Bolha:

Pior caso: 5999000 trocas

Tempo para rodar as listas: 3 minutos

Bolha2:

Pior caso: 5999000 trocas

Tempo para rodar as listas: 2:15

Bolha3:

Pior caso: 5999000 trocas

Tempo para rodar as listas: 2:13

Conclusão:

O número de trocas não varia conforme as modificações, mas as paradas antecipadas melhoram o tempo de execução.

Selection:

Pior caso: 8006001 Comparações

Tempo para rodar as listas: 40 segundos

SelectionV2:

Pior caso: 8006001 comparações

Tempo para rodar as listas: 36 segundos

Conclusão:

Usar o foldr1 melhora o tempo de execução, mas não faz variar as trocas.

Insertion:

Pior caso: 6001001 comparações

Tempo para rodar as listas: 41 segundos

QuickSort:

Pior caso: 8004000 comparações

Tempo para rodar as listas: 32 segundos

QuickSortVar1:

Pior caso: 4006001 comparações

Tempo para rodar as listas: 46 segundos

QuickSortVar2:

Pior caso: 3998006 comparações

Tempo para rodar as listas: 46 segundos

Conculsão:

Realizar as modificações no algoritmo original para tentar otimizar o tempo resultou em atrasos, tornando o algoritmo menos eficiente.

Porém, menos comparações foram feitas com as modificações.

MergeSort:

Pior caso: 26190 comparações

Tempo para rodar as listas: 7 segundos

Conclusão:

Mergesort é um algoritmo bastante eficiente para grandes conjuntos de dados, ganhando em disparada dos algoritmos anteriores.

Obs.: O fato de o mergesort ter um tempo tão eficiente pesa na questão de armazenamento, pois esse algoritmo consome mais memória que outros métodos.