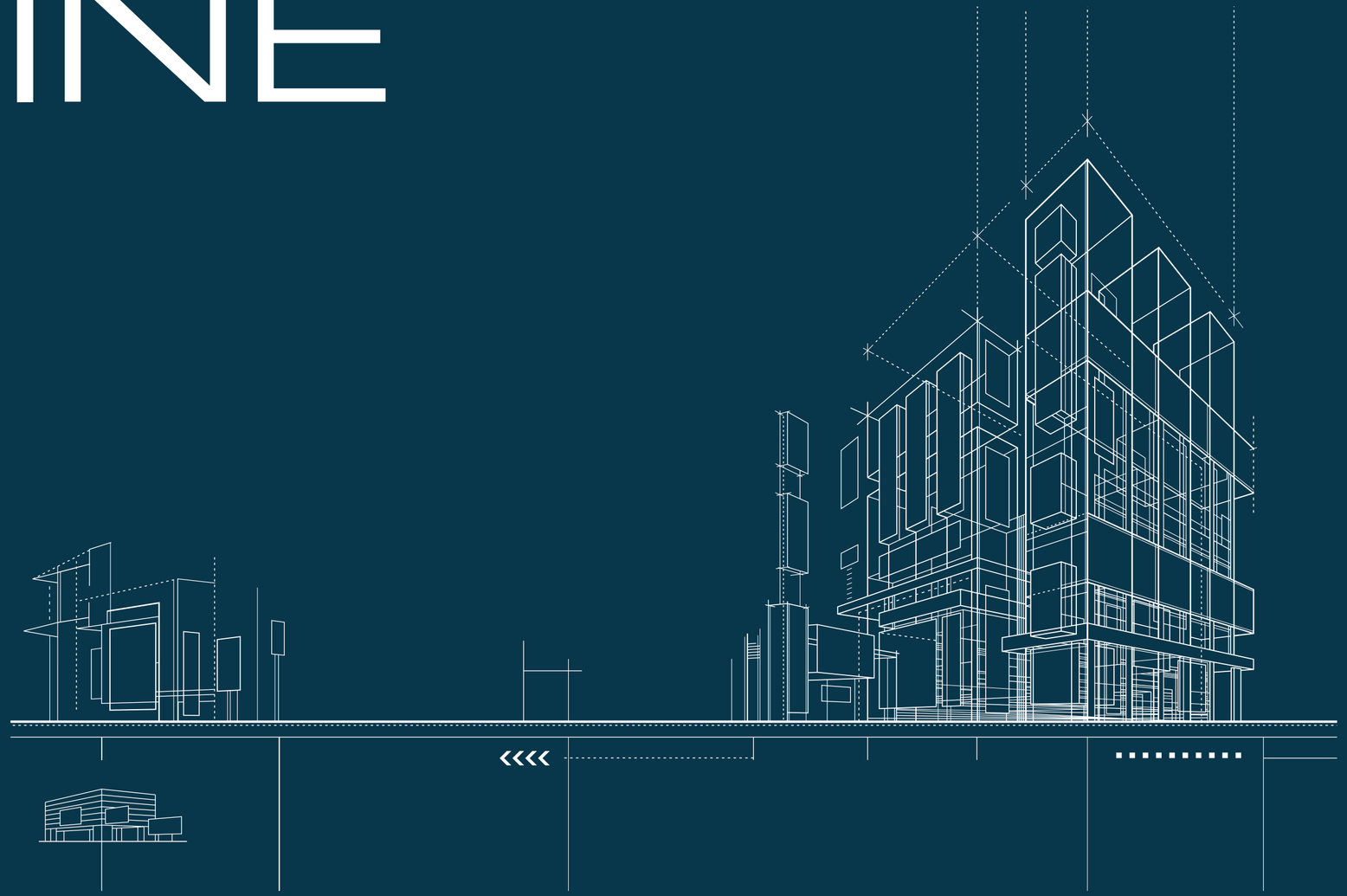
ALGORITMO SKYLINE



C.E.S.A.R school

Analise de Algoritmos
2025.1

AGENDA

Introdução
Entendendo o problema
Skyline

Dinâmica
Hands on

Divisão e Conquista
Solucionando o problema

Recursividade
Analise do Algoritmo

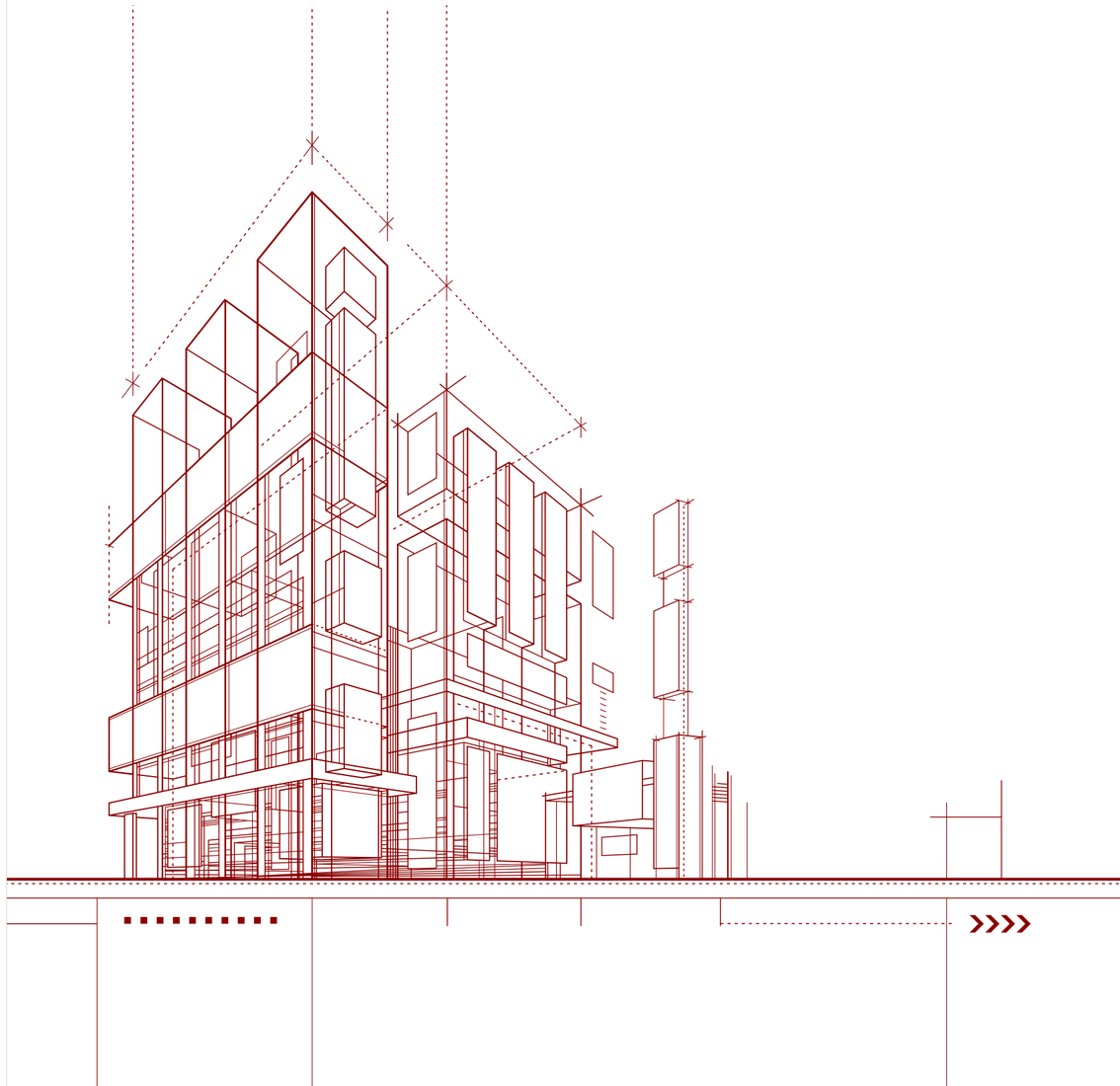
Teorema Mestre
Análise de Complexidade

Aplicações
Soluções Reais

Perguntas
Sanar eventuais
Dúvidas

Mentimeter
Fixação de Conhecimento

Quiz
Perguntas Sugeridas



INTRODUÇÃO

O que é o Problema do Skyline (Skyline Problem)?

Imagine que você está andando por uma cidade e olha para o horizonte, vendo apenas o contorno dos prédios. Aquela linha que desenha a silhueta dos edifícios contra o céu — isso é o **skyline** da cidade.
Mas... e se alguém te desse só as informações dos prédios, no formato onde cada um é representado por uma tripla (esquerda, altura, direita)?

E agora, como você vai descobrir os pontos onde a silhueta da cidade muda?

DINÂMICA

Resolva o problema do Skyline!

Sem código, como você descobriria o skyline dessa cidade, que possui os seguintes prédios?

PRÉDIO	Início (X1)	Altura(H)	Fim (X2)
A	2	10	9
B	3	15	7
C	5	12	12
D	13	10	16
E	15	8	20
F	8	13	14

OBS.: Cada prédio é representado por uma trinca (início, altura, fim).

1. Desenhem a linha do eixo X

Esse será seu ponto de observação.

2. Representação dos Prédios

Marquem as áreas dos prédios com blocos verticais, conforme os intervalos.

3. Objetivo

Descubram quais pontos representam mudanças de altura no contorno.

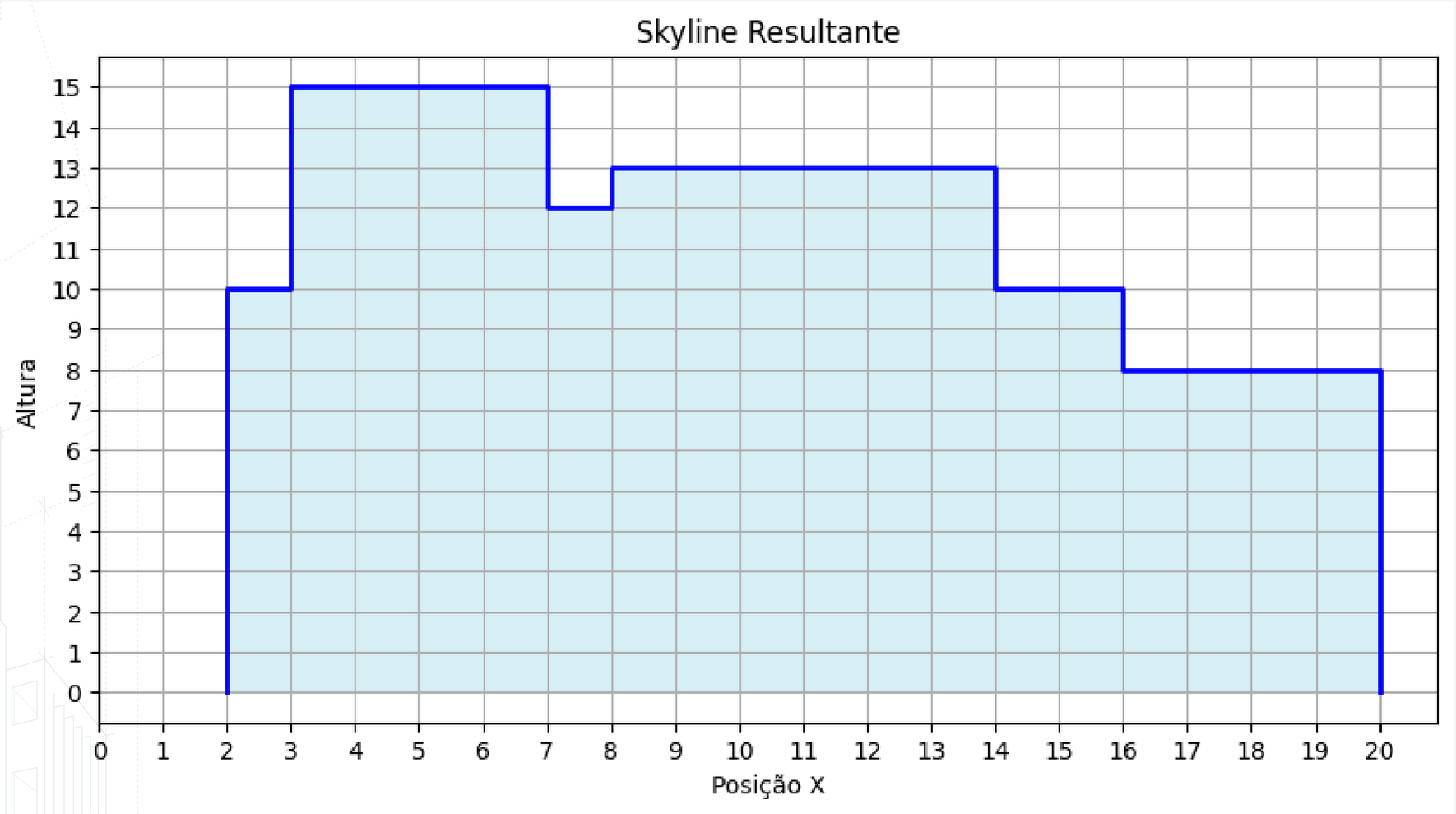
4. Montando o Skyline

Usem o lápis para marcar os “pontos críticos”, que representam quando a altura do skyline sobe ou desce.

5. Dicas

- Um prédio mais alto aparece por trás de outro;
- Um prédio termina, revelando o anterior;
- Todos os prédios terminam e a altura volta para zero.

RESULTADO


Skyline Resultante
Altura
Posição X
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20

DIVISÃO E CONQUISTA

A solução que iremos abordar

A ideia principal é:

1. Se houver apenas um prédio, o skyline é simples: vai do início até o fim com a altura do prédio.

2. Se houver vários prédios, dividimos a lista em duas partes:
   - Resolvemos cada metade recursivamente.
   - Mesclamos (merge) os dois skylines para formar o skyline final.

Perceba que isso segue a estrutura clássica de Dividir para Conquistar – ideal para aplicar o Teorema Mestre.

X

FILA DE EVENTOS

Não tem recursividade

A ideia principal é:

1. Para cada prédio, cria dois eventos:
   - Início do prédio: (x_inicio, -altura)
   - Fim do prédio: (x_fim, altura)

2. Ordena todos os eventos por x, e em caso de empate:
   - Começos vêm antes dos fins
   - Prédios mais altos vêm antes

3. Usa uma heap máxima para guardar as alturas ativas.

4. Quando a altura mais alta muda → isso é um novo ponto no skyline.

ALGORITMO SKYLINE

DIVIDIR

O array é dividido em duas partes

```
meio = len(pontos) // 2
esquerda = pontos[:meio]
direita = pontos[meio:]
```

CONQUISTAR (RECURSIVIDADE)

O Skyline é calculado recursivamente para cada uma das metades

```
skyline_esq = skyline_recursivo(esquerda)
skyline_dir = skyline_recursivo(direita)
```

COMBINAR

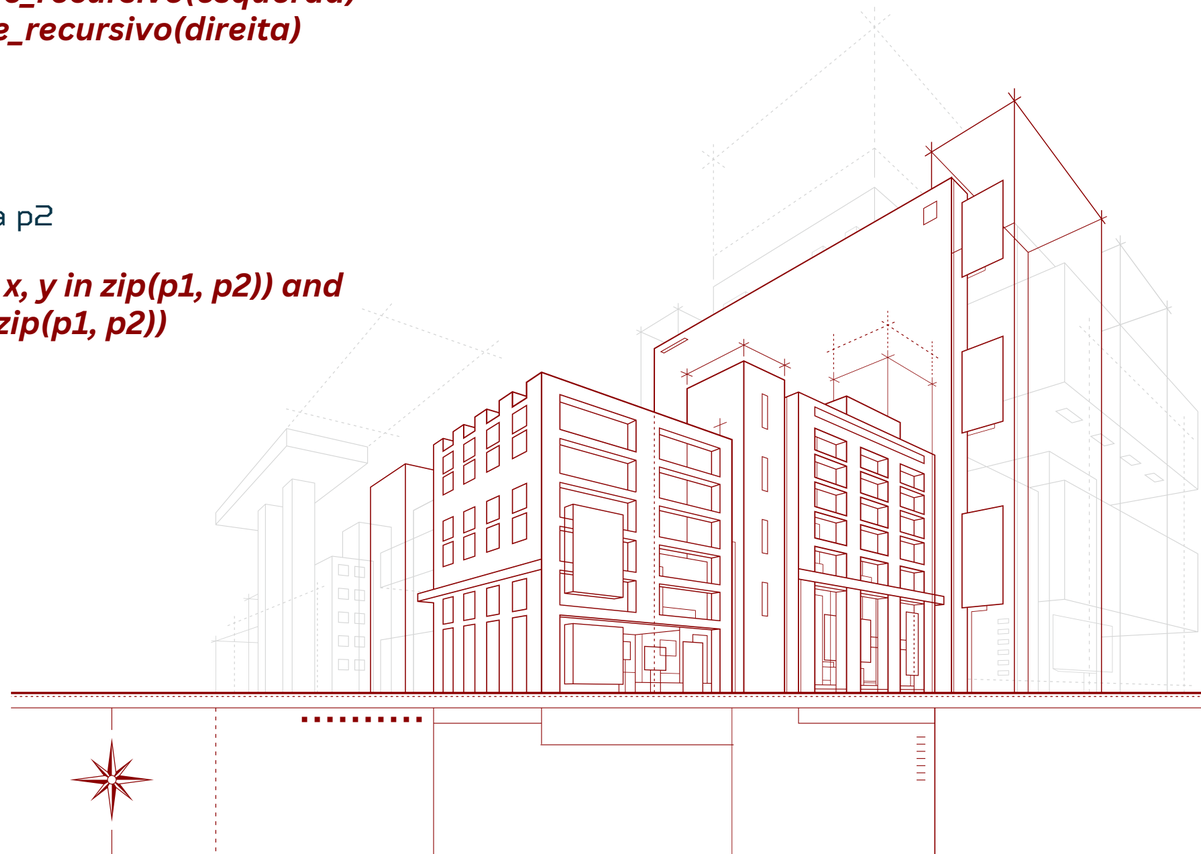
Os dois Skylines parciais serão combinados, removendo os pontos que são dominados por outros na união.

```
resultado = []
    for p in skyline_esq + skyline_dir:
        if not any(domina(outro, p) for outro in
resultado):
            resultado = [outro for outro in
resultado if not domina(p, outro)]
            resultado.append(p)
    return resultado
```

DOMINA

Verifica se p1 domina p2

```
return all(x <= y for x, y in zip(p1, p2)) and
any(x < y for x, y in zip(p1, p2))
```



CÓDIGO

```python
from typing import List, Tuple

def domina(p1: Tuple[float, ...], p2: Tuple[float, ...]) -> bool:

    return all(x <= y for x, y in zip(p1, p2)) and any(x < y for x, y in zip(p1, p2))

def dividir(pontos: List[Tuple[float, ...]]) -> Tuple[List[Tuple[float, ...]], List[Tuple[float, ...]]]:

    meio = len(pontos) // 2
    esquerda = pontos[:meio]
    direita = pontos[meio:]
    return esquerda, direita

def merge(skyline_esq: List[Tuple[float, ...]], skyline_dir: List[Tuple[float, ...]]) -> List[Tuple[float, ...]]:

    resultado = []
    for p in skyline_esq + skyline_dir:
        if not any(domina(outro, p) for outro in resultado):
            resultado = [outro for outro in resultado if not domina(p, outro)]
            resultado.append(p)
    return resultado

def skyline(pontos: List[Tuple[float, ...]]) -> List[Tuple[float, ...]]:

    if len(pontos) <= 1:
        return pontos

    esquerda, direita = dividir(pontos)

    skyline_esq = skyline(esquerda)
    skyline_dir = skyline(direita)

    return merge(skyline_esq, skyline_dir)

```

EXEMPLO 1

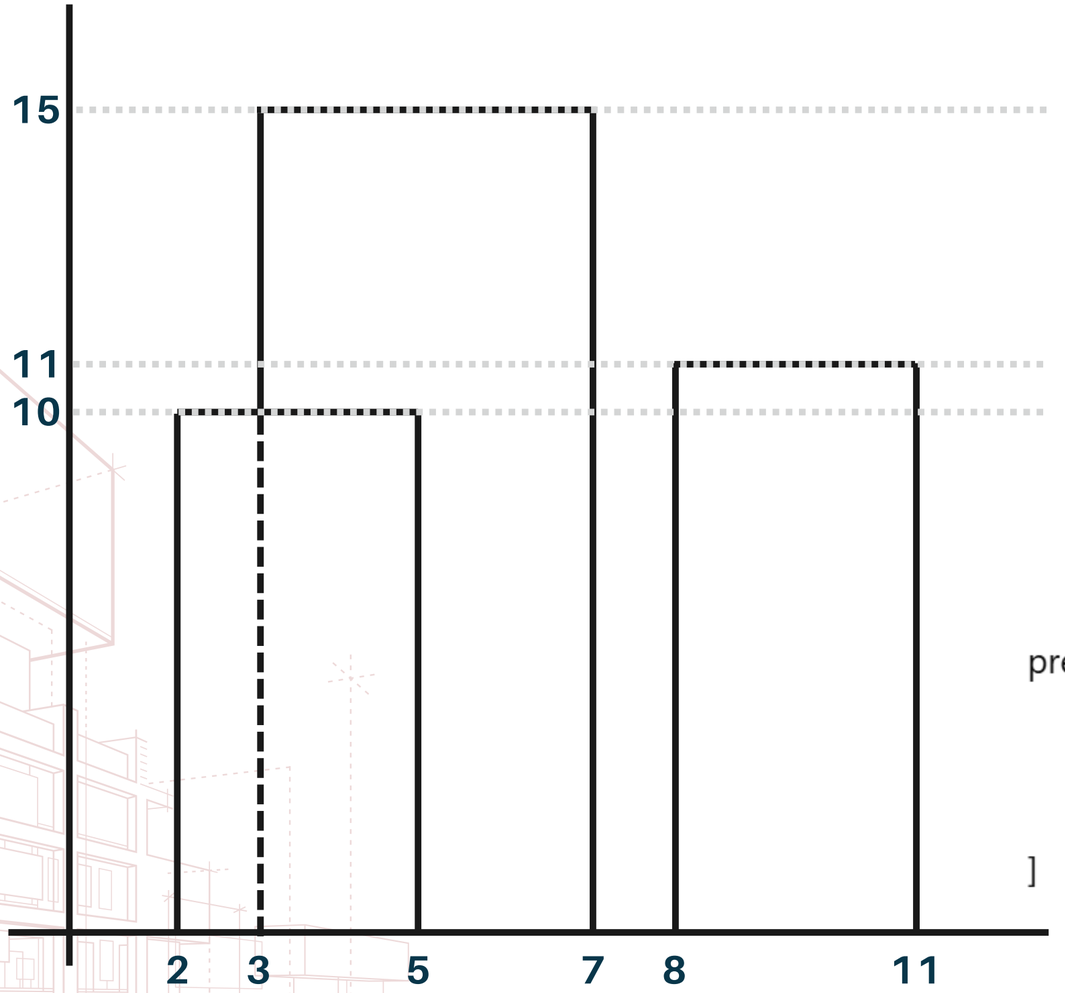
Fila de eventos

Passo 1:

```
eventos = [
    (2, -10),  # A começa
    (5, 10),   # A termina
    (3, -15),  # B começa
    (7, 15),   # B termina
    (8, -11),  # C começa
    (11, 11)   # C termina
]
```

Passo 2: ordenação

```
eventos_ordenados = [
    (2, -10),
    (3, -15),
    (5, 10),
    (7, 15),
    (8, -11),
    (11, 11)
]
```




```
prédios = [
    [2, 5, 10],  # prédio A
    [3, 7, 15],  # prédio B
    [8, 11, 11]   # prédio C
]
```

Passo 3: Simulação com heap

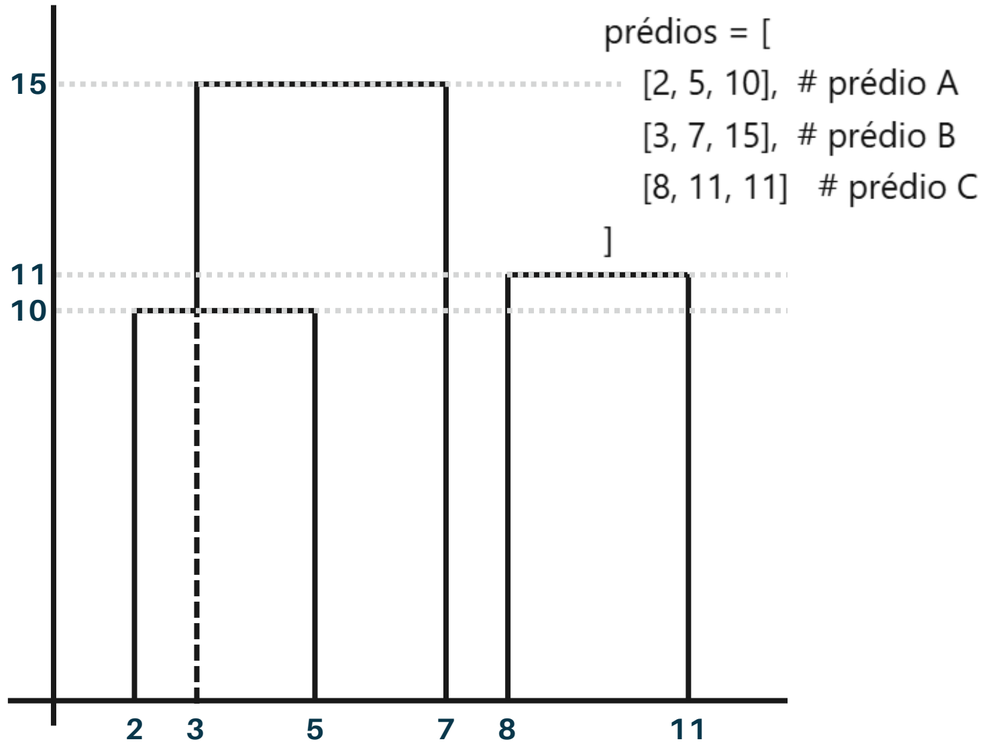
Iniciamos com uma heap que contém [**0**], representando o chão, e uma variável **altura_atual** = **0**.

Para cada evento ordenado, processamos da seguinte forma:

1. (**2,-10**): Começa o prédio A (altura 10). Inserimos -**10** na heap. A maior altura agora é 10, que é diferente da anterior (0), então adicionamos [**2, 10**] ao skyline.
2. (**3, -15**): Começa o prédio B (altura 15). Inserimos -**15**. Agora a maior altura é 15, diferente da anterior (10), então adicionamos [**3, 15**] .
3. (**5, 10**): Termina o prédio A. Removemos 10 da lista de alturas ativas. A altura máxima permanece 15, então nada muda no skyline.
4. (**7, 15**): Termina o prédio B. Removemos 15. Agora a única altura restante é 0 (chão), então a altura máxima muda para O. Adicionamos [**7, 0**].
5. (**8, -11**): Começa o prédio C (altura 11). Inserimos -**11**. A nova altura máxima é 11, diferente de 0, então adicionamos [**8, 11**].
6. (**11, 11**): Termina o prédio C. Removemos 11. A altura máxima volta para 0, então adicionamos [**11,0**].

SKYLINE FINAL

```
[[2, 10], [3, 15], [7, 0], [8, 11], [11, 0]]
```




```
eventos_ordenados = [
    (2, -10),
    (3, -15),
    (5, 10),
    (7, 15),
    (8, -11),
    (11, 11)
]
```

EXEMPLO 2

Dividir e conquistar

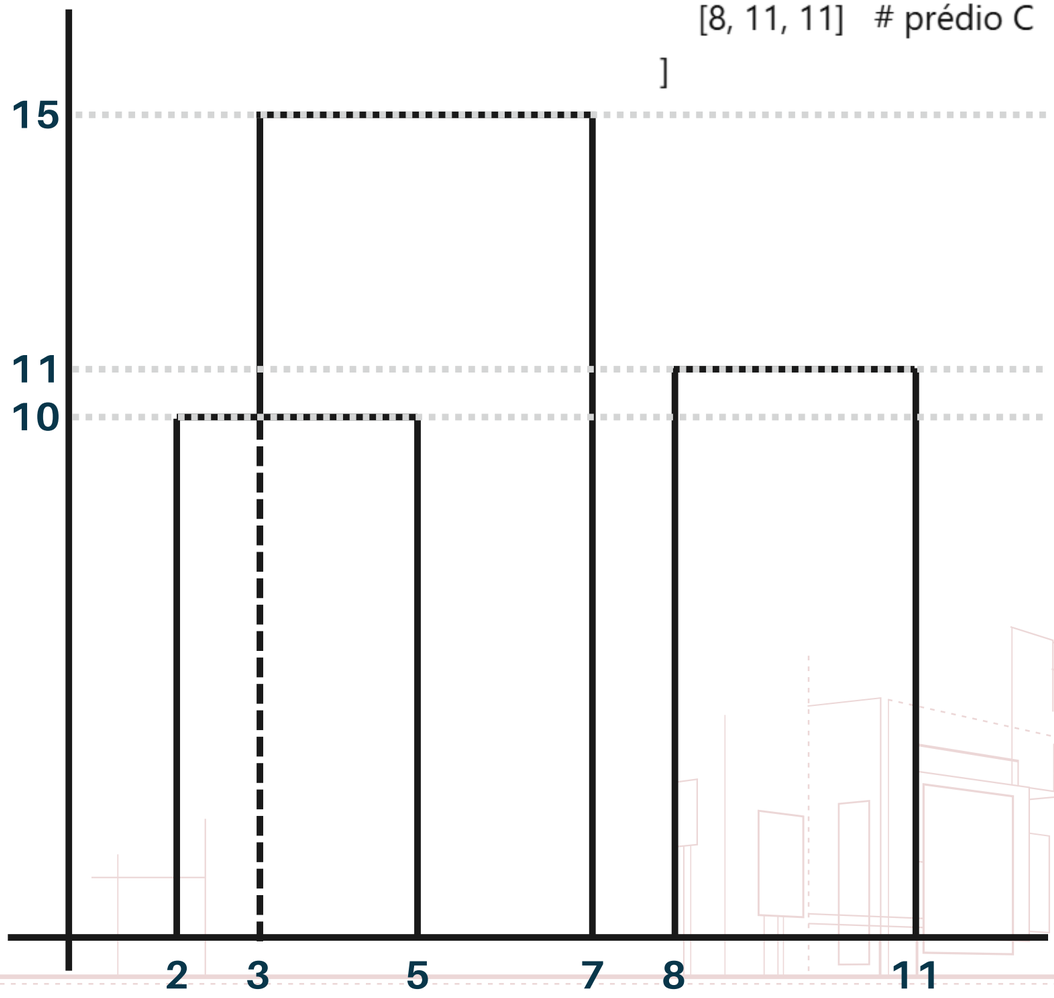
Passo 1: Divide a lista no meio até sobrar 1 prédio em cada sublista:

```
[ [2,5,10], [3,7,15], [8,11,11] ]

→ esquerda = [ [2,5,10] ]
→ direita  = [ [3,7,15], [8,11,11] ]

→ divide direita:
    esquerda_d = [ [3,7,15] ]
    direita_d  = [ [8,11,11] ]
```

Passo 2: Resolver os casos base (Gerando skylines individuais)

```
[2,5,10]     → [[2,10], [5,0]]
[3,7,15]     → [[3,15], [7,0]]
[8,11,11]    → [[8,11], [11,0]]
```

```
prédios = [
    [2, 5, 10],  # prédio A
    [3, 7, 15],  # prédio B
    [8, 11, 11]   # prédio C
]
```



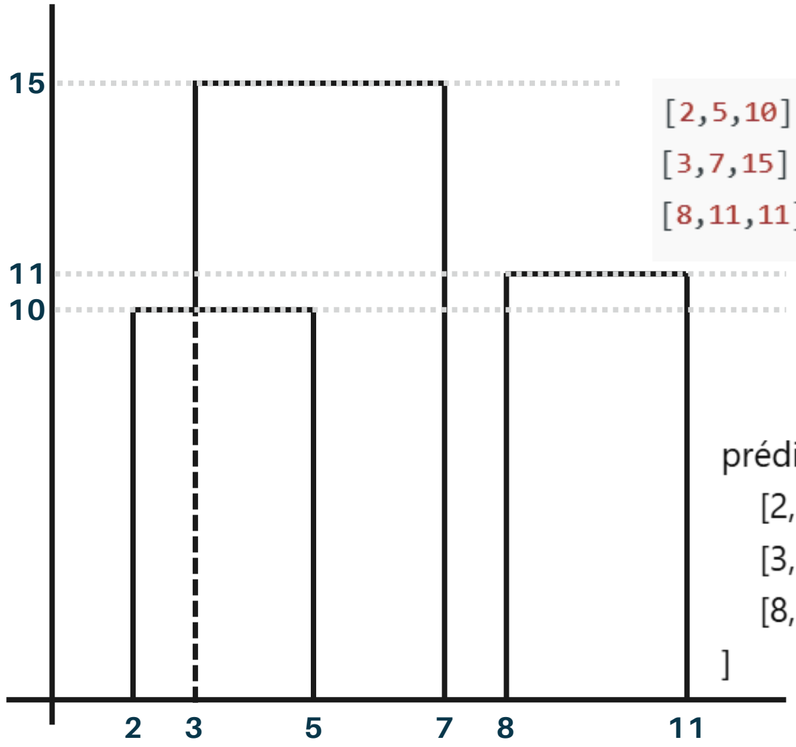
Passo 3: Combinar skylines

✅ **Primeiro merge:** `[[3,15],[7,0]]` **com** `[[8,11],[11,0]]`

Passo a passo do merge:

- `3 < 8` → pega `[3,15]`
- `7 < 8` → pega `[7,0]`
- `8, 11` → adiciona os dois restantes

➡️ Resultado intermediário: `[[3,15], [7,0], [8,11], [11,0]]`




```
[2,5,10]    → [[2,10], [5,0]]
[3,7,15]    → [[3,15], [7,0]]
[8,11,11]   → [[8,11], [11,0]]
```

```
prédios = [
    [2, 5, 10],  # prédio A
    [3, 7, 15],  # prédio B
    [8, 11, 11]   # prédio C
]
```

✅ **Segundo merge:** `[[2,10], [5,0]]` **com** `[[3,15], [7,0], [8,11], [11,0]]`

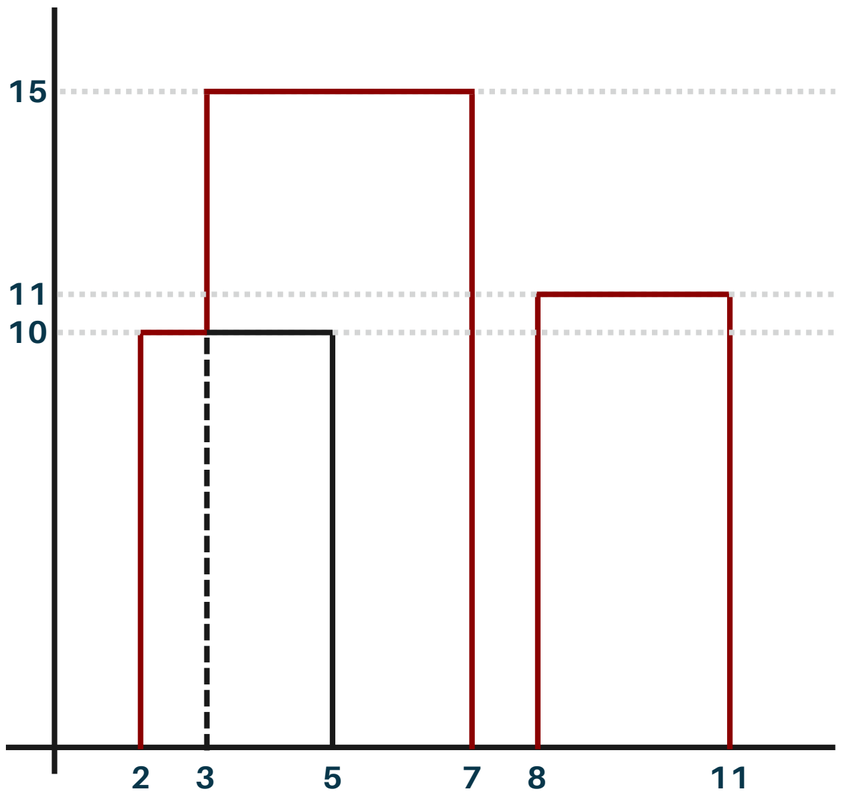
Passo a passo do merge:

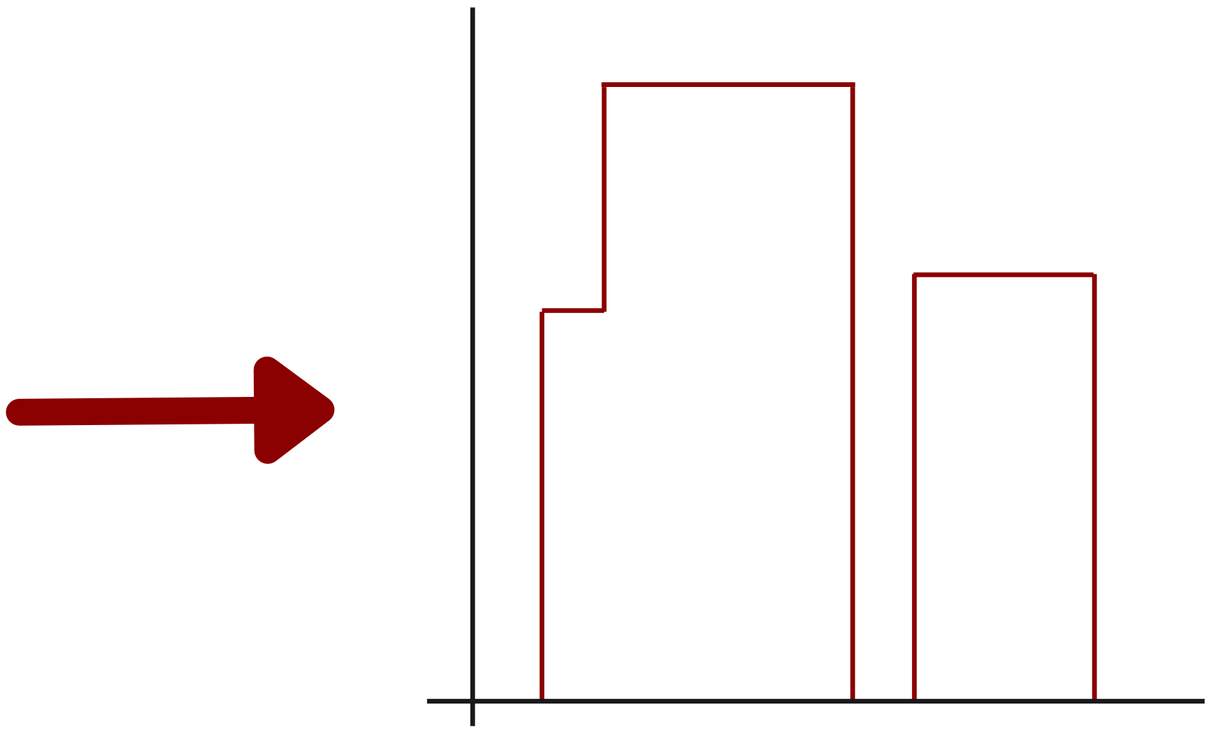
- `2 < 3` → pega `[2,10]`
  Altura atual: max(10, 0) = 10 → skyline: `[2,10]`
- `3 < 5` → pega `[3,15]`
  Altura atual: max(10, 15) = 15 → muda → `[3,15]`
- `5 < 7` → pega `[5,0]`
  Altura atual: max(0, 15) = 15 → não muda
- `7 < 8` → pega `[7,0]`
  Altura atual: max(0, 0) = 0 → muda → `[7,0]`
- Resto: `[8,11]`, `[11,0]` → adiciona

SKYLINE FINAL

```
[[2, 10], [3, 15], [7, 0], [8, 11], [11, 0]]
```

SKYLINE RESULTANTE DOS EXEMPLOS





SKYLINE FINAL

```
[[2, 10], [3, 15], [7, 0], [8, 11], [11, 0]]
```

TEOREMA MESTRE

Fórmula da Recorrência:

$$T(n) = aT(n/b) + f(n)$$

Parâmetros e Limites:

$a = 2$

$b = 2$

$f(n) = \Theta(n)$

Caso 1:

$f(n) = n^{\log_b(a-\varepsilon)}$

$n^{\log_2(2-\varepsilon)}$, supondo que $\varepsilon = 1$

$n^{\log_2(1)} = 0$

Falhou!!

Caso 2:

$f(n) = n^{\log_b(a)}$

$n^{\log_2(2)}$

$n^1 = n$

Verificação: resultado lg n = n log n

Complexidade: *n log n*

```
# Sample dataset of restaurants with ratings and prices
restaurants = [
    {"name": "Restaurant A", "rating": 4.5, "price": 30},
    {"name": "Restaurant B", "rating": 4.8, "price": 25},
    {"name": "Restaurant C", "rating": 4.2, "price": 20},
    {"name": "Restaurant D", "rating": 4.6, "price": 35},
]

# Function to check if point A dominates point B
def dominates(a, b):
    return a["rating"] >= b["rating"] and a["price"] <=
b["price"]

# Find skyline points
skyline = []

for restaurant in restaurants:
    is_skyline = all(not dominates(restaurant, other) for
other in restaurants)
    if is_skyline:
        skyline.append(restaurant)

# Print the skyline points
for restaurant in skyline:
    print(f"{restaurant['name']} - Rating:
{restaurant['rating']}, Price: ${restaurant['price']}")
```

APLICAÇÕES

Uso na análise de datasets:

Em datasets multidimensionais, as vezes queremos detectar os datapoints que mais influem em determinada dimensão, mas as vezes eles não são denominadas por outros pontos, sendo "independentes".

✒️ ***Skyline points***

- viabiliza a extração de insights direcionados ao propósito da análise
- **determinação do melhor restaurante baseado nos pontos mais altos de “rating” e “price”**
- Útil também para detectar anomalias em datasets de series temporais

Uso na detecção de anomalias em Machine Learning

Acompanhamento de performance

PERGUNTAS?

VAMOS PRATICAR?

Algoritmo Skyline - Mentimeter

Be heard, collaborate, and share ideas—make meetings and classes more engaging with real conversations.

menti.com

Código: ***84389134***

https://www.menti.com/alnt8gkdxeav

QUIZ

Perguntas Sugeridas

O que o algoritmo Skyline garante sobre o resultado final?

a) Que sempre terá número par de pontos
b) Que o skyline terá altura constante ao longo do tempo
c) Que o primeiro e último pontos têm mesma altura
d) Que os pontos são ordenados e representam mudanças de altura

Em que cenário a complexidade do merge pode ser $O(n^2)$?

a) Quando todos os prédios têm largura igual a 1
b) O skyline não estiver ordenado e for necessário reordenar durante o merge
c) Quando os prédios estão todos alinhados
d) Quando todos os prédios começam no mesmo ponto

Durante o merge de dois skylines, se dois pontos tiverem o mesmo X mas alturas diferentes, qual deve ser considerado?

a) O de menor altura
b) A média das alturas
c) Ambos devem ser descartados
d) O maior valor entre as alturas vigentes no momento

Em termos de stack de execução, quantas chamadas recursivas simultâneas teremos no pior caso para n prédios:

a) O(log n), devido à divisão binária
b) O(n)
c) O(n log n)
d) Constante, pois é tail-recursive

Em uma chamada recursiva do algoritmo Skyline, o que ocorre na etapa de merge?

a) Exclusão de prédios sobrepostos
b) Combinação dos skylines das subcidades mantendo ordem e altura
c) Substituição da menor altura pela maior
d) Normalização dos prédios

O que faz o algoritmo Skyline efetivamente eliminar partes invisíveis dos prédios?

a) A ordenação dos prédios
b) O uso de árvore binária
c) A lógica do merge que mantém apenas alterações na altura
d) A verificação de áreas com mesmo Y

Sobre a recorrência do Skyline, temos $T(n)=2T(n/2)+O(n)$. Dessa forma, é correto afirmar:

a) É um caso típico de recorrência exponencial
b) O Teorema Mestre não se aplica neste caso
c) O caso 2 do Teorema Mestre se aplica, resultando em O(n log n)
d) A solução é $O(n^2)$ pois o merge é quadrático

Ana Beatriz Alves

Caio Barreto

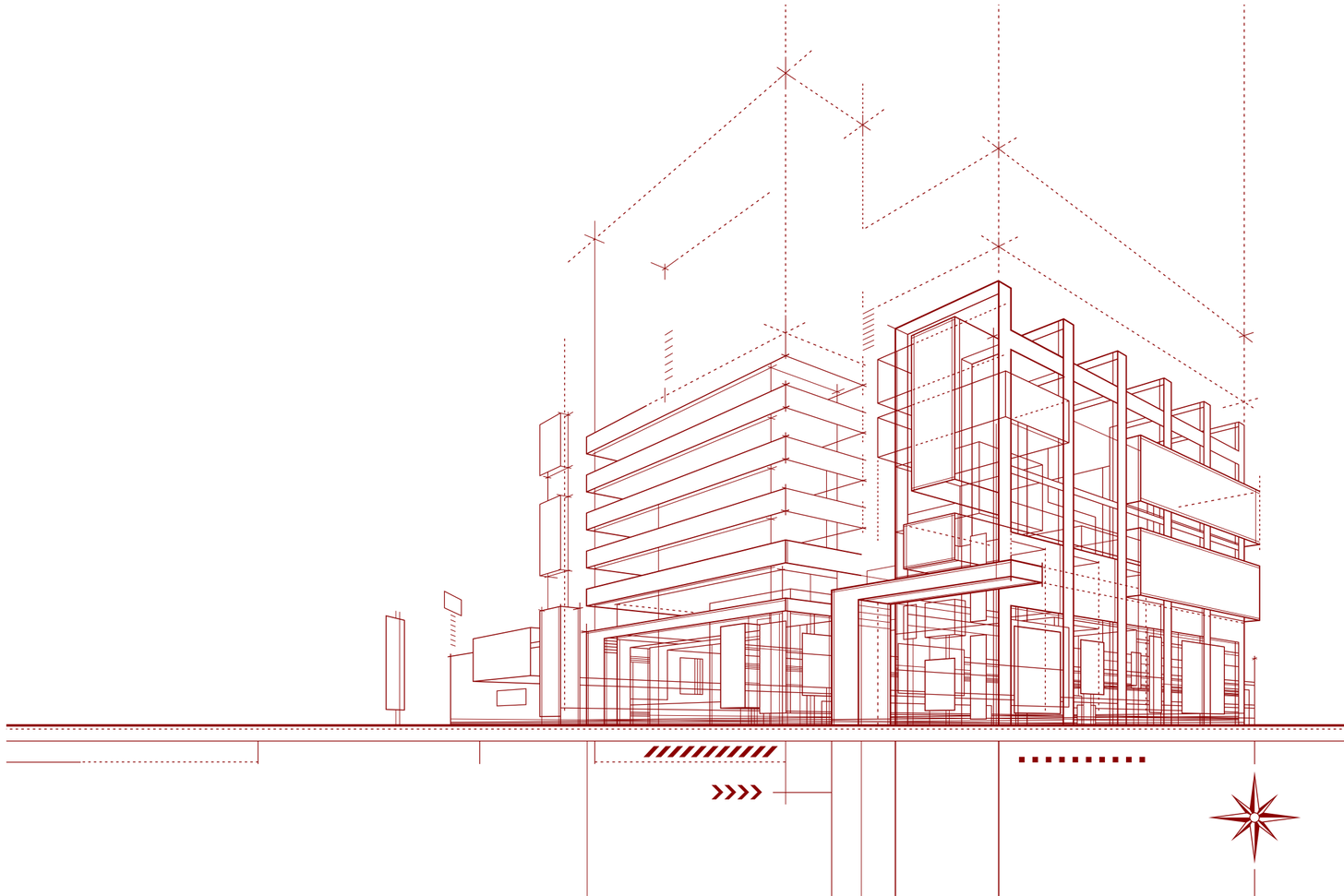
Maria Luísa Arruda

EQUIPE

Isabela Spinelli

Victor Hora

AGRADECEMOS!



C.E.S.A.R school

Analise de Algoritmos
2025.1