Relatório Projeto 4.4 AED 2021/2022

Nome: Mariana Lopes Paulino Nº Estudante: 2020190448
TP (inscrição): 1 *Login* no *Mooshak: 2020190448*

Registar os tempos computacionais das variantes em consideração para os diferentes tipos de sequências. Os tamanhos das sequências (N) devem ser: 25000, 50000, 75000, 100000, 125000. Só deve ser contabilizado o tempo de ordenamento. Exclui-se o tempo de leitura do input e de impressão dos resultados.

Gráfico para SEQ_ALEATORIA

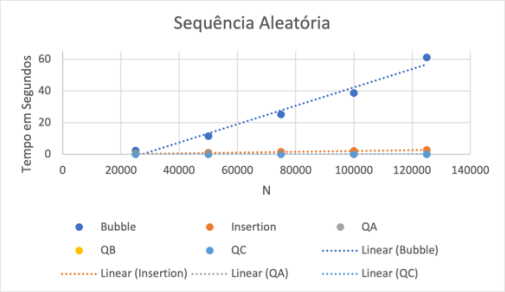


Gráfico para SEQ_ORDENADA_DECRESCENTE

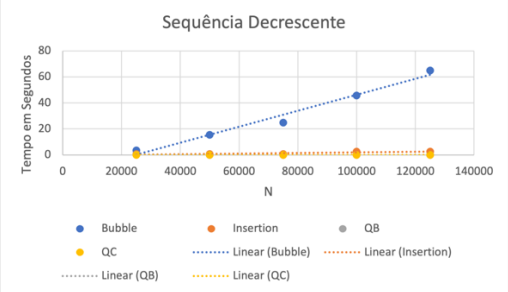


Gráfico para SEQ_QUASE_ORDENADA_1%

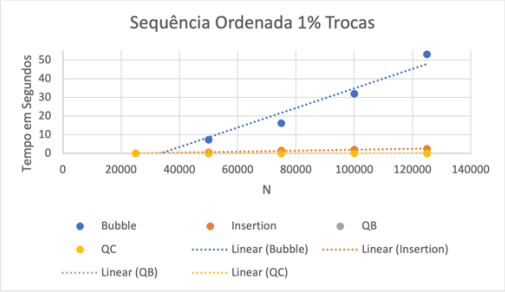
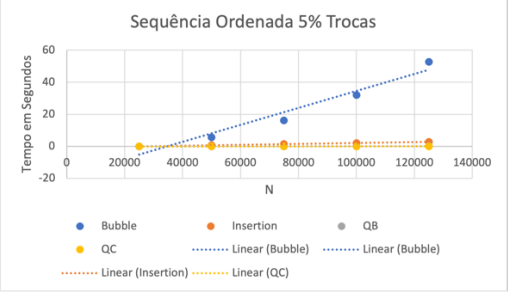


Gráfico para SEQ_QUASE_ORDENADA_5%



Análise dos resultados + discussão sobre a estabilidade de cada algoritmo de ordenamento e, nos casos em que o algoritmo não é estável, propor alterações ao algoritmo de forma a garantir a sua estabilidade + discussão sobre a possível criação de um algoritmo híbrido com base em 2 ou 3 destes algoritmos. Existirão vantagens gerais, vantagens para alguns tipos de sequência, ou nunca será vantajoso? Justifique:

Colocando todos os resultados dos diferentes algoritmos para o mesmo tipo de sequência no mesmo gráfico é notável a sua discrepância uma vez que, o Bubble Sort é o único que se destaca e onde conseguimos observar bem todos os seus tempos perfeitamente, nos outros algoritmos isso não acontece devido aos seus tempos extremamente pequenos comparando ao Bubble Sort. Entre os outros dois algoritmos (Insertion Sort e o Quick Sort) o Quick Sort vai ser o que tem tempos melhores dependendo da escolha do pivot sabendo que uma má escolha do pivot pode elevar a sua complexidade até O(n^2), a pior complexidade de todos os algoritmos estudados tal como é o caso do Bubble Sort. O melhor método estudado para escolher o pivot do Quick Sort é o método da mediana de 3 elementos (inicial, final, do meio) e a partir daí fixar sempre a mediana desses 3 como pivot o

que leva a uma complexidade de O(n*logn). No outro algoritmo estudado (Insertion Sort) sabemos também que tem complexidade O(n^2) em todos os seus casos gerais, mas tem uma particularidade que é bastante benéfica em sequências já ordenadas ou em sequências quase ordenadas onde a sua complexidade irá ser O(n) tendo assim a melhor complexidade estudada nestes algoritmos.

Quanto à estabilidade destes algoritmos o Bubble Sort e Insertion Sort são ambos algoritmos estáveis onde na sua execução sabemos que ao termos dois items iguais o que apareceu primeiro irá se manter em primeiro lugar, ao contrário do Quick Sort que é um algoritmo instável, uma das formas de o manter estável é quando temos dois ou mais elementos iguais e os estamos a organizar comparamos sempre os índices dos mesmos quando inseridos na lista uma vez que queremos preservar a ordem dos mesmos por isso devemos sempre selecionar primeiro o numero de índice mais pequeno.

A criação de algoritmos híbridos alivia os algoritmos já existentes uma vez que podemos combinar as melhores partes de ambos e criar um muito mais rápido, um algoritmo hibrido é bom quando implementado com recursividade, isto é algoritmos como o Quick Sort que utilizam recursividade em vez de iteratividade para se desenvolverem.

Um bom algoritmo hibrido seria a junção do Quick Sort com o Insertion Sort mas apenas em casos específicos tais como o de uma sequência mais pequena, uma vez que o Quick Sort apresenta os melhores tempos para sequências maiores mas mesmo assim a sua complexidade de O(n*logn) é superior à complexidade do Insertion Sort para sequências ordenadas ou quase ordenadas que é O(n) por isso, se utilizarmos o Quick Sort até a sequência ser pequena o suficiente para que compense a utilização do Insertion Sort para ordenar a sequência em vez da continua utilização do Quick Sort.

Este método iria utilizar a recursão aquando da utilização do Quick Sort e passaria para um método iterativo após o Quick Sort chegar a um certo número, especificando num caso concreto: Temos uma sequência de n elementos para ordenar e vamos verificando aquando das recursões feitas pelo Quick Sort se o n para ordenar é superior a um k definido para começar a utilizar o Insertion Sort, se n for superior continuamos a utilizar o Quick Sort, se n for igual ou inferior a k começamos a utilizar o Insertion Sort de modo a que este seja mais rápido formando assim um algoritmo mais rápido diminuindo o seu tempo de execução e a sua complexidade.