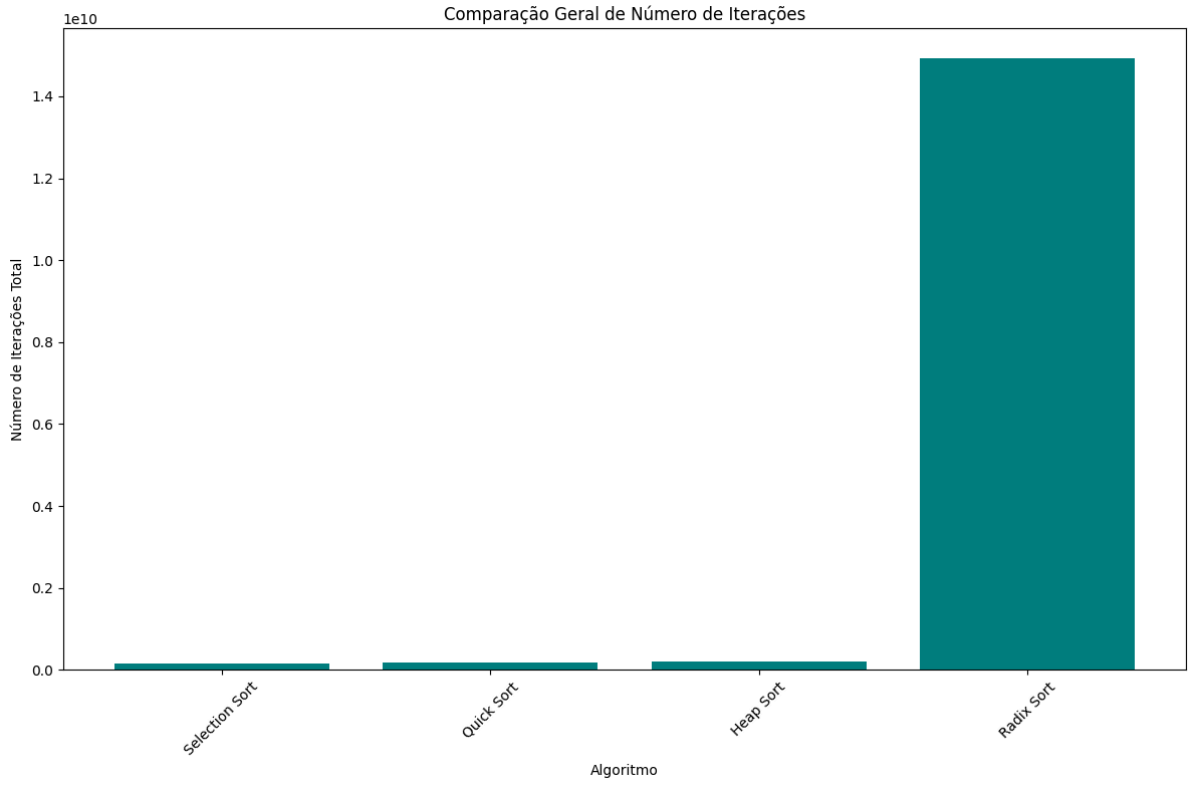
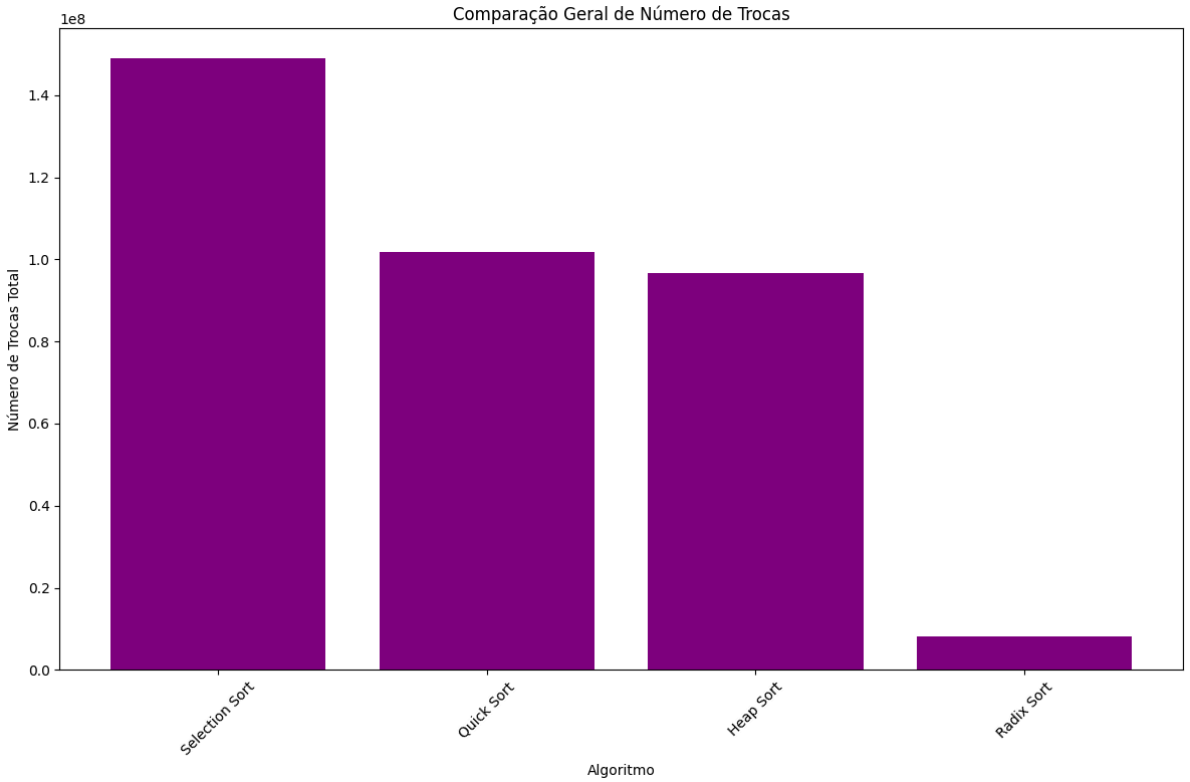
Relatório RA4

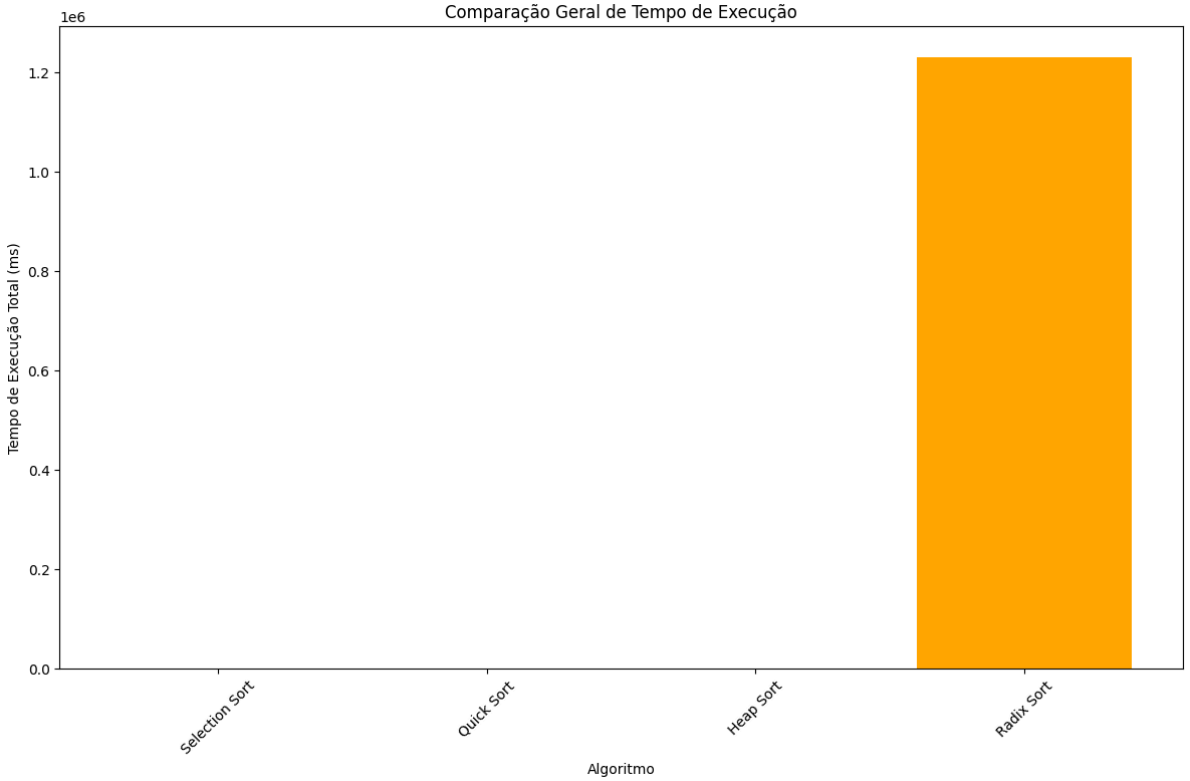
O gráfico apresentado demonstra de forma clara a disparidade na eficiência dos algoritmos de ordenação analisados. O Radix Sort se destaca significativamente, exigindo um número de iterações drasticamente inferior aos demais algoritmos. Essa superioridade pode ser atribuída à sua complexidade algorítmica, que o torna particularmente eficiente para conjuntos de dados com um número limitado de dígitos distintos. Em contrapartida, o Selection Sort apresenta o pior desempenho, evidenciando sua natureza ineficiente para grandes conjuntos de dados. O Quick Sort e o Heap Sort, embora apresentem um desempenho intermediário, demonstram a importância de escolher o algoritmo adequado com base nas características específicas do conjunto de dados a ser ordenado.



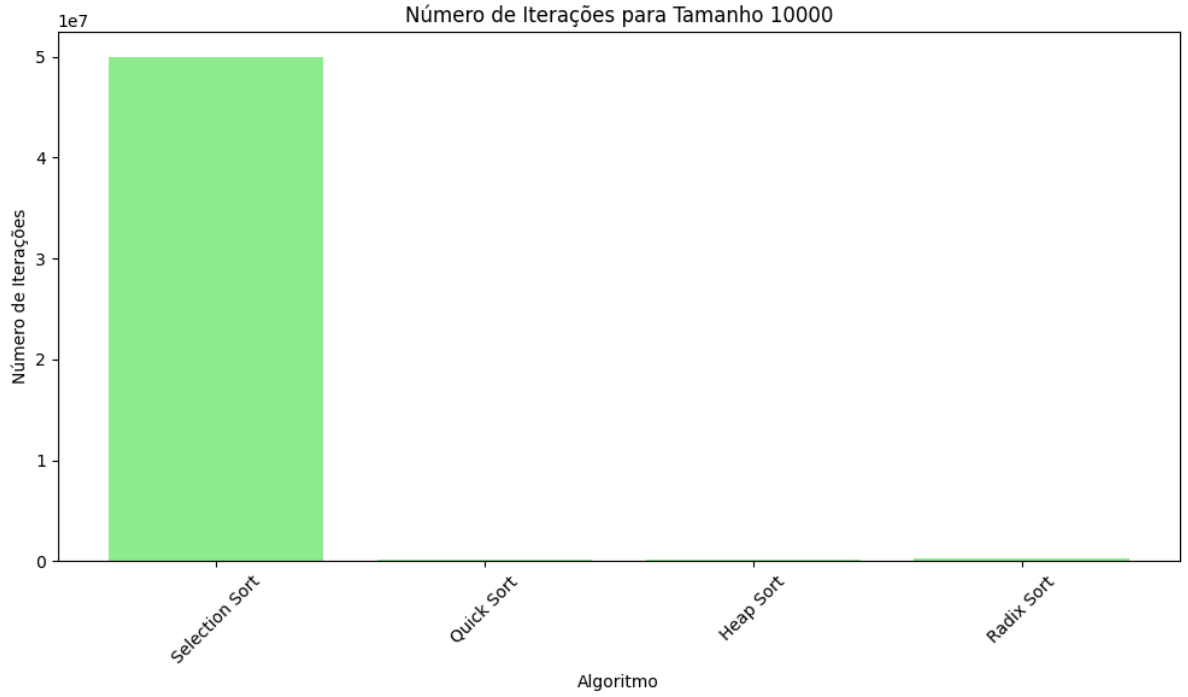
O gráfico evidencia uma disparidade significativa no número de trocas realizadas pelos algoritmos de ordenação analisados. O Selection Sort se destaca como o algoritmo que mais executa trocas, seguido pelo Quick Sort e Heap Sort. Em contraste, o Radix Sort apresenta um número de trocas consideravelmente menor. Essa diferença em desempenho pode ser atribuída à lógica de funcionamento de cada algoritmo, à complexidade algorítmica e às características dos dados de entrada. O Radix Sort, por explorar a representação digital dos números, demonstra uma eficiência superior em termos de número de trocas. Essa informação é crucial para a escolha do algoritmo mais adequado em diferentes cenários, levando em consideração fatores como o tamanho do conjunto de dados e a natureza dos dados.



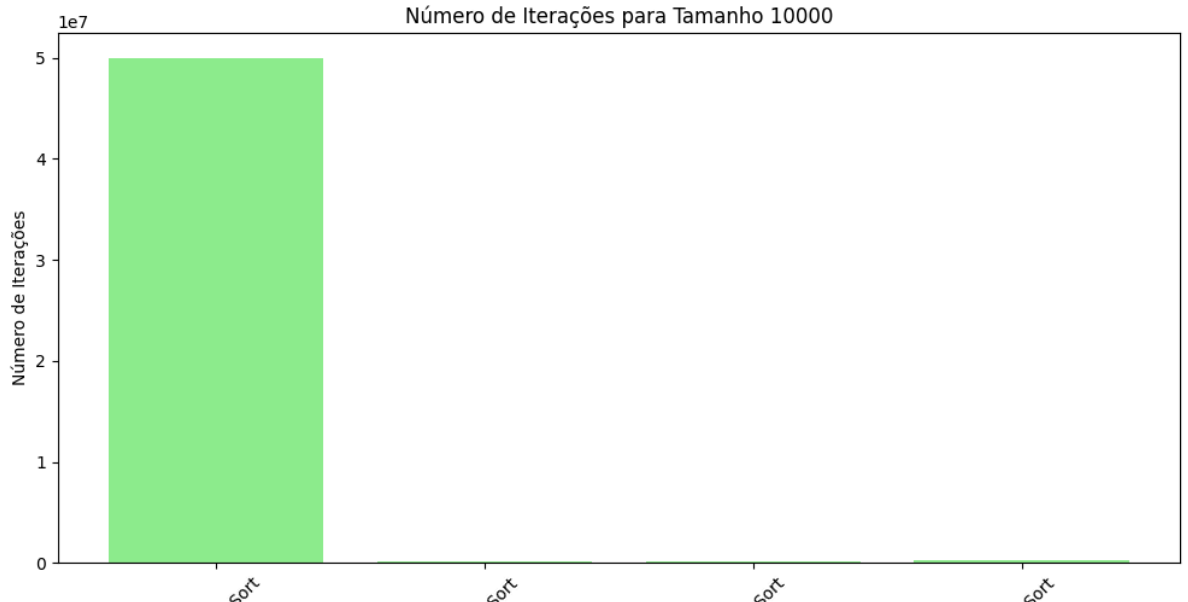
O gráfico apresentado evidencia uma disparidade significativa no tempo de execução dos algoritmos de ordenação analisados. O Radix Sort se destaca como o algoritmo com o tempo de execução mais elevado, seguido por uma diferença considerável dos demais algoritmos, que apresentam tempos de execução muito próximos de zero no gráfico.

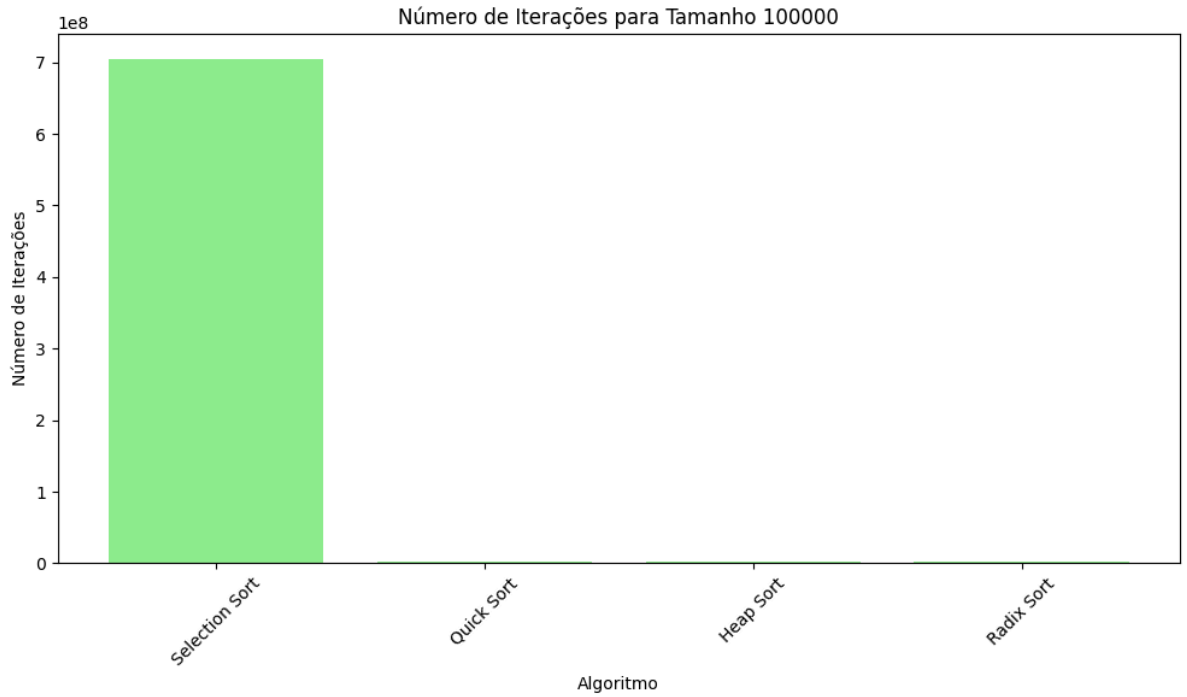


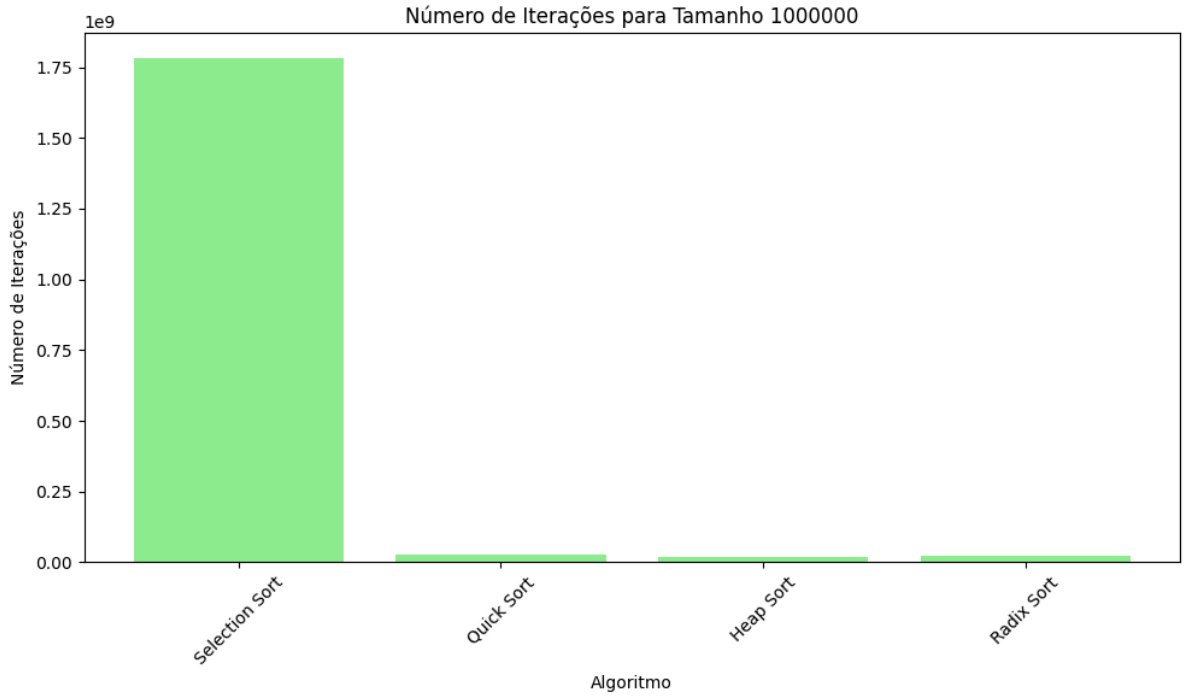
O gráfico evidencia uma clara superioridade do Radix Sort em relação aos demais algoritmos de ordenação quando o conjunto de dados possui 1000 elementos. O Selection Sort, como esperado, apresenta o maior número de iterações, refletindo sua natureza ineficiente. Os algoritmos Quick Sort e Heap Sort, embora apresentem um desempenho significativamente melhor que o Selection Sort, são superados pelo Radix Sort nesse cenário específico. A eficiência do Radix Sort para conjuntos de dados menores pode ser atribuída à sua complexidade algorítmica e à forma como ele explora a representação digital dos números. No entanto, é importante ressaltar que o desempenho de um algoritmo pode variar dependendo de diversos fatores, como a implementação, as características dos dados e o tamanho do conjunto de dados.

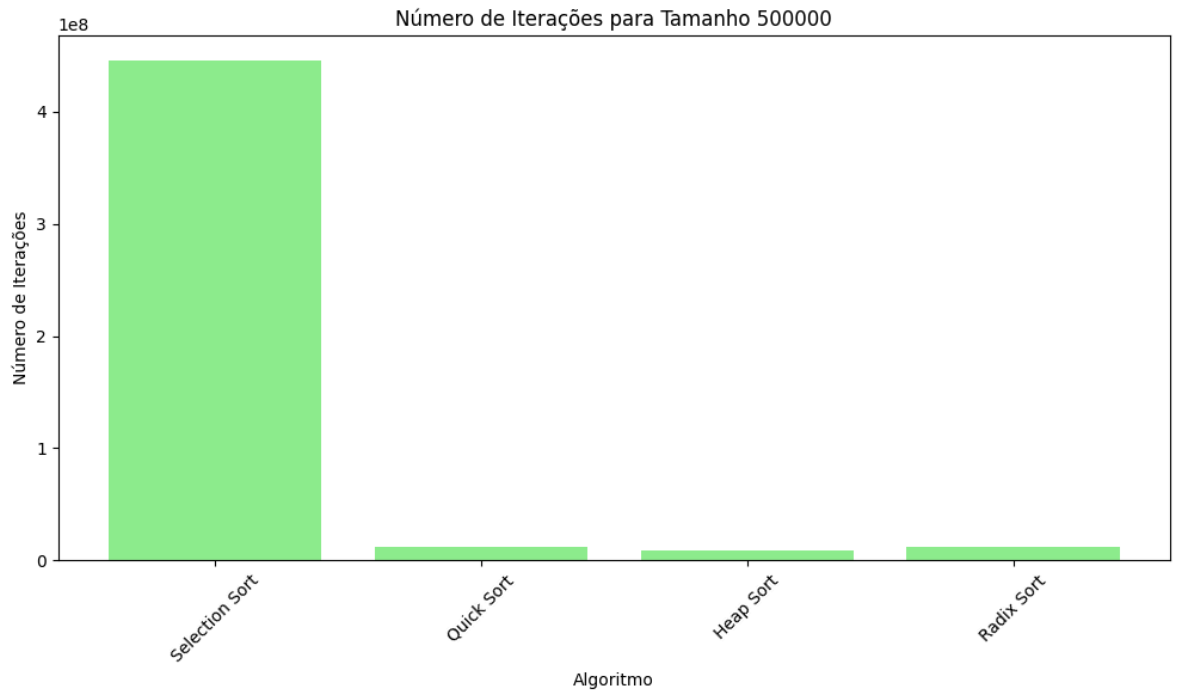


Os gráficos apresentados demonstram de forma clara a disparidade no número de iterações realizadas pelos algoritmos de ordenação ao variarem os tamanhos dos conjuntos de dados.

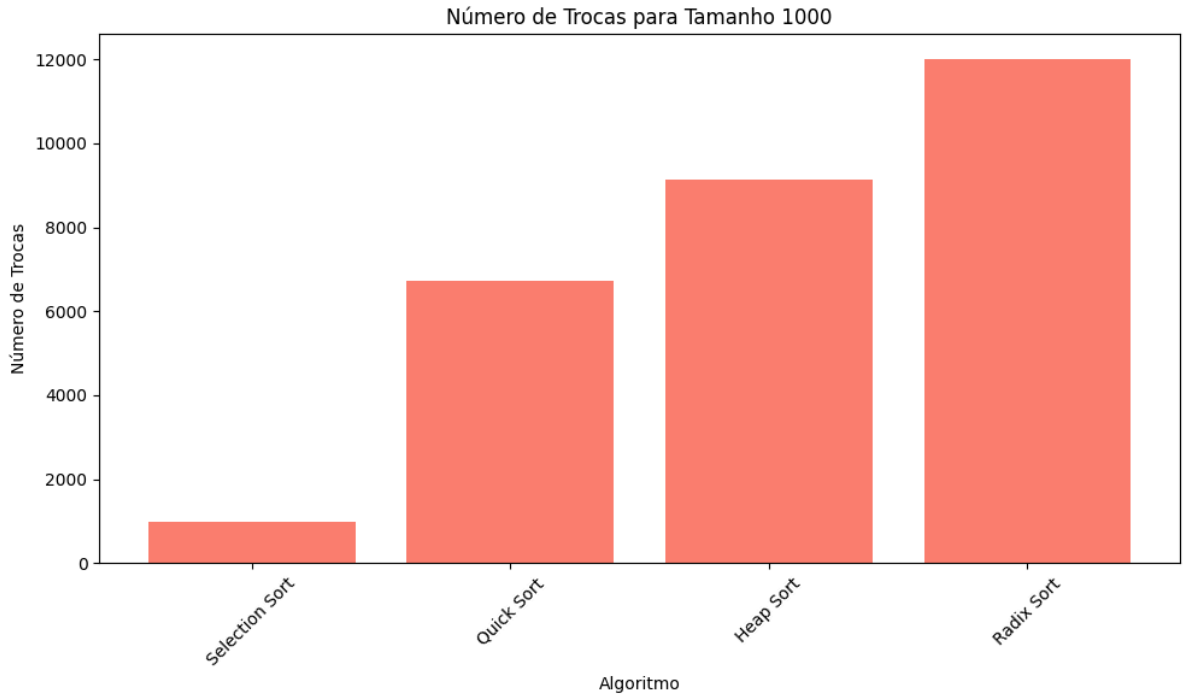


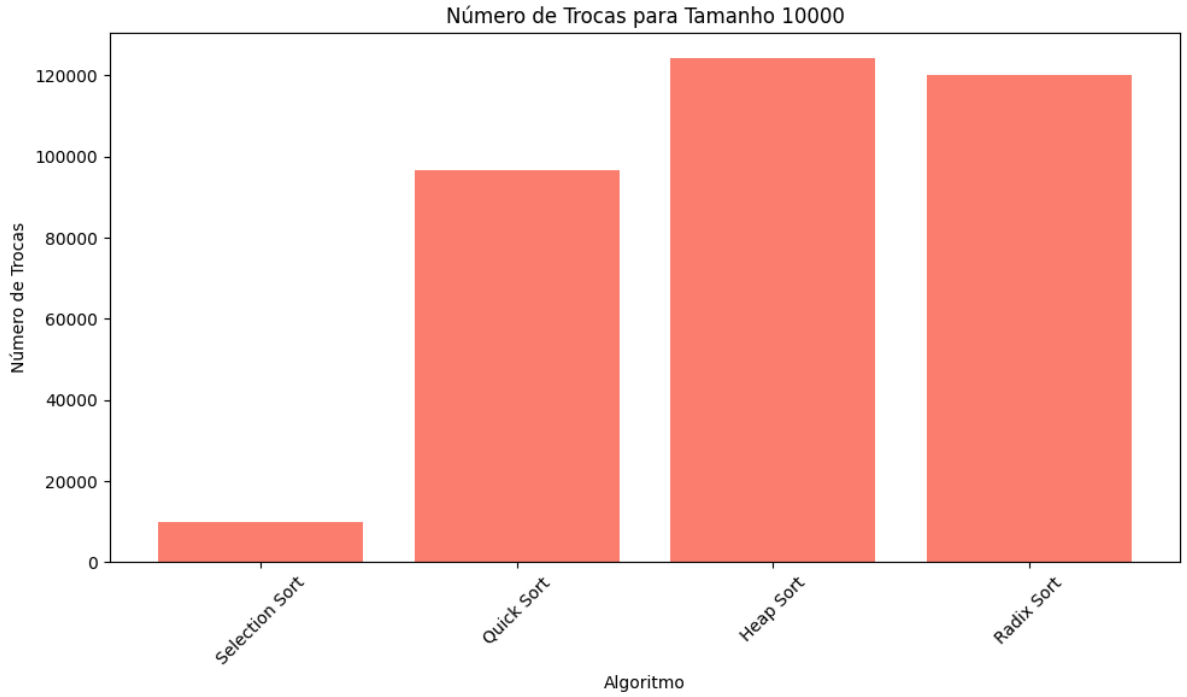

Número de Iterações para Tamanho 100000
1e8
7
6
5
4
3
2
1
0
Número de Iterações
Selection Sort
Quick Sort
Heap Sort
Radix Sort
Algoritmo

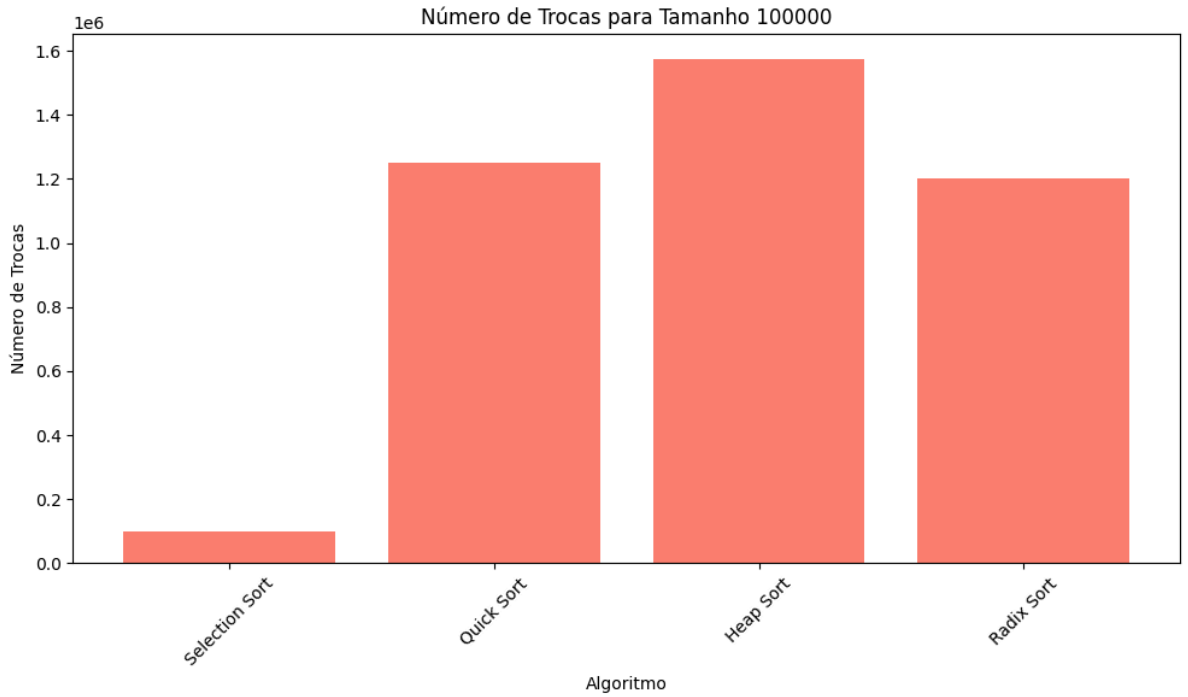

Número de Iterações para Tamanho 1000000
1e9
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
Número de Iterações
Selection Sort
Quick Sort
Heap Sort
Radix Sort
Algoritmo

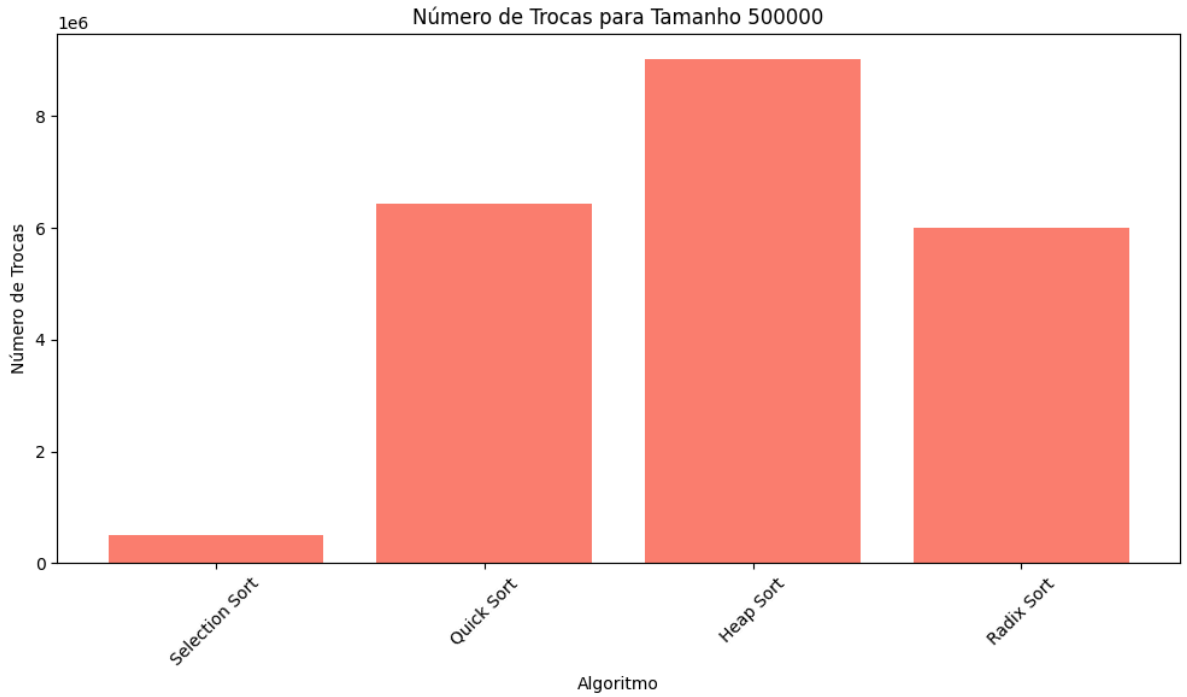


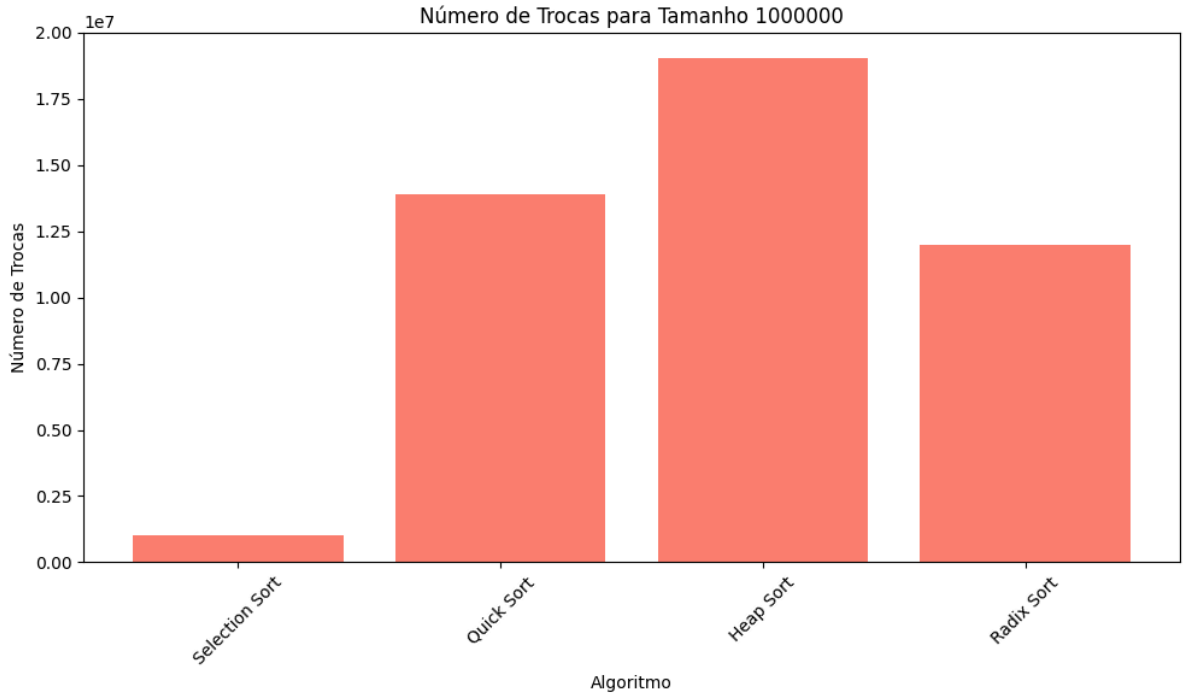
O gráfico revela nuances interessantes no desempenho dos algoritmos de ordenação quando avaliado o número de trocas para um conjunto de dados com 1000 elementos. O Radix Sort, apesar de ser conhecido por sua eficiência em determinados cenários, destaca-se com o maior número de trocas. Essa aparente contradição pode ser explicada pela natureza do algoritmo, que realiza várias passagens sobre os dados, trocando elementos com base em dígitos individuais. Por outro lado, o Selection Sort, embora tenha o menor número de trocas, apresenta um número significativamente maior de comparações, o que pode impactar seu desempenho geral, especialmente para conjuntos de dados maiores. O Quick Sort e o Heap Sort apresentam um desempenho intermediário, com o Heap Sort geralmente sendo mais eficiente em termos de número de trocas. A análise desse gráfico, em conjunto com outros indicadores de desempenho como tempo de execução e consumo de memória, é fundamental para escolher o algoritmo mais adequado para uma determinada aplicação, considerando as características específicas dos dados e os requisitos de desempenho da tarefa.


Número de Trocas para Tamanho 1000
Número de Trocas
0
2000
4000
6000
8000
10000
12000
Selection Sort
Quick Sort
Heap Sort
Radix Sort
Algoritmo


Número de Trocas para Tamanho 10000
Número de Trocas
0
20000
40000
60000
80000
100000
120000
Selection Sort
Quick Sort
Heap Sort
Radix Sort
Algoritmo


Número de Trocas para Tamanho 100000
1e6
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
Número de Trocas
Selection Sort
Quick Sort
Heap Sort
Radix Sort
Algoritmo

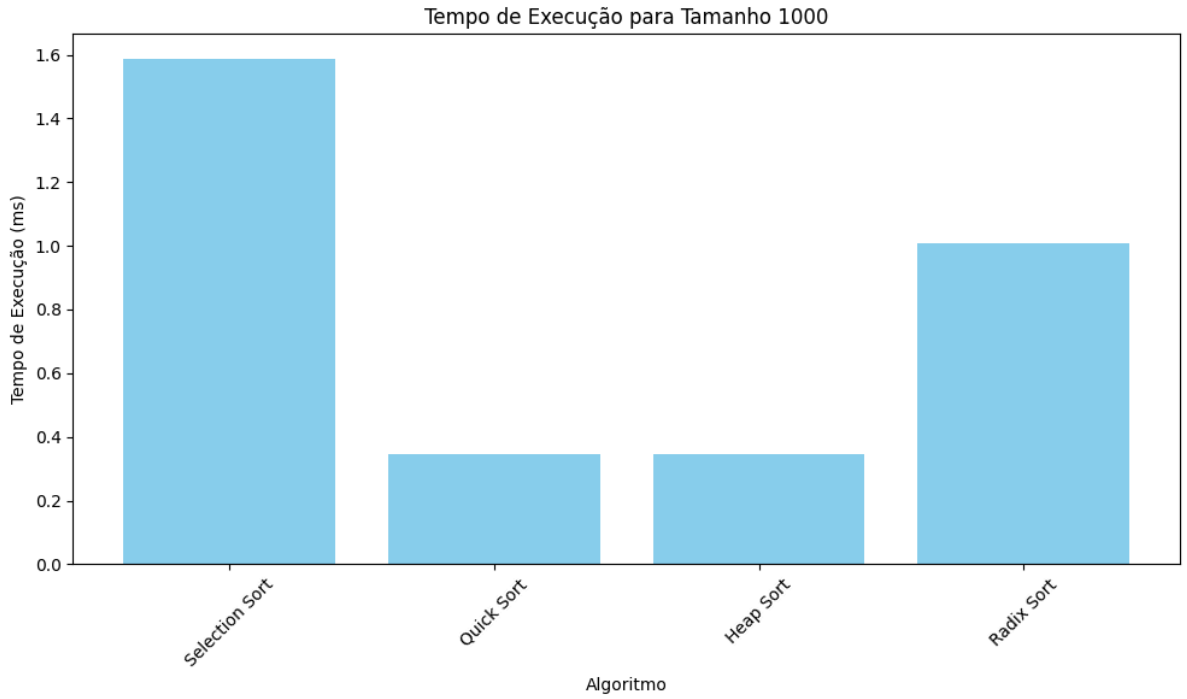




Em resumo, o gráfico demonstra que o número de trocas realizadas por um algoritmo de ordenação é apenas um dos fatores a serem considerados ao escolher o algoritmo mais adequado para uma determinada tarefa.

O gráfico demonstra claramente a superioridade do Radix Sort e dos algoritmos Quick Sort e Heap Sort em relação ao Selection Sort para ordenar um conjunto de 1000 elementos. O Selection Sort, por sua natureza, apresenta um desempenho significativamente pior devido ao grande número de comparações realizadas. Os

demais algoritmos, por sua vez, exibem tempos de execução muito mais eficientes, com o Radix Sort se destacando em muitos casos. Essa diferença de desempenho pode ser atribuída às diferentes estratégias de ordenação empregadas por cada algoritmo e à sua complexidade algorítmica. Em resumo, o Radix Sort se mostra como uma excelente opção para ordenar conjuntos de dados de tamanho médio, apresentando um bom desempenho em termos de tempo de execução. No entanto, a escolha do algoritmo ideal depende de diversos fatores, como as características dos dados e os requisitos específicos da aplicação.



O gráfico evidencia de forma clara a superioridade dos algoritmos Quick Sort, Heap Sort e Radix Sort em relação ao Selection Sort quando o conjunto de dados possui 10000 elementos ou mais. O Selection Sort, por sua natureza, apresenta um desempenho significativamente pior devido ao grande número de comparações realizadas. Os demais algoritmos, por sua vez, exibem tempos de execução muito mais eficientes, com o Radix Sort se destacando em muitos casos. Essa diferença de desempenho pode ser atribuída às diferentes estratégias de ordenação empregadas por cada algoritmo e à sua complexidade algorítmica. Em resumo, o Radix Sort se mostra como uma excelente opção para ordenar conjuntos de dados de grande porte, apresentando um bom desempenho em termos de tempo de execução. No entanto, a escolha do algoritmo ideal depende de diversos fatores, como as características dos dados e os requisitos específicos da aplicação.