Segundo Trabalho de Programação Funcional – 2020/1

Mateus Rocha Resende

Análise de algoritmos – exercícios 1 a 5

Exercício 1 – Bubble Sort)

1. O número de trocas foi **o mesmo** nos três algoritmos em todas as listas.
2. É possível observar que, em uma lista já ordenada (l1), os algoritmos nas variações 1 e 2 são mais eficientes que o original;
Em listas não ordenadas, a variação 1 e o original são executados com pouca diferença de tempo;
Nas listas l3, l5 e l7, em que é necessário mover elementos muitas vezes, a variação 2 é muito ineficiente;
A variação 2 é a mais eficiente na lista l6, uma vez que usa menos tempo e metade da memória por verificar se o resto da lista já está ordenado.

Exercício 2 – Selection Sort)

1. O número de trocas foi **o mesmo** nos três algoritmos em todas as listas, a quantidade de trocas é igual ao tamanho da lista.
2. O tempo de execução foi significantemente menor nas variações 1 e 2;
Os três algoritmos tomam muito tempo para organizar listas grandes.

Exercício 3 – Insertion Sort)

1. O número de comparações foi **o mesmo** nos dois algoritmos em todas as listas.
2. Foi possível notar uma pequena redução no tempo de processamento de listas maiores (l5, l6, l7) na variação 1.

Exercício 4 – Quick Sort)

1. O algoritmo original e a variação 1 realizaram o mesmo número de comparações, a variação 2 realizou menos.
2. Não houve mudança aparente entre o tempo de processamento da variação 1 e 2, porém foram executados em menos tempo que o algoritmo original.

Exercício 5 – Merge Sort)

1. O Mergesort é o que menos realiza comparações e o Quicksort é o que mais realiza.
2. O Mergesort é mais rápido em todas as situações;
O Selectionsort é o menos viável para listas grandes, muito devagar;
O Quicksort tem performance próxima à do Mergesort, porém um pouco mais lenta.

> especificações do sistema:
> CPU: AMD Ryzen 5 3600 @ 4GHz
> RAM: 16GB DDR4 @ 2800MHz
> OS: Win10 64-bit
> GHC 9.0.1

===== BUBBLE SORT =====

bolha l1 (3.68 secs, 3,046,286,560 bytes)
bolha l2 (4.36 secs, 3,260,702,944 bytes)
bolha l3 (3.75 secs, 3,049,458,064 bytes)
bolha l4 (4.42 secs, 3,263,564,400 bytes)
bolha l5 (17.31 secs, 12,679,672,080 bytes)
bolha l6 (17.65 secs, 12,677,050,640 bytes)
bolha l7 (16.85 secs, 12,890,708,944 bytes)

bolha1 l1 (0.13 secs, 7,081,744 bytes)
bolha1 l2 (4.97 secs, 3,260,801,456 bytes)
bolha1 l3 (3.81 secs, 3,049,826,088 bytes)
bolha1 l4 (4.49 secs, 3,263,663,096 bytes)
bolha1 l5 (17.16 secs, 12,680,040,104 bytes)
bolha1 l6 (16.33 secs, 12,677,050,848 bytes)
bolha1 l7 (17.82 secs, 12,890,774,688 bytes)

bolha2 l1 (1.24 secs, 745,330,912 bytes)
bolha2 l2 (4.02 secs, 2,771,899,704 bytes)
bolha2 l3 > 60 secs
bolha2 l4 (3.84 secs, 2,774,084,408 bytes)
bolha2 l5 > 60 secs
bolha2 l6 (10.87 secs, 6,984,335,352 bytes)
bolha2 l7 > 60 secs

=== SELECTION SORT ===

*listas 'x' foram utilizadas por ser um algoritmo recomendado para pequenos conjuntos

map selecao [x1,x2,x3,x4,x5,x6,x7] (1.49 secs, 960,015,232 bytes)

map selecao1 [x1,x2,x3,x4,x5,x6,x7] (0.01 secs, 976,560 bytes)

map selecao2 [x1,x2,x3,x4,x5,x6,x7] (0.01 secs, 951,480 bytes)

=== INSERTION SORT ===

insercao l1 (0.14 secs, 9,251,112 bytes) 1999
insercao l2 (1.17 secs, 976,821,144 bytes) 1999
insercao l3 (0.14 secs, 10,201,240 bytes) 2000
insercao l4 (1.20 secs, 977,431,104 bytes) 2000
insercao l5 (2.66 secs, 1,988,792,576 bytes) 4000
insercao l6 (2.49 secs, 1,956,160,040 bytes) 4000
insercao l7 (3.77 secs, 2,956,463,296 bytes) 4000

insercao1 l1 (0.13 secs, 7,406,464 bytes) 1999
insercao1 l2 (0.95 secs, 492,757,136 bytes) 1999
insercao1 l3 (0.13 secs, 7,906,400 bytes) 2000
insercao1 l4 (0.96 secs, 492,758,960 bytes) 2000
insercao1 l5 (1.94 secs, 1,016,567,376 bytes) 4000
insercao1 l6 (1.94 secs, 986,984,456 bytes) 4000
insercao1 l7 (2.77 secs, 1,502,416,144 bytes) 4000

```
===== QUICK SORT =====

quicksort l1 (1.33 secs, 377,032,976 bytes) 2003000
quicksort l2 (1.28 secs, 655,978,400 bytes) 2003000
quicksort l3 (1.34 secs, 377,003,536 bytes) 2003003
quicksort l4 (1.25 secs, 656,287,384 bytes) 2005002
quicksort l5 (2.62 secs, 754,580,632 bytes) 4010002
quicksort l6 (2.71 secs, 1,332,887,320 bytes) 4010002
quicksort l7 (2.50 secs, 1,333,083,928 bytes) 4010002

quicksort1 l1 (0.75 secs, 121,272,360 bytes) 2003000
quicksort1 l2 (0.91 secs, 332,831,072 bytes) 2003000
quicksort1 l3 (0.75 secs, 121,275,008 bytes) 2003003
quicksort1 l4 (0.89 secs, 333,011,400 bytes) 2005002
quicksort1 l5 (1.56 secs, 242,356,520 bytes) 4010002
quicksort1 l6 (1.81 secs, 692,821,416 bytes) 4010002
quicksort1 l7 (1.71 secs, 693,018,032 bytes) 4010002

quicksort2 l1 (0.74 secs, 121,161,192 bytes) 2001002
quicksort2 l2 (0.90 secs, 332,477,800 bytes) 2001002
quicksort2 l3 (0.73 secs, 121,164,000 bytes) 2001006
quicksort2 l4 (0.97 secs, 332,624,616 bytes) 2003003
quicksort2 l5 (1.45 secs, 242,021,432 bytes) 4006006
quicksort2 l6 (1.86 secs, 692,244,544 bytes) 4006006
quicksort2 l7 (1.88 secs, 692,441,152 bytes) 4006006

===== MERGE SORT =====

map mergeSort [x1,x2,x3,x4,x5,x6,x7] (0.01 secs, 831,808 bytes)
map mergeSort [l1,l2,l3,l4,l5,l6,l7] (1.36 secs, 243,452,712 bytes)

mergeSort l1 (0.15 secs, 23,746,216 bytes) 1999
mergeSort l2 (0.14 secs, 23,776,144 bytes) 1999
mergeSort l3 (0.14 secs, 23,724,656 bytes) 2000
mergeSort l4 (0.13 secs, 23,626,824 bytes) 2000
mergeSort l5 (0.27 secs, 50,296,976 bytes) 3999
mergeSort l6 (0.27 secs, 50,047,992 bytes) 3999
mergeSort l7 (0.27 secs, 49,295,912 bytes) 4000
```