Trabalho Pr?tico de Matem?tica Discreta

November 6, 2018

Trabalho Prático de Matemática Discreta: Análise de Algoritmos de Ordenação

Membros	Nome
01	Juan Manoel
02	Robin Hoodix
03	Yang Jin Samara cavalari
04	Gustavo Jeromine
05	Vladimir da Silva Borges

0.0.1 1 - Introdução

Deve conter: * Uma breve descrição do objetivo do trabalho * O que são algoritmos de ordenação, o que fazem, como funcionam, onde podem ser aplicados, importância etc * Uma breve introdução aos algoritmos escolhidos

0.0.2 2 - Descrição e Análise do Algoritmo 1

Deve conter: * Descrição completa do primeiro algoritmo escolhido: nome, origem, estratégia usada, e como funciona * Estrutura do algoritmo em pseudocódigo * Citações para a descrição do algoritmo e pseudocódigo

```
In [22]: import time
```

```
In [68]: def merge(llist, rlist):
                final = []
                while llist or rlist:
                        if len(llist) and len(rlist):
                                if llist[0] < rlist[0]:
                                        final.append(llist.pop(0))

                                else:
                                        final.append(rlist.pop(0))


                        if not len(llist):
                                        if len(rlist):
                                              final.append(rlist.pop(0))
```

```
                    if not len(rlist):
                        if len(llist):
                            final.append(llist.pop(0))


        return final

    def merge_sort(list):
        """
        Sort the list passed by argument with merge.
        """
        if len(list) < 2: return list
        mid = len(list) / 2
        return merge(merge_sort(list[:mid]), merge_sort(list[mid:]))
```

```
In [72]: data01 = []
         for p in range(10):
          ini01 = time.time()
          merge_sort([5,3,1,7,2,9,8])
          fim01 = time.time()
          print "Tempo de Execução: ", fim01-ini01
          tempo01 = fim01-ini01
          data01.append(tempo01)
```

```
Tempo de Execução:  4.91142272949e-05
Tempo de Execução:  4.41074371338e-05
Tempo de Execução:  0.000198125839233
Tempo de Execução:  4.31537628174e-05
Tempo de Execução:  0.000176906585693
Tempo de Execução:  4.31537628174e-05
Tempo de Execução:  4.29153442383e-05
Tempo de Execução:  4.10079956055e-05
Tempo de Execução:  4.29153442383e-05
Tempo de Execução:  3.69548797607e-05
```

0.0.3 3 - Descrição e Análise do Algoritmo 2

Deve conter: * Descrição completa do segundo algoritmo escolhido: nome, origem, estratégia usada, e como funciona * Estrutura do algoritmo em pseudocódigo * Citações para a descrição do algoritmo e pseudocódigo

```
In [66]: def bubble_sort(lista):
             elementos = len(lista)-1
             ordenado = False
             while not ordenado:
                  ordenado = True
```

```
                for i in range(elementos):
                    if lista[i] > lista[i+1]:
                        lista[i], lista[i+1] = lista[i+1],lista[i]
                        ordenado = False
                        #print(lista)
            return lista
```

```
In [71]: data02 = []

         for p in range(10):
          ini = time.time()
          bubble_sort([5,3,1,7,2,9,8])
          fim = time.time()
          print "Tempo de Execução: ",fim-ini
          tempo2 = fim-ini
          data02.append(tempo2)
```

```
Tempo de Execução:  9.05990600586e-06
Tempo de Execução:  8.10623168945e-06
Tempo de Execução:  8.10623168945e-06
Tempo de Execução:  6.91413879395e-06
Tempo de Execução:  7.15255737305e-06
Tempo de Execução:  6.91413879395e-06
Tempo de Execução:  6.91413879395e-06
Tempo de Execução:  6.19888305664e-06
Tempo de Execução:  5.96046447754e-06
Tempo de Execução:  5.96046447754e-06
```

```
In [73]: print data02
         print data01
```

```
[9.059906005859375e-06, 8.106231689453125e-06, 8.106231689453125e-06, 6.9141387939453125e-06,
[4.9114227294921875e-05, 4.410743713378906e-05, 0.00019812583923339844, 4.315376281738281e-05,
```

0.0.4 4 - Análise Assintótica Comparativa

Deve conter: * Representação em Ozão ou Theta da complexidade do algoritmo 1 no melhor e pior caso em relação ao tempo de execução * Representação em Ozão ou Theta da complexidade do algoritmo 2 no melhor e pior caso em relação ao tempo de execução * Discussão a respeito de qual algoritmo é o mais eficiente

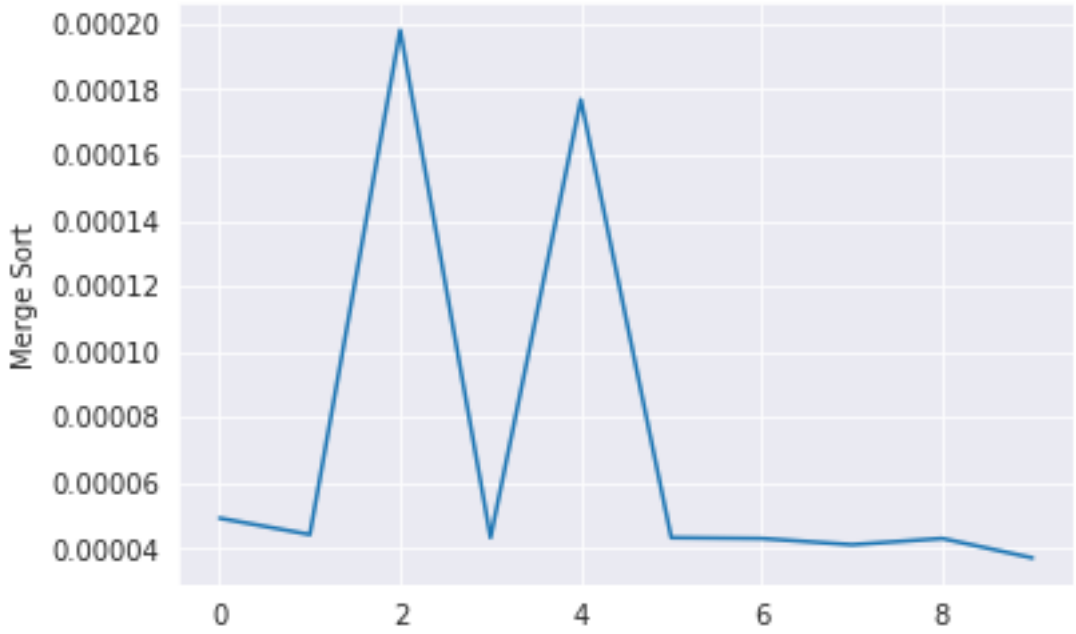
0.0.5 5 - Análise Experimental Opcional

Esta seção opcional deve conter: * A descrição da configuração da análise experimental conduzida: implementação utilizada, configurações da máquina utilizada, tamanhos de entrada utilizados, tipo de dados usados como entrada, e método utilizado de quantificação de tempo/operações primitivas. * Gráfico comparando ambas curvas de desempenho obtidas pelos algoritmos de ordenação escolhidos.

0.0.6 Plot Grafico de Desempenho Merge Sort

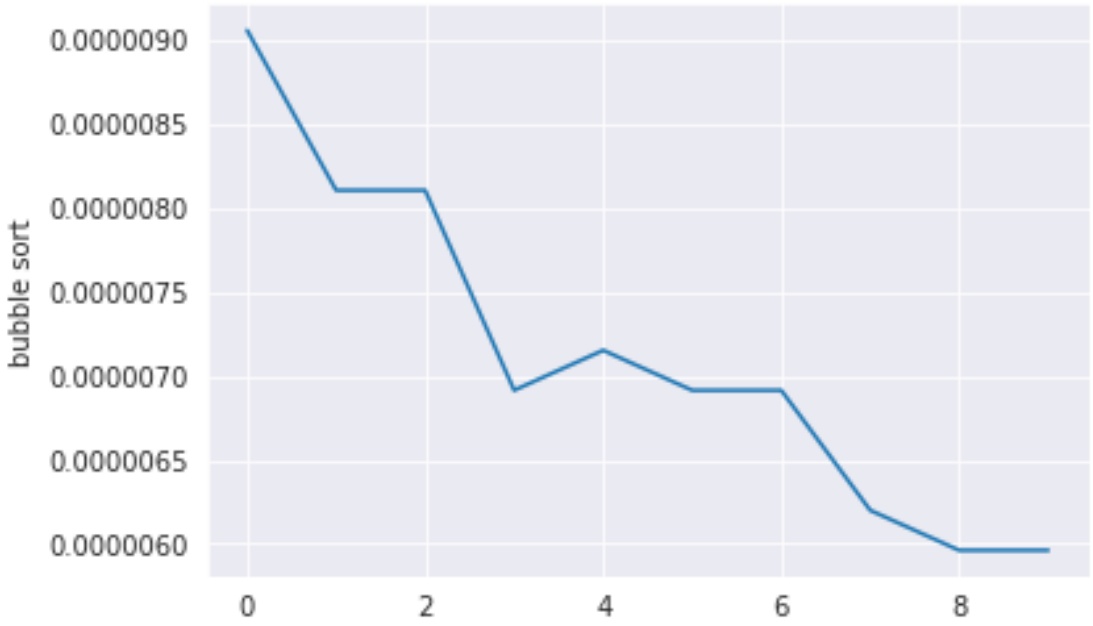
```
In [45]: import matplotlib.pyplot as plt
         import seaborn as sns
         %matplotlib inline
```

```
In [74]: plt.plot(data01)
         plt.ylabel('Merge Sort')
         plt.show()
```



0.0.7 Plot Grafico de Desempenho bubble sort

```
In [75]: plt.plot(data02)
         plt.ylabel('bubble sort')
         plt.show()
```



0.0.8 6 - Referências

Esta seção deve conter uma linha para cada referência utilizada. Exemplo:

Lee, J., & Yeung, C. Y. (2018). Personalizing Lexical Simplification. In Proceedings of the 27th International Conference on Computational Linguistics. Mancini, P. (2011). Leader, president, person: Lexical ambiguities and interpretive implications. European Journal of Communication, 26(1). Saggion, H. (2018). LaSTUS/TALN at Complex Word Identification (CWI) 2018 Shared Task. In Proceedings of the Thirteenth Workshop on Innovative Use of NLP for Building Educational Applications