Estrutura de Dados e Algoritmos

Ano Letivo 2020/2021 – 2.º Semestre

Trabalho realizado por:

Sebastião Manuel Inácio Rosalino, n.º 98437, turma CDA1

Trabalho 3 – EDA

Questão 1. Estudo do algoritmo de Shellsort

Criado por Donald Shell em 1959, o Shellsort é o mais eficiente algoritmo no que toca aos algoritmos de complexidade quadrática.

Este algoritmo é tido como bastante eficiente e é baseado no algoritmo Insertionsort. Este algoritmo destaca-se por apresentar maior eficiência em conjuntos de médio a elevado tamanho, evitando grandes deslocamentos que não seriam possíveis de evitar no caso do Insertionsort, nomeadamente se o mínimo estiver no extremo direito e necessitar de ser movido para o extremo esquerdo.

Tem como ideias de funcionamento principais: i) a divisão do conjunto de elementos a ordenar em sublistas (uma ordenada e a outra desordenada); ii) o cálculo do valor de intervalo (gap) entre cada uma das listas (esse gap deverá descer à medida que a lista vai sendo ordenada); e iii) a aplicação do algoritmo Insertionsort. O Shellsort repete este processo até que o conjunto de elementos fique totalmente ordenado.

Devido a sua complexidade possui excelentes desempenhos em conjuntos muito grandes, podendo, inclusive, ser melhor que o Mergesort em determinadas situações.

Questão 2. Criação da Classe Ordena

A Classe Ordena está criada no código disponibilizado no ficheiro: Trabalho_3_EDA_Sebastião_Rosalino_98437.py

Através de funções de Python, foram criados seguintes algoritmos de ordenação de listas: Bubblesort; Selectionsort; Insertionsort; Mergesort; Quicksort; Shellsort e Python Sorted.

Questão 3.

a) Lista dos tempos médios de execução

Processo:

1.º - Criar listas com valores inteiros gerados aleatoriamente e a variar no conjunto {1, 2, ..., 10000};

2.º - A partir do procedimento anterior, são geradas listas (num total de 10) com "n" número de elementos a variar em {50, 100, . . . , 500}, com intervalos (passo) de 50;

3.º - Cronometrar e guardar os tempos médios de 30 execuções para cada um dos algoritmos da classe Ordena, incluindo o algoritmo nativo do Python (Python Sorted).

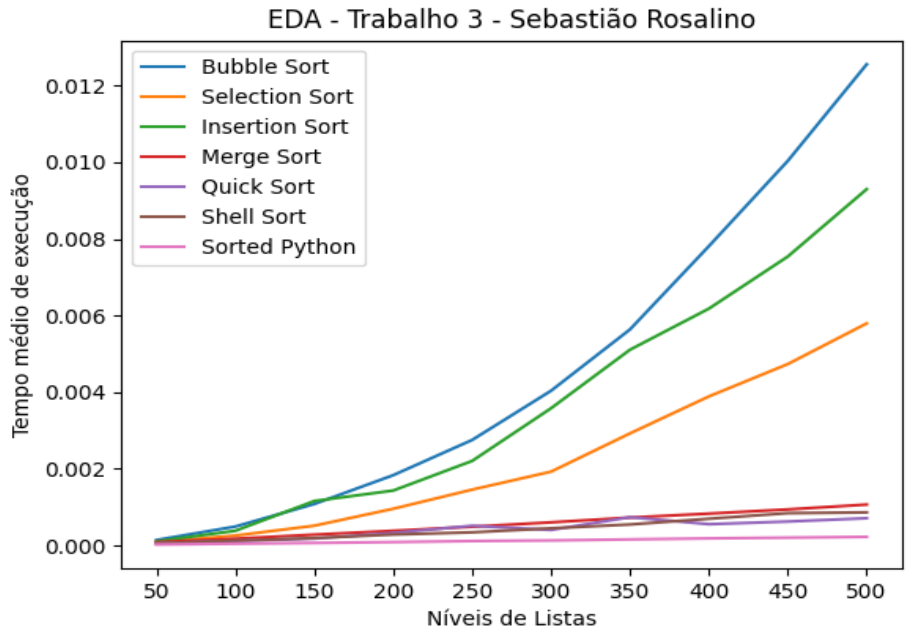
Resultado:

Algoritmo de ordenação	Tempo (seg) de execução das listas a variar em {50, 100, ..., 500} (passo 50) elementos									
	50	100	150	200	250	300	350	400	450	500
Bubblesort	0,0002532	0,0010948	0,0029827	0,0043022	0,0142017	0,0122504	0,0094059	0,0148991	0,0158264	0,0173235
Selectionsort	0,0001781	0,0007356	0,0020894	0,0028788	0,0051685	0,0094450	0,0144328	0,0079095	0,0066609	0,0079944
Insertionsort	0,0001993	0,0008959	0,0029228	0,0072949	0,0136656	0,0181084	0,0068830	0,0083690	0,0107906	0,0133335
Mergesort	0,0001546	0,0003119	0,0004959	0,0006932	0,0008965	0,0010189	0,0013731	0,0017321	0,0030084	0,0049078
Quicksort	0,0002857	0,0005963	0,0008778	0,0011369	0,0014433	0,0015565	0,0011250	0,0011456	0,0009509	0,0014302
Shellsort	0,0001009	0,0002162	0,0003817	0,0005455	0,0006321	0,0010472	0,0018949	0,0015613	0,0013224	0,0019095
Python Sorted	0,0000381	0,0000726	0,0001129	0,0001381	0,0001715	0,0002664	0,0002871	0,0003815	0,0004962	0,0005553

Como antecipado, o algoritmo de ordenação mais eficiente é o Python Sorted, seguido, por ordem decrescente de eficiência, pelo Quicksort, Shellsort, Mergesort, Selectionsort, Insertsort e, por último, pelo Bubblesort, que confirmou ser o menos eficiente entre todos.

Nota-se que, entre o Shellsort e Mergesort há algumas alternâncias de posição, em termos de eficiência, ao longo do tempo de execução das listas.

b) Gráfico de visualização dos tempos médios de execução



A análise do gráfico, em escala linear, confirma os resultados transpostos para o quadro da alínea anterior, permitindo verificar que o Bubblesort e Insertsort se destacam pela negativa e que, com o aumento do número de elementos nas listas, as diferenças nos tempos de execução se vão alargando significativamente.

c) Investigue qual o método(s) que estará a usar com o sort() pré-implementado do Python.

O Python Sorted foi criado por Tim Peters (razão pela qual é muitas vezes denominado Timsort) em 2002 para ser usado como o algoritmo de ordenação padrão e predefinido da linguagem Python (desde a versão 2.3).

A funcionalidade sorted em Python é um algoritmo de ordenação baseado nos algoritmos Insertionsort e Mergesort. Nesse sentido, o Python Sorted é considerado um algoritmo de ordenação misto e, por isso, apresenta-se como um dos mais eficientes e mais rápidos que se encontram disponíveis, na medida em que faz a combinação das principais vantagens de eficiência que o Insertionsort e o Mergesort oferecem.

Tem como ideias de funcionamento a divisão do conjunto de elementos em blocos designados por Run's. Em sequência, procede à ordenação desses Run's usando o Insertionsort, um por um, aplicando seguidamente o Mergesort a cada Run. Se o tamanho do conjunto de dados for menor do que o Run, o conjunto será ordenado usando apenas o Insertionsort.

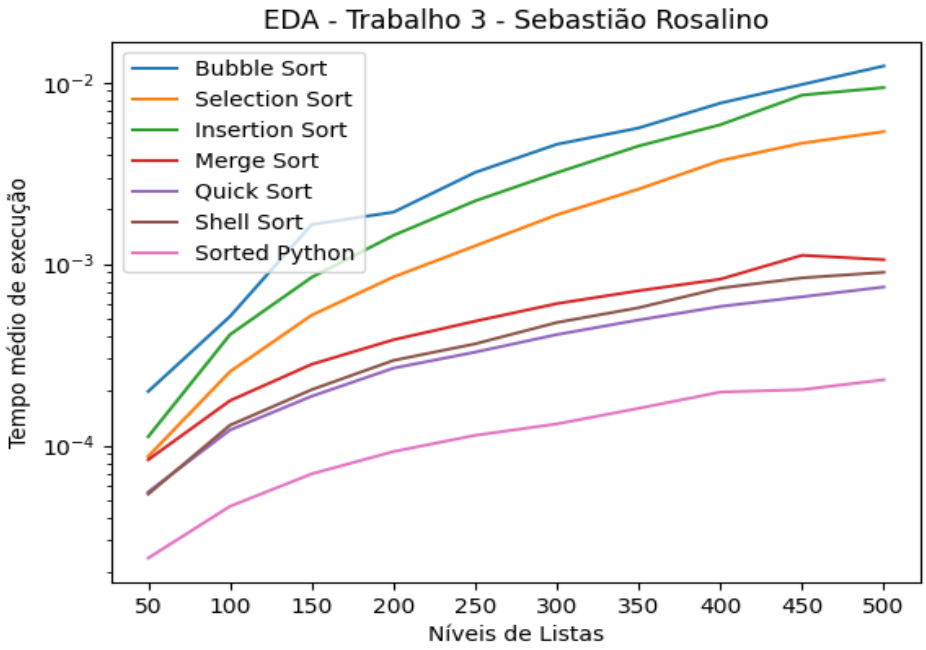
Uma das grandes vantagens é a enorme eficiência que resulta da utilização do Mergesort quando o conjunto apresenta uma dimensão de média a elevada ordem, e a rapidez do Insertionsort no que toca a dimensões pequenas de conjuntos de dados.

A combinação destes dois algoritmos confere Python Sorted uma elevada eficiência, numa lógica de utilização do melhor do que cada um daqueles algoritmos pode oferecer.

Assume as seguintes classificações nas variadas ordens de complexidade:

- O(n) – Melhor caso
- O(n log n) – Pior caso
- O(n log n) – Caso Médio

d) Gráfico de visualização dos tempos médios de execução em escala logarítmica



A análise do gráfico, em escala logarítmica, permite observar em maior detalhe a representação dos dados tratados, oferecendo maior amplitude nos intervalos entre os algoritmos.

Com este aumento de escala de visualização é possível perceber que todos os algoritmos, mesmo os mais eficientes, apresentam tendências de crescimento no tempo de execução das listas em função do aumento do número de elementos que a integram.

No entanto, os resultados apresentados comprovam as conclusões já expostas nas alíneas anteriores quanto à eficácia relativa de cada um dos algoritmos face aos demais.

e) Usando o conhecimento obtido em 3.c), discuta os resultados observados em 3.b) e 3.d), fazendo a ligação com as ordens de complexidade teóricas conhecidas.

Ordem de complexidade dos algoritmos estudados:

Algoritmo	Ordem de Complexidade		
	Melhor caso	Médio caso	Pior caso
Bubble sort	O(n)	$O(n^2)$	$O(n^2)$
Insertion sort	O(n)	$O(n^2)$	$O(n^2)$
Selection sort	$O(n^2)$	$O(n^2)$	$O(n^2)$
Shell sort	O(n log n)	O(n log n)	O(n log n)
Merge sort	O(n log n)	O(n log n)	O(n log n)
Quick sort	O(n log n)	O(n log n)	$O(n^2)$
Python Sorted	O(n)	O(n log n)	O(n log n)

Complexidade quadrática

Complexidade poli-logarítmica

Os algoritmos analisados podem classificar-se em duas ordens de complexidade: o Bubblesort, Selectionsort e o Insertionsort são de complexidade quadrática e os restantes - Mergesort, Quicksort, e Python Sorted - são de complexidade poli-logarítmica.

Em geral, a ordem de complexidade é considerada para o pior caso.

Os algoritmos com complexidade quadrática têm como característica principal o facto de os elementos serem analisados ou percorridos duas vezes ao longo da sua execução. Ou seja, possuem no seu código duas funcionalidades cíclicas (ciclos While ou For, por exemplo) e, em geral, são algoritmos pouco eficientes, com aplicações limitadas à resolução de pequenos problemas.

Por sua vez, os algoritmos com complexidade poli-logarítmica são significativamente mais eficientes, já que possuem como paradigma central de funcionamento a divisão e o tratamento dos elementos em sucessivos subconjuntos do todo inicial, utilizando a técnica "*divide to conquer*".

Através da visualização dos gráficos podemos concluir que à medida que o volume de dados aumenta, o tempo de execução dos algoritmos de ordem poli-logarítmica apresenta um crescimento constante, mas a um ritmo bastante mais lento, em comparação com os algoritmos de ordem quadrática.

O gráfico em escala logarítmica permite, pela maior amplitude de análise, perceber claramente um alinhamento de posicionamento, em termos de eficiência, entre os algoritmos de ordem quadrática, por um lado, e os algoritmos de ordem poli-logarítmica por outro. Os primeiros apresentam um pior desempenho ao longo de todo o gráfico, com relativa proximidade entre si. Os segundos apresentam-se claramente mais eficientes, com grande estabilização ao longo da curva de desempenho.

O gráfico em escala linear, embora permita diferenciar claramente os algoritmos mais eficientes dos menos eficientes, não oferece uma capacidade de perceção analítica dos respetivos desempenhos, sobretudo entre os que têm uma ordem de complexidade poli-logarítmica (que podem ser melhor observados na escala logarítmica).

Por último, é possível identificar que o Python Sorted é o algoritmo mais eficiente (e isso percebe-se claramente na escala logarítmica). Este algoritmo apresenta uma ordem de eficiência poli-logarítmica tanto no médio, como no pior caso. Acresce que, no melhor caso, este consegue atingir a ordem linear.

Tal é possível visualizar nos gráficos apresentados nas alienas 3.b) e 3.d). A razão pela qual isto acontece, como explicado na aliena 3.c), deve-se ao facto deste algoritmo combinar “o melhor dos dois mundos” dos algoritmos Mergesort e Insertionsort.