Faculdade Boa Viagem

Graduação em Ciência da Computação

Projeto Análise e Monitoramento de Desempenho

Comparativo entre os algoritmos de ordenação

de listas das linguagens Java e Python

Carlos Romano Pereira da Silva

Área de Concentração

Análise de Desempenho de Sistemas

Professor

Thiago Rodrigues

Recife, Junho/2018.

**1 INTRODUÇÃO**

Com o surgimento de diversas linguagens de programação de alto nível nas últimas décadas, muitas dúvidas são formadas acerca de qual linguagem utilizar para uma dada situação, com o propósito de se obter software e algoritmos eficientes e robustos para atender às necessidades que a tecnologia da informação busca suprir.

Pensando nisto, este trabalho tem por objetivo principal identificar qual o algoritmo de ordenação de lista mais eficiente, em termo de tempo gasto para executar a ordenação. As linguagens de programação utilizadas para realizar o experimento serão as linguagens Java e Python.

Os resultados obtidos pelo experimento em cada linguagem, separadamente, serão comparados entre si, com o objetivo de auxiliar na identificação sobre qual linguagem apresenta melhores resultados para a finalidade em questão.

A proposta é utilizar os recursos nativos das linguagens, e comparar os resultados obtidos, bem como implementar a estrutura de dados lista, em cada uma das linguagens e, também coletar dados através de simulações e comparar os resultados entre as duas versões implementadas, a implementação na versão Java e na versão Python.

**2 METODOLOGIA**

O experimento será realizado sob duas perspectivas. Sob a perspectiva dos recursos nativos das linguagens, no caso do Java será utilizada a estrutura de dados ArrayList e em Python será utilizada a estrutura de lista disponível nativamente na linguagem. A segunda perspectiva será sobre a implementação das estruturas de dados de lista em ambas as linguagens. O tipo de dado com o qual as listas serão populadas, é o tipo Double.

Serão realizadas 14 coletas (ordenações) para cada quantidade de elementos (as quantidades podem ser vistas na seção de 5.2) na lista. Após coletado os resultados, serão removidos os dois maiores tempos e os dois menores, mantendo 10 registros de tempo gasto pelo algoritmo, a partir disso, é obtido a média.

**3 CENÁRIO**

**3.1 CENÁRIO 1**

Utilizando a implementação nativa da estrutura de dados ArrayList e o algoritmo de ordenação nativo da linguagem Java, a lista é instanciada e populada, variando o número de elementos para cada nível do experimento.

**3.2 CENÁRIO 2**

Para este cenário o autor implementou uma estrutura de dados do tipo lista, bem como um algoritmo de ordenação utilizando a linguagem Java. A lista é instanciada e populada, variando o número de elementos para cada nível do experimento.

**3.3 CENÁRIO 3**

Utilizando a implementação nativa da estrutura de dados array e o algoritmo de ordenação nativo da linguagem Python, a lista é instanciada e populada, variando o número de elementos para cada nível do experimento.

**3.4 CENÁRIO 4**

Para este cenário o autor implementou uma estrutura de dados do tipo lista, bem como um algoritmo de ordenação utilizando a linguagem Python. A lista é instanciada e populada, variando o número de elementos para cada nível do experimento.

**4 TEST BED**

O computador utilizado para o experimento, possui:

* Arquitetura 64BITs;
* Sistema operacional Windows 10;
* Processador Intel core i7 7700HQ de 3.8GHz (frequência máxima), com 6MB de cache;
* Memória RAM de 32GB;
* SSD 480GB;

**5 FATORES, NÍVEIS e MÉTRICAS**

**5.1 FATORES**

* Linguagem de programação Java;
* Linguagem de programação Python;
* Estrutura de dados Array;
* Algoritmo de ordenação Bubble Sort;
* Algoritmo de ordenação TimSort;

Para a ordenação da lista nativa da linguagem Java e Python, o algoritmo de ordenação nativo das linguagens é o TimSort (ORACLE, 2018).

O algoritmo de ordenação implementado pelo autor para a realização dos experimentos foi o Bubble Sort.

**5.2 NÍVEIS**

As coletas são realizadas seguindo os seguintes níveis, para cada tipo de lista:

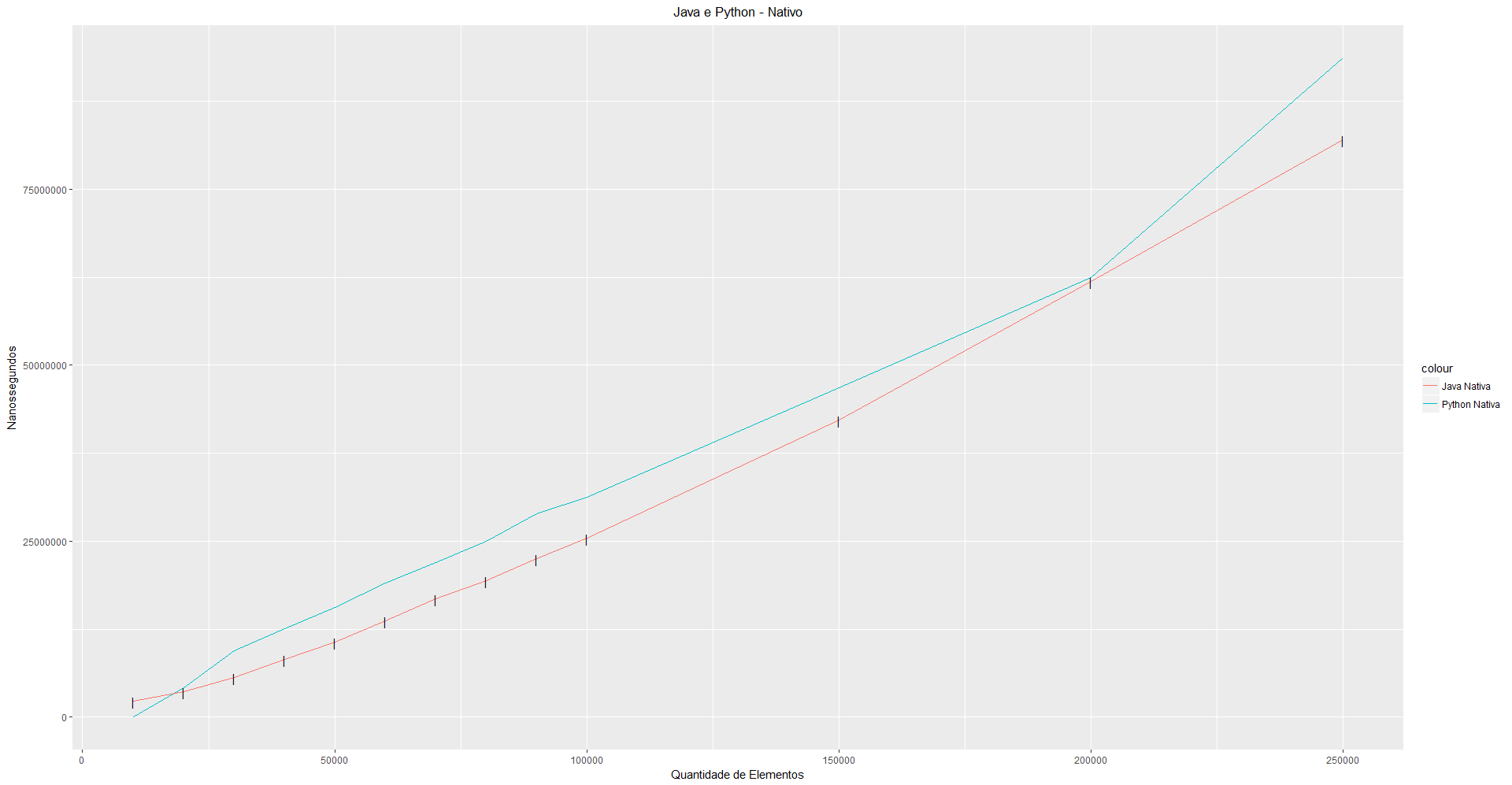
|  |  |
| --- | --- |
| Quantidade Elementos na Lista | Quantidade de Coletas |
| 10.000 | 14 |
| 20.000 | 14 |
| 30.000 | 14 |
| 40.000 | 14 |
| 50.000 | 14 |
| 60.000 | 14 |
| 70.000 | 14 |
| 80.000 | 14 |
| 90.000 | 14 |
| 100.000 | 14 |
| 150.000 | 14 |
| 200.000 | 14 |
| 250.000 | 14 |

**3.3 MÉTRICAS**

A métrica utilizada é o tempo na unidade de medida de tempo nanosegundos. Assim que a ordenação é iniciada é coletado o tempo em nanosegundos e assim que o algoritmo de ordenação é finalizado o tempo é coletado novamente. A diferença entre esses dois valores obtidos (início e fim) é tomado como o tempo gasto pelo algoritmo para realizar a ordenação.

**4 RESULTADOS**

A imagem 1, demonstra a comparação entre os resultados obtidos pela execução dos algoritmos de ordenação nativo das linguagens Java e Python, com base na métrica tempo.



Figura

Os resultados exibidos na figura 1, foram coletados conforme descrito nas seções 2 e 5.2. A linha horizontal marca o número de elementos com os quais as listas foram populadas, que inicia em 10.000 indo até 250.000 elementos.

A figura 2, demonstra a comparação entre os resultados obtidos pela execução dos algoritmos de ordenação implementados pelo autor nas linguagens Java e Python, com base na métrica tempo.

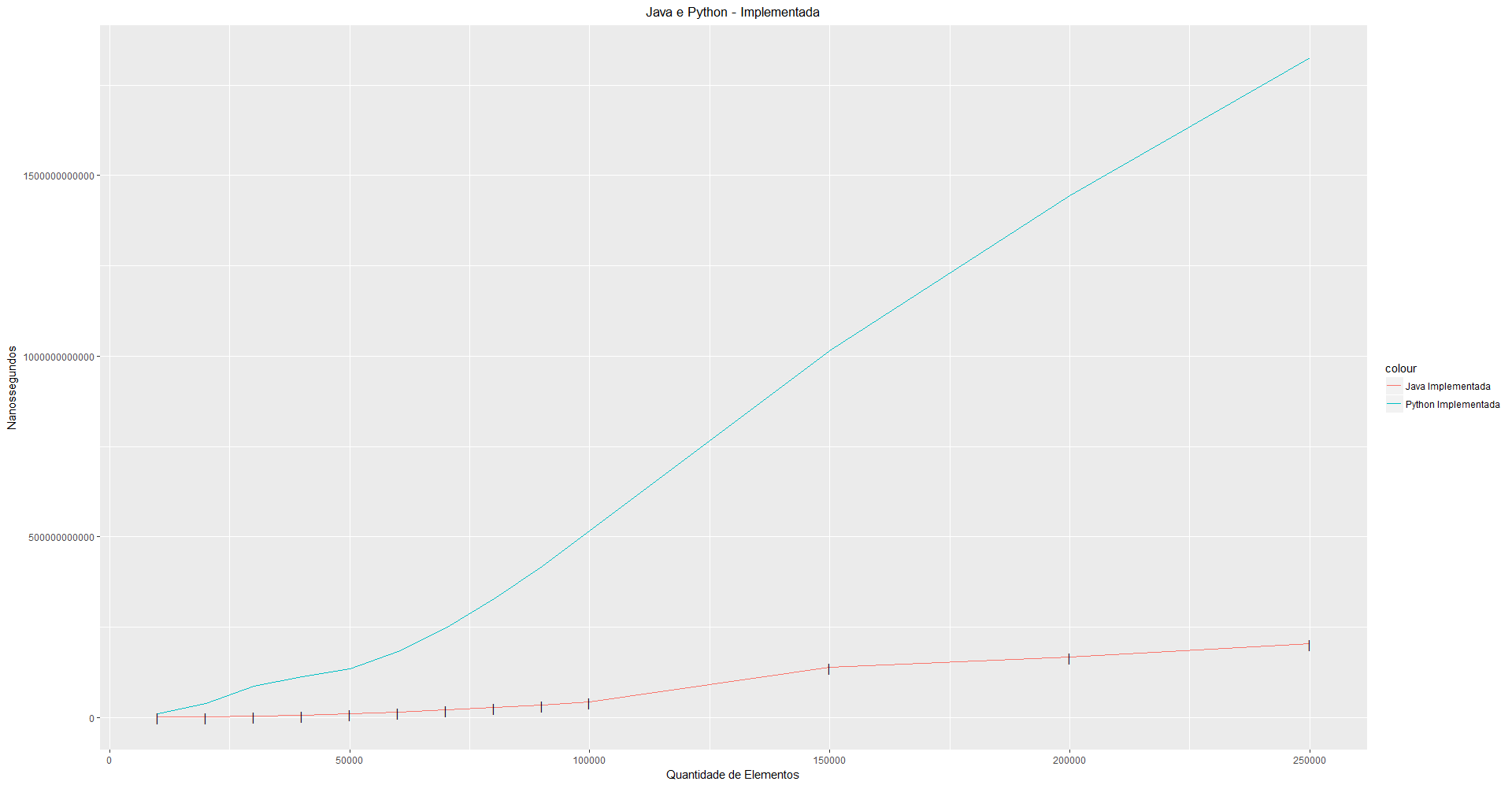
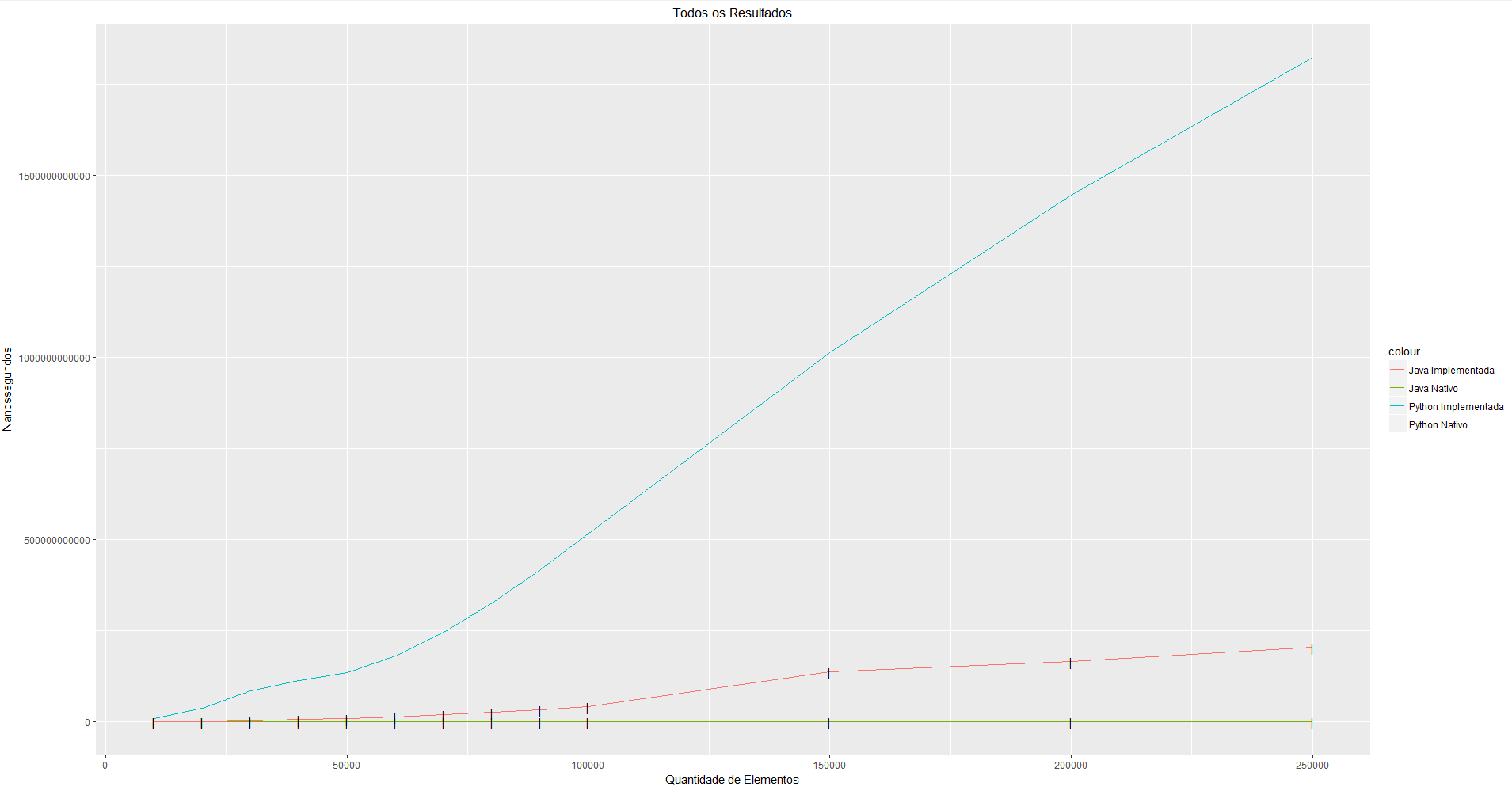


Figura 2

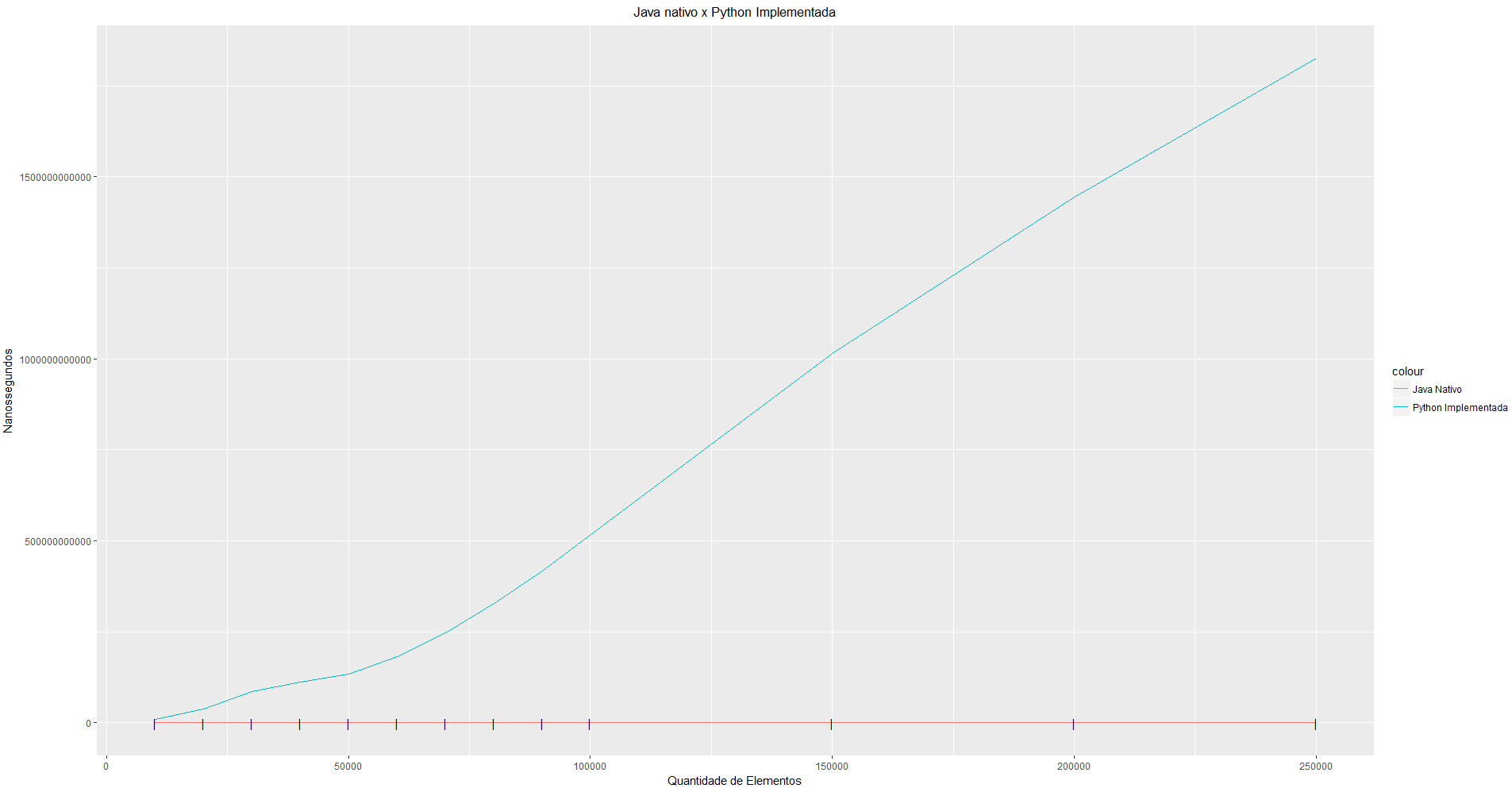
Os resultados exibidos na figura 2, foram coletados conforme descrito nas seções 2 e 5.2. A linha horizontal marca o número de elementos com os quais as listas foram populadas, que inicia em 10.000 indo até 250.000 elementos.

A figura 3 apresenta o comparativo entre todos os resultados obtidos. A comparação é feita entre a versão implementada do Java, versão implementada do Python, versão nativa do Python e versão nativa do Java.



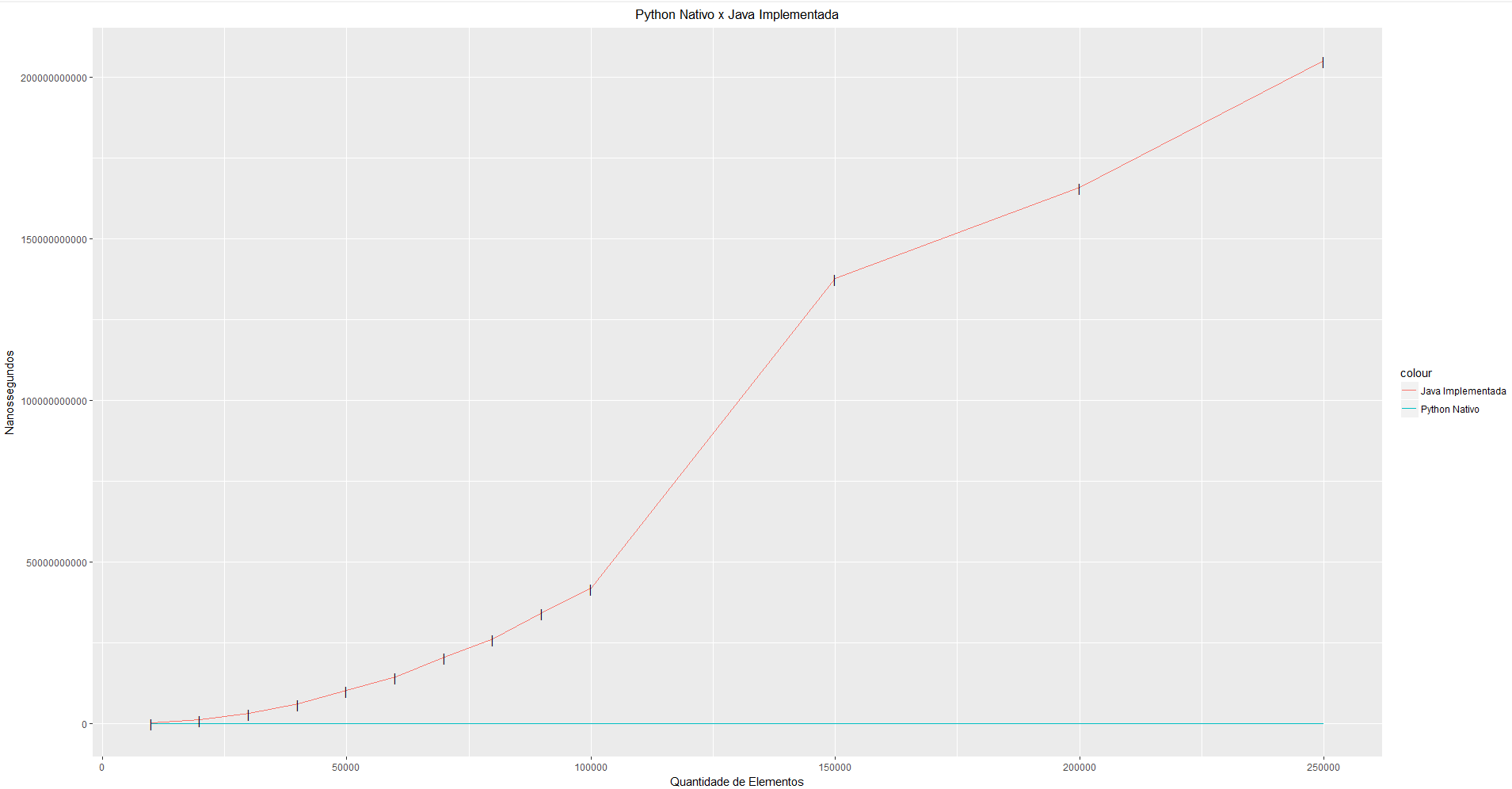
Figura

A figura 4 apresentada o resultado entre a implementação realizada pelo autor na linguagem Python e o resultado obtido da versão Java nativo.



Figura

A figura 5 apresenta a comparação entre os resultados das versões Java implementada e Python nativo.



Figura

**5 DISCUSSÃO**

Como pode ser visto na seção de resultados, no experimento entre a estrutura de dados e o algoritmo de ordenação nativos, as linguagens alternam a posição com relação ao tempo gasto, podemos ver que até os 20.000 elementos, Python se mantém com um menor tempo gasto para a ordenação.

Vale lembrar que a implementação do algoritmo de ordenação é o mesmo para as duas linguagens, o TimSort, conforme mencionado no artigo referenciado da Oracle Java docs.

Em relação ao experimento com a estrutura de dados e o algoritmo de ordenação implementados pelo autor, o Java demostrou, disparadamente, um melhor desempenho quanto ao tempo total gasto para ordenação em todos os níveis estabelecidos.

Assim como o algoritmo de ordenação nativo das linguagens, o algoritmo de ordenação implementado também são comuns entre as linguagens, o Bubble Sort.

**6 CONCLUSÃO**

Com base nos resultados obtidos, pode-se afirmar que a implementação do TimSort que apresenta melhor performance, vai depender da quantidade de elementos que serão ordenados.

Visto que, até cerca de 18.000 elementos, a implementação da linguagem Python apresenta uma melhor performance, porém, ao ultrapassar a quantidade de 20.000 elementos na lista, em relação à implementação Java, pode-se observar que o Python fica para trás.

Sendo assim, a melhor opção para a ordenação de uma lista de elementos do tipo Double, com mais 20.000 elementos, será o Java, demonstrando um crescimento mais linear que o Python, o qual não mantém um crescimento constante ao passo que a quantidade de elementos s adicionados à lista de forma incremental.

**7 REFERÊNCIAS**

ORACLE, [Documentação oracle linguagem de programação Java](https://docs.oracle.com/javase/8/docs/api/java/util/List.html#sort-java.util.Comparator-). Disponível em: <https://docs.oracle.com/javase/8/docs/api/java/util/List.html#sort-java.util.Comparator-/> Acesso em: 14 de junho de 2018