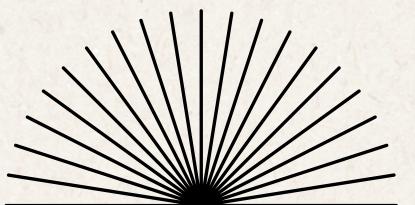


# TIMSORT

**Estrutura de Dados**

Eduardo Netto - 22409922  
Igor Durando - 22408431  
Rafael Fontenele - 22409349



# Sumário

03	<b>Introdução</b>
04	<b>Como funciona</b>
05	<b>RUNs</b>
06	<b>Código no Colab</b>
07	<b>Gráfico</b>
08	<b>Complexidade</b>
09	<b>Conclusão</b>

# Introdução

03

**01** É um algoritmo híbrido, combinando o melhor de dois algoritmos, sendo eles o Insertion Sort e o Merge Sort

**02** - Insertion Sort é usado para pequenos blocos de dados, muito rápido para blocos pequenos ou quase ordenados.

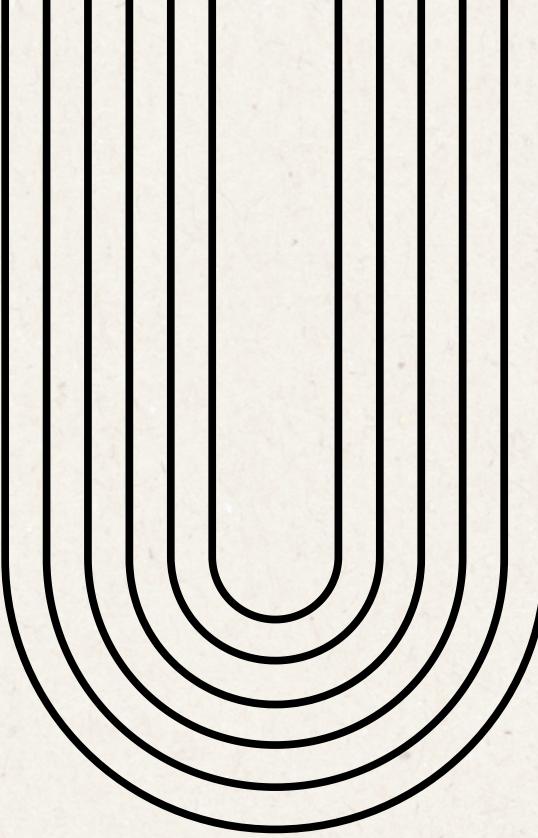
- Merge Sort é usado para combinar os blocos maiores ordenados pelo insertion Sort. É eficiente em listas grandes.

**03** • Estabilidade  
O Timsort é um algoritmo estável, pois preserva a ordem relativa de elementos com chaves iguais. Isso significa que, se dois valores idênticos aparecem na lista original, eles continuarão na mesma ordem após a ordenação.

• Adaptatividade  
O Timsort é adaptativo porque seu desempenho melhora quando a lista de entrada já possui algum nível de ordenação. Ele identifica e aproveita essas sequências já ordenadas (“runs”), tornando-se especialmente eficiente em dados reais, que raramente estão totalmente desorganizados.

# Como Funciona

O Timsort analisa a lista e identifica trechos já ordenados. Cada trecho é organizado e ajustado para ter um tamanho mínimo. Depois, esses trechos são ordenados internamente com insertion sort (quando pequenos) e então combinados usando um merge otimizado. Durante o processo, o algoritmo adapta suas operações para aproveitar padrões da lista e reduzir o número de comparações e cópias.

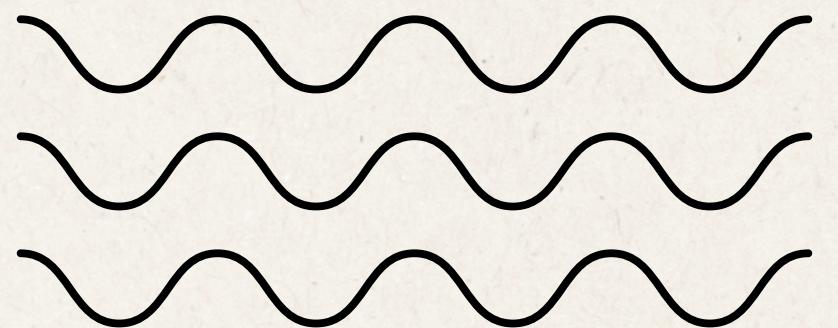


As RUNs são sequências consecutivas da lista que já estão ordenadas, seja de forma crescente ou decrescente.

O Timsort identifica essas sequências naturalmente presentes na lista e as usa como blocos iniciais da ordenação.

Se uma RUN for muito pequena, ela é ampliada usando insertion sort até atingir um tamanho mínimo definido.

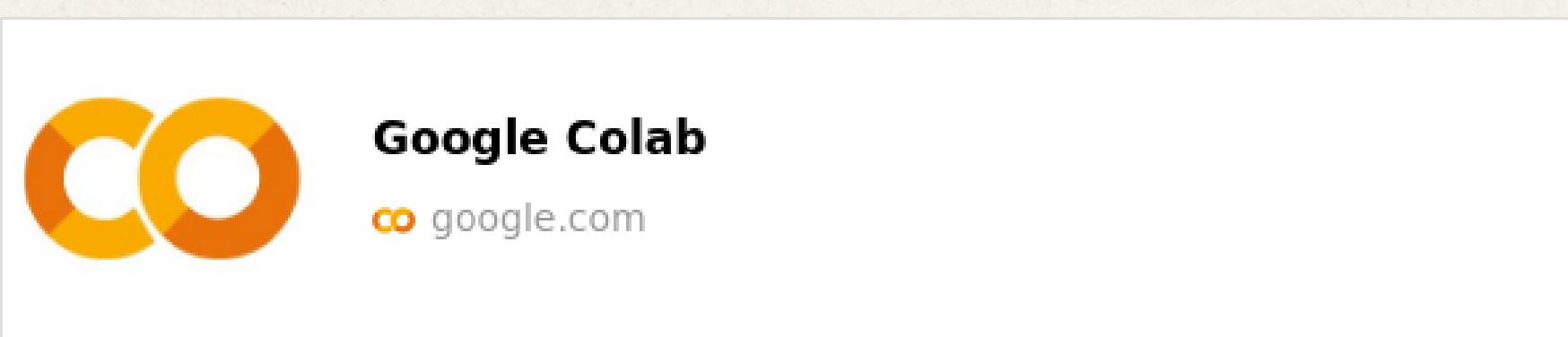
Em seguida, todas as RUNs são organizadas e mescladas em uma ordem específica, garantindo eficiência e equilíbrio no processo. Aproveitar essas RUNs é o que torna o Timsort extremamente rápido em dados reais.



## RUNS

(o coração do Timsort)

# Código no Colab



# Gráfico

Escolha a ordenação:

1 - Crescente

2 - Decrescente

Digite 1 ou 2: 1

Digite números separados por espaço:

5 7 19 29

Resultado da ordenação:

[5, 7, 19, 29]

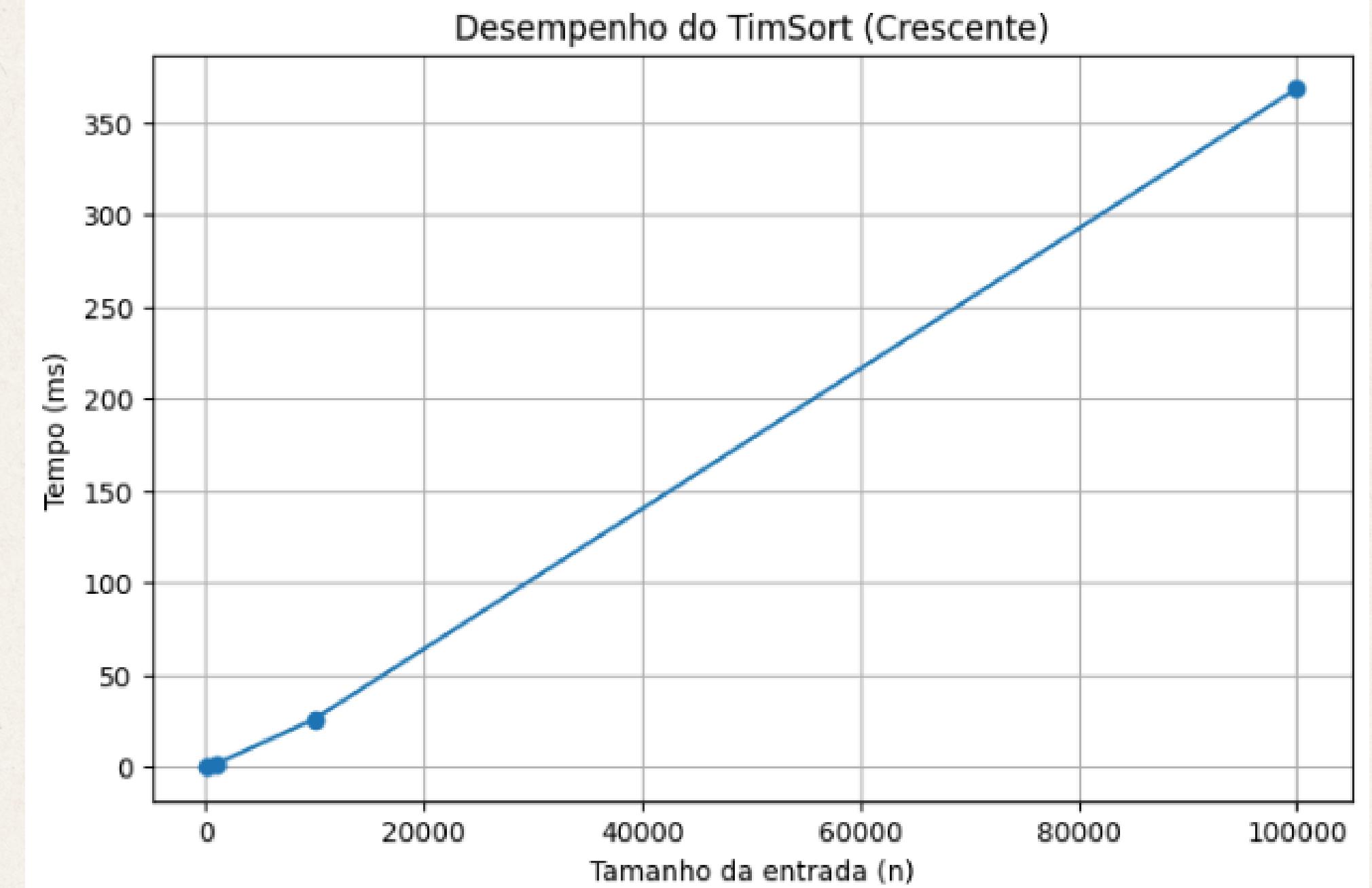
--- INICIANDO TESTE DE DESEMPENHO ---

Entrada 100: 0.121 ms

Entrada 1000: 1.821 ms

Entrada 10000: 26.054 ms

Entrada 100000: 368.366 ms



# Complexidade

## 1. Melhor Caso – $O(N)$

Quando a lista já está ordenada (ou quase), o Timsort reconhece rapidamente essas runs naturais e praticamente não realiza trabalho extra.

Essa é uma das maiores vantagens do algoritmo: ele aproveita a ordem já existente, tornando-se extremamente eficiente para dados reais.

## 2. Pior Caso – $O(N \log N)$

Mesmo quando a lista está totalmente desordenada, o Timsort mantém um desempenho ótimo, alcançando o melhor tempo assintótico possível para algoritmos de ordenação baseados em comparação.

Isso garante que, mesmo no cenário mais difícil, ele continua tão eficiente quanto Merge Sort e outros algoritmos de alto desempenho.

# Conclusão

Com base nas execuções realizadas, percebemos que o processo se mantém estável, apresentando apenas pequenas variações naturais entre uma run e outra. A média obtida representa bem o desempenho geral e não foram identificados outliers significativos que pudesse comprometer o resultado. No geral, os testes mostram que repetir as execuções é importante para ter uma visão real do comportamento do processo e evitar conclusões baseadas em uma única medição.