PARTE A - Algoritmos de ordenação

BUBBLE

	Original Bolha	Variação 1	Variação 2
l1	trocas 0 tempo: ~ 0.57064 s	trocas 0 tempo: Instantâneo	trocas 0 tempo: Instantâneo
l2	trocas: 1999000 tempo: ~ 0.65216 s	trocas: 1999000 tempo: ~ 0,28532 s	trocas: 1999000 tempo: ~ 0,36684 s
l3	trocas: 2000 tempo: ~0.59102 s	trocas: 2000 tempo: ~ 0,26494 s	trocas: 2000 tempo: ~ 0,26494 s
l4	trocas: 1999000 tempo: ~ 0.65216 s	trocas: 1999000 tempo: ~ 0,32608 s	trocas: 1999000 tempo: ~ 0,32608 s
l5	trocas: 4000000 tempo: ~ 2,97548 s	trocas: 4000000 tempo: ~ 1,2228 s	trocas: 4000000 tempo: ~ 1,3247 s
l6	trocas: 4000000 tempo: ~ 2,93472 s	trocas: 4000000 tempo: ~ 0,69292 s	trocas: 4000000 tempo: ~ 0,69292 s
l7	trocas: 5999000 tempo: ~ 3,34232 s	trocas: 5999000 tempo: ~ 1,24318 s	trocas: 5999000 tempo: ~ 1,4266 s

x 1	trocas: 0 tempo: instantâneo	trocas: 0 tempo: instantâneo	trocas: 0 tempo: instantâneo
x 2	trocas: 190 tempo: instantâneo	trocas: 190 tempo: instantâneo	trocas: 190 tempo: instantâneo
x 3	trocas: 100 tempo: instantâneo	trocas: 100 tempo: instantâneo	trocas: 100 tempo: instantâneo
x 4	trocas: 90 tempo: instantâneo	trocas: 90 tempo: instantâneo	trocas: 90 tempo: instantâneo
x 5	trocas: 95 tempo: instantâneo	trocas: 95 tempo: instantâneo	trocas: 95 tempo: instantâneo
x 6	trocas: 66 tempo: instantâneo	trocas: 66 tempo: instantâneo	trocas: 66 tempo: instantâneo
x 7	trocas: 94 tempo: instantâneo	trocas: 94 tempo: instantâneo	trocas: 94 tempo: instantâneo

Análise Bubble

A variação 1 do algoritmo bubble trouxe como vantagem a remoção do processamento desnecessário na situação em que a lista se encontra ordenada, por isso diminuiu consideravelmente o tempo da implementação original para a sua variação. Já em relação à variação 2 ocorreu aumento pequeno no tempo comparado a variação 1.
As trocas em todas permaneceram as mesmas.

SELECTION

	Original Selection	Variação 1	Variação 2
l1	trocas: 0 tempo: ~ 0,3528 s	trocas: 1999000 tempo: 0,2016	trocas: 0 tempo: 0,07056
l2	trocas: 1999000 tempo: ~ 0,3024 s	trocas: 0 tempo: ~ 0,06048 s	trocas: 1999000 tempo: ~ 0,22176 s
l3	trocas: 1 tempo: ~ 0,34272 s	trocas: 1999000 tempo: ~ 0,21168 s	trocas: 1 tempo: ~ 0,07056 s
l4	trocas: 1999000 tempo: ~ 0,32256 s	trocas: 2000 tempo: ~ 0,06048 s	trocas: 1999000 tempo: ~ 0,23184 s
l5	trocas: 5999 tempo: ~ 1,57248 s	trocas: 4000000 tempo: ~ 0,6048 s	trocas: 5999 tempo: ~ 0,42336 s
l6	trocas: 4002000 tempo: ~ 1,3507 s	trocas: 4000000 tempo: ~ 0,54432 s	trocas: 4002000 tempo: ~ 0,58464 s
l7	trocas: 4003999 tempo: ~ 1,4515 s	trocas: 2001000 tempo: ~ 0,39312 s	trocas: 4003999 tempo: ~ 0,6552 s
x1	trocas: 0 tempo: instantâneo	trocas: 190 tempo: instantâneo	trocas: 0 tempo: instantâneo

x2	trocas: 190 tempo: instantâneo	trocas: 0 tempo: instantâneo	trocas: 190 tempo: instantâneo
x3	trocas: 10 tempo: instantâneo	trocas: 90 tempo: instantâneo	trocas: 10 tempo: instantâneo
x4	trocas: 90 tempo: instantâneo	trocas: 100 tempo: instantâneo	trocas: 90 tempo: instantâneo
x5	trocas: 30 tempo: instantâneo	trocas: 95 tempo: instantâneo	trocas: 30 tempo: instantâneo
x6	trocas: 22 tempo: instantâneo	trocas: 124 tempo: instantâneo	trocas: 22 tempo: instantâneo
x7	trocas: 39 tempo: instantâneo	trocas: 96 tempo: instantâneo	trocas: 39 tempo: instantâneo

Análise Selection

A variação 1 do algoritmo selection trouxe como vantagem a remoção do processamento ao procurar o menor e removê-lo (percorrendo duas vezes a lista), a melhoria na variação remove o menor percorrendo a lista uma única vez, por isso diminuiu consideravelmente o tempo de processamento. Já em relação à variação 2 ocorreu aumento pequeno no tempo comparado à variação 1 e houve casos em que o tempo foi bem melhor.

INSERTION

	Original Insertion	Variação 1
l1	comparações: 1999 tempo: ~ 0 s	comparações: 1999000 tempo: Instantâneo
l2	comparações: 1999000 tempo: ~ 0,34524 s	comparações: 1999 tempo: ~ 0,27948 s
l3	comparações: 3999 tempo: ~ 0 s	comparações: 1999001 tempo: ~ 0,28496 s
l4	comparações: 1999001 tempo: ~ 0,34524 s	comparações: 3999 tempo: ~ 0 s
l5	comparações: 4002001 tempo: ~ 0,696 s	comparações: 4002001 tempo: ~ 0.70692 s
l6	comparações: 4004000 tempo: ~ 0,7562 s	comparações: 4004000 tempo: ~ 0,56444 s
l7	comparações: 6001001 tempo: ~ 1,1234 s	comparações: 2005000 tempo: ~ 0.35072 s
x1	comparações: 19 tempo: instantâneo	comparações: 190 tempo: instantâneo

x2	comparações: 190 tempo: instantâneo	comparações: 19 tempo: instantâneo
x3	comparações: 118 tempo: instantâneo	comparações: 100 tempo: instantâneo
x4	comparações: 100 tempo: instantâneo	comparações: 118 tempo: instantâneo
x5	comparações: 109 tempo: instantâneo	comparações: 105 tempo: instantâneo
x6	comparações: 83 tempo: instantâneo	comparações: 137 tempo: instantâneo
x7	comparações: 110 tempo: instantâneo	comparações: 115 tempo: instantâneo

Análise Insertion

A variação 1 do algoritmo insertion usando foldr é basicamente uma inserção ordenada que ocorre do final da lista para o começo, então as quantidades de trocas tiveram situações contrárias nas listas que eram o inverso da outra (l1, l2). Quanto ao quesito de tempo, não houve uma diferença notável entre as duas variações, já que uma é o contrário da outra.

QUICK

	Original Quick	Variação 1	Variação 2
l1	comparações: 3998000 tempo: ~ 0,39507 s	comparações: 1999000 tempo: 0,16208	comparações: 1997000 tempo: 0,18234
l2	comparações: 3998000 tempo: ~ 0,37481 s	comparações: 1999000 tempo: ~ 0,33429 s	comparações: 1997000 tempo: ~ 0,35455 s
l3	comparações: 3998002 tempo: ~ 0,37481 s	comparações: 1999001 tempo: ~ 0,17221 s	comparações: 1997002 tempo: ~ 0,17221 s
l4	comparações: 4002000 tempo: ~ 0,4052 s	comparações: 2001000 tempo: ~ 0,37481 s	comparações: 1998999 tempo: ~ 0,3039 s
l5	comparações: 8004000 tempo: ~ 0,84079 S	comparações: 4002000 tempo: ~ 0,4052 s	comparações: 3998002 tempo: ~ 0,48624 s
l6	comparações: 8004000 tempo: ~ 0,96235 s	comparações: 4002000 tempo: ~ 0,68884 s	comparações: 3998002 tempo: ~ 0,7699 s
l7	comparações: 8004000 tempo: ~ 0,85092 s	comparações: 4002000 tempo: ~ 0,75975 s	comparações: 3998002 tempo: ~ 0,56728 s
x1	comparações: 380	comparações: 190	comparações: 170

	tempo: instantâneo	tempo: instantâneo	tempo: instantâneo
x2	comparações: 380 tempo: instantâneo	comparações: 190 tempo: instantâneo	comparações: 170 tempo: instantâneo
x3	comparações: 200 tempo: instantâneo	comparações: 100 tempo: instantâneo	comparações: 100 tempo: instantâneo
x4	comparações: 200 tempo: instantâneo	comparações: 100 tempo: instantâneo	comparações: 100 tempo: instantâneo
x5	comparações: 160 tempo: instantâneo	comparações: 80 tempo: instantâneo	comparações: 80 tempo: instantâneo
x6	comparações: 168 tempo: instantâneo	comparações: 84 tempo: instantâneo	comparações: 72 tempo: instantâneo
x7	comparações: 168 tempo: instantâneo	comparações: 81 tempo: instantâneo	comparações: 72 tempo: instantâneo

Análise Quick

A variação 1 do algoritmo quick trouxe como vantagem a separação da lista em duas com os menores e maiores em relação ao pivô em uma único percorrimento da lista, o que diminuiu drasticamente o número de comparações/tempo pela metade. A variação 2 se mostrou muito parecida com a 1, tendo diminuição de tempo/comparações em determinadas situações, e o aumento em outras.

MERGE

	Original Merge	Selection Variação 1	Quick Variação 2
l1	comparações 10864 tempo: ~ 0,00835 s	trocas: 1999000 tempo: 0,2016	comparações 1997000 tempo: 0,18234
l2	comparações: 11088 tempo: ~ 0,00525 s	trocas: 0 tempo: ~ 0,06048 s	comparações: 1997000 tempo: ~ 0,35455 s
l3	comparações: 10880 tempo: ~ 0,0052 s	trocas: 1999000 tempo: ~ 0,21168 s	comparações: 1997002 tempo: ~ 0,17221 s
l4	comparações: 11102 tempo: ~ 0,0042 s	trocas: 2000 tempo: ~ 0,06048 s	comparações: 1998999 tempo: ~ 0,3039 s
l5	comparações: 25966 tempo: ~ 0,00525 s	trocas: 4000000 tempo: ~ 0,6048 s	comparações: 3998002 tempo: ~ 0,48624 s
l6	comparações: 25958 tempo: ~ 0,00625 s	trocas: 4000000 tempo: ~ 0,54432 s	comparações: 3998002 tempo: ~ 0,7699 s
l7	comparações: 26190 tempo: ~ 0,01025 s	trocas: 2001000 tempo: ~ 0,39312 s	comparações: 3998002 tempo: ~ 0,56728 s
x1	comparações: 40	trocas: 190	comparações: 170

	tempo: instantâneo	tempo: instantâneo	tempo: instantâneo
x2	comparações: 48 tempo: instantâneo	trocas: 0 tempo: instantâneo	comparações: 170 tempo: instantâneo
x3	comparações: 40 tempo: instantâneo	trocas: 90 tempo: instantâneo	comparações: 100 tempo: instantâneo
x4	comparações: 48 tempo: instantâneo	trocas: 100 tempo: instantâneo	comparações: 100 tempo: instantâneo
x5	comparações: 49 tempo: instantâneo	trocas: 95 tempo: instantâneo	comparações: 80 tempo: instantâneo
x6	comparações: 60 tempo: instantâneo	trocas: 124 tempo: instantâneo	comparações: 72 tempo: instantâneo
x7	comparações: 65 tempo: instantâneo	trocas: 96 tempo: instantâneo	comparações: 72 tempo: instantâneo

Análise Merge

No quesito de eficiência, não há dúvidas que o mergesort vence em tempo e em comparações, pois a sua definição recursiva se encaixou muito bem no paradigma funcional. O Quicksort também possui uma definição recursiva, porém o fato do uso da concatenação para unir o pivô aos menores e maiores, aumentou consideravelmente a complexidade do algoritmo. O algoritmo selection conseguiu se sobressair em questão de

tempo em relaçao ao quick na maioria, mas isso ocorreu devido ao gargalo da concatenação no quick.

Sobre as análises do tempo de processamento dos algoritmos

Para obter os resultados utilizamos o recurso de profiling do haskell que gera um arquivo .prof que dá as estatísticas do tempo de execução do arquivo, na função main apenas forçamos o “evaluate” das listas (pois foram construídas por enumeração e o haskell precisaria avaliar toda vez que executasse o algoritmo, isso deixou o arquivo mais performático em geral pois evitou de avaliar a lista a cada execução de cada função), todos os arquivos estão na pasta “benchmark” .

Analisando os resultados do tempo percebemos que o pior algoritmo de todos em questão de tempo é o Bubble Sort e o melhor é o Merge, isso se deve a natureza recursiva do Mergesort, que diferente do do Quicksort que também é recursivo não necessita concatenar a lista a todo momento.

Quanto a comparação das variações do Mergesort com o melhor do Insertion e o melhor do Quick tivemos que descobrir a melhor versão dos dois últimos, para isso somamos os tempos de execução das variação 1 e 2, pois em ambos algoritmos algumas listas eram mais rápidas na variação 1 e outras na variação 2, então para decidir o melhor algoritmo entre as duas variações somamos todos os tempos das listas, obtivemos os seguintes resultados:

- Selectionsort Variação 1: 1,53216 segundos
- Selectionsort Variação 2: 2,2579 segundos
- Quicksort Variação 1: 2,8972 segundos
- Quicksort Variação 2: 2,8364 segundos