Análise de Algoritmos
Projeto 01 - Análise de Métodos de Ordenação
Bacharelado em Ciência da Computação

Prof. Daniel Saad Nogueira Nunes

1 Introdução

Existem diversos métodos de ordenação, cada qual com a sua complexidade de tempo e espaço. Além disso, podemos caracterizá-los de acordo com o critério de ser *in-place* ou estável.

Este projeto visa a comparação experimental dos algoritmos de ordenação *Bubblesort*, *Heapsort* e *Mergesort*, considerando os recursos de tempo e espaço. O ferramental de análise de algoritmos deve ser utilizado para embasar a análise dos resultados experimentais.

2 Métodos de Ordenação

Os métodos *Bubblesort*, *Mergesort* e *Heapsort* deverão ser implementados em uma Linguagem de Programação cuja escolha é livre, e experimentos deverão ser realizados para averiguar o comportamento destes métodos no que tange o tempo e o espaço (consumo de memória).

Ao menos três cenários de entrada devem ser considerados para a discussão experimental:

- Melhor caso.
- Pior caso.
- Aleatório.

Ressalta-se que o melhor/pior caso de um método nem sempre é igual ao de outro.

Uma discussão acerta dos critérios de *in-place* e estabilidade dos métodos também deverá ser realizada.

Gráficos deverão ser gerados para ilustrar os diferentes comportamentos dos métodos. Uma opção é trabalhar com tamanhos de entrada nas faixas de $\{10^5, 10^6, 10^7, 10^8, 10^9\}$ elementos nos experimentos. É importante observar que, devido a complexidade de tempo, alguns métodos podem não executar em tempo hábil para tamanhos de entradas mais elevados. Isso deverá ser considerado na apresentação dos resultados e explicado no relatório.

Um relatório técnico deverá ser produzido e deve conter, minimamente, a seguinte organização:

- Introdução: contextualiza o problema a ser resolvido.
- Métodos de Ordenação: apresenta os métodos de maneira breve e com exemplos próprios.
- Materiais e Métodos: especifica os materiais e métodos utilizados na execução dos experimentos. As informações aqui contidas devem ser suficientes para que um terceiro consiga reproduzir os experimentos realizados. Não se trata de copiar e colar código, mas explicar detalhadamente como os experimentos foram conduzidos para a coleta dos resultados. As informações contidas aqui deverão ser suficientes para que uma pessoa consiga reproduzir os resultados apresentados.

- Análise de Dados: apresenta os dados obtidos, em formato de gráfico, e realiza a discussão. Os resultados deverão ser justificados com suporte formal do ferramental de análise de algoritmos (análise assintótica, relações de recorrência, etc).

- Considerações finais: realiza a discussão sobre os resultados observados e apresenta as considerações finais de acordo com os objetivos do projeto.

O relatório deverá utilizar o modelo de artigos LaTeX da Sociedade Brasileira de Computação.

3 Considerações

- Este projeto pode ser executado individualmente ou em dupla.

- Detecção de plágio automaticamente acarretará nota 0 para os envolvidos. Medidas disciplinares também serão tomadas.

- O trabalho deve ser entregue dentro de uma pasta zipada com a devida identificação do(s) aluno(s) através da sala de aula virtual da disciplina.

- Os códigos utilizados para implementação dos métodos e execução dos experimentos devem ser anexados junto ao relatório.

3.1 Suporte Experimental

- Para automatização dos experimentos, podem ser utilizadas linguagens de *script*, como por exemplo: `Python`, `Perl` e `Bash`.

- A maioria das linguagens de programação modernas possuem mecanismos para mensurar o tempo de execução entre dois trechos de códigos. Estes mecanismos podem ser utilizados para mensurar o tempo. O mesmo pode ser dito sobre a geração aleatória de números.

- Existem diversas ferramentas capazes de mensurar o consumo de memória RAM durante a execução de um processo, tais como:

  - `https://gist.github.com/netj/526585`;
  - `https://github.com/jhclark/memusg/blob/master/memusg`;
  - `https://github.com/bingmann/malloc_count`;
  - `https://github.com/mpetri/mem_monitor`;

  . Outras ferramentas também podem ser utilizadas.

- É interessante compilar (caso se aplique) os códigos-fonte com as *flags* de otimização, para que o programa possa executar mais rápido.