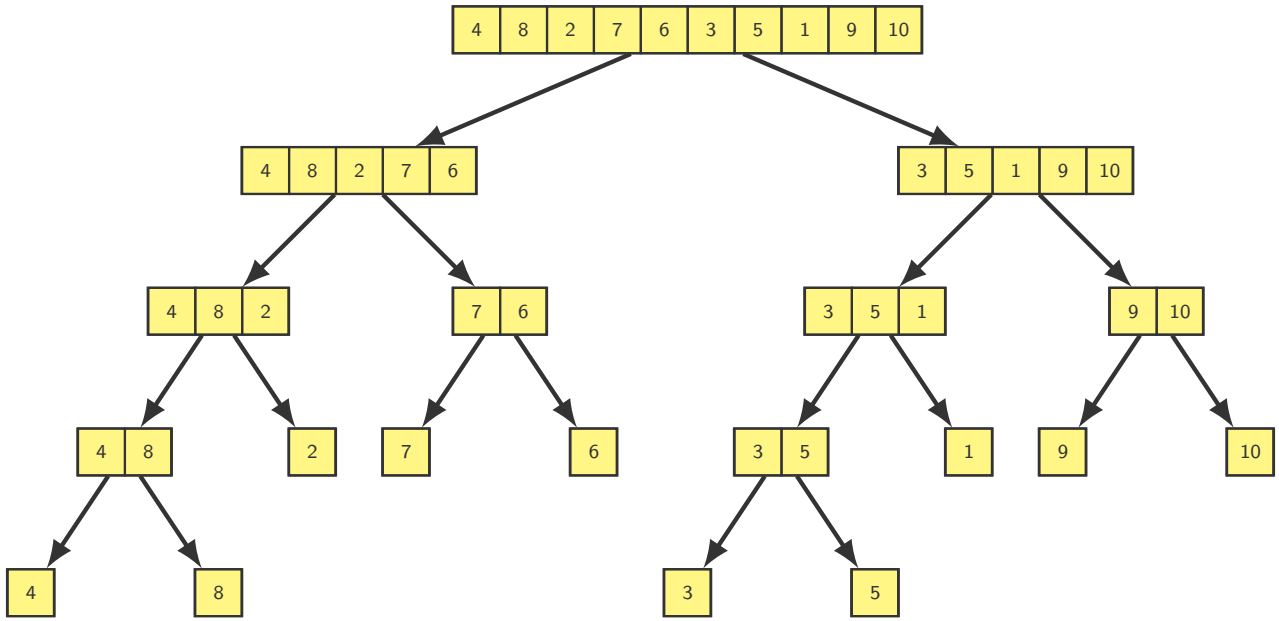
UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ



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

Tempo de execução para n qualquer

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

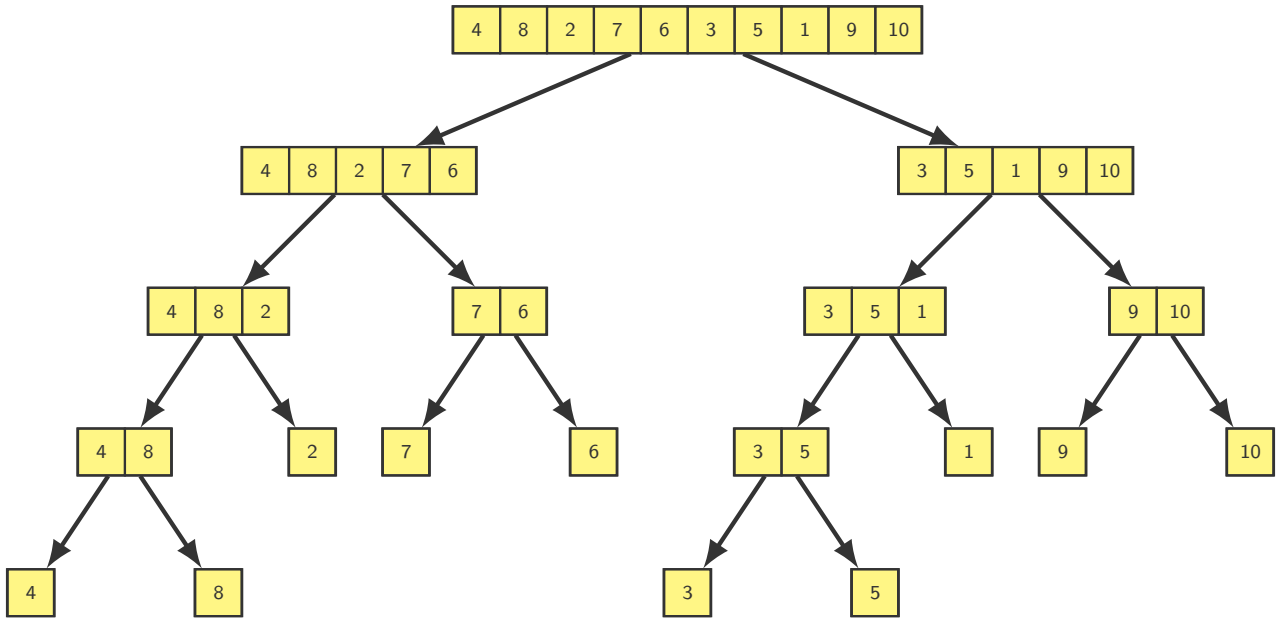


Qual o tempo de execução para n que não é potência de 2?

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

Tempo de execução para n qualquer

Universidade Federal do Ceará
Campus Quixadá

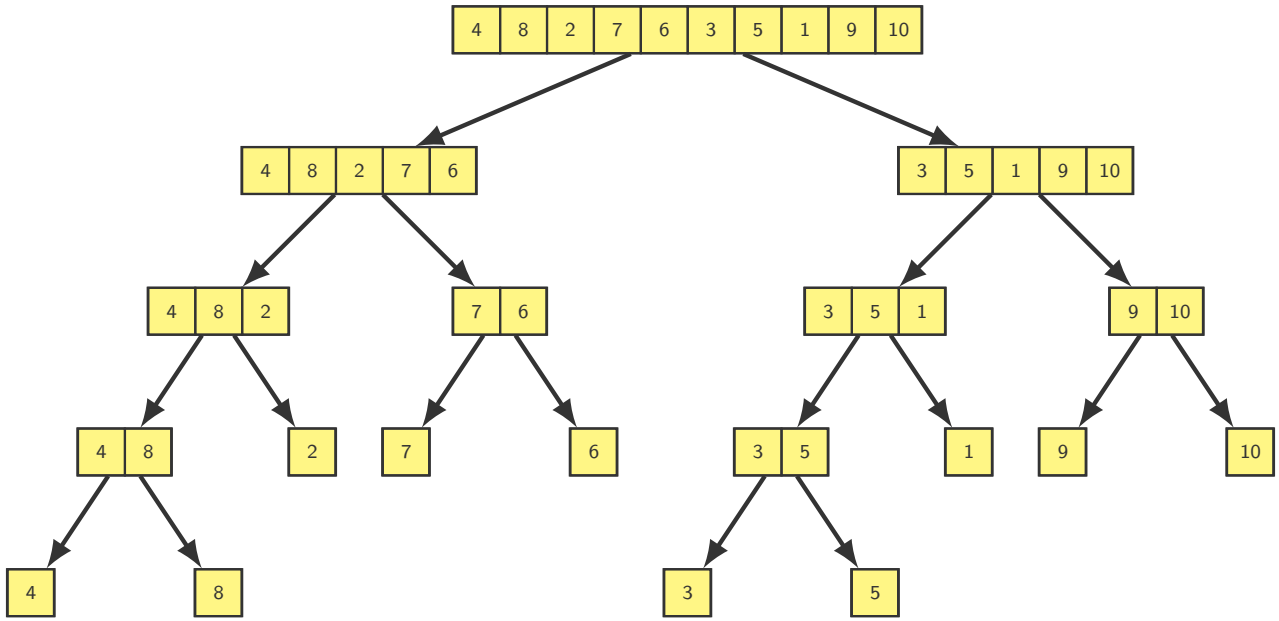


Qual o tempo de execução para n que não é potência de 2?

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

Tempo de execução para n qualquer

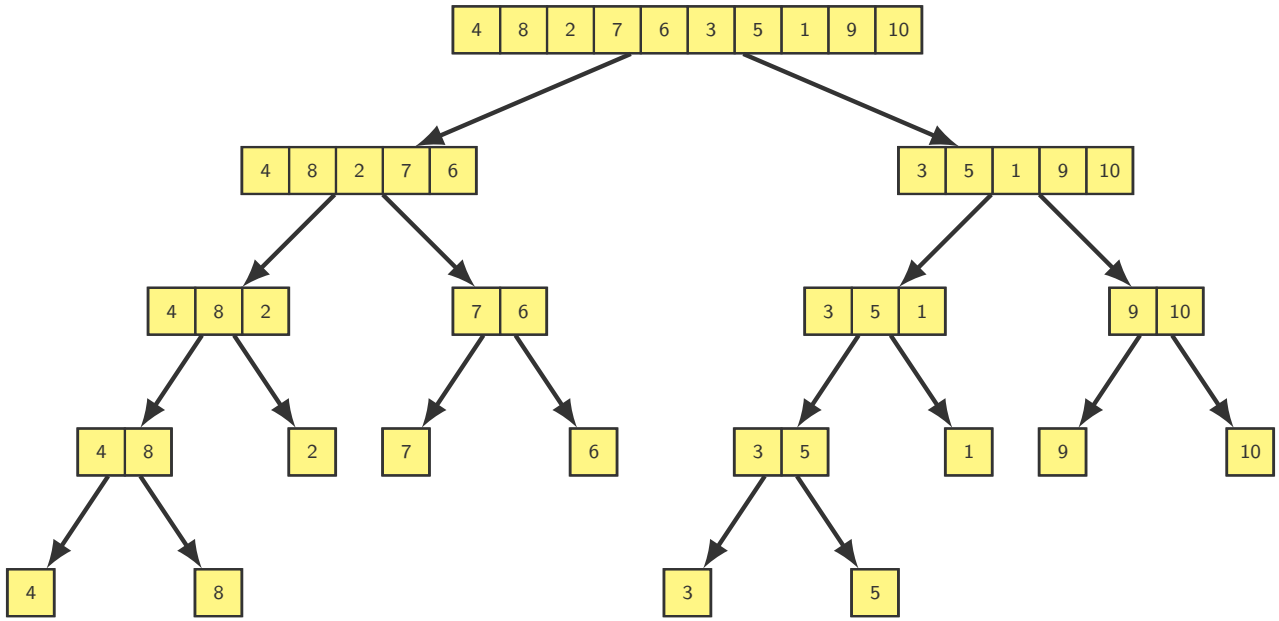
Universidade Federal do Ceará
Campus Quixadá



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, $2^k < 2n$.

Tempo de execução para n qualquer

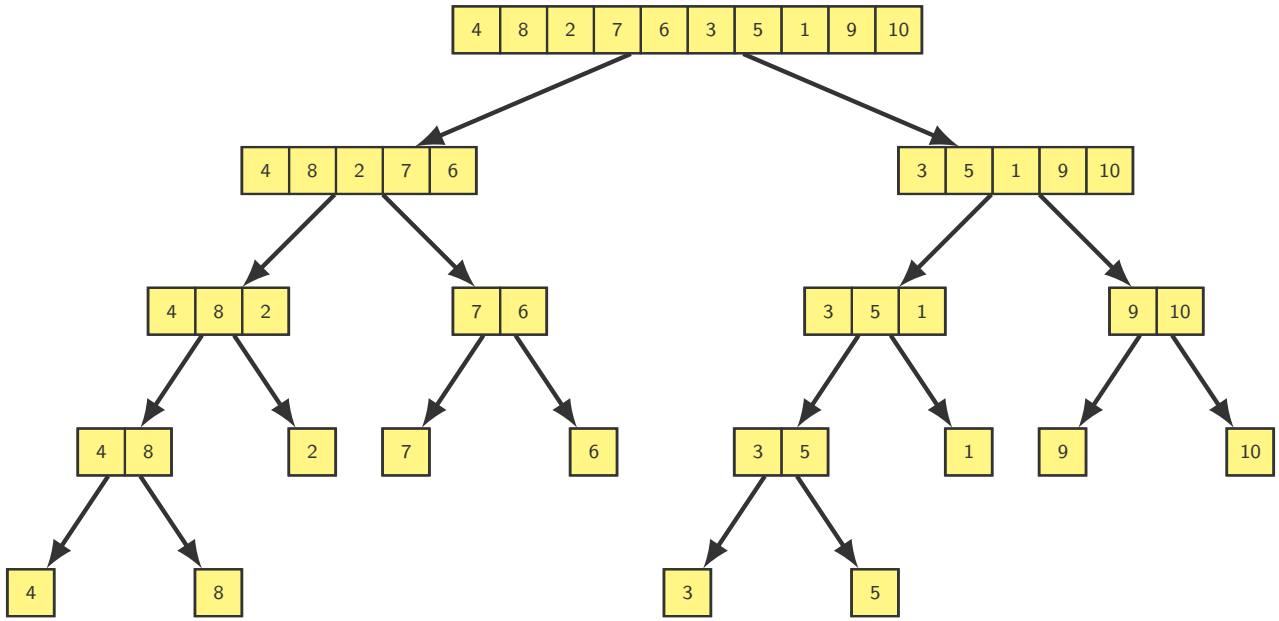


Qual o tempo de execução para n que não é potência de 2?

- 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

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ



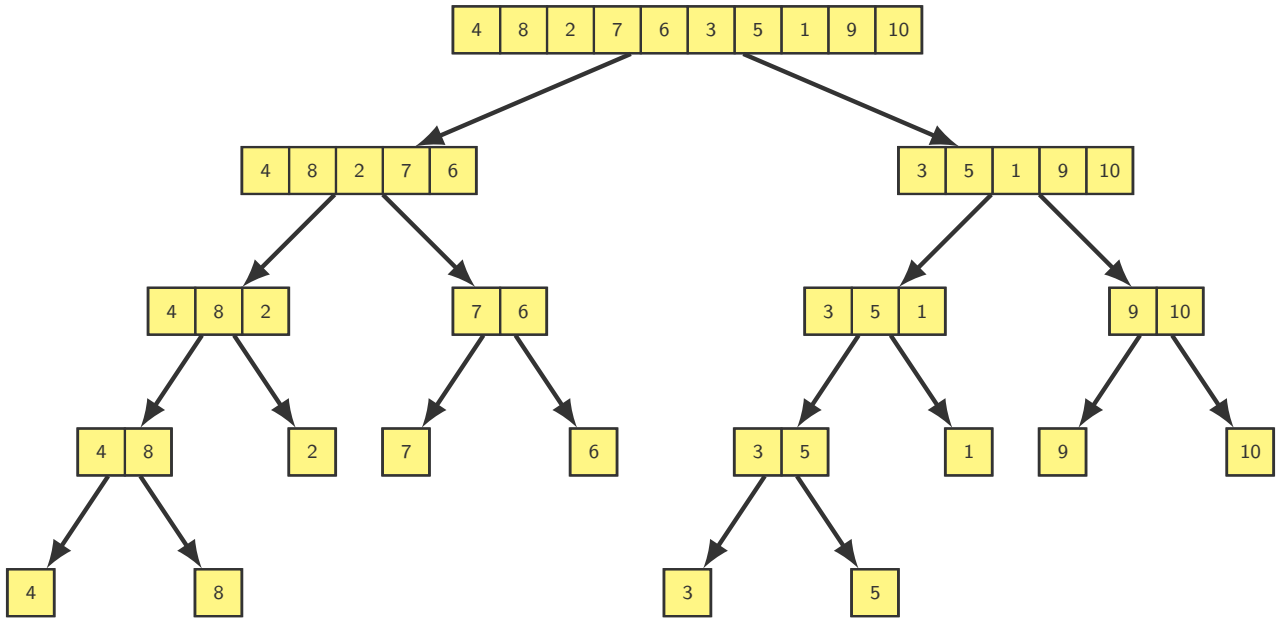
Qual o tempo de execução para n que não é potência de 2?

- 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

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

Tempo de execução para n qualquer

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

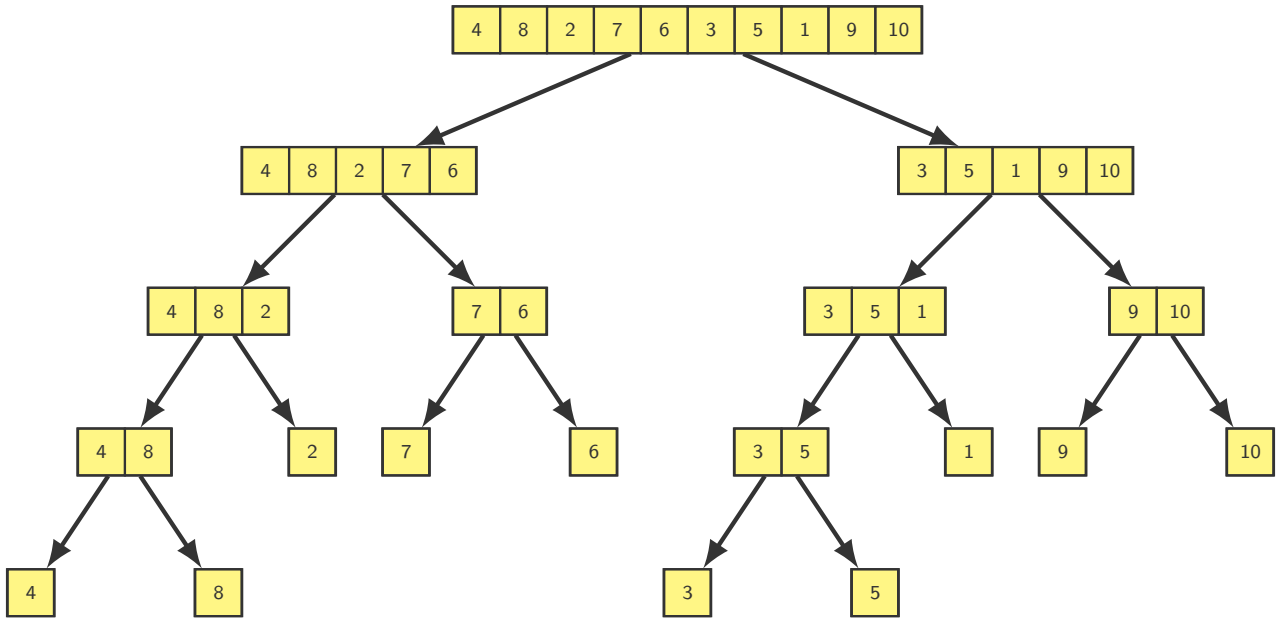


Qual o tempo de execução para n que não é potência de 2?

- 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

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

Tempo de execução para n qualquer

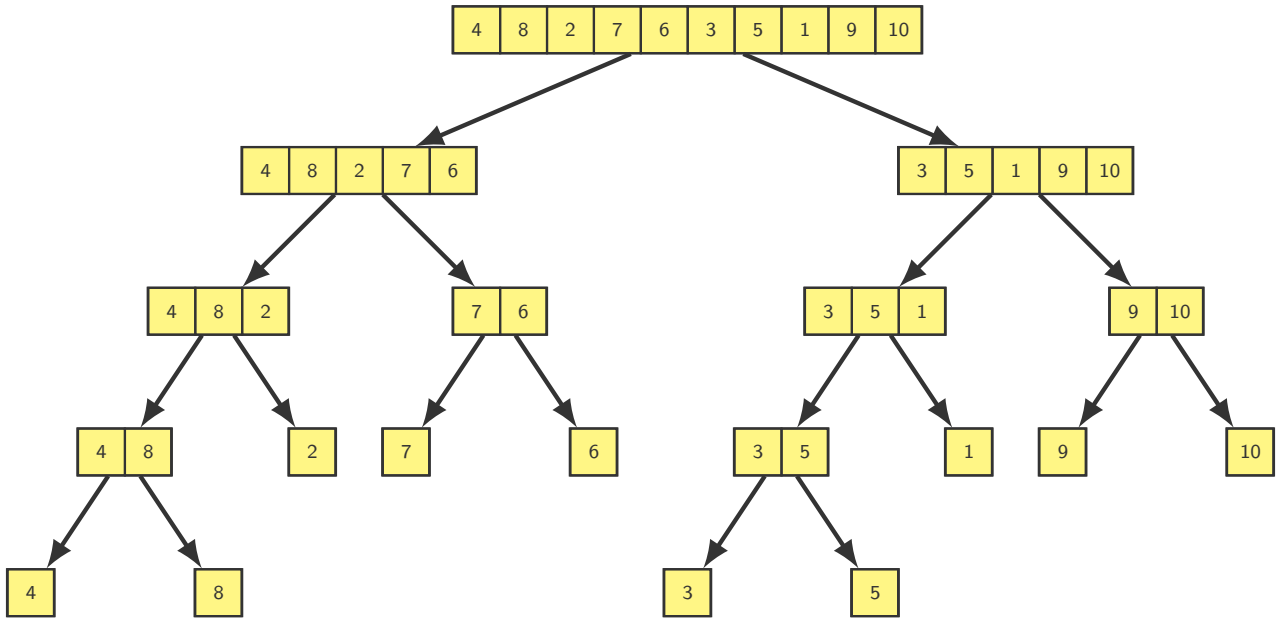


Qual o tempo de execução para n que não é potência de 2?

- 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



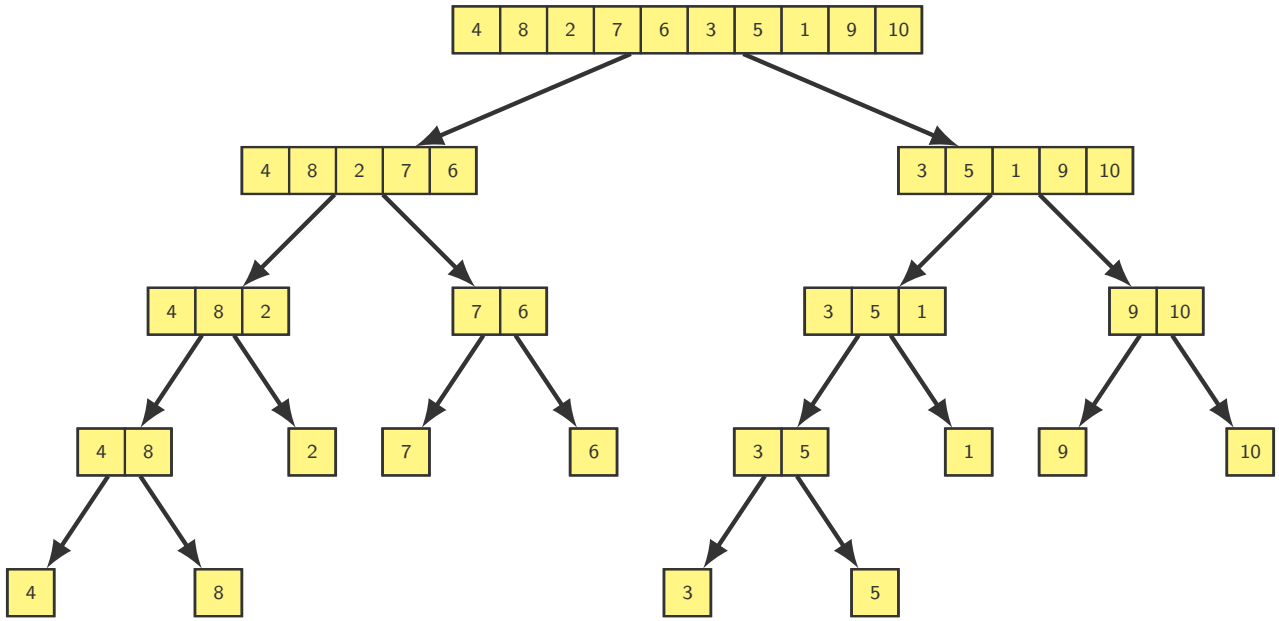
Qual o tempo de execução para n que não é potência de 2?

- 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

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ



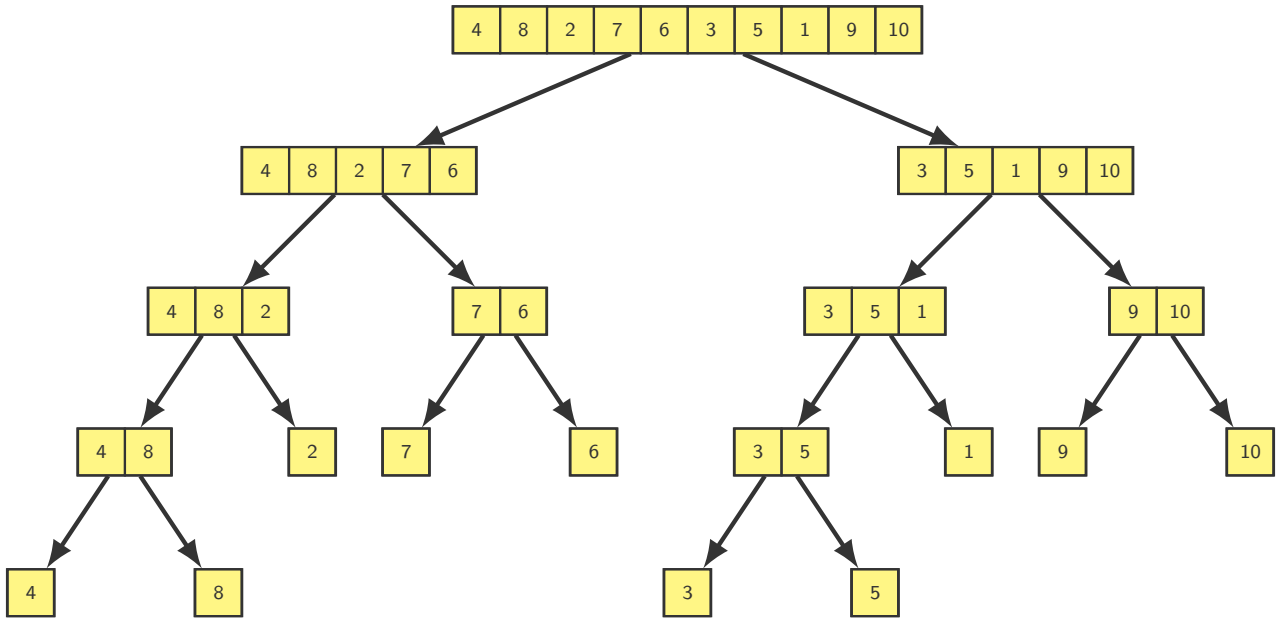
Qual o tempo de execução para n que não é potência de 2?

- 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

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

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

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

Localidade e Estabilidade

- O mergesort é um algoritmo in loco?
- Ele é estável?

UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

FIM