

# Estruturas de Dados e Introdução à Projeto e Análise de Algoritmos

## Roteiro de Laboratório – *Quick sort*

## 1 Objetivo

O objetivo deste laboratório é implementar uma série de variantes do *quick sort* e fazer uma análise empírica (experimental) do seu desempenho.

## 2 Configuração dos experimentos

Baixe os arquivos de entrada disponibilizados no Classroom. Estes arquivos possuem entradas de 100K, 1M e 10M inteiros para ordenação. As variantes da entrada são as mesmas usadas anteriormente: aleatório, ordenada, ordenada reversa e parcialmente ordenada.

Utilize o programa cliente desenvolvido no Laboratório de Métodos Elementares de Ordenação para realizar os testes dos algoritmos de *quick sort*. Meça o tempo de execução de cada uma das variantes do *quick sort* para cada uma das entradas de teste.

## 3 Variantes do *quick sort*

Implemente, teste e meça o tempo de execução de cada uma das versões do *quick sort* abaixo. **MUITO IMPORTANTE:** Não faça o embaralhamento da entrada nas suas implementações, exceto na versão em que ele for explicitamente incluído. O objetivo destes experimentos é verificar como o *quick sort* se comporta diante dos diferentes tipos de entrada.

- **Versão 1:** *quick sort* clássico *top-down* recursivo sem nenhuma otimização. (Veja os slides 10 e 11 da Aula 16.)
- **Versão 2:** *quick sort* *top-down* recursivo com *cut-off* para *insertion sort*. Implemente o seu código de forma que seja fácil modificar o valor de *cut-off*. (Veja o slide 23 da Aula 16.) Varie o valor de *cut-off* a partir de 1 e determine o valor ideal para a sua implementação.
- **Versão 3:** *quick sort* *top-down* recursivo com mediana de 3. (Veja o slide 24 da Aula 16.) Implemente essa versão a partir da Versão 1, isto é, não use *cut-off*. Assim, comparando separadamente as versões você pode determinar o ganho individual de cada otimização.
- **Versão 4:** fusão das Versões 2 e 3, isto é, usar as duas otimizações: *cut-off* e mediana de 3.
- **Versão 5:** *quick sort* *bottom-up* sem nenhuma otimização. (Veja o slide 25 da Aula 16.)
- **Versão 6:** altere a Versão 5 para incluir *cut-off* e mediana de 3.
- **Versão 7:** altere a melhor versão que você obteve até aqui para incluir o embaralhamento (*shuffle*) da entrada.

Faça uma análise das 7 versões do algoritmo segundo o desempenho. Há alguma que se destacou? Qual versão você escolheria e por quê? Em particular, procure entender por que a Versão 7 não é utilizada na prática.

## 4 *Quick sort* vs. *Merge sort*

Compare a melhor implementação do *quick sort* obtida até aqui com a melhor implementação do *merge sort* obtida no Laboratório passado. Utilize as entradas com 10M inteiros. É possível eleger o melhor algoritmo? Justifique.

## 5 Dijkstra 3-way partitioning

**Versão 8:** implemente o *quick sort top-down* com o particionamento 3-way de Dijkstra. (Veja o slide 36 da Aula 16.)

- Teste a Versão 8 vs. a versão mais eficiente obtida até agora para a entrada aleatória com 10M inteiros. Qual versão foi melhor? Justifique esse resultado.
- Repita o teste anterior para a entrada `few_uniq/10M_const_keys_unif_rand.txt`. Esse arquivo possui somente chaves entre 0 e 99. Assim, cada chave aparece em média 100K vezes no arquivo. Qual foi o resultado obtido dessa vez? Explique o que mudou.