Relatório Projeto 4.2 AED 2020/2021

Nome: Tomás Batista Mendes Nº Estudante: 2019232272

TP (inscrição): 2 *Login* no *Mooshak:*

Nº de horas de trabalho: 6 *H* *Aulas Práticas de Laboratório:* 0 *H* *Fora de Sala de Aula:* 6 *H*

(A Preencher pelo Docente) CLASSIFICAÇÃO:

Comentários:

Registar os tempos computacionais do QS e das 4 variantes selecionadas do QS+IS para os diferentes tipos de sequências. O tamanho das sequências (N) deve ser crescente e terminar em 10,000,000. Só deve ser contabilizado o tempo de ordenamento. Exclui-se o tempo de leitura do input e de impressão dos resultados. Devem apresentar e discutir as regressões para a melhor variante em cada tipo de sequência.

Gráfico para SEQ_ALEATORIA

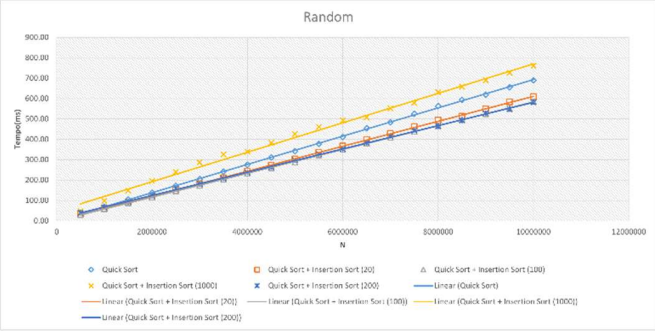


Gráfico para SEQ_ORDENADA_DECRESCENTE

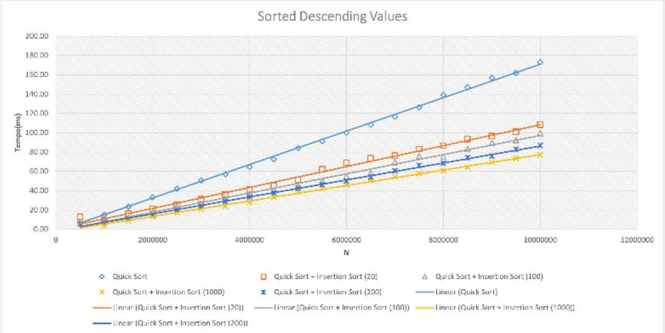


Gráfico para SEQ_QUASE_ORDENADA_1%

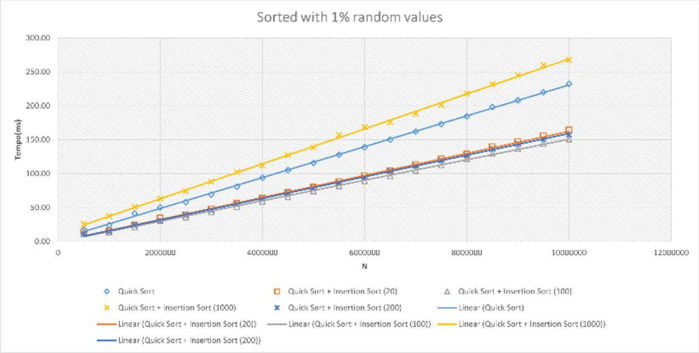
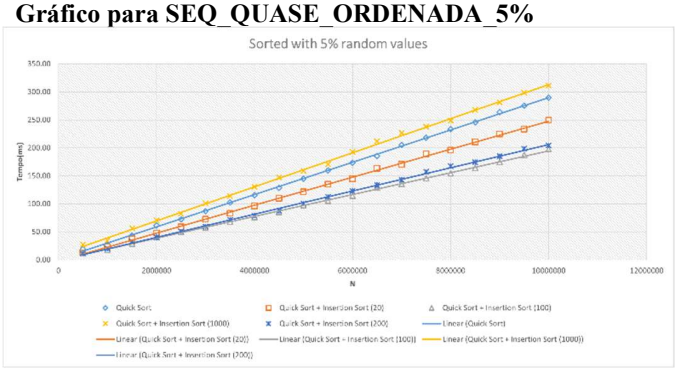


Gráfico para SEQ_QUASE_ORDENADA_5%



Análise dos resultados:

Tal como esperado, o quick sort apresenta tempos de execução muito baixos com uma complexidade O(N log N) e não precisa de mais *overhead* de memória, dando *sort* ao *array* sem ser preciso cópias do mesmo. Na implementação feita deste algoritmo, foi considerado o elemento central de cada *sub-array* como *pivot* do algoritmo. Como sugerido, foi também implementado um *Insertion sort* para "acabar" o trabalho do quick sort, evitando assim muitas chamadas recursivas quando o *array* a dar *sort* já é pequeno. O *Insertion sort* apesar de ter um complexidade O(N^2) no pior caso, apresenta tempos baixos para quando um *array* já se encontra quase ordenado, o que acontece quando o *Quick sort* chega a *sub-arrays* pequenos. Pelos testes feitos, e pelo *dataset* em questão, o *Insertion sort* diminuiu em geral os tempos de *sorting,* sendo que mostrou ser o mais eficiente para *sub-arrays* com tamanho 200. Para a sequência de valores ordenada descendente, um valor alto alto, como 1000, diminui ainda mais os tempos de execução.