AULA 4 - ANÁLISE DA COMPLEXIDADE DE ALGORITMOS

1 – Considere uma sequência (*array*) de n elementos inteiros, ordenada por **ordem não decrescente**. Pretende-se determinar se a sequência é uma **progressão aritmética de razão 1**, i.e., a[i+1] – a[i] = 1.

- Implemente uma função **eficiente** (utilize um algoritmo em lógica negativa) e **eficaz** que verifique se uma sequência com n elementos (n > 1) define uma sequência contínua de números. A função deverá devolver 1 ou 0, consoante a sequência verificar ou não essa propriedade.
  Depois de validar o algoritmo apresente-o no verso da folha.

- Determine experimentalmente a **ordem de complexidade do número de adições/subtrações** efetuadas pelo algoritmo e envolvendo elementos da sequência. Considere as seguintes 10 sequências de 10 elementos inteiros, todas diferentes, e que cobrem as distintas situações possíveis de execução do algoritmo. Determine, para cada uma delas, se satisfaz a propriedade e qual o número de operações de adição/subtração efetuadas pelo algoritmo.

1	3	4	5	5	6	7	7	8	9	Resultado	0	N° de operações	1
1	2	4	5	5	6	7	8	8	9	Resultado	0	N° de operações	2
1	2	3	6	8	8	8	9	9	9	Resultado	0	N° de operações	3
1	2	3	4	6	7	7	8	8	9	Resultado	0	N° de operações	4
1	2	3	4	5	7	7	8	8	9	Resultado	0	N° de operações	5
1	2	3	4	5	6	8	8	9	9	Resultado	0	N° de operações	6
1	2	3	4	5	6	7	9	9	9	Resultado	0	N° de operações	7
1	2	3	4	5	6	7	8	8	9	Resultado	0	N° de operações	8
1	2	3	4	5	6	7	8	9	9	Resultado	0	N° de operações	9
1	2	3	4	5	6	7	8	9	10	Resultado	1	N° de operações	9

Depois da execução do algoritmo responda às seguintes questões:

- Qual é a sequência (ou as sequências) que corresponde(m) ao melhor caso do algoritmo?

A primeira sequência, quando falha a progressão na primeira comparação.

- Qual é a sequência (ou as sequências) que corresponde(m) ao pior caso do algoritmo?

A 2 últimas, a progressão aritmética falha na última comparação ou não falha.

- Determine o número de adições efetuadas no caso médio do algoritmo (**para n = 10**).

$9p+(1-p)\times 5$, se $p=\frac{1}{2}$, $4.5+2.5=7$

- Qual é a ordem de complexidade do algoritmo?

$O(n)$

- Determine formalmente a ordem de complexidade do algoritmo nas situações do melhor caso, do pior caso e do caso médio, considerando uma sequência de tamanho n. Tenha em atenção que deve obter expressões matemáticas exatas e simplificadas. **Faça as análises no verso da folha.**

- Calcule o valor das expressões para n = 10 e compare-os com os resultados obtidos experimentalmente.

Para n=10 o melhor caso é 1 e o pior 9, que vai de acordo aos resultados experimentais

NOME: José Costa **Nº MEC:** 92996

APRESENTAÇÃO DO ALGORITMO

```c
#include <stdio.h>

int main()
{
    int list[] = {9,10,11,12,13,14,15,16,17,18};
    int size = 10;
    int i;
    int ops = 0;
    for(i = 1; i < size;i++){
        if(!((list[i] - list[i-1]) == 1)){
            printf("%d",ops);
            return 0;
        }
        ops++;
    }
    printf("%d",ops);
    return 1;
}
```

ANÁLISE FORMAL DO ALGORITMO

MELHOR CASO - B(N) = 1

PIOR CASO - W(N) = $n - 1$

Melhor Caso:

Na primeira comparação (dos 2 primeiros valores) a diferença é maior que 1, logo só efetuamos 1 comparação.

Pior Caso:

No Pior Caso o algoritmo tem que efetuar todas as comparações, provando assim que se trata de uma progressão aritmética de razão 1 com n-1 comparações

CASO MÉDIO - A(N) = $P(n-1) + (1-P) \times \sum_{i=1}^{n-1} (i) \times \frac{1}{n-1} = P(n-1) + (1-P) \times \frac{n}{2}$

Caso Médio:

$$P \times (n-1) + (1-P) \times \left(\sum_{i=1}^{n-1} (i) \times \frac{1}{n-1} \right) = A(n) \iff A(n) = P(n-1) + (1-P) \times \frac{n(n-1)}{2(n-1)}$$
$$\iff A(n) = P(n-1) + (1-P) \times \left(\frac{n}{2}\right)$$

-> Sucesso de probabilidade P com n-1 comparações

-> Insucesso de probabilidade (1-P) em que o insucesso pode ocorrer da 1ª comparação à n-1 comparação e multiplicado por $\frac{1}{n-1}$ pois esta é a igual probabilidade de o insucesso ocorrer da 1ª à n-1 comparação.

NOME: José Luís Nº MEC: 92996

2 – Considere uma sequência (array) não ordenada de n elementos inteiros. Pretende-se eliminar os elementos repetidos existentes na sequência, sem fazer uma pré-ordenação e sem alterar a posição relativa dos elementos. Por exemplo, a sequência { 1, 2, 2, 2, 3, 3, 4, 5, 8, 8 } com 10 elementos será transformada na sequência { 1, 2, 3, 4, 5, 8 } com apenas 6 elementos. Por exemplo, a sequência { 1, 2, 2, 2, 3, 3, 3, 3, 8, 8 } com 10 elementos será transformada na sequência { 1, 2, 3, 8 } com apenas 4 elementos. Por exemplo, a sequência { 1, 2, 3, 2, 1, 3, 4 } com 7 elementos será transformada na sequência { 1, 2, 3, 4 } com apenas 4 elementos. Mas, a sequência { 1, 2, 5, 4, 7, 0, 3, 9, 6, 8 } permanece inalterada.

- Implemente uma função **eficiente** e **eficaz** que elimina os elementos repetidos numa sequência com n elementos (n > 1). A função deverá ser *void* e alterar o valor do parâmetro indicador do número de elementos efetivamente armazenados na sequência (que deve ser passado por referência).
  Depois de validar o algoritmo apresente-o no verso da folha.

- Determine experimentalmente a **ordem de complexidade do número de comparações** e **do número de deslocamentos** envolvendo elementos da sequência. Considere as sequências anteriormente indicadas de 10 elementos e outras à sua escolha. Determine, para cada uma delas, a sua configuração final, bem como o número de comparações e de deslocamentos efetuados.

Depois da execução do algoritmo responda às seguintes questões:

- Indique uma sequência inicial com 10 elementos que conduza ao **melhor caso do número de comparações** efetuadas. Qual é a sequência final obtida? Qual é o número de comparações efetuadas? Qual é o número de deslocamentos (i.e., cópias) de elementos efetuados?

Inicial	9	9	9	9	9	9	9	9	9	9	N° de comparações	9
Final	9										N° de cópias	1

Justifique a sua resposta: Só necessita de comparar a totalidade do array com o 9 adicionado, como vai ser sempre igual só vai ser percorrido 1 vez

- Indique uma sequência inicial com 10 elementos que conduza ao **pior caso do número de comparações** efetuadas. Qual é a sequência final obtida? Qual é o número de comparações efetuadas? Qual é o número de deslocamentos (i.e., cópias) de elementos efetuados?

Inicial	8	7	10	20	99	11	13	1	2	3	N° de comparações	45
Final	8	7	10	20	99	11	13	1	2	3	N° de cópias	10

Justifique a sua resposta: Como todos os valores são diferentes, todos precisam de ser comparados com os próximos e adicionados pois são todos diferentes

- Determine formalmente a ordem de complexidade do algoritmo nas situações do **melhor caso** e do **pior caso**, considerando uma sequência de tamanho n. Tenha em atenção que deve obter expressões matemáticas exatas e simplificadas. **Faça as análises no verso da folha.**

NOME: José Luís **Nº MEC:** 92996

APRESENTAÇÃO DO ALGORITMO

```c
#include <stdio.h>

void removeDuplicates(int* list, int *size)
{
    int troca = 0;
    int comp = 0;
    int i;
    int p;
    int c = 0;
    int d = 0;
    int newSize = 0;
    for (i = 0; i < *size; i++)
    {
        for (p = 0; p < newSize; p++){
            comp++;
            if(list[i] != list[p]){
                c++;
            }
        }
        if(c == newSize){
            list[d] = list[i];
            d++;
            newSize++;
            troca++;
        }
        c = 0;
    }
    *size = newSize;
    printf("\ncomp: %d\nCopias: %d\nsize: %d\n", comp, troca, *size);
}

int main()
{
    int list[] = {9, 9, 9, 9, 9, 9, 9, 9, 9, 9};
    int size = 10;
    removeDuplicates(list, &size);
    printf("\nArray elements after deleting duplicates : ");
    for (int s = 0; s < size; s++)
    {
        printf("%d | ", list[s]);
    }
    printf("\n");
    return 1;
}
```

ANÁLISE FORMAL DO ALGORITMO

Nº DE COMPARAÇÕES

MELHOR CASO - B(N) = $(n-1)$ → compara o array e todos são iguais, logo só percorre 1 vez

PIOR CASO - W(N) = $\sum_{i=0}^{n-1}(i) = \frac{n(n-1)}{2}$ → Compara cada posição do array com as posições seguintes

Nº DE DESLOCAMENTOS DE ELEMENTOS (Cópias)

MELHOR CASO - B(N) = 1 , adicionar o único elemento que é igual no array inteiro

PIOR CASO - W(N) = n , adicionar os elementos do array que são todos distintos