**Trabalho de Aspectos Teóricos da Computação**

**Instituição: UNIP – Universidade Paulista – Campus São José do Rio Preto**

**Curso: Ciência da Computação**

**Alunos:**

**Filipi Yukio Iwakami Itoyama – RA N4453J-1**

**Lucas de Oliveira Brandolezi – RA D9380G-9**

**Luiz Gabriel Zeferino Duarte – RA N454CD-8**

**William Rossi do Carmo Ruiz – RA N473GF-8**

**Tema: Relatório com os gráficos de comportamento e devidos comentários sobre a implementação dos algoritmos de ordenação: Bubble Sort, Insertion Sort e Quick Sort.**

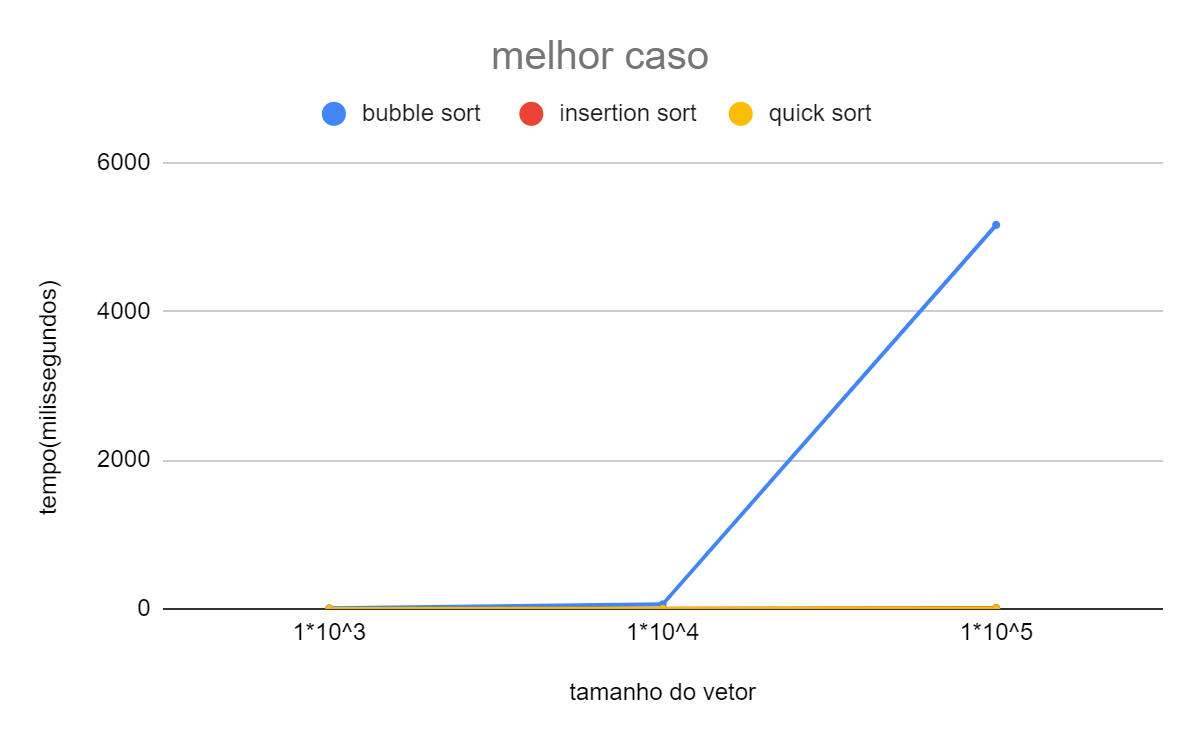
O objetivo deste trabalho é apresentar o relatório com os gráficos gerados a partir dos dados retirados com os testes em 3 dos algoritmos de ordenação populares. São eles o Bubble Sort, Insertion Sort e Quick Sort. No trabalho proposto, foi pedido para que fossem feitos testes criando um vetor com quantidades de números inteiros aleatórios variados (entre 1000 e 100.000) e ordenando-os de forma crescente, assim, medindo o tempo no qual cada algoritmo gastaria para ordenar tais vetores. Junto a esse enunciado, foi proposto também algumas restrições na hora da ordenação, uma delas é o fato de ordenar um vetor já ordenado de forma crescente, assim, considerando-se como sendo o melhor caso de ordenação. Outra restrição foi que precisava ser ordenado um vetor já ordenado também, porém, de forma decrescente, sendo assim considerado o pior caso. E por fim, a última restrição imposta foi de ordenar um vetor com números aleatórios gerados em ordem aleatória, também conhecido como o caso médio.

Antes de começar com as demonstrações dos gráficos, é bom lembrar de que, os testes neste trabalho foram feitos cada um caso a caso, ou seja, compilados teste por teste para que não houvesse qualquer influência de alguma forma se compilados todos os testes juntos. Outro adendo, cada um dos testes feitos foram compilados e testados exatamente 10 vezes, e logo em seguida era tirada a média dos tempos ganhos nesses 10 testes. Foi feito isso para que seja evitada algum dado, ou resultado que fossem errados por azar, pois estávamos gerando vetores com números aleatórios. E por fim, todos os testes feitos estão no limite máximo do que conseguimos rodar em nossos computadores, ou seja, tivemos limitações com hardware para maiores testes.

Feito a introdução e todos os adendos vamos ao relatório gerado pelos gráficos:

|  |  |  |  |
| --- | --- | --- | --- |
| melhor caso | | | |
|  | bubble sort | insertion sort | quick sort |
| 1\*10^3 | 7,4 | 0 | 0 |
| 1\*10^4 | 61,7 | 0,9 | 0,5 |
| 1\*10^5 | 5162,3 | 11,2 | 4,2 |

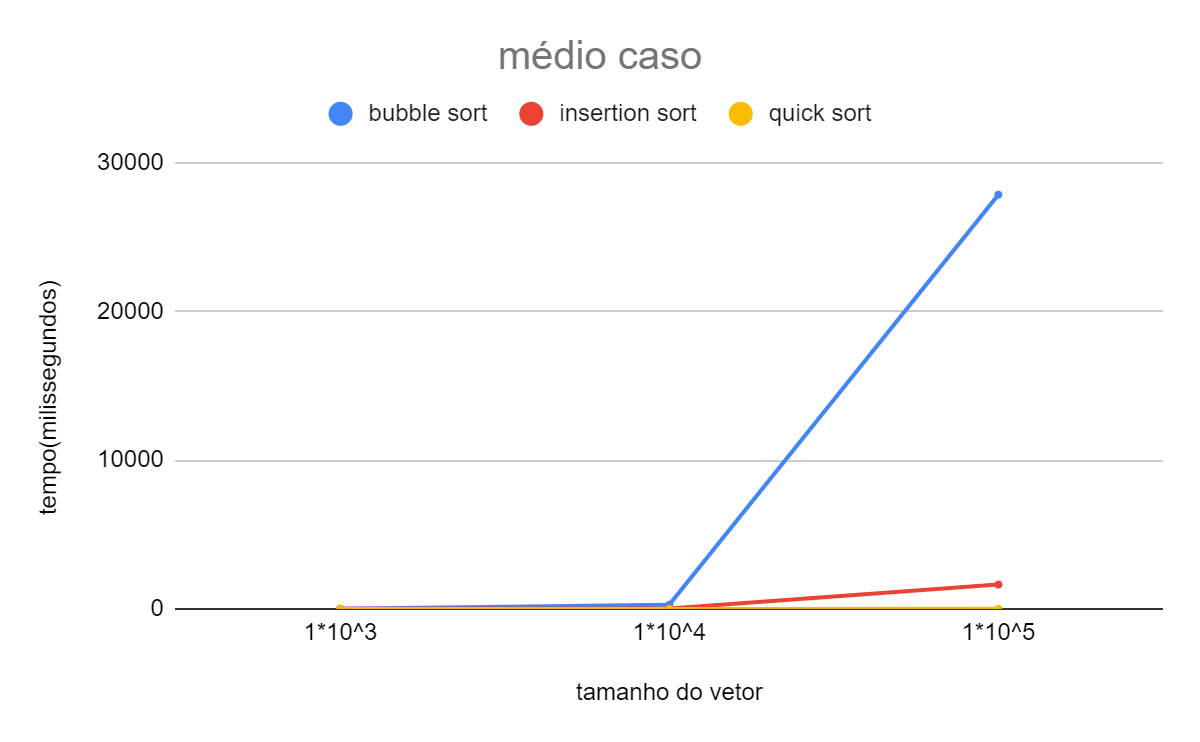
Nesta primeira tabela temos os resultados obtidos através dos testes realizados no melhor caso (no qual o vetor que está sendo ordenado JÁ está ordenado de forma crescente), cada coluna está definindo qual algoritmo foi usado para o teste, e cada linha, definindo a quantidade de números inteiros que foram geradas no vetor para o teste. O resultado obtido é o tempo em que cada algoritmo demorou para terminar de ordenar todo o vetor em milissegundos.



Este é o gráfico no qual foi plotado através dos resultados obtidos na tabela anterior. Pode-se observar que neste melhor caso, o bubble sort é disparadamente o algoritmo mais lento, aumentando drasticamente o tempo de execução dele de acordo com o tamanho do vetor no qual está sendo ordenado.

|  |  |  |  |
| --- | --- | --- | --- |
| médio caso | | | |
|  | bubble sort | insertion sort | quick sort |
| 1\*10^3 | 9,3 | 3,9 | 0,2 |
| 1\*10^4 | 277,5 | 25,9 | 1,1 |
| 1\*10^5 | 27841,5 | 1636,8 | 11,1 |

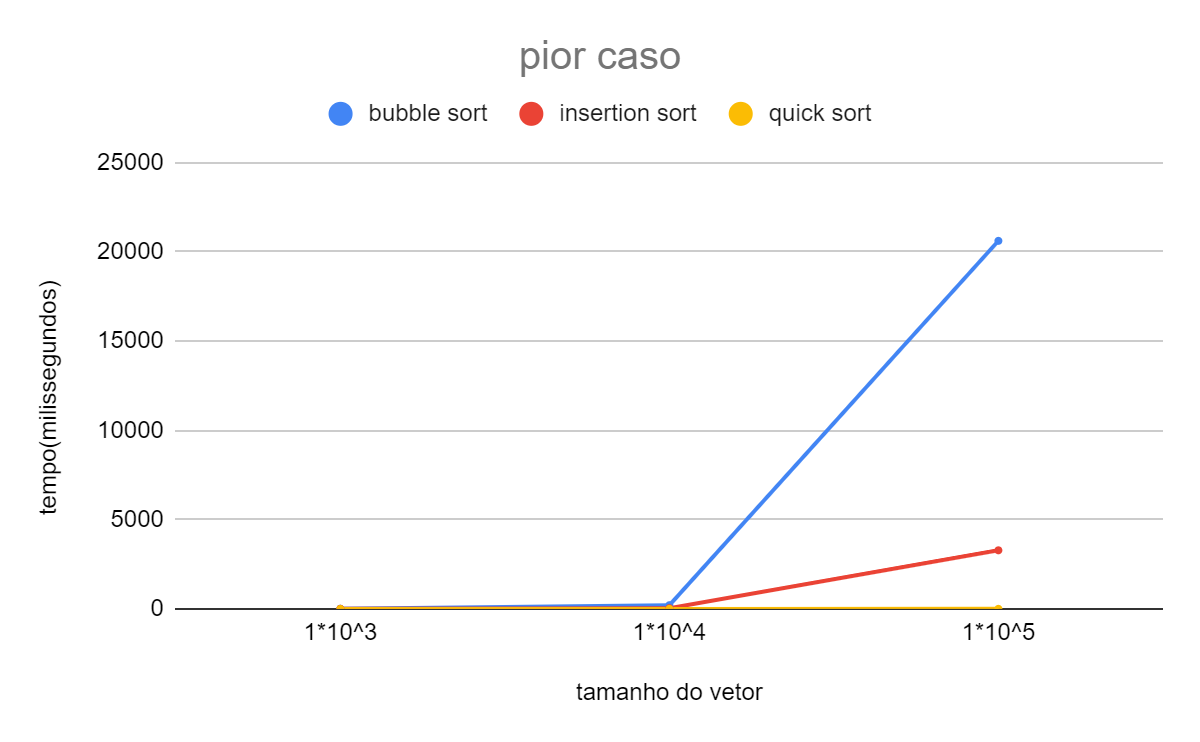
Esta é a tabela com a mesma arquitetura da tabela anterior, porém com os resultados obtidos através dos testes feitos em médio caso (aquele no qual os vetores não estão ordenados de nenhuma forma).



Este é o gráfico plotado com os resultados obtidos na tabela acima. Nele também conseguimos analisar a mesma situação do gráfico anterior, no qual o bubble sort é de longe o mais lento algoritmo entre esses 3 algoritmos testados.

|  |  |  |  |
| --- | --- | --- | --- |
| pior caso | | | |
|  | bubble sort | insertion sort | quick sort |
| 1\*10^3 | 10 | 5,7 | 0,2 |
| 1\*10^4 | 207,8 | 41,5 | 0,35 |
| 1\*10^5 | 20615,3 | 3281,3 | 3,8 |

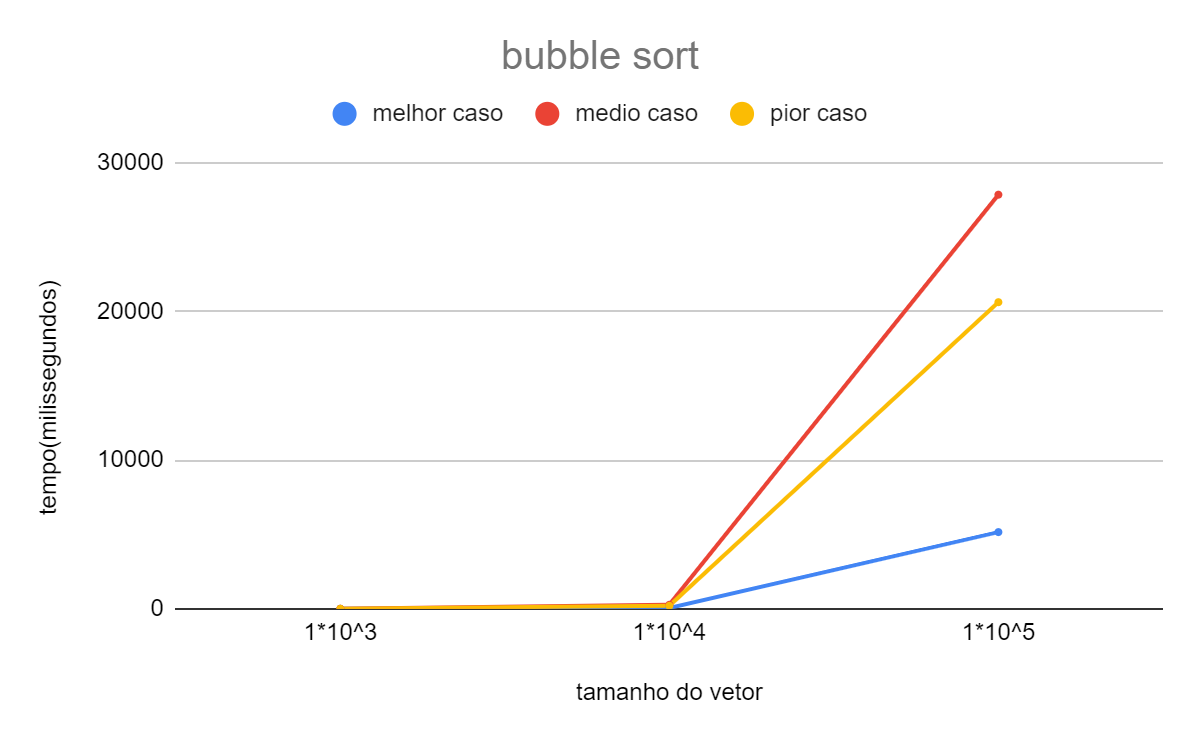
E por fim, esta é a tabela do pior caso, no qual os testes são feitos com os vetores ordenados de forma decrescente.



Este é o gráfico para a representação do pior caso. Igualmente como em todos os outros gráficos, temos a conclusão de que o bubble sort, dentre estes 3 algoritmos testados, não importando o tamanho do vetor, sempre é o mais lento e vai aumentando o seu tempo de execução muito mais do que os outros dois algoritmos (insertion sort e quick sort) conforme aumenta o tamanho de seu vetor.

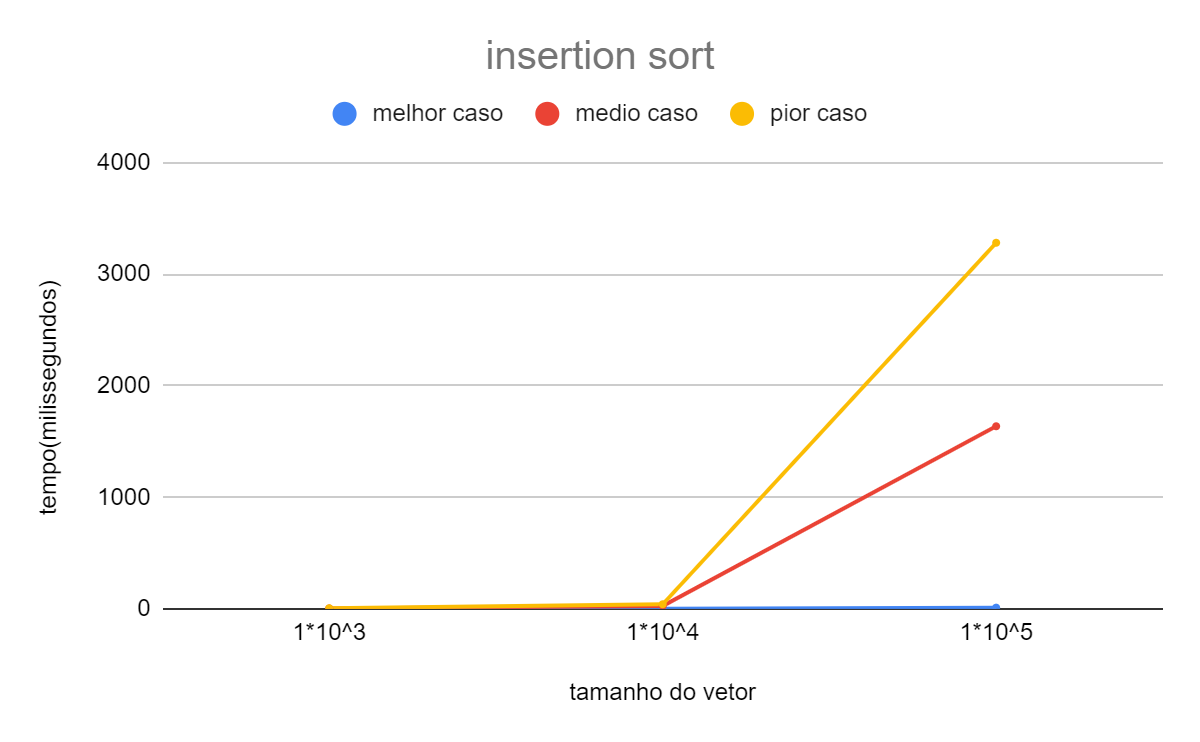
Foi também, montado o relatório de cada um dos algoritmos, para ver suas diferenças de acordo com o tamanho do vetor em cada caso.

|  |  |  |  |
| --- | --- | --- | --- |
| bubble sort | | | |
|  | melhor caso | medio caso | pior caso |
| 1\*10^3 | 7,4 | 9,3 | 10 |
| 1\*10^4 | 61,7 | 277,5 | 207,8 |
| 1\*10^5 | 5162,3 | 27841,5 | 20615,3 |



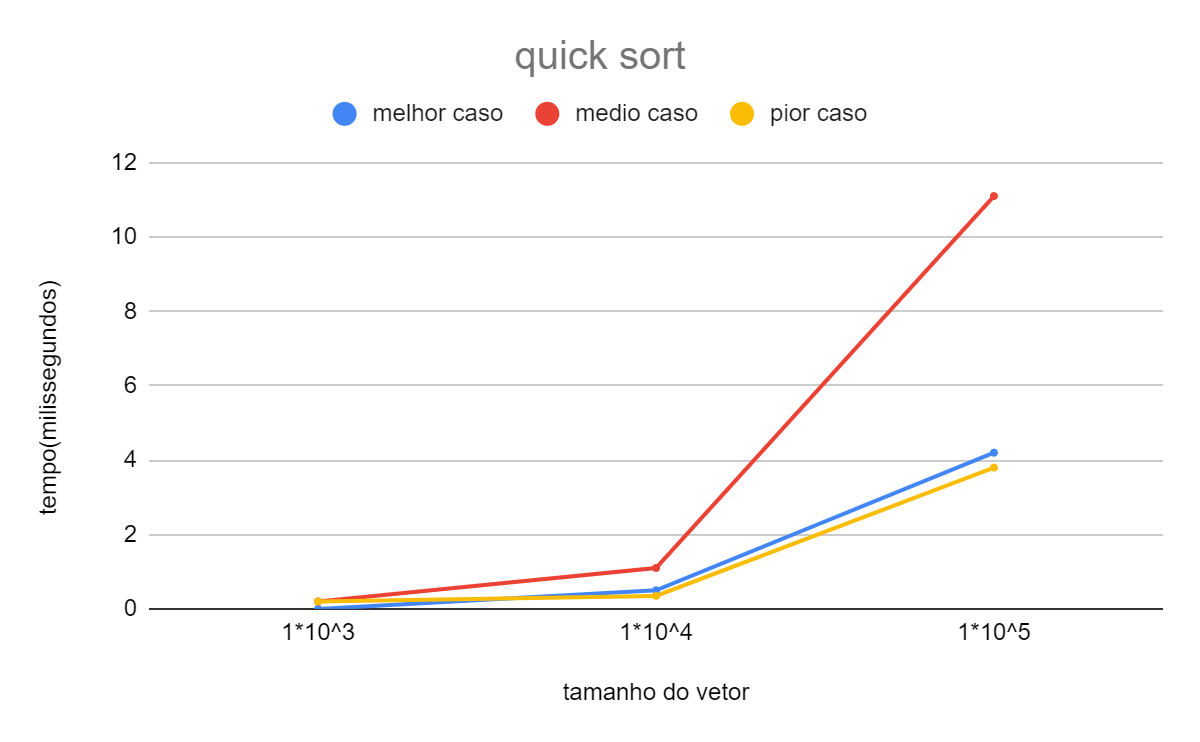
Como pode ser visto através do gráfico, no bubble sort, o caso no qual leva mais tempo para ser ordenado, quando o vetor é menor, é o pior caso. Porém, de acordo com o aumento do tamanho do vetor, o caso mais demorado passa a ser o médio caso.

|  |  |  |  |
| --- | --- | --- | --- |
| insertion sort | | | |
|  | melhor caso | medio caso | pior caso |
| 1\*10^3 | 0 | 3,9 | 5,7 |
| 1\*10^4 | 0,9 | 25,9 | 41,5 |
| 1\*10^5 | 11,2 | 1636,8 | 3281,3 |



Já no insertion sort, o pior caso é sempre o caso mais demorado, independentemente da quantia números que estão sendo ordenados.

|  |  |  |  |
| --- | --- | --- | --- |
| quick sort | | | |
|  | melhor caso | medio caso | pior caso |
| 1\*10^3 | 0 | 0,2 | 0,2 |
| 1\*10^4 | 0,5 | 1,1 | 0,35 |
| 1\*10^5 | 4,2 | 11,1 | 3,8 |



Enquanto isso, o quick sort, passa a ter uma característica similar ao bubble sort, porém, com um único detalhe diferente, que seria o fato de médio caso SEMPRE ser o mais lento, enquanto no bubble sort o médio caso só ficava mais lento conforme era aumentado o tamanho do vetor.