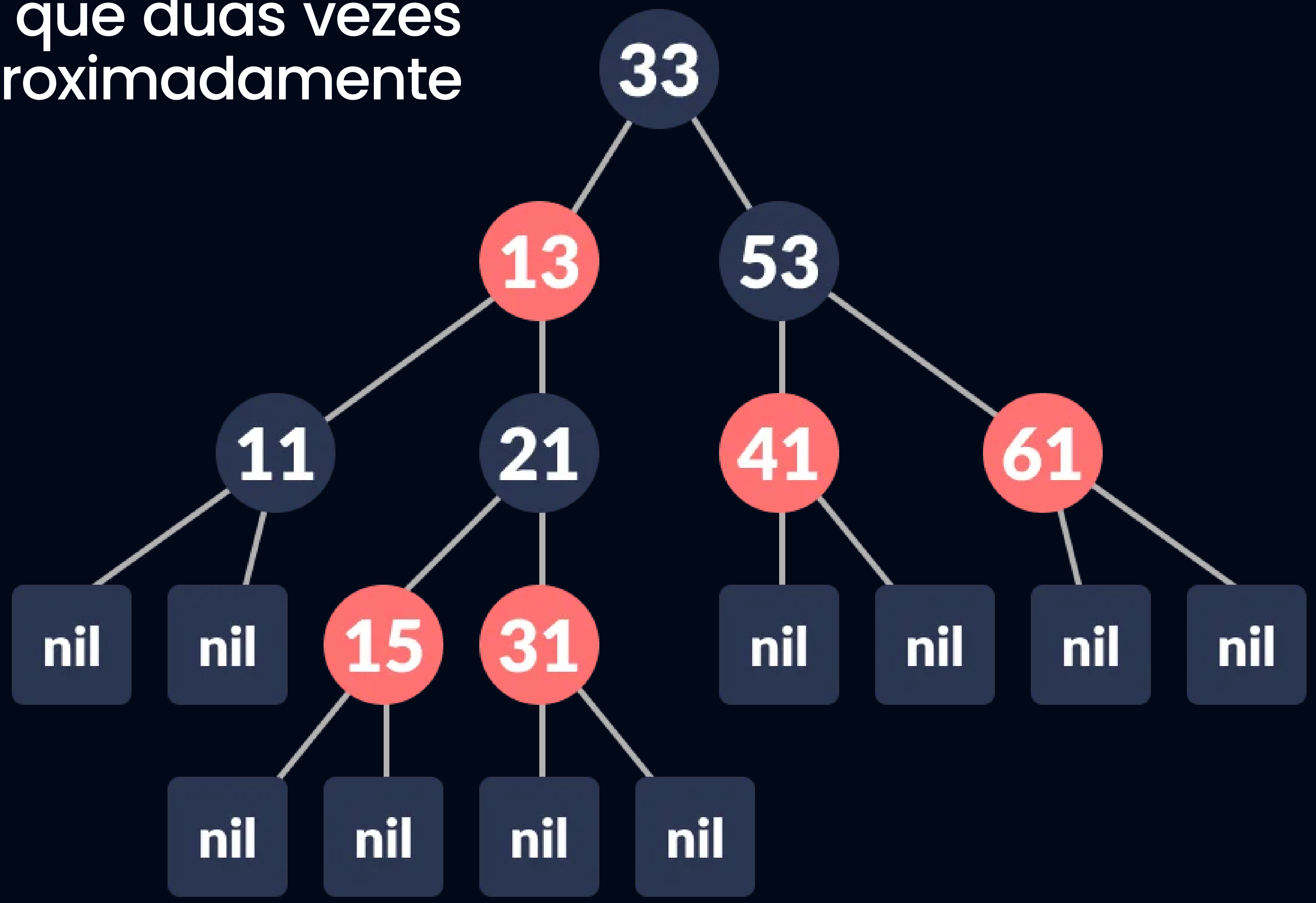
Aluno: Kevin Willyn

Inserção Árvore Red black

Árvore Vermelha e Preta.

Uma árvore vermelho-preto é uma árvore de busca binária com um bit extra de armazenamento por nó: sua cor pode ser VERMELHA ou PRETA. Restringindo as cores dos nós em qualquer caminho simples da raiz até uma folha, as árvores vermelho-preto asseguram que o comprimento de nenhum desses caminhos seja maior que duas vezes o de qualquer outro, de modo que a árvore é aproximadamente balanceada



Propriedades

Árvore Vermelha e Preta

- Cada nó Tem uma cor: vermelho ou preto.
- A raiz é preta
- Cada folha (NIL) é preta
- o pai de um nó vermelho é sempre preto.
- Para cada nó, todos os caminhos simples do nó até folhas descendentes contêm o mesmo número de nós pretos

Pontos importantes para a Inserção

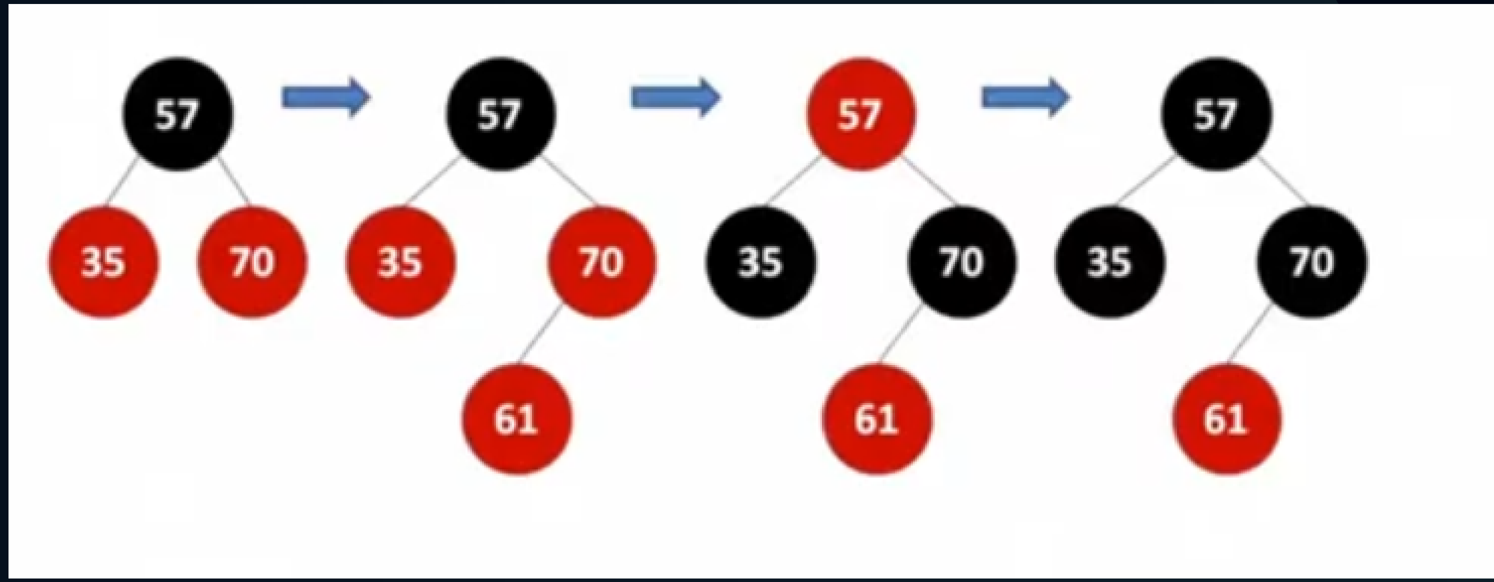
Árvore Vermelha e Preta

- Cada nó inserido , por definição possui cor vermelha.
- A inserção e exatamente igual a de uma ABB.
- Apos a inserção, verifica as propriedades que se mantem:
- Lembrando:
  -> a raiz da arvore e sempre negra
  -> se o pai do novo nó inserido for preto, se mantem todas as propriedades
  -> se o pai do novo nó inserido for vermelho,rotações e alteração de cores precisam ser feitas.

Caso da Inserção

Árvore Vermelha e Preta

- **Caso 1: O pai e o tio do novo nó são vermelhos.**

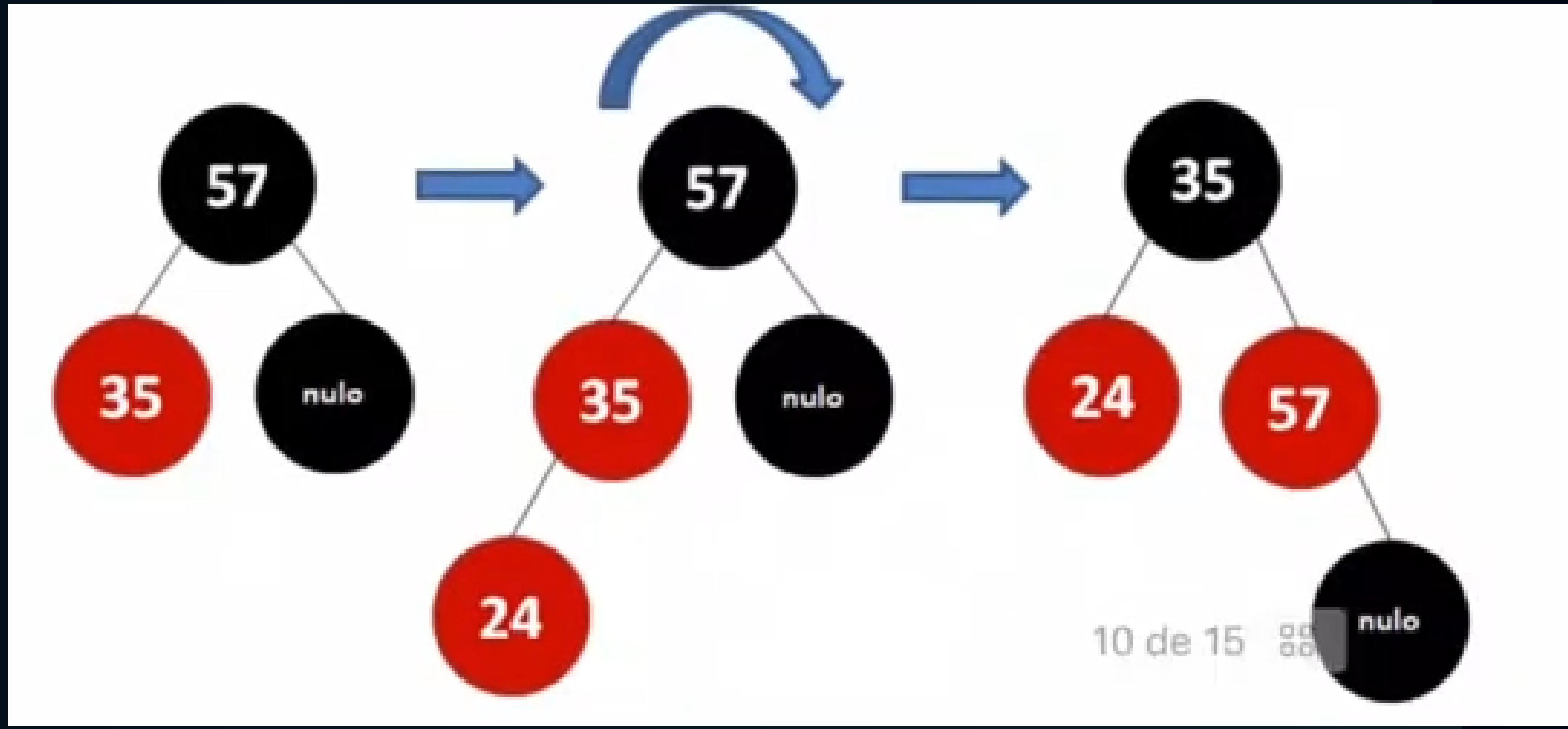


- **Caso 2: O pai e vermelho e o tio e preto**
- **Caso 2a: o novo nó inserido é o filho esquerdo**
- **Caso 2b: o pai do novo nó inserido é o filho direito.**

Caso da Inserção

Árvore Vermelha e Preta

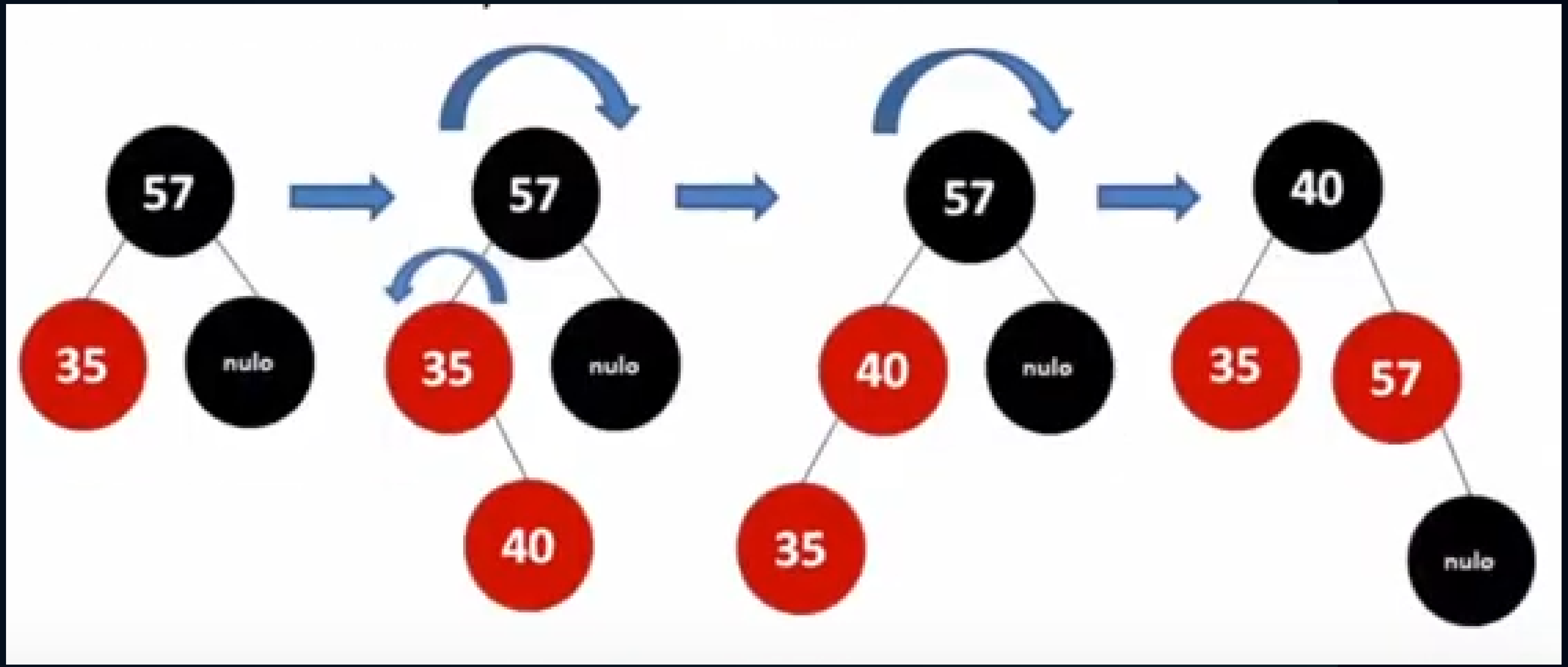
- **Caso 2: O pai é vermelho e o tio e preto**
- **Caso 2a: o novo nó inserido é o filho esquerdo**



Caso da Inserção

Árvore Vermelha e Preta

- **Caso 2: O pai e vermelho e o tio e preto**
- **Caso 2b: o novo nó inserido é o filho direito.**



Pseudocódigo

```
Função inserir(vermelhoNegro, valor):
    novoNodo = criarNodo(valor)
    se raiz(vermelhoNegro) == nulo:
        raiz(vermelhoNegro) = novoNodo
        cor(novoNodo) = preto
    senão:
        nodoAtual = raiz(vermelhoNegro)
        enquanto verdadeiro:
            se valor(novoNodo) < valor(nodoAtual):
                se esquerda(nodoAtual) == nulo:
                    esquerda(nodoAtual) = novoNodo
                    pai(novoNodo) = nodoAtual
                    sair do loop
                senão:
                    nodoAtual = esquerda(nodoAtual)
            senão se valor(novoNodo) > valor(nodoAtual):
                se direita(nodoAtual) == nulo:
                    direita(nodoAtual) = novoNodo
                    pai(novoNodo) = nodoAtual
                    sair do loop
                senão:
                    nodoAtual = direita(nodoAtual)
            senão:
                retornar # valor já existe na árvore
        ajustarInsercao(vermelhoNegro, novoNodo)
```

```
Função ajustarInsercao(vermelhoNegro, nodo):
    enquanto cor(pai(nodo)) == vermelho:
        se pai(nodo) == esquerda(pai(pai(nodo))):
            tioNodo = direita(pai(pai(nodo)))
            se cor(tioNodo) == vermelho:
                cor(pai(nodo)) = preto
                cor(tioNodo) = preto
                cor(pai(pai(nodo))) = vermelho
                nodo = pai(pai(nodo))
            senão:
                se nodo == direita(pai(nodo)):
                    nodo = pai(nodo)
                    rotacaoEsquerda(vermelhoNegro, nodo)
                cor(pai(nodo)) = preto
                cor(pai(pai(nodo))) = vermelho
                rotacaoDireita(vermelhoNegro, pai(pai(nodo)))
        senão:
            tioNodo = esquerda(pai(pai(nodo)))
            se cor(tioNodo) == vermelho:
                cor(pai(nodo)) = preto
                cor(tioNodo) = preto
                cor(pai(pai(nodo))) = vermelho
                nodo = pai(pai(nodo))
            senão:
                se nodo == esquerda(pai(nodo)):
                    nodo = pai(nodo)
                    rotacaoDireita(vermelhoNegro, nodo)
                cor(pai(nodo)) = preto
                cor(pai(pai(nodo))) = vermelho
                rotacaoEsquerda(vermelhoNegro, pai(pai(nodo)))
    cor(raiz(vermelhoNegro)) = preto
```

-> FUNÇÃO DE CUSTO

Vamos considerar a árvore inicialmente vazia e um novo nodo sendo inserido:

1. Inserção do novo nodo:

- Criar um novo nodo: O(1)
- Verificar se a raiz é nula: O(1)
- Se a raiz for nula, definir o novo nodo como raiz e colori-lo de preto: O(1)
- Caso contrário, percorrer a árvore para encontrar a posição correta para inserir o novo nodo:
  - Percorrer a árvore: **O(log n)** no pior caso, onde n é o número de nodos na árvore
  - Comparar o valor do novo nodo com o valor do nodo atual: O(1)
  - Verificar se o filho esquerdo ou direito do nodo atual é nulo: O(1)
  - Se o filho esquerdo ou direito for nulo, inserir o novo nodo nessa posição: O(1)
  - Caso contrário, continuar percorrendo a árvore
- Executar a função de ajuste de inserção: **O(log n)** no pior caso, onde n é o número de nodos na árvore

-> COMPLEXIDADE: O(LOG N)

-> COMPLEXIDADE: O(LOG(N))

A complexidade da árvore Red-Black é determinada pela altura da árvore em relação ao número de elementos armazenados nela. A altura da árvore Red-Black é limitada superiormente por 2*log(n+1). Isso significa que a altura da árvore nunca será maior do que duas vezes o logaritmo do número de elementos mais um.

ARVORE VERMELHO E PRETA VS AVL

- **Inserção e exclusão:** Ambas as estruturas de dados oferecem operações de inserção e exclusão eficientes, mas as árvores RB são geralmente mais rápidas para inserções e exclusões.

- **Eficiência de pesquisa:** As árvores AVL são mais eficientes em operações de pesquisa do que as árvores RB.

- **Espaço necessário:** As árvores RB geralmente requerem menos espaço do que as árvores AVL, já que as árvores AVL precisam armazenar um fator de equilíbrio adicional em cada nó para manter a estrutura equilibrada.

- **Uso em cenários específicos:** As árvores AVL são geralmente preferíveis em cenários onde a eficiência de pesquisa é crucial, enquanto as árvores RB são preferíveis em cenários onde as operações de inserção e exclusão são mais frequentes do que as operações de pesquisa.

Cada tipo de árvore binária de busca tem seus próprios prós e contras, e a escolha depende do cenário e dos requisitos específicos da aplicação.

Grafico

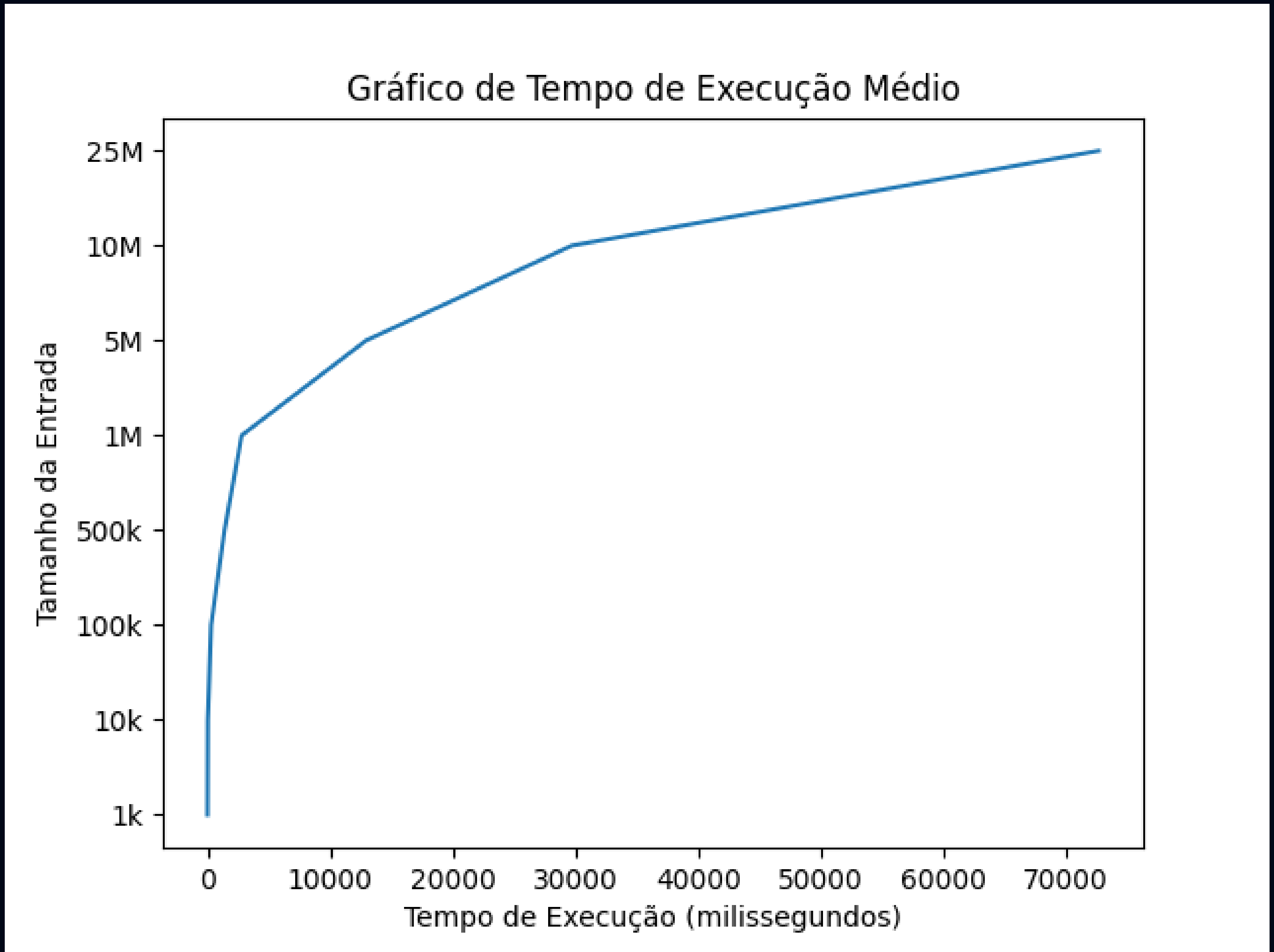


Gráfico de Tempo de Execução Médio
Tamanho da Entrada
25M
10M
5M
1M
500k
100k
10k
1k
0
10000
20000
30000
40000
50000
60000
70000
Tempo de Execução (milissegundos)

The end