



INTROSORT

Amanda Scalari
Emilly Pereira
Hannah Oliveira
Matheus Junqueira
Rodrigo Rocha





O QUE É INTROSORT?

Algoritmo de ordenação híbrido criado por David Musser na década de 90

Mistura de aspectos de:

Quicksort + Insertion sort + Heapsort

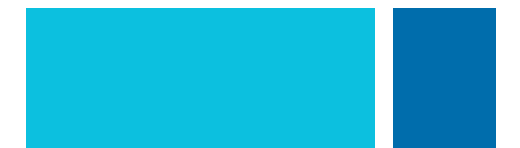




CARACTERÍSTICAS



- Complexidade $O(n \log n)$ em todos os casos
- Controle da profundidade da pilha de execução do Quicksort (através do Heapsort)
- Ordenação em estruturas pequenas pelo Insertion






CARACTERÍSTICAS



Algoritmo	Melhor caso	Médio caso	Pior caso
Insertion	$O(n)$	$O(n^2)$	$O(n^2)$
Selection	$O(n^2)$	$O(n^2)$	$O(n^2)$
Quick	$O(n \log n)$	$O(n \log n)$	$O(n^2)$
Merge	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$
Intro	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$



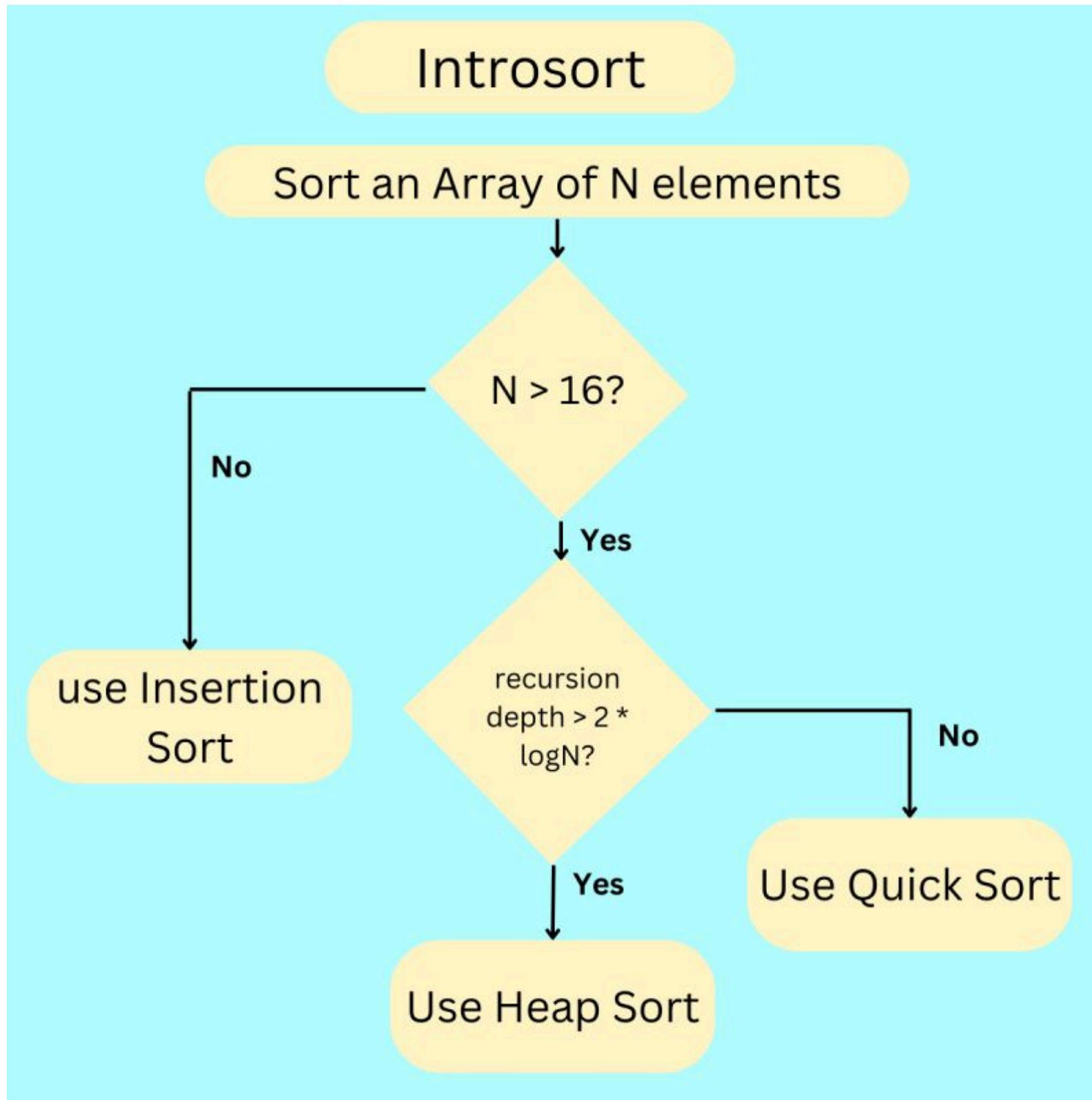
PSEUDOCÓDIGO

Algoritmo 1: IntroSort

Entrada: Vetor *vet*, inteiros *ini*, *fim*, profundidade máxima *prof_max*, estrutura *dados*, índice *i*

Saída : Vetor ordenado

```
1 tam ← (fim − ini) + 1;
2 se tam ≤ 16 então
3   chamar InsertionIntro(vet, ini, fim, dados, i);
4   retornar;
5 fim
6 se prof_max = 0 então
7   chamar HeapSort(vet, tam, dados, i);
8   retornar;
9 fim
10 senão
11   pivo ← Particiona(vet, ini, fim, dados, i);
12   chamar IntroSort(vet, ini, pivo − 1, prof_max − 1, dados, i);
13   chamar IntroSort(vet, pivo + 1, fim, prof_max − 1, dados, i);
14 fim
15 end
```

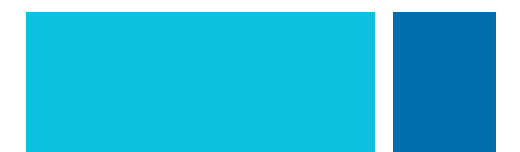
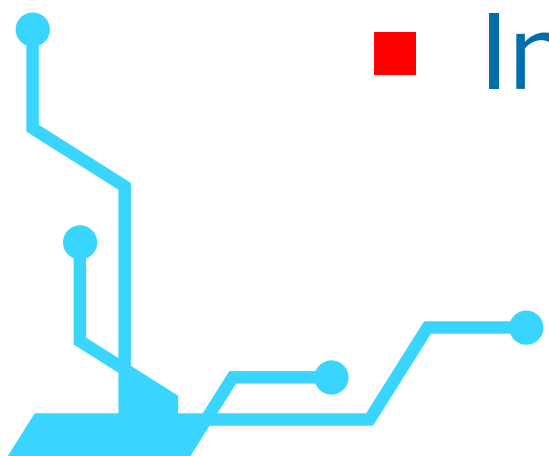


FLUXOGRAMA



VANTAGENS E DESVANTAGENS

- Complexidade confiável de $O(n \log n)$
- Pouco uso de memória extra
- Implementação e controle de três algoritmos distintos
- Instável

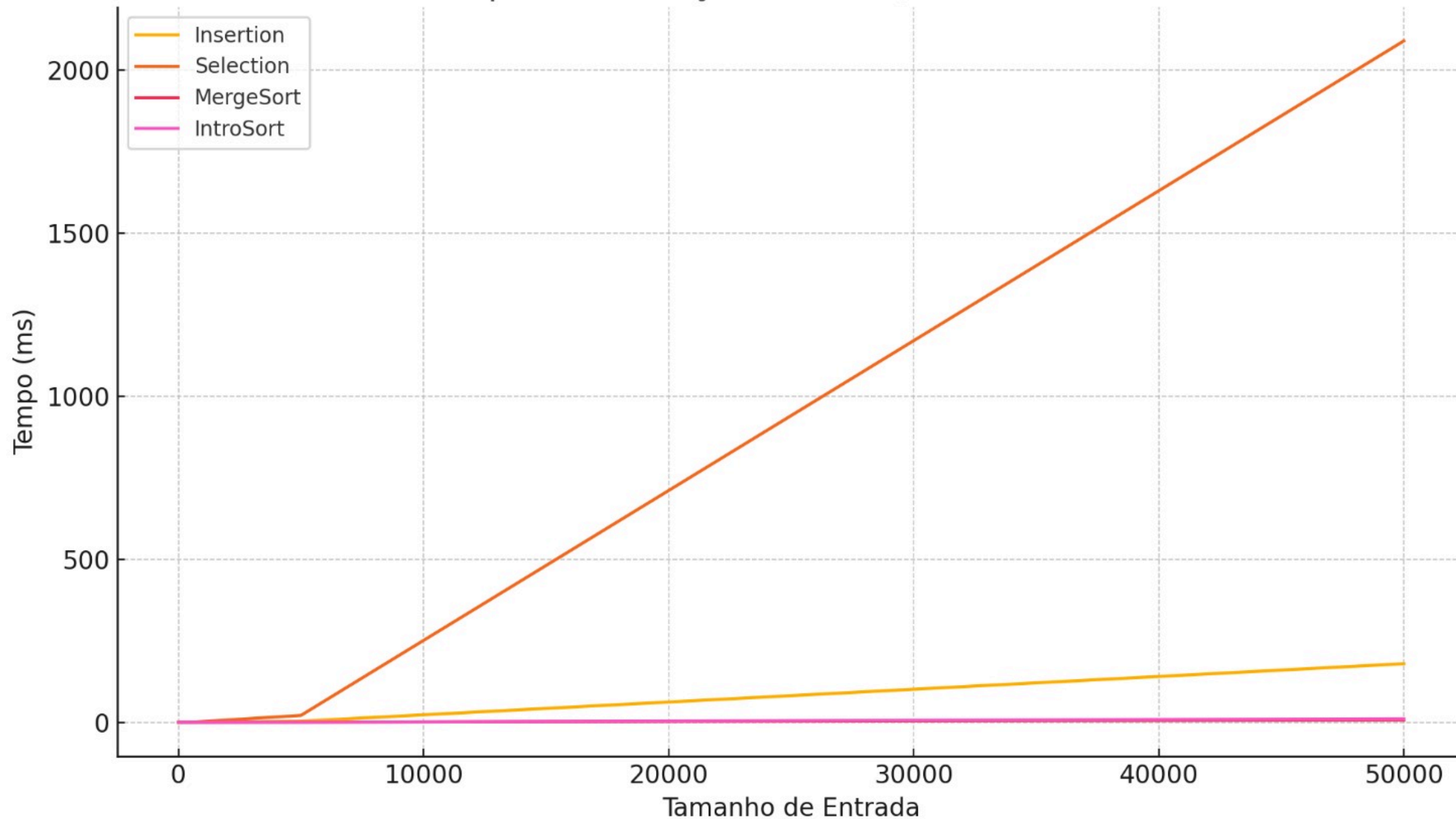


EXPECTATIVAS

CASOS	INSERTION	SELECTION	MERGESORT	QUICKSORT	INTROSORT
VETOR ALEATÓRIO	Pouca eficiência	Pouca eficiência	Alta eficiência	Alta eficiência	Alta eficiência
VETOR ORDENADO	Alta eficiência	Pouca eficiência	Pouca eficiência	Pouca eficiência	Alta eficiência
VETOR PARCIALMENTE ORDENADO	Média eficiência	Pouca eficiência	Média eficiência	Média eficiência	Alta eficiência
VETOR DECRESCENTE	Pouca eficiência	Pouca eficiência	Alta eficiência	Pouca eficiência	Alta eficiência

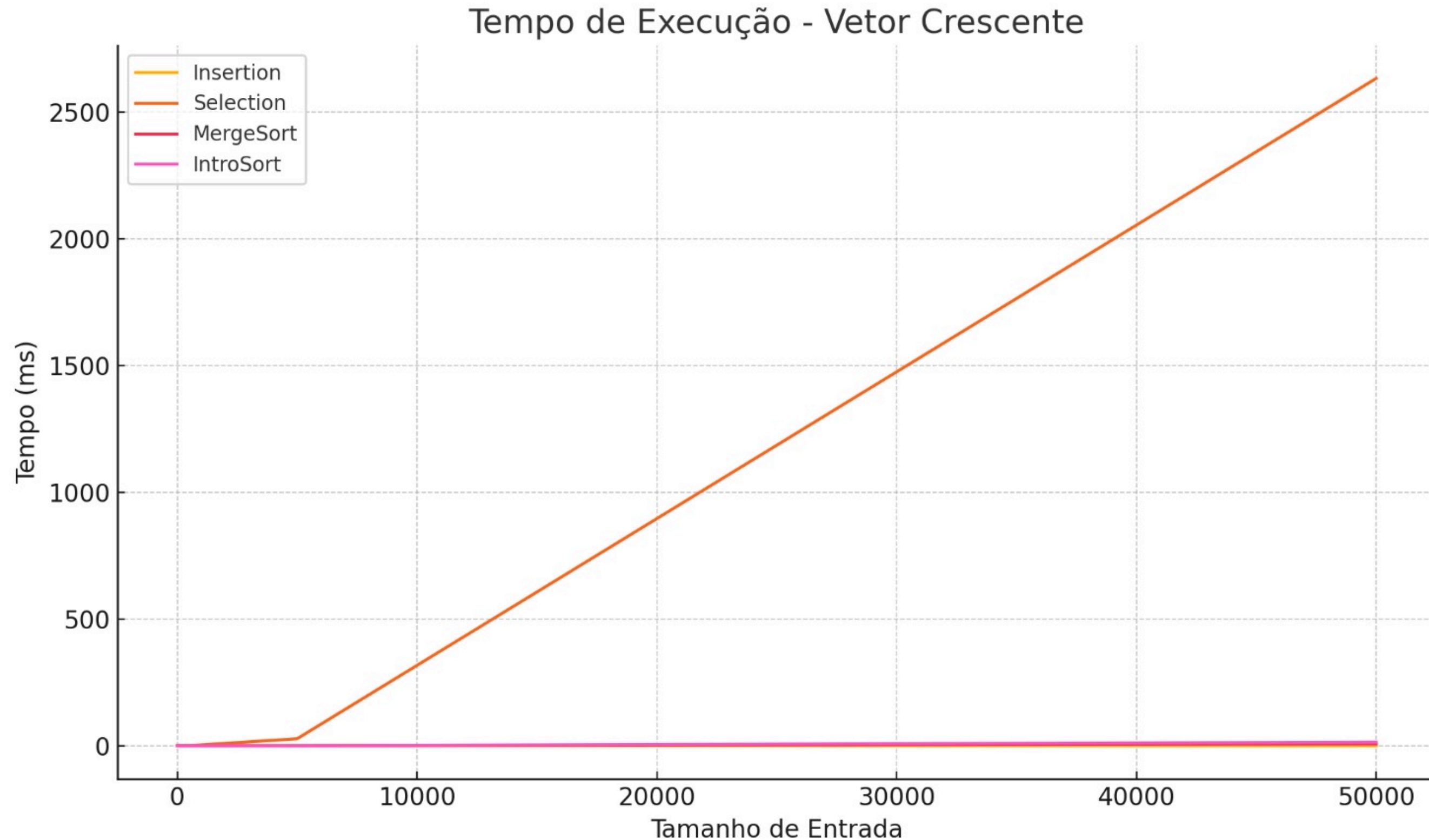
COMPARATIVO

Tempo de Execução - Vetor Quase Ordenado



- Insertion possui um tempo baixo (esperado, devido aos poucos elementos movidos)
- Selection sem sensibilidade à ordenação
- Merge e Intro muito rápidos

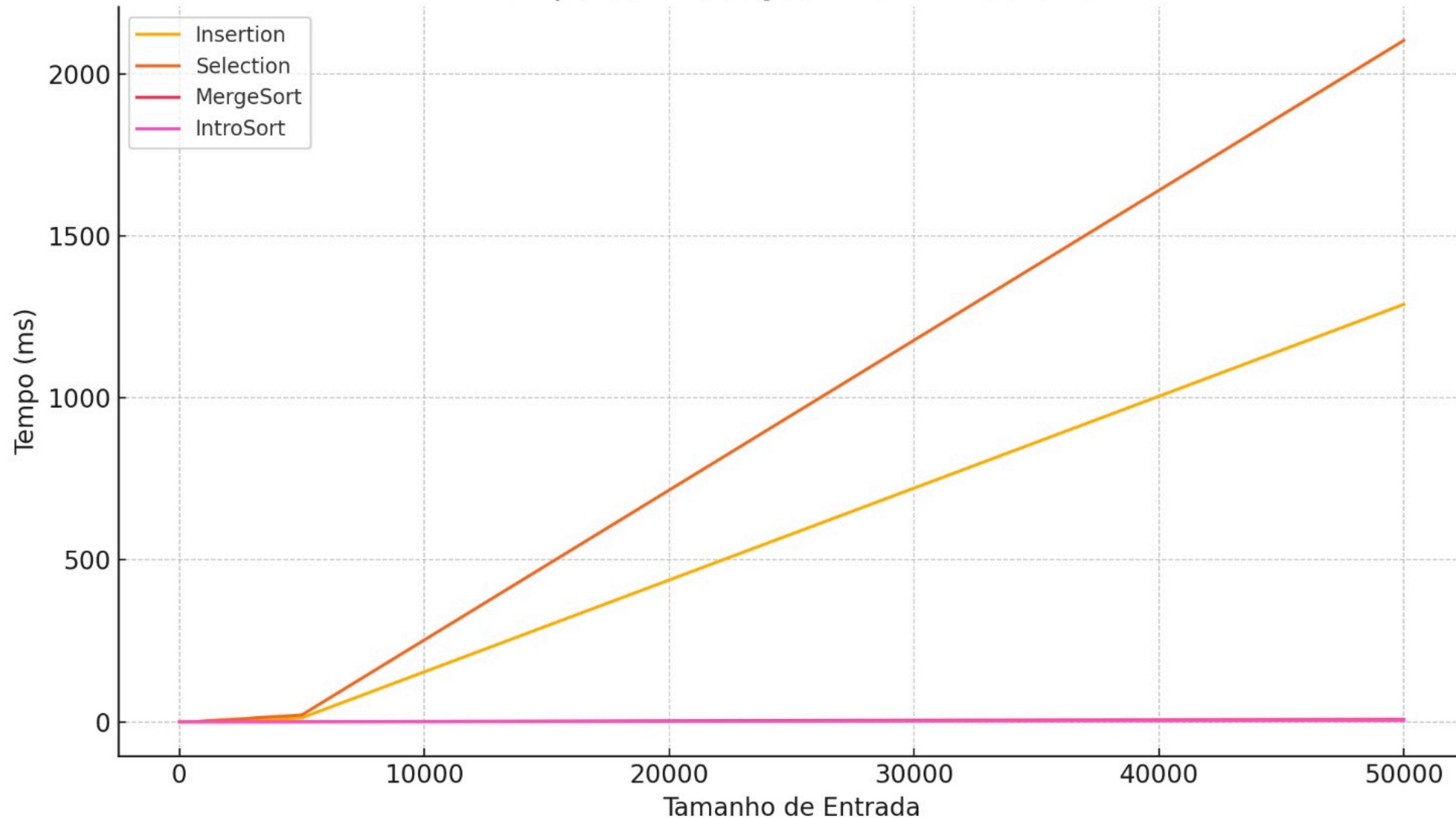
COMPARATIVO



- Insertion com tempo quase nulo (esperado devido ao seu melhor caso)
- Selection constantemente ineficiente
- Merge e Intro mantêm desempenho baixo e constante

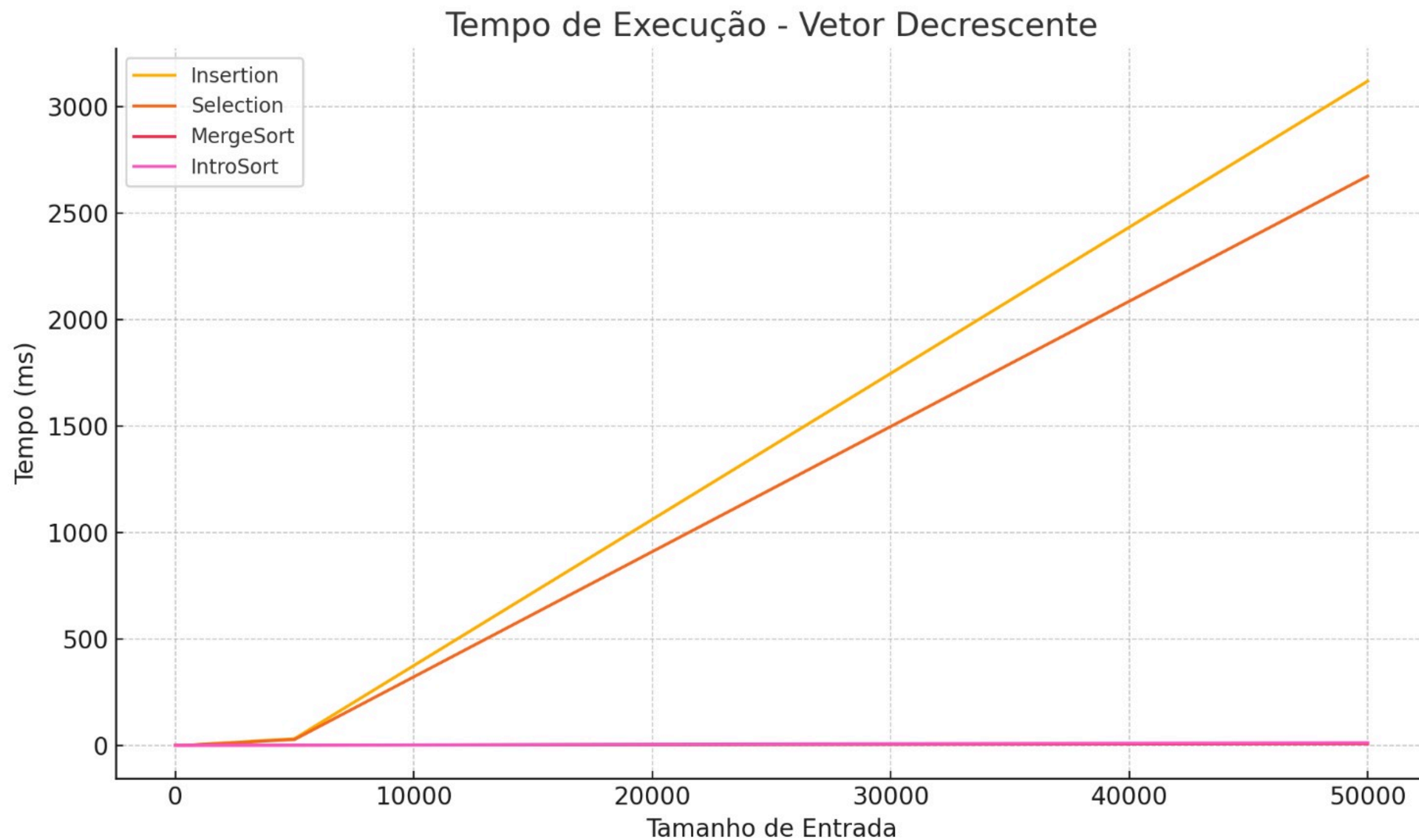
COMPARATIVO

Tempo de Execução - Vetor Aleatório



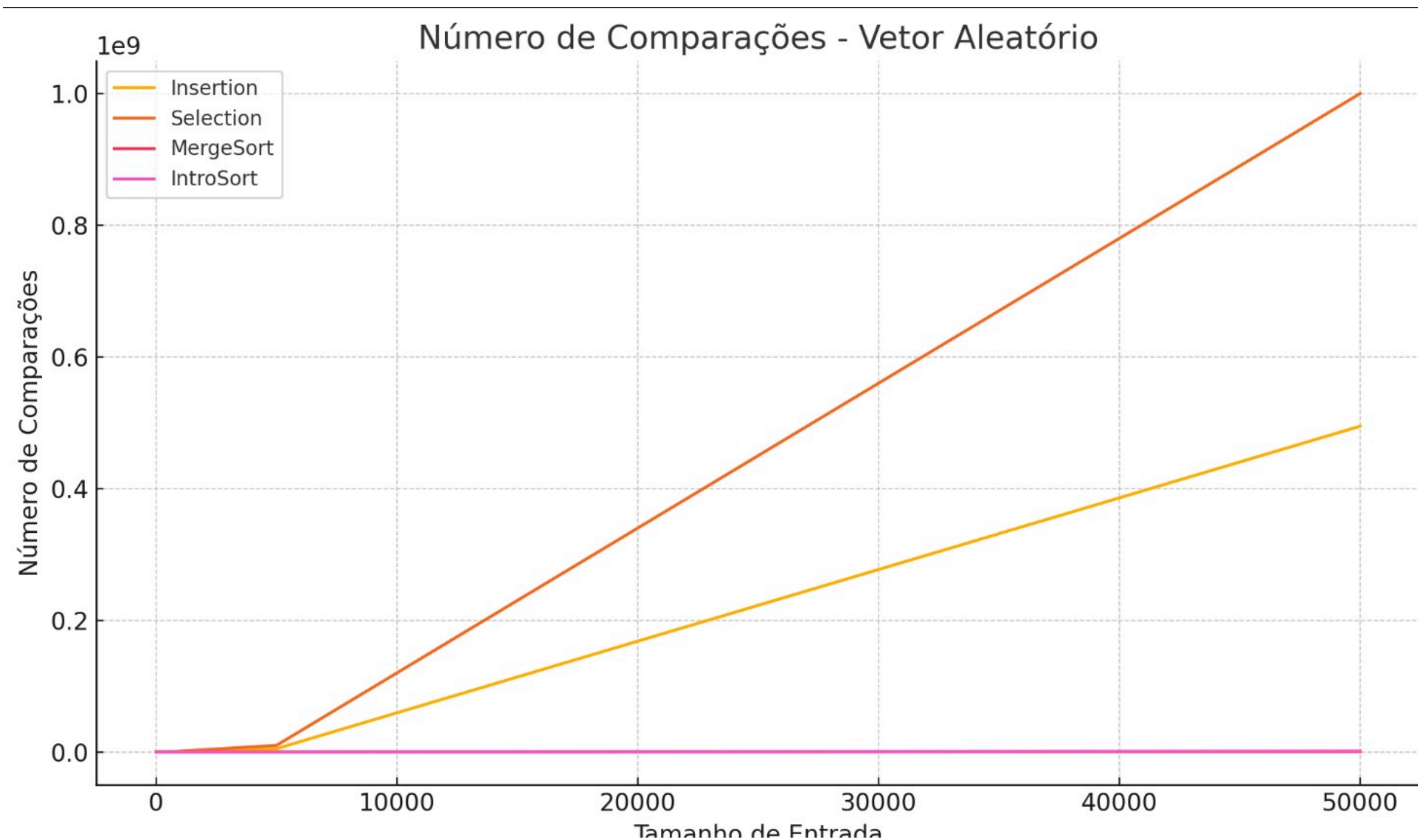
- Insertion possui um crescimento rápido (condizente com sua complexidade)
- Selection possui seu tempo ainda maior
- Merge e Intro crescem de maneira suave (Intro sendo ligeiramente mais rápido)

COMPARATIVO



- Insertion com tempo significativamente elevado (todos os elementos precisam ser movidos)
- Selection sem variação significativa (esperado)
- Merge e Intro ainda são rápidos, com um pequeno aumento de tempo)

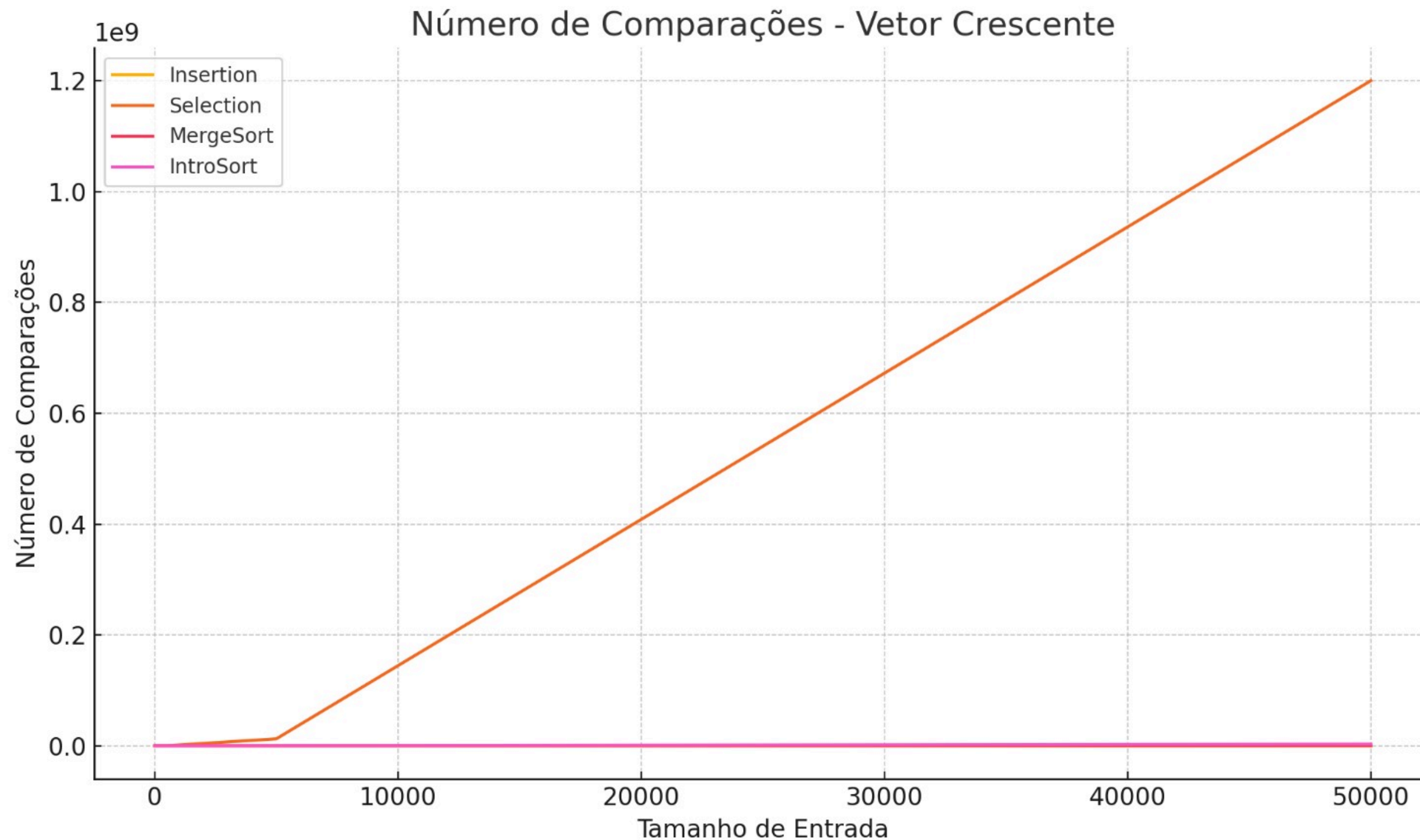
COMPARATIVO



■ Insertion e Selection fazem muitas comparações (esperado)

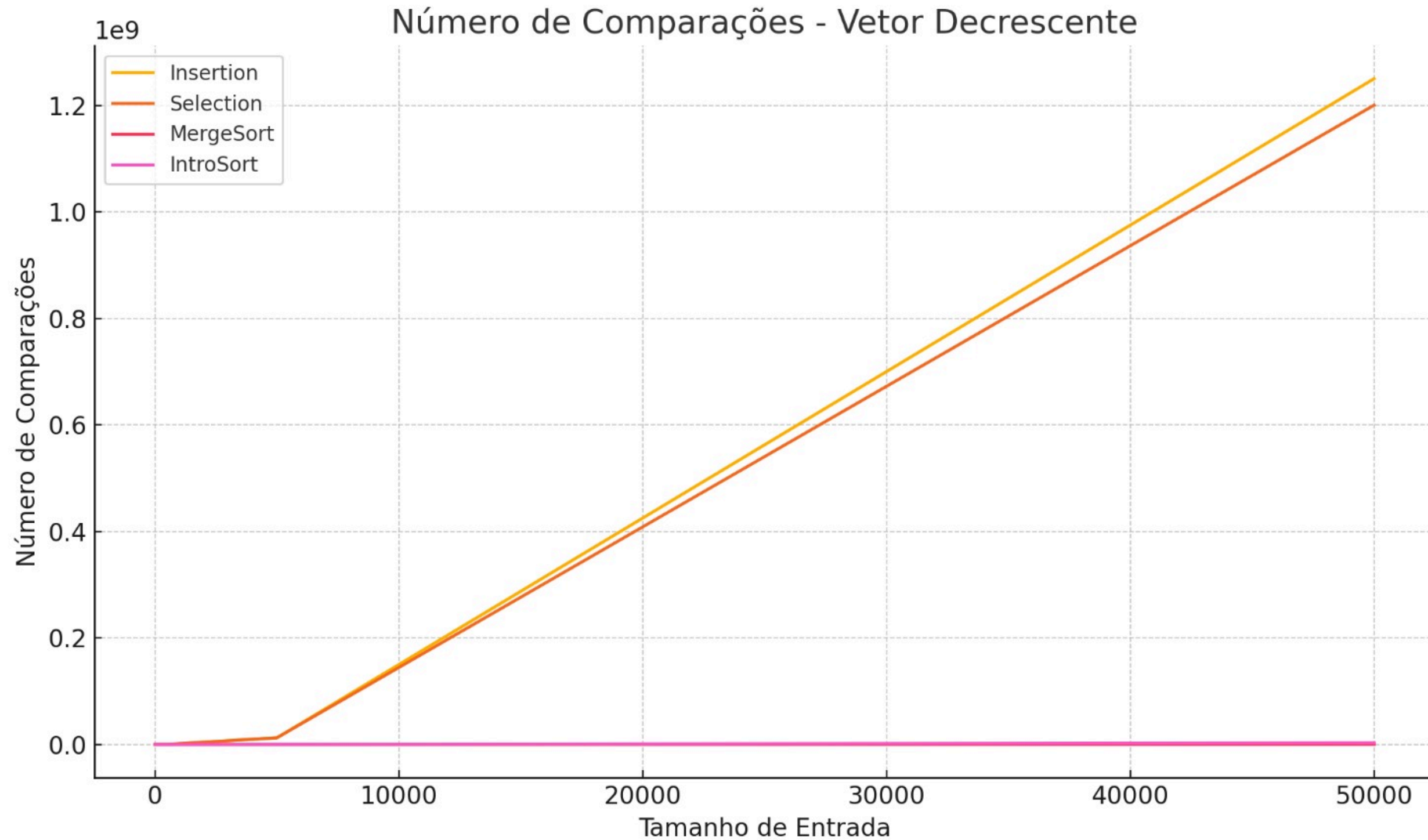
Intro oscila levemente mais que o Merge, mas permanece na faixa esperada

COMPARATIVO



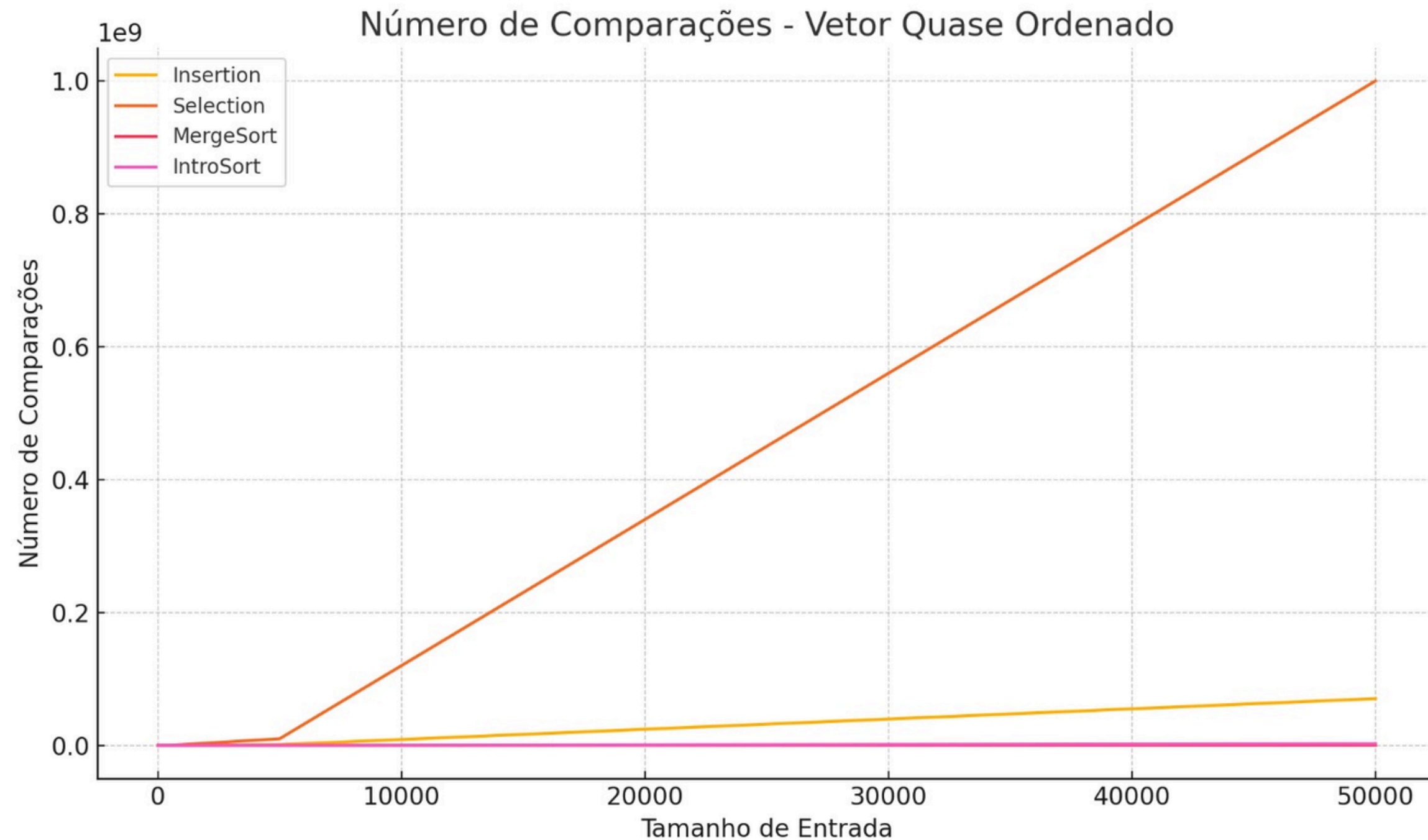
- Insertion faz menos comparações e o Selection mantém as suas comparações (esperado)
- Merge e Intro mantém seu padrão sem melhora perceptível

COMPARATIVO



- Pior caso do Insertion Sort, fazendo mais comparações do que o Selection (que mantém idêntico)
- Merge se mantém sem grandes alterações (esperado)
- Intro mantém um número razoável, indicando eficiência em vetores decrescentes

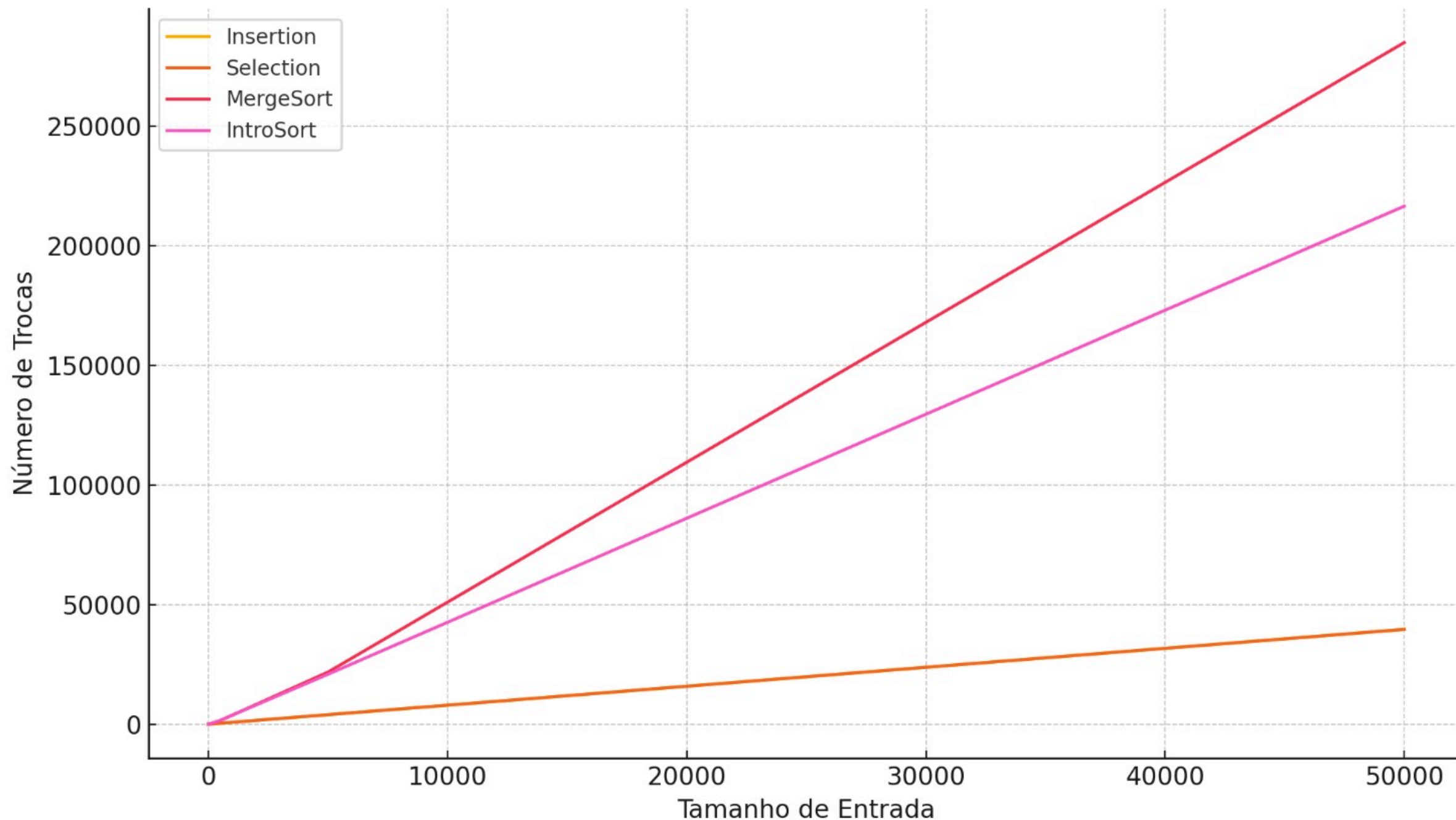
COMPARATIVO



- Insertion realiza poucas comparações (bom desempenho)
- Selection Inalterado
- Merge semelhante a outras entradas
- IntroSort possui bom desempenho (porém, é sensível a pequenos desarranjos)

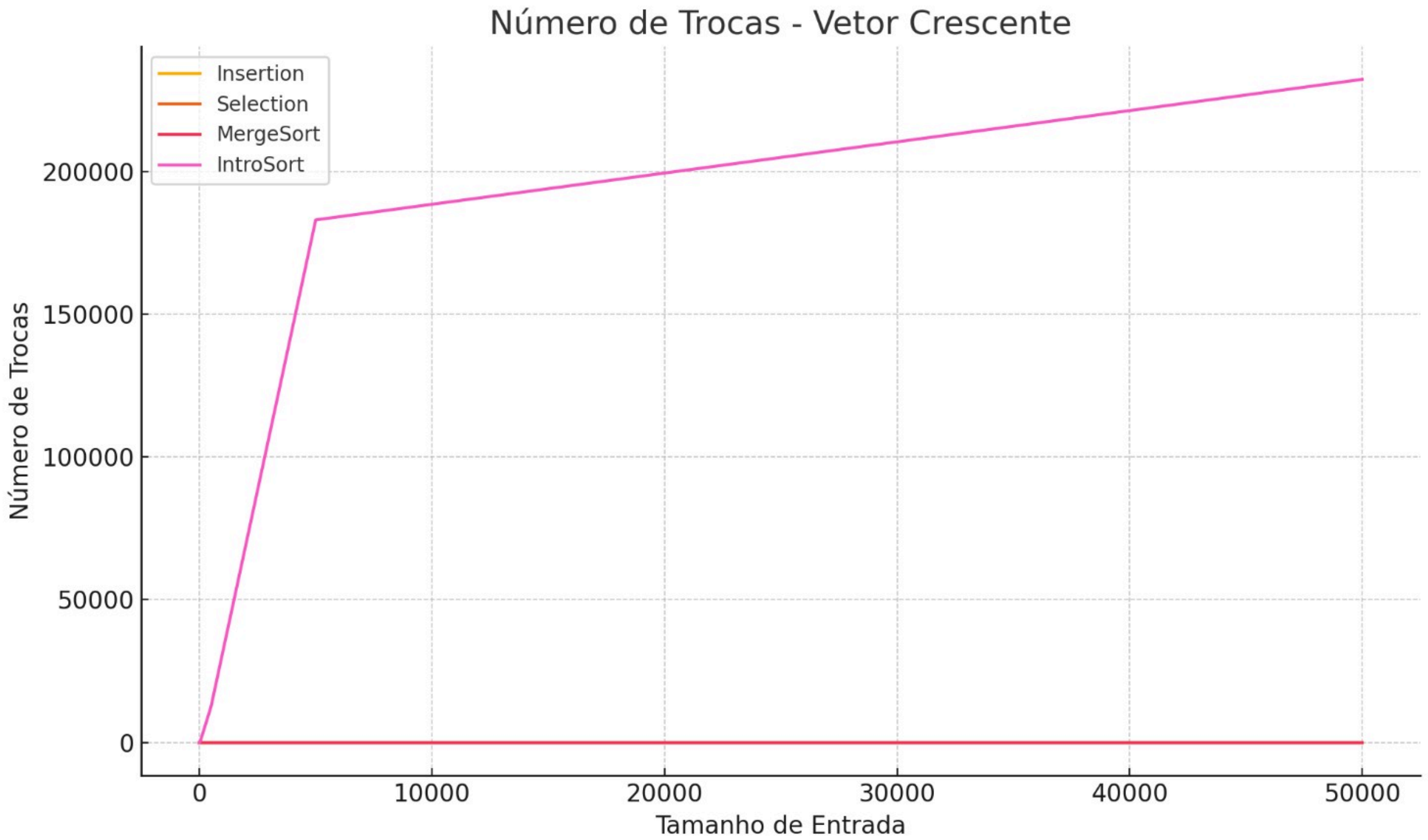
COMPARATIVO

Número de Trocas - Vetor Aleatório



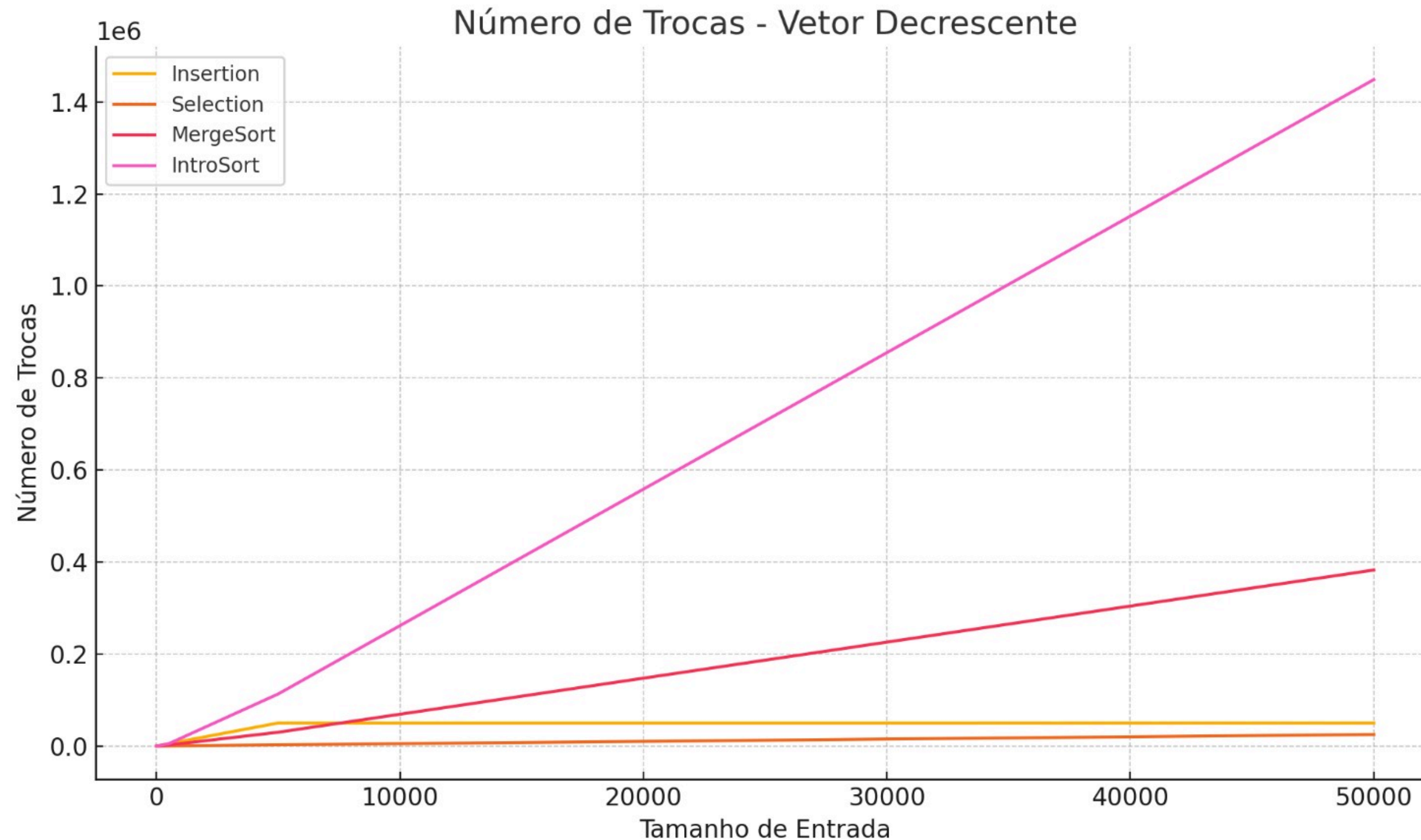
- Insertion e Selection realizam trocas significativas (mesmo com o princípio de ordenação do Selection)
- Merge faz muitas “trocas” (cópias) devido à criação de vetores auxiliares
- Intro realiza um volume razoável de trocas (menos que o Merge)

COMPARATIVO



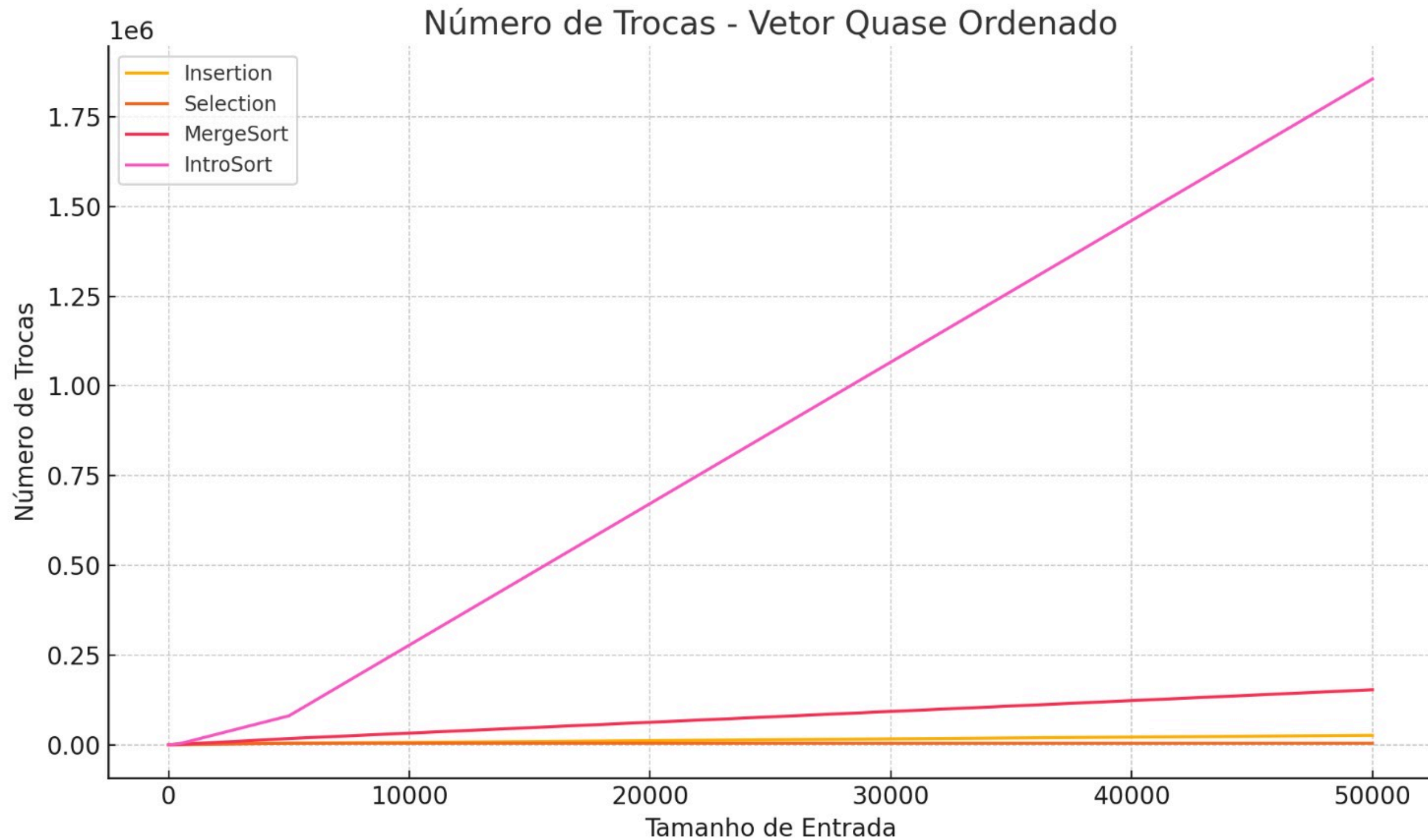
- Insertion e Selection realizam 0 trocas (ideal)
- Intro realiza trocas inesperadamente altas

COMPARATIVO



- Insertion realiza seu número máximo de trocas (todos elementos deslocados)
- Selection realiza uma por posição
- Merge e Intro fazem volumes significativos (esperado)

COMPARATIVO



- Insertion e Selection realizam poucas trocas
- Merge e Intro possuem um número elevado


COMPARATIVO

Algoritmo	Vantagens	Desvantagens	Ideal para...
Insertion	Simples e eficiente para vetores pequenos	Péssimo para vetores grandes ou decrescentes	Vetores pequenos ou já ordenados
Selection	Simples e poucas trocas	Sempre ineficiente	Situações didáticas
Merge	Estável, previsível e relativamente eficiente	Alto custo de memória	Grandes volumes que demandam estabilidade
Intro	Eficiência adaptativa e baixo tempo	Implementação complexa	Aplicações que exigem robustez e rapidez



CONCLUSÕES



- Bom desempenho na maioria dos casos
 - Combina número moderado de comparações com um número variável e eficiente de trocas;
 - Se adapta bem aos piores casos (utilizando o HeapSort como fallback);
 - Semelhante ao MergeSort ($O(n \log n)$), porém, utiliza menos memória auxiliar.
- 



**OBRIGADO
PELA
ATENÇÃO**