Relatório da parte A do trabalho

bolha original
completou a ordenação com aproximadamente 25 seg com 5999000 trocas lista ultilizada l7

as versões v2 e v3 não tiveram um ganho de permanece aparente

selecao original
usando a lista x7 e instantâneo ele faz 19 trocas

Variação1: Refaça o código original para que a busca pelo menor elemento (função mínimo) e a eliminação desse menor elemento da lista a ser ordenada (função remove) ocorra numa mesma função (remove_menor), sem a necessidade de se percorrer a lista duas vezes a cada iteração.

usando a lista x7 e instantâneo ele faz 94 trocas

- Variação 2: Refaça a implementação do algoritmo Seleção usando funções genéricas (foldr ou foldr1) .

usando a lista x7 e instantâneo ele faz 19 trocas

Inserção

usando a lista x7 e instantâneo ele faz 115 comparações
usando a l7 levou alguns segundos mais foi bem rápido ele fez 2005000 comparações

Variação 1: Refaça a implementação do algoritmo Inserção usando funções genéricas (foldr ou foldr1).

usando a l7 levou alguns segundos tive a imprensão que demorou um pouco mais do que o original ele fez 6001001 comparações

quicksort
- Original Quicksort: implementação do código visto em aula.

Ele bem mais rápido, tem permanece bem melhor do que os anteriores e faz menos comparações 4001

Variação 1: modifique o algoritmo original para que ao invés dos elementos maiores e menores serem encontrados com buscas independentes, que seja elaborada e utilizada a função divide que percorre a lista uma única vez, retornando os elementos menores em uma lista e os elementos menores em outra.

Essa variação faz com que ele fique um pouco mais lento do que o original, comparações 4002000

- Variação 2: modifique a variação 1 para que o elemento pivô seja obtido a partir da análise dos 3 primeiros elementos da lista, sendo que o pivô será o elemento mediano entre eles. Exemplo: na lista [3, 9, 4, 7, 8, 1, 2], os elementos 3, 9 e 4 seriam analisados e o pivô escolhido seria 4. Caso a lista a ser analisada tenha menos que 3 elementos, o pivô é sempre o primeiro.

Essa variação faz com que ele fique um pouco mais lento do que o original, e um pouco mais rapido do que a variação anterior comparações 3998005

mergesort

sem duvidada o merge_sort o mais rapido de todos mais nos meus testes ele não consegue fazer a ordenação da l3 para frente ele da essa mensagem de erro

(*** Exception: resolucao.hs:(416,1)-(421,38): Non-exhaustive patterns in function dividT

então eu tenho conclusão se fosse para escolher para listas grande a minha implementação do marge_sort não funciona, então escolheria o quicksort