Trabalho de Aula 2 CAL

Aluno: Pedro Vargas Tannuri

Introdução:

Neste trabalho, foi feita a implementação, bem como a análise e comparação da complexidade de diversos algoritmos de ordenação e de busca, todos sobre um vetor de números inteiros.

A análise da complexidade dos algoritmos foi feita medindo o tempo de execução (em Microssegundos (µs)) que cada método apresentou, dado um tamanho de entrada N, que variou de zero até 3000 elementos.

A implementação destes algoritmos foi feita em linguagem C. Cada algoritmo cria um arquivo de texto contendo, em cada linha, o custo temporal de execução. Após a criação destes arquivos, um script simples em Python utiliza a biblioteca matplotlib para plotar dois gráficos de comparação.

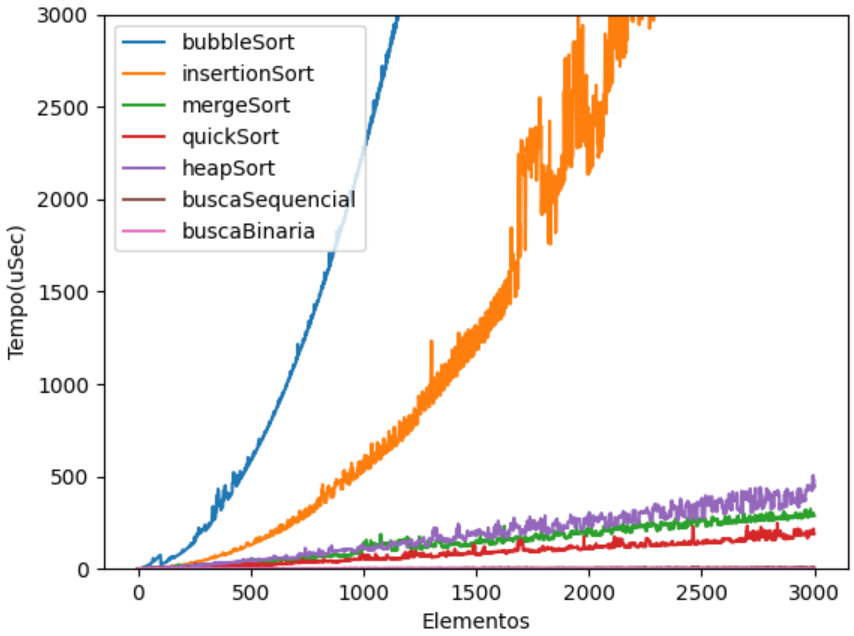
Os algoritmos de ordenação analisados foram: Bubble-Sort, Insertion-Sort, Heap-Sort, Merge-Sort e Quick-Sort.

Os algoritmos de busca analisados foram: Busca Sequencial e Busca Binária.

A medição do tempo foi feita realizando a chamada da função gettimeofday() antes e depois da chamada de cada algoritmo, sendo assim, relativamente precisa.

Análise dos gráficos obtidos:

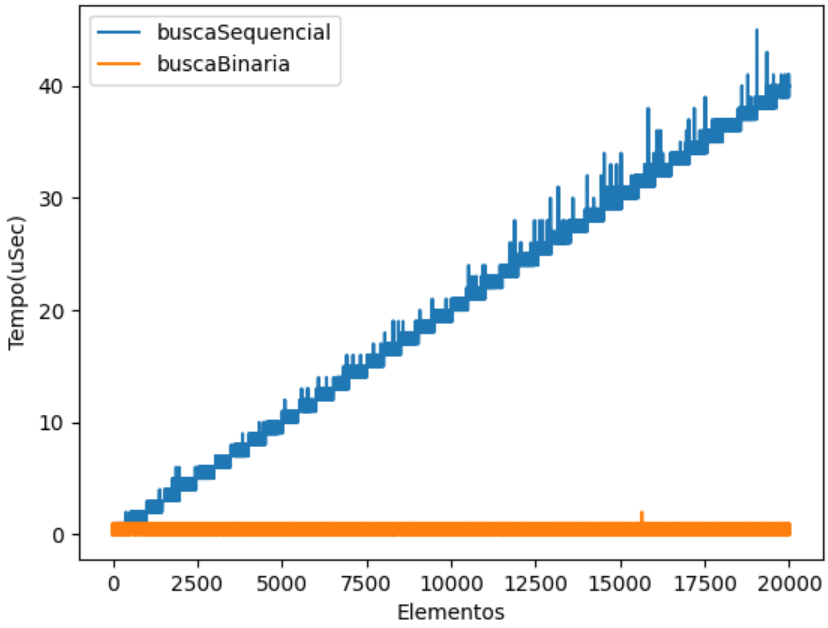
Primeiro Gráfico: Comparação geral, entrada variando de 0 até 3000 elementos



Observando este primeiro gráfico, podemos comparar suas taxas de crescimento com suas respectivas complexidades:

- Bubble-Sort: $O(n^2)$ em todos os casos.
- Insertion-Sort: $O(n^2)$ no pior caso e caso médio e O(n) no melhor caso. Isto explica por que a taxa de crescimento do tempo se assemelha à taxa do Bubble-Sort, porém é notavelmente menor.
- Heap-Sort: O(n log(n)) em todos os casos.
- Merge-Sort: O(n log(n)) em todos os casos. O Merge-Sort apresenta um melhor desempenho de tempo se comparado com o Heap-Sort para valores muito grandes de entrada, porém, pela sua natureza ser recursiva, acaba apresentando um pior uso da memória.
- Quick-Sort: O(n log(n)) no melhor caso e caso médio e $O(n^2)$ no pior caso. Mesmo apresentando essa instabilidade, se comparado com o Heap e Merge-Sort, o Quick-Sort utiliza melhor o princípio de localidade de dados (como visto nas aulas de EDA2), portanto, para vetores, como foi utilizado neste trabalho, ele apresenta um desempenho significativamente melhor do que o Merge e o Heap-Sort.
- Busca Sequencial e Busca Binária: Respectivamente O(n) e O(log n). Não é possível, dada a escala do gráfico 1, analisar a diferença entre estes dois algoritmos de busca, portanto, faz-se necessário um segundo gráfico, com escala modificada para melhor compreensão da gigantesca diferença entre eles.

Segundo Gráfico: Comparação entre Busca Sequencial e Busca Binária, entrada variando de 0 até 20000 elementos.



Com esta nova escala, fica evidente a diferença entre uma simples busca sequencial (O(n)) e uma poderosa busca binária (O(log n)). A única desvantagem desta em relação a outra é o fato dela ter uma pré-condição necessária para realizar a busca, essa sendo a ordenação dos dados contidos no vetor.

Conclusão:

Os resultados obtidos de forma empírica foram consonantes com a teoria apresentada pela professora em sala de aula.

Note que, mesmo que o esperado de cada uma das funções fosse um comportamento monótono crescente, isto é, estritamente crescente, na prática, pequenas questões físicas relacionadas ao hardware do computador podem fazer com que uma entrada de dez elementos demore mais do que uma entrada de onze.