# Prácticas de Programación
PEC4 - 20221

## Fecha límite de entrega: **20 / 12 / 2022**

## Formato y fecha de entrega

La PEC debe entregarse antes del día **20 de diciembre de 2022** a las 23:59.

Es necesario entregar un fichero en formato **pdf** con las respuestas a todos los ejercicios. El documento debe indicar en su primera página el **nombre y apellidos** del estudiante que hace la entrega.

Es necesario hacer la entrega en el apartado de entregas de EC del aula de teoría antes de la fecha de entrega.

## Objetivos

- Adquirir los conceptos teóricos explicados sobre los métodos de búsqueda.
- Interpretar los algoritmos de búsqueda, pudiendo simular su funcionamiento dada una entrada.
- Implementar cualquier método de búsqueda en un algoritmo y evaluar su rendimiento.
- Adquirir los conceptos teóricos explicados sobre las técnicas de análisis de algoritmos.
- Analizar la complejidad de una función, calculando su función de tiempo de ejecución y expresar esta complejidad con notación asintótica.

# Criterios de corrección:

Cada ejercicio tiene asociada su puntuación sobre el total de la actividad. Se valorará tanto que las respuestas sean correctas como que también sean completas.

- No seguir el **formato de entrega**, tanto por lo que se refiere al **tipo y nombre de los ficheros** como al contenido solicitado, comportará una **penalización importante** o la calificación con una **D de la actividad**.
- En los ejercicios en que se pide lenguaje algorítmico, se debe respetar el **formato.**
- En el caso de ejercicios en lenguaje C, estos **deben compilar para ser evaluados**. Si compilan, se valorará:
    - Que **funcionen** tal como se describe en el enunciado.
    - Que se respeten los **criterios de estilo y** que el código esté **debidamente comentado**.
    - Que las **estructuras** utilizadas sean las correctas.

# Aviso

Aprovechamos para recordar que **está totalmente prohibido copiar en las PECs** de la asignatura. Se entiende que puede haber un trabajo o comunicación entre los estudiantes durante la realización de la actividad, pero la entrega de esta debe que ser individual y diferenciada del resto. Las entregas serán analizadas con **herramientas de detección de plagio.**

Así pues, las entregas que contengan alguna parte idéntica respecto a entregas de otros estudiantes serán consideradas copias y todos los implicados (sin que sea relevante el vínculo existente entre ellos) suspenderán la actividad entregada.

**Guía citación:** https://biblioteca.uoc.edu/es/contenidos/Como-citar/index.html
**Monográfico sobre plagio:**
http://biblioteca.uoc.edu/es/biblioguias/biblioguia/Plagio-academico/

# Observaciones

Esta PEC continúa el proyecto que se desarrolla durante las distintas actividades del semestre.

En este documento se utilizan los siguientes símbolos para hacer referencia a los bloques de diseño y programación:

 Indica que el código mostrado es en **lenguaje** algorítmico.

 Indica que el código mostrado es en **lenguaje** C.

 Muestra la ejecución de un programa en **lenguaje** C.

# Enunciado

En la PEC2 se introdujo el concepto de disponibilidad (availability) de los trabajadores de una ONG. En esta PEC vamos a entrar un poco más en detalle en la gestión de esta disponibilidad y trabajaremos algunos conceptos relacionados con el cálculo de complejidad algorítmica, y los algoritmos de búsqueda.

## Ejercicio 1: Conceptos sobre búsqueda y complejidad [20%]

En la PEC2 implementamos los siguientes tipos de datos para guardar la disponibilidad:

```
const
      MAX_AVAILABILITY: integer = 100;
end const

type
      tAvailability = record
            person : tPerson;
            start : tDate;
            end : tDate;
      end record

      tAvailabilityData = record
            elems : vector [MAX_AVAILABILITY] of tAvailability;
            count : integer;
      end record

end type
```

Considerando esta implementación, y que los registros de disponibilidad están **ordenados utilizando el documento** de la persona, responde a las siguientes preguntas, razonando tu respuesta:

**a)** **[5%]** ¿Se puede buscar de forma eficiente la disponibilidad de una persona en una lista ***tAvailabilityData*** mediante la búsqueda binaria?

Sí, al tener los elementos ordenados por el documento de la persona, podemos utilizar la búsqueda binaria.

Sin embargo, dado que una persona puede estar disponible en uno o más períodos de tiempo (tener disponibilidad en distintos rangos de fechas), después de localizar una disponibilidad con la búsqueda binaria, sería necesario emprender una búsqueda lineal a la izquierda y a la derecha del elemento localizado para encontrar la disponibilidad completa de una persona.

**b)** **[5%]** En caso de que una persona no tenga ninguna disponibilidad registrada, ¿sería igual de eficiente comprobarlo con el algoritmo de búsqueda secuencial que con el de búsqueda binaria?

Si el elemento no se encuentra en el vector y éste contiene N elementos, en el peor caso de búsqueda secuencial el número de iteraciones que se deben realizar es N, mientras que en el peor caso de la búsqueda binaria serán $\log_2(N)+1$. Por tanto, si consideramos el peor caso la búsqueda binaria realizará menos iteraciones y será más eficiente.

**c) [10%]** Calcula la complejidad de las siguientes funciones de tiempo de ejecución utilizando la notación asintótica y ordénalas de más eficiente a menos eficiente:

1) $T(n) = n^2 + 2 + 20n$
2) $T(n) = 45^2 + \log(50)$
3) $T(n) = 25 + 5n$
4) $T(n) = n + 5n + 2^{n+2}$
5) $T(n) = n \cdot \log_2(5n)$
6) $T(n) = 100n + 50$

Las complejidades computacionales ordenadas son:

| | | |
|---|---|---|
| 2) | $O(1)$ | constante |
| 3) | $O(n)$ | lineal |
| 6) | $O(n)$ | lineal |
| 5) | $O(n \log(n))$ | cuasi-lineal |
| 1) | $O(n^2)$ | cuadrática |
| 4) | $O(2^n)$ | exponencial |

## Ejercicio 2: Algoritmos de búsqueda [30%]

Hemos ordenado la disponibilidad de los trabajadores (***tAvailabilityData***) por documento de identidad para facilitar la búsqueda por persona. No obstante, en este ejercicio, en lugar de trabajar con el vector de elementos ***tAvailability***, vamos a trabajar con un vector de identificadores numéricos (enteros).

Así pues, dado el siguiente vector **ordenado**:

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 5 | 10 | 24 | 25 | 28 | 29 | 40 | 42 | 55 | 59 | 60 | 63 |

**a) [5%]** Aplica el algoritmo de **búsqueda lineal** para encontrar el elemento 40.

```
El elemento 40 se encuentra en la posición 7 del
vector:

índice 1:   5 ≠ 40
índice 2:  10 ≠ 40
índice 3:  24 ≠ 40
índice 4:  25 ≠ 40
índice 5:  28 ≠ 40
índice 6:  29 ≠ 40
índice 7:  40 = 40
```

**b) [5%]** Aplica el algoritmo de **búsqueda lineal** para encontrar el elemento 27.

```
El elemento 27 no se encuentra en el vector:

índice 1:   5 ≠ 27
índice 2:  10 ≠ 27
índice 3:  24 ≠ 27
índice 4:  25 ≠ 27
índice 5:  28 ≠ 27
índice 6:  29 ≠ 27
índice 7:  40 ≠ 27
índice 8:  42 ≠ 27
índice 9:  55 ≠ 27
índice 10: 59 ≠ 27
índice 11: 60 ≠ 27
índice 12: 63 ≠ 27
```

```
Dado que el vector se encuentra ordenado, también es
correcta la solución que detiene la búsqueda en el
índice 5, puesto que a partir de este índice todos los
elementos serán mayores o iguales que 28, y, por lo
tanto, el 27 no estará entre ellos.
```

**c) [10%]** Aplica el algoritmo de **búsqueda binaria** para encontrar el elemento 26.

```
El elemento 26 no se encuentra en el vector:

I=1 D=12      (1+12)/2 = 13/2 = 6   índice 6:  29 > 26
I=1 D=5       (1+5)/2 = 6/2 = 3     índice 3:  24 < 26
I=4 D=5       (4+5)/2 = 9/2 = 4     índice 4:  25 < 26
I=5 D=5       (5+5)/2 = 10/2 = 5    índice 5:  28 > 26
I=5 D=4                             No encontrado
```

**d) [10%]** Aplica el algoritmo de **búsqueda binaria** para encontrar el elemento 60.

```
El elemento 60 se encuentra en la posición 11 del
vector:

I=1 D=12      (1+12)/2 = 13/2 = 6    índice 6:  29 < 59
I=7 D=12      (7+12)/2 = 19/2 = 9    índice 9:  55 < 59
I=10 D=12     (10+12)/2 = 22/2 = 11  índice 11: 60 = 60
```

## Ejercicio 3: Uso de los algoritmos de búsqueda [20%]

A partir de la siguiente definición de la lista de disponibilidad de los trabajadores de la ONG:

```
const
      MAX_AVAILABILITY: integer = 100;
end const

type
      tDate = record
            day : integer;
            month : integer;
            year : integer;
      end record

      tPerson = record
            document : string;
            name : string;
            surname : string;
            email : string;
            address : string;
            cp : string;
            birthday : tDate;
      end record

      tAvailability = record
            person : tPerson;
            start : tDate;
            end : tDate;
      end record

      tAvailabilityData = record
            elems : vector [MAX_AVAILABILITY] of tAvailability;
            count : integer;
      end record

end type
```

Asumiendo que los elementos de la lista ***tAvailabilityData*** se encuentran ordenados por el campo *document* de la persona, se pide implementar en **lenguaje algorítmico** la función:

```
function checkHasAvailability (availabilities: tAvailabilityData,
document: string) : boolean
```

Que dada la lista de disponibilidad de los trabajadores ***tAvailabilityData*** y el documento de un trabajador, nos devuelva **true** en caso de que la persona tenga al

menos un rango de disponibilidad ***tAvailability*** en la lista y **false** en caso contrario. Para la implementación, hay que seguir el método de **búsqueda binaria**.

```
function checkHasAvailability (availabilities: tAvailabilityData, document:
string) : boolean

var
    left, right, middle : integer;
    found : boolean;
end var

    { Initialize variables }
    left := 1;
    right := availabilities.count;
    found := false;

    { Loop using binary search }
    while (left ≤ right) and not found do
       middle := (left + right) div 2;

       if availabilities.elems[middle].person.document = document then
           found := true;
       else
          if availabilities.elems[middle].person.document < document then
             left := middle + 1;
          else
             right := middle -1;
          end if
       end if
    end while

    return found;

end function
```

## Ejercicio 4: Análisis de algoritmos [30%]

Responde a las preguntas siguientes:

**a)** **[15%]** Dada la siguiente definición del tiempo de ejecución, calcula la función T(n) sin que aparezca ninguna definición recursiva y explica el proceso que has seguido para obtener el resultado.

$$\begin{cases} T(n) = 6, & \text{si } n=0 \\ T(n) = T(n-1) + 4n, & \text{si } n>0 \end{cases}$$

El método para resolver este tipo de ecuaciones consiste en aplicar la definición recursiva de forma repetida un total de *i* veces, hasta que vemos qué forma toma la expresión resultante:

T(n) =
= **T(n-1) + 4n** = (T(n-1-1) + 4(n-1)) + 4n = T(n-2) + 4n - 4 + 4n =
= **T(n-2) + 8n - 4** = (T(n-2-1) + 4(n-2)) + 8n - 4 =
= **T(n-3) + 12n - 12** = (T(n-3-1) + 4(n-3)) + 12n - 12 =
= **T(n-4) + 16n - 24** = … =

$$T(n-i) + 4in - 4\sum_{j=0}^{i-1} j$$

Averiguamos qué valor debe tomar *i* para poder aplicar el caso base de la definición de T(n):

n-i = 0
n=i

A continuación, completamos el cálculo aprovechando que la expresión T(n-i) cuando i = n se transforma en T(0), que según la definición del enunciado es 6:

$$T(n) = T(n-n) + 4nn - 4\sum_{j=0}^{n-1} j = T(0) + 4n^2 - 4\frac{n(n-1)}{2} = 6 + 4n^2 - 2(n^2 - n) = 2n^2 + 2n + 6$$

**b) [15%]** Calcula la complejidad computacional de la función **num_equals** y explica el proceso que has seguido para obtener el resultado:

```
function num_equals (a: vector[N] of real, b: vector[N] of
real) : integer
   var
      res: integer;
      i: integer;
   end var
   res := 0;                                          (1)
   for i:=1 to N do                                   (2)
     if a[i] = b[i] then                              (3)
       res := res + 1;                                (4)
     end if                                           (5)
   end for                                            (6)
   return res;                                        (7)
end function
```

La función **num_equals** empieza con dos asignaciones (línea 1 y 2) que son operaciones elementales con un tiempo constante que se pueden agrupar en $k_1$.

Después hay un bucle (líneas 2-6) que se ejecuta N veces, siendo N la longitud del vector.

El siguiente paso es calcular el coste de una iteración del bucle, que es la suma de una secuencia de operaciones elementales con coste constante: las condiciones (líneas 2 y 3), la asignación con operación aritmética (línea 4), el incremento de la variable contador *i* del bucle (línea 2) y los accesos a los vectores a y b (línea 3). El resultado de esta suma es el coste de una iteración y lo agruparemos en el valor constante $k_2$.

El tiempo total del bucle es la multiplicación del coste de una iteración por el número de iteraciones: $k_2 \cdot N$

Finalmente, la función del tiempo de ejecución de **num_equals** es:

$$T(N) = k_1 + k_2 N$$

Por tanto, la complejidad es **lineal**, O(N).