UNIVERSIDADE DE COIMBRA
FACULDADE DE CIÊNCIAS E TECNOLOGIA

Departamento de Engenharia Informática

Projeto #1
Algoritmos e Estruturas de Dados

2023-2024 – 2º Semestre

Submissão de relatório e código (InforEstudante): 19 de fevereiro 23:59

Anotações: *Em anexo template do relatório a ser entregue no InforEstudante.*

É incentivado que os alunos discutam ideias e questões relativas ao trabalho proposto, mas é entendido que quer a reflexão final sobre os resultados obtidos, quer o código desenvolvido, são da autoria de cada estudante.

Objetivos:

Com o desenvolvimento deste projeto pretende-se que o aluno consolide os conhecimentos sobre a importância da análise de complexidade O-Grande de um algoritmo e a viabilidade das implementações propostaws. Na análise de complexidade vamo-nos concentrar no fator tempo**.**

Tarefas:

A. Implementação de três algoritmos para o mesmo problema com complexidades diferentes.

B. Análise empírica de complexidade dos três algoritmos.

Conceitos: Algoritmos

Um algoritmo deve ter as seguintes propriedades (Donald E. Knuth, The Art of Computer Programming, vol. 1, 3rd edition, 2016):

- ***Finiteness*****:** deve terminar num número finito de passos;
- ***Definiteness:*** todos os passos de um algoritmo devem ser definidos com rigor;
- ***Input*****:** deve receber zero ou mais dados à entrada;
- ***Output*****:** deve retornar um ou mais resultados à saída;
- ***Effectiveness*****:** todos os passos devem ser possíveis de resolver sem ambiguidade em tempo finito com os recursos disponíveis.

Conceitos: Análise de Complexidade

Análise teórica: análise da complexidade de um algoritmo *a priori* sem ser necessária a sua implementação ou execução. Vamos falar sobre isto nas sessões teóricas.

Análise empírica: análise do comportamento empírico de uma implementação do algoritmo *a posteriori* com base na recolha de dados.

Problema a usar neste trabalho: Encontrar a soma na lista.

Dada uma sequência de inteiros verifique que existe um par de números **distintos** cuja soma é igual a um inteiro *k*.

Exemplos:

Input: [1, 5, 11, 5], k=16

Output: True

Uma vez que 5 + 11 = 16

Input: [80, 98, 83, 92, 1, 38, 37, 54, 58, 89], k=181

Output: True

Uma vez que 92 + 89 = 181

Entrada:

Como entrada o programa deve receber uma lista com os elementos da sequência do tipo inteiro.

Saída:

Como saída o programa deve devolver se encontrou um par de valores ou não.

Algoritmo 1: Solução exaustiva

Resolver o problema recorrendo a uma solução exaustiva, ou seja, ter dois ciclos para verificar todos os pares de valores possíveis.

Algoritmo 2: Solução c/ ordenamento

Recorra a uma solução que comece por fazer o ordenamento da sequência. Tire partido deste ordenamento para construir a solução.

Nota: *utilize uma função de ordenamento disponível na linguagem de programação utilizada (C:* `qsort`*, C++:* `sort`*, Java:* `Collections.sort`*, Python:* `list.sort` *ou* `sorted`*).*
Não esqueça que a complexidade temporal deste algoritmo também depende da complexidade do algoritmo de ordenamento utilizado.

Algoritmo 3: Solução elaborada

Encontre uma solução que seja superior em termos de complexidade temporal. Esta solução não recorre ao ordenamento. A ideia desta solução é que como é apenas um par de valores, a cada número que procuramos é apenas necessário saber se a sequência possui o valor que somado ao atualmente considerado resulta no valor procurado.

Tarefa A

Implementar as diferentes soluções.

Tarefa B

Análise empírica das soluções implementadas (ver exemplo nos slides da aula teórica sobre análise de complexidade). Recomendações a ter em conta:

- Medir o tempo de execução das soluções para sequências de diferentes tamanhos (ex. 10000, 20000, ... 100000).
- Organizar os resultados das medições num gráfico (identificar os eixos do gráfico e as unidades utilizadas).
- No gráfico mostrar o resultado da regressão escolhida para sumariar os valores obtidos.

Relatório

O relatório deve seguir o *template* disponibilizado, incluindo:

- Gráficos com as medições e resultado da regressão para cada solução desenvolvida.
- Reflexão crítica sobre o resultado da regressão e possíveis *outliers.*
- Análise de complexidade com base nos resultados empíricos obtidos.