Estrutura de Dados 2
Roteiro de Laboratório 5 – *Merge sort*
Luiza Batista Laquini - 2019107786

1 Objetivo

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

2 Configuração dos experimentos

Utilize o arquivo `input.zip` disponibilizado no AVA para testar os programas que você desenvolver neste laboratório. Este arquivo possui 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 Exercício 2 do Laboratório 3 para realizar os testes dos algoritmos de *merge sort*. Meça o tempo de execução de cada uma das variantes do *merge sort* para cada uma das entradas de teste.

3 Variantes do *merge sort*

Implemente, teste e meça o tempo de execução de cada uma das versões do *merge sort* abaixo:

- **Versão 1**: *merge sort* clássico *top-down* recursivo sem nenhuma otimização. (Veja os slides 7 e 8 da Aula 11.)
- **Versão 2**: *merge 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 15 da Aula 11.) Varie o valor de *cut-off* a partir de 1 e determine o valor ideal para a sua implementação.
- **Versão 3**: *merge sort top-down* recursivo com *merge skip*. (Veja o slide 16 da Aula 11.) 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 *merge skip*.
- **Versão 5**: *merge sort bottom-up* sem nenhuma otimização. (Veja o slide 19 da Aula 11.)
- **Versão 6**: altere a Versão 5 para implementar o que seria o "*cut-off*" na versão *bottom-up*.
- **Versão 7**: altere a Versão 6 para implementar o *merge skip* na versão *bottom-up*.

Faça uma análise das 7 versões do algoritmo segundo o desempenho. Há alguma que se destacou? Qual versão você escolheria e por quê?

R: Eu escolheria a versão 4 do algoritmo, pois, a partir da medição dos tempos que fiz, observei que todas as entradas (*nearly sorted*, *reverse sorted*, *sorted* e *unif ramdom*) obtiveram uma melhoria significativa com a implementação do *cut-off* para *insertion sort* da versão 2 do algoritmo e algumas melhoraram ainda mais com a implementação do *merge skip* da versão 3. Sendo a versão 4 a união das versões 2 e 3, seria ela a melhor escolha a se fazer. A seguir irei acrescentar 4 tabelas de tempos (uma para cada entrada) que sustentam minha argumentação.

nearly_sorted	
Versão	Tempo
1	0.089
2	0.078
3	0.025
4	0.020
5	0.081
6	0.081
7	0.081

reverse_sorted	
Versão	Tempo
1	0.087
2	0.077
3	0.087
4	0.077
5	0.082
6	0.082
7	0.082

sorted	
Versão	Tempo
1	0.085
2	0.075
3	0.016
4	0.0.14
5	0.082
6	0.082
7	0.082

unif_rand	
Versão	Tempo
1	0.174
2	0.161
3	0.175
4	0.161
5	0.173
6	0.173
7	0.173