Além dos gráficos, descrever os seguintes aspectos: -

**- Como foram gerados os vetores aleatórios;**

R. Usamos a função **Math.random** do java para gerar os valores aleatórios. A saída do **random** gera um double de 0 até 0.99, cujo resultado é multiplicado por 101. No final gerando um número inteiro até 99;

A função **Math.random** foi colocada estrategicamente dentro de um for que tem o objetivo de inserir o valor gerado pelo **random** dentro de um vetor, que é executado até que seu tamanho seja totalmente preenchido:  
  
Tamanhos: 5, 10, 50, 100, 1000, 10000;

Tudo isto dentro de um método GenerateRandVet que recebe o tamanho setado na main.

**- Como foram adaptados os algoritmos para realizar a contagem do número de comparações;   
R**: Criamos uma classe abstrata que tem um atributo comparações, que guarda o numero de comparações de cada algoritmo. Cada algoritmo deve então herdar essa classe e implementar a contagem de comparações

**- O que foi desenvolvido para que cada algoritmo ordenasse os mesmos vetores que os demais;**

R: Ainda usando a mesma classe Abstrata, criamos um método setVetor, que copia um vetor paraum atributo interno, vetor. O algoritmo ordena apenas o atributo criado internamente.

**- Explicar os resultados dos experimentos com base na literatura;**

Com base nos dados extraídos do algoritmo feito pelo grupo, conseguimos chegar a algumas conclusões:

Algoritmos:

* Bubble Sort: Este algoritmo é eficiente apenas para entradas de tamanho até 5, a partir disto se torna o pior algoritmo para ordenação de vetores maior que 5.
  + Complexidade quadrática.
  + No melhor caso se torna O(n)
* Selection Sort: Falando em comparações é melhor do que o Bubble, mas ainda sim atinge nossa penúltima colocação se fomos pensar em um ranking de algoritmos quadráticos, este algoritmo tem bom desempenho para vetores de tamanho até 50.
  + Complexidade quadrática.
* Insertion Sort: Entre os algoritmos é o que possui melhor comportamento de acordo com nossa tabela de comparações, possui bom desempenho para vetores de tamanho até 100.
  + Complexidade quadrática.

Os três algoritmos acima, possuem um péssimo desempenho quando se é necessário ordenar vetores com valores altos. Indo ao oposto destes algoritmos temos os algoritmos abaixo:

* HeapSort
* MergeSort
* QuickSort
* CountSort
* RadixSort
* BucketSort

Todos estes algoritmos não tem um bom desempenho quando se é necessário ordenar pequena quantidade de dados, porém quando falamos de um alto volume, todos possuem um bom desempenho, se colocarmos em ranking de comparações os algoritmos acima, chegaríamos no seguinte resultado:

* CountSort possui o menor número de comparação dentre os algoritmos acima, ele começa a apresentar seu bom desempenho a partir de vetores de tamanho 10, se mantando eficiente para grande volume de dados.