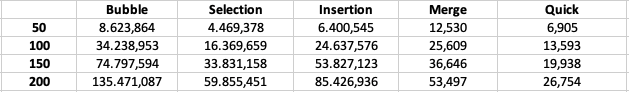
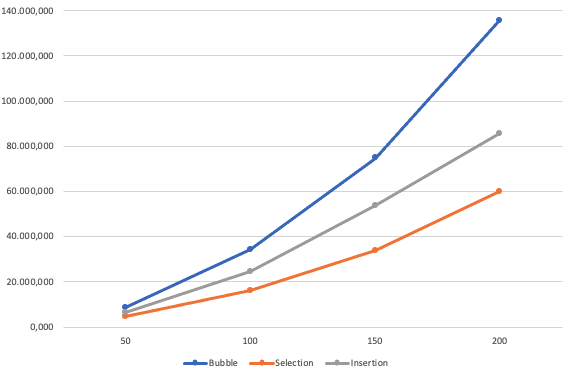
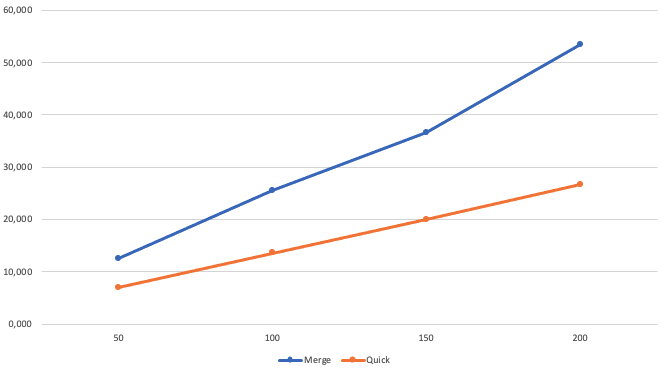
**Trabalho de estruturas de dados**

1. O algoritmo escolhido foi o quicksort, uma vez que olhando comparações com os demais é o mais eficiente para ordenação da maioria de conjuntos de dados aleatórios.
2. Código Implementado está anexado.
3. Ok.
4. Seguem os tempos de cada algoritmo para 50, 100, 150 e 200 mil números no vetor.



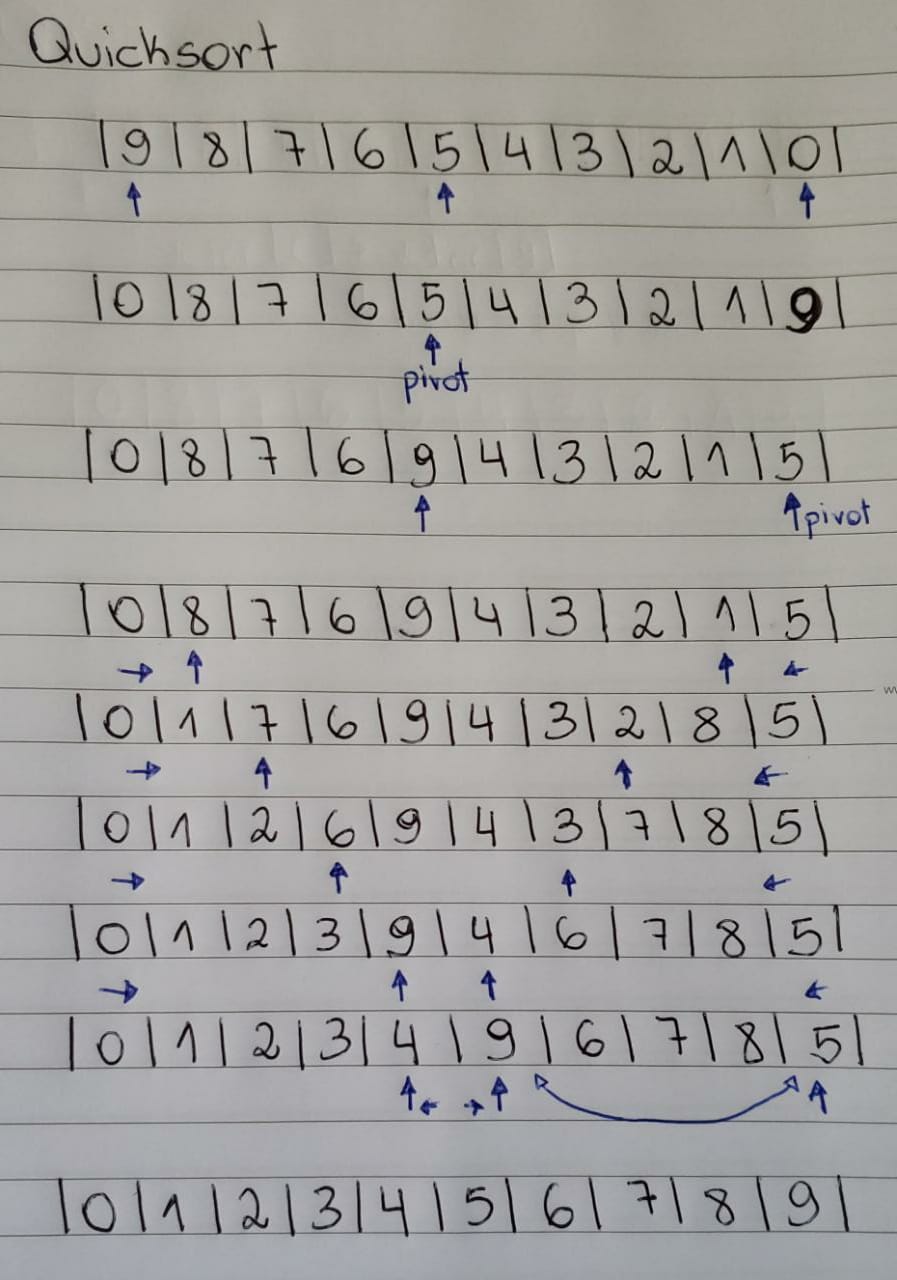
1. A escala dos algoritmos mais lentos (bubble, selection e insertion) é muitas vezes maior que os mais eficientes (merge e quick). Por isso, fiz dois gráficos em escalas diferentes que permitem uma análise mais precisa.





Com esses números, conseguimos perceber que o merge sort e quick sort, mesmo no pior caso, apresentam performance muito superior aos demais algoritmos. Além disso o quick sort é consideravelmente mais rápido que o merge sort.

1. O processo de ordenação do quicksort funciona da seguinte forma:
   1. Escolhe-se um pivot. A técnica utilizada para isso foi a mediana entre o primeiro, o central e último elemento.
   2. Após isso, posiciona-se o pivot no último elemento do vetor.
   3. Então, partindo da esquerda, busca-se o primeiro elemento que seja maior que o pivot. Após isso, partindo da direita, busca-se o primeiro elemento que seja menor que o pivot.
   4. Ao encontrar esses elementos, devem ser invertidos.
   5. Se o primeiro menor elemento encontrado estiver à esquerda que o primeiro maior elemento encontrado, significa que todos os menores elementos estão a esquerda de todos os maiores elementos. Nesse caso, troca-se o primeiro maior elemento com o pivot.
   6. Por ser um algoritmo recursivo, esse processo é reexecutado tantas vezes quanto necessárias.



Nesse caso, apenas uma execução do algoritmo foi necessária para ordenar o vetor, não tendo sido necessária a chamada recursiva. Isso acontece principalmente pelo método de escolha do pivot, que afeta bastante a performance do algoritmo. Alguns algoritmos selecionam o último elemento do vetor como pivot. Nesse caso, seriam necessárias várias chamadas recursivas para o algoritmo, resultando no pior caso de complexidade do quick sort que é O(n^2).

Essa é a explicação conceitual que julguei mais fácil de entender e representar. Inicialmente implementei essa solução que necessita, contudo, de 2 loops. Em busca de algoritmos mais eficientes encontrei outra maneira, que mantém o conceito, mas mais performática, que é a implementada. Nesse caso, ao invés de haver 2 loops, o vetor é percorrido apenas uma vez. A lógica utilizada é:

* Inicia o algoritmo com indiceElementoEsquerda e indiceElementoDireita como o primeiro valor do vetor.
* Incrementa-se indiceElementoDireita até que o valor apontado seja menor que o pivot.
* Quando for menor, inverte-se o elementoEsquerda e o elementoDireita e incrementa-se 1 no indiceElementoEsquerda.
* Dessa maneira, sabemos que todos os elementos a esquerda de indiceElementoEsquerda são menores que o pivot.
* Quando o indiceElementoDireita chegar ao final do vetor, significa todos os elementos maiores que o pivot estão à esquerda de indiceElementoEsquerda e todos os elementos maiores estão a direita de indiceElementoEsquerda, fazendo com que esse seja o novo índice correto para o pivot.
* Por ser recursiva, repete-se o processo para o vetor de valores maiores e o vetor de valores menores que o pivot.