UNIVERSIDADE DE COIMBRA
FACULDADE DE CIÊNCIAS E TECNOLOGIA

*Departamento de Engenharia Informática*

# Trabalho Prático #2
# Algoritmos e Estruturas de Dados

## 2019-2020 – 2º Semestre

**Submissão no Mooskak:**
até 15mn do final da última aula P#2

***Anotações:*** *esta ficha foi preparada para ser resolvida maioritariamente no espaço das duas sessões práticas. O código deve ser submetido até 15mn do final da segunda sessão prática.*

*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. Procedimentos contrários ao que é dito acima, nomeadamente cópia de código desenvolvido por colegas ou obtido da net é entendido como fraude. Para além de a fraude denotar uma grave falta de ética e constituir um comportamento não admissível num estudante do ensino superior e futuro profissional licenciado, esta prejudica definitivamente o processo de aprendizagem do infrator.*

*A avaliação do trabalho realizado incide sobre o desenvolvido na sessão prática presencial.*

**Objetivos:**

Pretende-se que o aluno consolide conhecimentos sobre a importância da complexidade O-Grande de um algoritmo na viabilidade ou não da respetiva implementação. Na análise de complexidade vamo-nos concentrar no fator tempo**.**

**Tarefas:**

**A.** Análise de complexidade de dois algoritmos para o mesmo problema

**B. e C.** Implementação e aplicação de estruturas de dados.

---

**Conceitos: Algoritmos**

Um algoritmo deve ter as seguintes propriedades:
(www.tutorialspoint.com/data_structures_algorithms/algorithms_basics.htm)

- **não ser ambíguo:** conjunto de instruções claro e sem ambiguidades;
- **entradas (dados):** deve ter ZERO ou mais dados de entrada bem definidos;
- **saídas (resultados):** deve ter UM ou mais resultados bem definidos;
- **finitude:** deve terminar ao fim de um número finito de passos;
- **viabilidade:** deve ser viável com os recursos disponíveis;
- **independência:** deve definir uma execução passo a passo, independente da linguagem usada.

**Conceitos: Análise de Complexidade**

**Análise *a priori*:** é uma análise teórica da complexidade do algoritmo, efetuada antes de este ser implementado e corrido… vamos falar sobre isto nas sessões teóricas.

**Análise *a posteriori*:** é uma análise empírica da complexidade do algoritmo, realizada sobre uma implementação do mesmo. A implementação é corrida e um conjunto de estatísticas são recolhidas.

---

**Problema a usar neste trabalho: Elemento maioritário**

Dado um *array*, determinar se existe um elemento maioritário e imprimi-lo se existir. Se não existir, imprimir "Sem elemento.". Um elemento maioritário num *array* A[] de tamanho n, é um elemento que apareça mais de n/2 vezes (pelo que há no máximo um elemento).

**Exemplos :**
Input : {3, 3, 4, 2, 4, 4, 2, 4, 4}
Output : 4

Input : {3, 3, 4, 2, 4, 4, 2, 4}
Output : Sem elemento.

Como entrada o programa (quer para a tarefa A quer para a B) recebe uma linha com o número de elementos do *array*, exemplo:

9[\n]

seguido de uma linha com elementos do *array* do tipo int na gama [0 .. 9999999]. Todos os valores são separados por espaço, com exceção do último valor em cada linha, a que se segue \n, exemplo:

34 2 21 298765 3 4 45 54 54[\n]

**ENTRADA**
3[\n]
2 6 6[\n]
**SAÍDA**
6[\n]

**ENTRADA**
3[\n]
2 6 8[\n]
**SAÍDA**
Sem elemento.[\n]

**ENTRADA**
11[\n]
2 4 4 6 8 8 0 0 1 5 2 5 4 6 6 2 8 0 9 5[\n]
**SAÍDA**
Sem elemento.[\n]

**Solução exaustiva**

Resolver o problema recorrendo a uma solução exaustiva, ou seja, ter dois ciclos e ter registo do número de vezes que cada elemento diferente aparece. Se esse número de vezes ultrapassar n/2 podemos parar a contagem e devolver o elemento em causa. Caso tal não aconteça então não existe elemento maioritário.

**Solução melhorada**

Resolver o problema recorrendo a uma solução melhorada. Uma possibilidade é resolvemos o problema com a ajuda da representação binária dos números presentes na matriz. A tarefa é encontrar o elemento que aparece mais de n/2 vezes, ou seja, que aparece mais do que todos os outros números combinados.

*Se um número é maioritário os bits que o representam a "1" terão necessariamente de aparecer em número maioritário, e serão os únicos nessa condição. (caso esta afirmação lhe ofereça dúvidas discuti-la quer com o docente quer com os seus colegas)*

Partindo da representação binária de cada número do *array*, contamos para cada posição binária no *array* as ocorrências de "1"s. Para as posições binárias em que a contagem de "1"s for superior a n/2 consideramos essa posição a "1" na representação do elemento maioritário. Esta abordagem funciona porque, para todos os outros números combinados, a contagem de bits definida não pode ser maior que n/2, pois o elemento maioritário está presente mais de n/2 vezes.

Vamos ver com a ajuda do exemplo:

```
Input : {3, 3, 4, 2, 4, 4, 2, 4, 4}
Representação binária:

3 - 0 1 1
3 - 0 1 1
4 - 1 0 0
2 - 0 1 0
4 - 1 0 0
4 - 1 0 0
2 - 0 1 0
4 - 1 0 0
4 - 1 0 0
----------
  - 5 4 0
```

Aqui n=9, então n/2 = 4 e apenas o terceiro bit a contar da direita satisfaz a contagem>4, sendo definido como elemento maioritário.

Portanto, nosso elemento majoritário é 1 0 0, que é 4.

Notem que essa abordagem funciona quando o elemento maioritário está presente no *array*, mas se não estiver, podemos incorrer em erro.
Vamos ver com a ajuda deste exemplo:

```
Input : {3, 3, 6, 2, 4, 4, 2, 4}
Representação binária:

3 – 0 1 1
3 – 0 1 1
6 – 1 1 0
2 – 0 1 0
4 – 1 0 0
4 – 1 0 0
2 – 0 1 0
4 – 1 0 0
----------
  – 4 5 0
```

Aqui n é 8, então n/2 = 4 e apenas o segundo bit a contar da direita satisfaz a contagem>4 sendo, portanto, considerado o elemento maioritário. Desta forma, o elemento maioritário seria 0 1 0, que é 2. Mas, na verdade, o 2 não está presente no *array*.

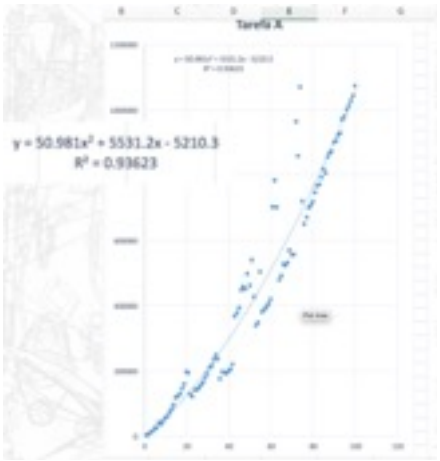
Para garantir que estas situações são despistadas o que é feito é para o elemento identificado como maioritário por este método fazer uma passagem pelo *array* para contar efetivamente quantas vezes ele ocorre. Só se ocorrer mais de n/2 vezes podemos então identificá-lo como maioritário.

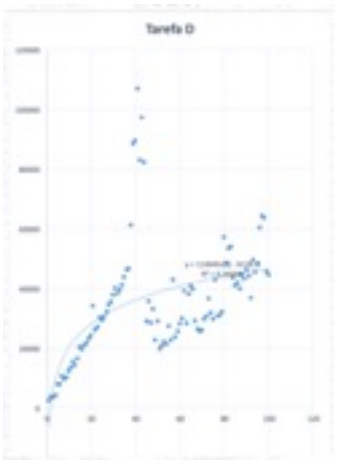
---

**Tarefa A:** *tem como objetivo central fazer a análise a posteriori dos algoritmos a implementar*.

**Exemplo de uma análise a posteriori para a implementação de um algoritmo estudado nas sessões teóricas.**

| A | B | C | D |
|---|---|---|---|
| 2966 | 27634 | 22244 | 2439 |
| 4319 | 4299 | 1972 | 2446 |
| 8949 | 6166 | 2140 | 3393 |
| 13255 | 9178 | 2640 | 4347 |
| 16833 | 14594 | 3213 | 8452 |
| 23732 | 17680 | 4943 | 8934 |
| 30356 | 17714 | 4150 | 7533 |
| 36447 | 20465 | 6096 | 9912 |
| 45170 | 24069 | 5128 | 9618 |
| 56300 | 29916 | 7030 | 12147 |
| 65112 | 31172 | 10108 | 13743 |
| 73251 | 36880 | 6772 | 12996 |
| 82780 | 40225 | 8342 | 14114 |
| 96433 | 44051 | 9206 | 16627 |
| 117062 | 48169 | 9569 | 17806 |
| 120369 | 52890 | 10361 | 20007 |
| 128716 | 56173 | 10829 | 20749 |
| 146861 | 61141 | 16189 | 35066 |
| 159141 | 68488 | 12429 | 23810 |
| 212484 | 70017 | 12428 | 24791 |



As figuras acima representam, da esquerda para a direita, uma tabela com valores, neste caso em ms, da corrida de um programa nas versões A a D para um conjunto de dados de dimensão crescente (indicados em abcissa nos gráficos central e à direita); a figura central representa os resultados e a função f(N) para a versão A. A figura à direita representa os resultados, e a função f(N) para a versão D.

Conclui-se neste caso que a versão A tem complexidade assimptótica, de acordo com a análise empírica realizada, de $O(N^2)$ e a versão D tem complexidade O(N logN).

**Relatório (a submeter no inforestudante):**

O relatório a realizar sobre esta parte do trabalho (com base no formulário disponibilizado) deve incluir e ter em conta as seguintes recomendações:

- definir a escala (ex. linear, logarítmica) de valores para as dimensão dos dados de entrada;
- não esquecer de colocar no eixo das ordenadas a unidade de tempo utilizada
- incluir gráficos para a solução exaustiva e melhorada de dimensões que permitam fácil leitura, sem ocupar demasiado espaço
- refletir e elaborar sobre possíveis razões para o aparecimento de *outliers* (ex. os que aparecem no gráfico à direita na figura)
- concluir sobre em que medida os resultados obtidos são ou não os esperados, justificando a conclusão.

**É esperado que no final da realização do trabalho o aluno:**

- tenha boa perceção do impacto da complexidade O-grande na viabilidade de um algoritmo;
- reconheça que quando temos uma solução inviável em termos de complexidade temporal (e/ou espacial) precisamos de revisitar o desenho do algoritmo e estruturas de dados utilizadas.
- saiba distinguir entre análise *a priori* e *a posteriori* e tenha adquirido experiência na preparação e obtenção de conclusões de uma análise *a posteriori.*

**Formato do relatório:**

O relatório, deve fazer uso do *template disponibilizado também no infoestudante* e que compreende os seguintes pontos:

- Gráfico Excel incluindo regressão para a solução exaustiva.
- Gráfico Excel incluindo regressão para a solução melhorada.

- Conclusões sobre a complexidade O-grande de cada algoritmo e sucinta análise crítica do resultado.
- Reflexão crítica sobre possíveis *outliers* (caso estes apareçam na sua análise empírica).

---

**Tarefa B:** *Implementar e submeter no Mooshak a solução exaustiva*

---

**Tarefa C:** *Implementar e submeter no Mooshak uma solução melhorada*