ipluso INSTITUTO POLITÉCNICO DA LUSOFONIA

DWDM/AICD/EIA

1º semestre 2022/2023

Prof.: Carlos Limão

Trabalho Prático 4

Pesquisa Binária e Algoritmos de Ordenação

Introdução

Vimos que, em termos do tempo de execução, a pesquisa binária é francamente melhor do que a pesquisa linear. Por esse motivo, os algoritmos de ordenação têm um papel importante no aumento da eficiência das tarefas de pesquisa. No entanto, vimos também que os algoritmos de ordenação têm aplicação frequente noutras áreas, como por exemplo, para:

- Permitir o processamento de dados numa determinada ordem.
- Tornar mais eficiente a junção de várias sequências.

Objetivos

Os objetivos deste trabalho são, por um lado, melhorar a compreensão dos algoritmos estudados nas aulas teóricas e, por outro, verificar na prática as conclusões tiradas, em termos da notação assimptótica, para os algoritmos de pesquisa linear estudados.

Exercício 1 - Algoritmos de Ordenação

Neste exercício vamos medir experimentalmente os tempos de execução dos quatro algoritmos de ordenação que estudámos nas aulas teóricas para confirmarmos as conclusões que retirámos após realizarmos a sua análise recorrendo à notação assimptótica.

Se precisar, reveja os *slides* da aula em que analisámos o desempenho destes algoritmos. Em resumo, as conclusões quanto ao tempo de execução foram as seguintes:

Selection Sort: $\Theta(n^2)$ (ou seja, proporcional a $\mathbf{n}^2$ no melhor e no pior caso)

Insertion Sort: $\Theta(n^2)$ no pior caso e $\Theta(n)$ no melhor caso

Merge Sort: $\Theta(n.\lg n)$ para todos os casos

Quicksort: $\Theta(n.\lg n)$ para os casos típicos (melhores fatores constantes que o Merge Sort)

Passo 1 - Criação de um projeto

Vamos começar por criar um novo projeto no PyCharm chamado SortAlgorithms.

Passo 2 - Adicionar ficheiros Python ao projeto

Neste projeto vamos começar por adicionar um ficheiro por cada um dos algoritmos a implementar:

- Um ficheiro Python chamado ***selectionsort.py***.
- Um ficheiro Python chamado ***insertionsort.py***.
- Um ficheiro Python chamado ***mergesort.py***.
- Um ficheiro Python chamado ***quicksort.py***.

Vamos também adicionar:

- O ficheiro Python chamado ***profile.py*** cujo conteúdo foi **fornecido** na aula anterior.
- Um ficheiro Python chamado ***profile_sort.py***.

Pergunta 1: Mostre o ecrã resultante, após a adição destes ficheiros ao projeto.

> ⚠ Não se esqueça de instalar as *packages* **plotly** e **pandas**, usadas pelo ficheiro *profile.py*.

Passo 3 - Escrita dos programas

Em cada um dos ficheiros ***selectionsort.py***, ***insertionsort.py***, ***mergesort.py***, e ***quicksort.py*** deverá escrever as respetivas funçõe necessárias para implementar os referidos algoritmos, tal como estudados na aula teórica.

> ⚠ Não se esqueça de incluir *docstrings* (*documentation strings*) descrevendo os **parâmetros de entrada** e a **saída** de cada função.

Em cada um destes módulos deverá escrever um pequeno programa de teste, semelhante ao seguinte, que permita validar rapidamente se o algoritmo (simbolizado pela função sort_algorithm) está corretamente implementado.

```python
if __name__ == '__main__':
    import random

    LIST_LEN = 30

    arr = random.sample(range(1, LIST_LEN + 1), LIST_LEN)

    print("Array desordenado: ")
    for item in arr:
        print(item, end=" ")

    sort_algorithm(arr, len(arr))

    print("\nArray ordenad: ")
    for item in arr:
        print(item, end=" ")
```

Ficheiro ***profile.py***

Este é o módulo Python fornecido juntamente no trabalho anterior. Tal como anteriormente, apenas terá de **copiar este ficheiro para o seu projeto**.

Ficheiro ***profile_sort.py***

Neste ficheiro deverá escrever um programa que, tirando partido dos anteriores (importando-os), gerará os gráficos com o resultado da análise experimental de cada um dos algoritmos de pesquisa linear.

> ℹ Note que pode executar as 4 avaliações em sucessão no mesmo programa, como ilustrado de seguida. Nesse caso serão gerados 4 gráficos independentes:

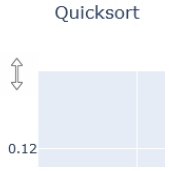
```python
ns = [10000, 20000, 30000, 40000, 50000, 60000, 70000, 80000, 90000]
profile.profile_algorithm(test_selectionsort, ns, 'Selection Sort', False, )
profile.profile_algorithm(test_insertionsort, ns, 'Insertion Sort', False, )
profile.profile_algorithm(test_mergesort, ns, 'Merge Sort', False, )
profile.profile_algorithm(test_quicksort, ns, 'Quicksort', False)
```

Passo 4 - Execução do programa

Tenha em atenção que deverá escolher valores de **n** que permitam executar os 4 algoritmos num intervalo de tempo razoável.

> ⚠ Lembre-se que, para que os valores sejam minimamente fiáveis deverá garantir que no melhor caso (*quicksort*) o tempo de execução seja próximo de (ligeiramente inferior a) 1 segundo para o mais pequeno **n**. Isto porque os algoritmos *selectionsort* e *insertionsort* são **bastante mais** demorados.

Pergunta 2: Mostre os gráficos gerados para cada um dos 4 algoritmos avaliados e explique-os à luz da análise realizada na aula teórica. Explique, em particular, se os resultados da análise experimental confirmam ou não os resultados esperados na sequência da análise teórica.

> ⓘ Para que os gráficos resultantes da avaliação de diferentes algoritmos sejam facilmente **comparáveis**, estes podem ser redimensionados de forma a que todos possuam uma escala semelhante no eixo dos **Y**. Para isso, basta colocar o cursor no lado esquerdo do topo do eixo até que o cursor mude para o formato de redimensionamento, e ajustar.
>
> 
>

Pergunta 3: Na submissão do trabalho, deve incluir, num ficheiro com extensão .zip, um diretório **exercício1** contendo todos os módulos Python que escreveu para este exercício.

> ⚠ Como sempre, deve ter em atenção que todos os ficheiros escritos por si devem inclui um cabeçalho com as características indicadas no enunciado do 1º trabalho prático.

Exercício 2 - Algoritmo de Pesquisa Binária

Neste exercício vamos medir experimentalmente o tempo de execução do algoritmo de pesquisa binária. Recorde que, em termos de notação assimptótica vimos que este algoritmo é **O(lg n)**.

Passo 5 - Criação de um projeto

Vamos começar por criar um novo projeto no PyCharm chamado BinarySearch.

Passo 6 - Adicionar ficheiros Python ao projeto

Neste projeto vamos adicionar os seguintes ficheiros:

- Um ficheiro Python chamado ***bsearch.py***.
- Um ficheiro Python chamado ***lsearch.py***.
- Um ficheiro Python chamado ***profile.py***.
- Um ficheiro Python chamado ***profile_bsearch.py***.

Pergunta 4: Mostre o ecrã resultante, após a adição destes ficheiros ao projeto.

> ⚠ Não se esqueça de instalar as *packages* **plotly** e **pandas**, usadas pelo ficheiro *profile.py*.

Passo 7 - Escrita dos programas

> ⚠ Não se esqueça de incluir *docstrings* (*documentation strings*) descrevendo a funcionalidade, os **parâmetros de entrada** e a **saída** de cada função.

Em cada um dos ficheiros, deve escrever o seguinte código:

Ficheiro ***lsearch.py***

A implementação dos algoritmos de **pesquisa linea**r que realizou no trabalho anterior.

Ficheiro b***search.py***

Uma implementação do algoritmo de **pesquisa binária**, tal como estudado na aula teórica.

Ficheiro ***profile.py***

Este é o módulo Python fornecido juntamente no trabalho anterior. Tal como anteriormente, apenas terá de **copiar este ficheiro para o seu projeto**.

Ficheiro ***profile_bsearch.py***

Neste ficheiro deve escrever um programa que permita usar o módulo ***profile.py*** para comparar a execução dos algoritmos de pesquisa linear e de pesquisa binária.

Na escrita deste ficheiro tenha em atenção que o *array* a fornecer ao algoritmo de pesquisa binária deve estar ordenado. O quinto parâmetro da função ***profile_algorithm()*** permite especificar se o *array* deve ou não ser previamente ordenado antes de invocar o algoritmo a avaliar. Por defeito **não é** ordenado.

```
ns = [1000000, 2000000, 3000000, 4000000, 5000000, 6000000, 7000000, 8000000, 9000000]
profile.profile_algorithm(test_bsearch, ns, 'Pesquisa Binária', False, True)
profile.profile_algorithm(test_lsearch, ns, 'Pesquisa Linear', False)
```

> ⚠ Como sempre, deve ter em atenção que todos os ficheiros escritos por si devem inclui um cabeçalho com as características indicadas no enunciado do 1º trabalho prático.

Passo 8 - Execução do programa

Pergunta 5: Mostre os gráficos gerados para a avaliação da pesquisa linear e da pesquisa binária e explique-os à luz da análise realizada na aula teórica. Explique, em particular, se os resultados da análise experimental confirmam ou não os resultados esperados, na sequência da análise teórica.

Pergunta 6: Na submissão do trabalho, deve incluir, num ficheiro com extensão .zip, um diretório **exercício2** contendo todos os módulos Python que escreveu para este exercício.

Conclusão do Trabalho

A entrega do trabalho faz-se submetendo, no Moodle, um ficheiro **PDF**, devidamente identificado (aluno, curso, disciplina, trabalho, data), com as **respostas** às perguntas colocadas, eventualmente justificadas com cópias dos ecrãs.

Caso tal seja solicitado, deve submeter também os ficheiros que contêm os eventuais programas que tenham sido pedidos no enunciado. Neste caso os ficheiros devem ter nomes sugestivos, que indiquem claramente a que se referem (p.e.: algoritmoX.py ou exercícioY.py admitindo que são ficheiros com programas em Python).

Notas:

1. Não se limite a usar imagens do ecrã inteiro e esperar que o professor encontre nelas a resposta correta ou a justificação para a sua resposta!
2. Use uma imagem **indicando claramente** (sublinhado, círculo, etc.) a **resposta** ou a **justificação** para uma resposta que deu previamente, consoante o caso.
3. Por outro lado, recortes muito pequenos podem não permitir perceber a que contexto se referem. Os recortes devem incluir o contexto suficiente para que se perceba a que se referem e de onde foram retirados.

Dica: Pode usar a **Ferramenta de Recorte** (**Snip & Sketch** em Inglês) para cortar facilmente zonas do ecrã que queira usar nas suas respostas.