**1.3 - Análise (RA3)**

Para cada algoritmo de ordenação utilizado, faça a análise de complexidade (Notação Big O). Justifique como e porque a complexidade de tempo do algoritmo se comporta em relação ao número de itens na lista de itens.

**Bubble Sort**

No Bubble Sort o número de comparações é constante, independentemente de quão embaralhada ou não a lista esteja. Porém cada novo item adicionado na lista aumenta exponencialmente o tempo que vai demorar para processar o algoritmo(n²), porque cada novo item adiciona uma nova comparação para cada item que já existia na lista. O Bubble Sort também tem o potencial de “desperdiçar” muita energia em uma lista já resolvida. Concluindo então o Bubble Sort é bom em pequenos bancos de dados, mas potencialmente muito custoso.

**Selection Sort**

O Selection Sort é muito similar ao Bubble Sort por conta de suas comparações serem idênticas, 2 loops fazendo uma comparação por item por item ou n², a diferença está no número de trocas ou swaps que ocorrem no Selection sort sendo só de uma constante n, um para cada elemento da lista. Assumindo que não exista algo que cheque se a troca é necessária ou não. Também é bom em pequenas Databases e piora exponencialmente dependendo do aumento no número de itens.

**Merge Sort**

O Merge Sort difere grandemente dos últimos dois algoritmos na questão de quanto tempo ele pode demorar para processar, seguindo uma fórmula constante de nlogn independente da situação da lista a ser ordenada. Isso significa que ele demora um tempo relativamente alto para cada item na lista, tendo sempre que dividir a lista no meio, mas que aumenta constantemente, fazendo ele ser um algoritmo muito mais eficiente para Databases maiores. O drawback é a necessidade de memória adicional de n para poder processas o algoritmo, ou seja, uma capacidade adicional igual ao tamanho da lista

**Quick Sort**

Quick Sort segue o Merge Sort em sua complexidade de tempo, mas difere na margem que pode existir, no melhor caso, ou seja, escolhendo o pivot perfeito e sempre dividindo o array em partes iguais, já no pior caso o Quick sort pode chegar até a n². Porém um Quick Sort bem implementado raramente teria o problema de cair no pior caso, estabelecendo-o como um algoritmo bom para Databases grandes, gastando bem menos memória do que o Merge Sort, chegando a no máximo igualar a complexidade de n no pior dos casos.